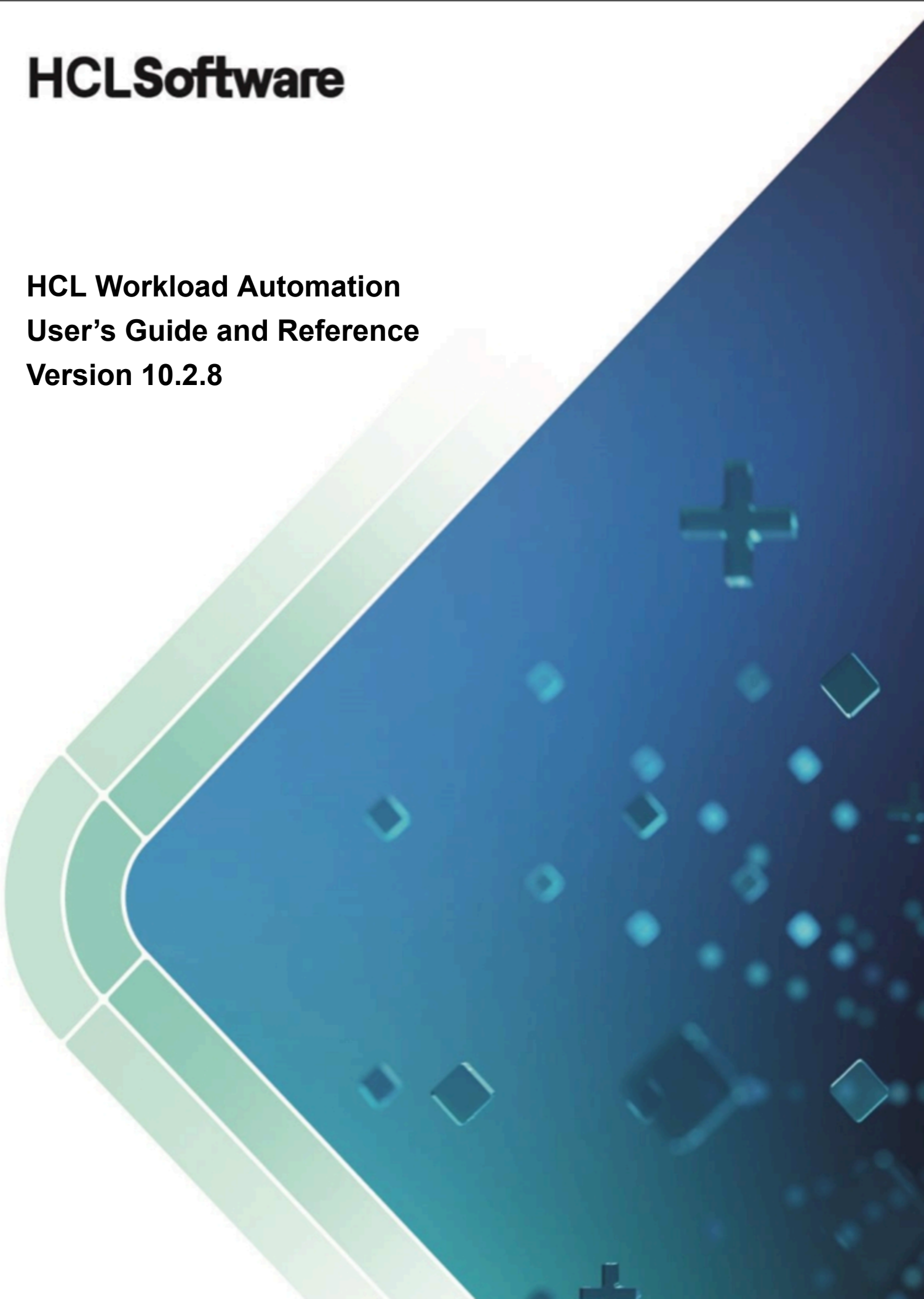
HCLSoftware

HCL Workload Automation
User's Guide and Reference
Version 10.2.8



Note

Before using this information and the product it supports, read the information in Notices on page mcdxxxvi.

This edition applies to version 10, release 2, modification level 8 of HCL Workload Automation (program number 5698-T09) and to all subsequent releases and modifications until otherwise indicated in new editions.

Contents

List of Figures..... x
List of Tables..... xii
About this publication..... xvi
What is new in this release..... xvi
What is new in this publication..... xvi
Conventions used in this publication..... xvi
Typeface conventions..... xvi
Operating system-dependent variables and paths..... xvii
Command syntax..... xvii
Chapter 1. HCL Workload Automation..... 19
Understanding basic concepts..... 19
HCL Workload Automation database objects..... 19
The HCL Workload Automation network..... 40
Configuring your runtime environment..... 41
Defining scheduling activities..... 42
Controlling job and job stream processing..... 42
Managing production scheduling activities..... 50
Automating workload using event rules..... 51
HCL Workload Automation user interfaces..... 52
Starting production..... 54
Chapter 2. Understanding basic processes and commands..... 57
Issuing commands on Windows operating systems..... 57
HCL Workload Automation workstation processes..... 57
Starting and stopping processes on a workstation..... 63
Workstation inter-process communication..... 66
HCL Workload Automation network communication..... 70
Support for Internet Protocol version 6..... 73
Chapter 3. Configuring the job environment..... 74
Job environment overview..... 74
Environment variables exported by jobman..... 75
Customizing date formatting in the stdlist..... 82
Customizing job processing on a UNIX workstation - jobmanrc..... 83
Customizing the MAIL_ON_ABEND section of jobmanrc..... 85
Customizing job processing for a user on UNIX workstations-.jobmanrc..... 86
Customizing job processing on a Windows workstation..... 88
Customizing the MAIL_ON_ABEND section of jobmanrc.cmd..... 91
Customizing job processing on a Windows workstation - djobmanrc.cmd..... 91
Setting up options for using the user interfaces..... 93
Chapter 4. Managing the production cycle..... 96
Plan management basic concepts..... 96
Preproduction plan..... 98
Identifying job stream instances in the plan..... 99
Managing external follows dependencies for jobs and job streams..... 100
Production plan..... 114
Understanding carry forward options..... 115
Trial plan..... 117
Forecast plan..... 118
Customizing plan management using global options..... 119
Creating and extending the production plan..... 125
JnextPlan..... 127
Planman command line..... 129
Creating an intermediate production plan..... 130
Creating an intermediate plan for a plan extension..... 132
Retrieving the production plan information..... 133
Creating a trial plan..... 134
Creating a trial plan of a production plan extension..... 135
Creating a forecast plan..... 136
Deploying rules..... 138
Unlocking the production plan..... 139
Resetting the production plan..... 139
Removing the preproduction plan..... 140
Replicating plan data in the database..... 140
Monitoring the replication of plan data in the database..... 142
The stageman command..... 143
Managing concurrent accesses to the Symphony file..... 145
Scenario 1: Access to Symphony file locked by other HCL Workload Automation processes..... 145
Scenario 2: Access to Symphony file locked by stageman..... 145
Managing follows dependencies using carry forward prompt..... 146
The logman command..... 146
Starting production plan processing..... 149
Automating production plan processing..... 150
Detecting loops between jobs and job streams in current and trial plans..... 151
Chapter 5. Using workload service assurance..... 153
Enabling and configuring workload service assurance..... 153
Planning critical jobs..... 157

Processing and monitoring critical jobs.....159
Workload service assurance scenario.....161
Promoting jobs scheduled on dynamic pools.....162
Chapter 6. Customizing your workload using variable tables.....164
The default variable table..... 164
Data integrity for variable tables..... 165
Locking mechanism for variable tables..... 165
Variable table security.....165
Variable resolution.....166
Chapter 7. Condition-based workload automation..... 169
A business scenario..... 172
Chapter 8. Running event-driven workload automation.....174
The event rule management process.....177
Using the involved interfaces and commands... 179
Defining event rules..... 182
Security checks on event rules.....184
Event rule examples.....185
Rule operation notes.....192
Triggered rule elements..... 194
Defining custom events.....195
Chapter 9. Defining objects in the database..... 196
Defining scheduling objects.....196
Workstation definition.....201
Workstation class definition.....224
Domain definition..... 227
Job definition.....228
User definition..... 281
Calendar definition..... 286
Folder definition.....287
Variable and parameter definition.....289
Variable table definition.....295
Prompt definition.....298
Resource definition..... 300
Run cycle group definition.....301
Job stream definition.....314
Job stream definition keyword details.....325
Variable table.....394
Event rule definition..... 394
Workload application definition.....412
Security object definition..... 414
Chapter 10. Managing objects in the database - composer.....434
Setting up the composer command-line program..... 434
Setting up the composer environment..... 435
Running the composer program..... 437
Running commands from composer.....441
Filters and wildcards.....442
Delimeters and special characters.....446
Composer return codes..... 447
COMPOSER COMMANDS.....447
Referential integrity check..... 449
Add..... 454
Authenticate.....457
Chfolder.....457
Continue..... 459
Delete..... 460
Display.....465
Edit.....471
Exit.....471
Extract..... 472
Help..... 478
List.....479
Listfolder..... 488
Lock..... 488
Mkfolder..... 493
Modify..... 495
New..... 501
Print..... 504
Redo..... 504
Rmfolder.....505
Rename..... 506
Renamefolder..... 510
Replace.....511
System command..... 512
Unlock..... 513
Update.....517
Validate..... 520
Version..... 521
Organizing scheduling objects into folders.....521
Chapter 11. Managing workload applications.....527
Reusing a workload in another environment.....528
Resolving the mapping file.....529
Using regular expressions to modify the mapping file.....535
Deploying a workload application.....537
Running the wappman program..... 538
Chapter 12. Managing objects in the plan - conman..... 545
Setting up the conman command-line program..... 545
Setting up the conman environment.....546
Running the conman program.....548
Running commands from conman..... 552
Wildcards..... 553
Delimiters and special characters.....554
Conman commands processing..... 555

Selecting jobs in commands..... 556
Syntax.....556
Arguments..... 556
Selecting job streams in commands.....571
Syntax.....571
Arguments..... 571
Conman return codes.....581
Conman commands..... 582
Adddep job.....586
Addep Sched..... 589
Altjob..... 592
Altpass..... 593
Altpri..... 595
Cancel job..... 596
Cancel sched..... 598
Checkhealthstatus.....599
Chfolder.....600
Confirm.....602
Console..... 605
Continue..... 607
Deldep job..... 607
Deldep sched..... 609
Deployconf..... 611
Display..... 612
Exit..... 614
Fence..... 615
Help..... 616
Kill..... 618
Limit cpu..... 619
Limit sched..... 620
Link..... 621
Listfolder..... 623
Listsym..... 623
Listsucc..... 626
Recall..... 627
Redo..... 628
Release job..... 630
Release sched..... 632
Reply..... 634
Rerun..... 635
Rerunsucc..... 639
ResetFTA..... 642
Resource..... 643
Setsym..... 644
Showcpus..... 645
Showdomain..... 654
Showfiles..... 656
Showjobs..... 659
Showprompts..... 683
Showresources..... 686
Showschedules..... 690
Shutdown..... 696
Start..... 697
Startappserver..... 699
Starteventprocessor..... 700
Startmon..... 700
Status..... 701
Stop..... 702
Stop ;progressive..... 703
Stopappserver..... 704
Stopeventprocessor..... 705
Stopmon..... 706
Submit docommand..... 707
Submit file..... 711
Submit job..... 715
Submit sched..... 720
Switcheventprocessor..... 725
Switchmgr..... 726
System command..... 727
Tellop..... 728
Unlink..... 729
Version..... 730
Chapter 13. Using Orchestration Query Language..... 732
Chapter 14. Orchestration CLI..... 757
Configuring Orchestration CLI..... 757
Running commands from Orchestration CLI..... 762
Authenticating Orchestration CLI using API Keys..... 763
Orchestration CLI commands..... 764
Access required to run Orchestration CLI commands..... 766
Context commands..... 770
Model commands..... 775
Plan commands..... 808
help..... 881
version..... 881
Special characters as wildcards and delimiters..... 882
Orchestration CLI return codes..... 884
Managing multiple config.yaml files..... 887
Managing multiple formats..... 888
Chapter 15. Modifying context parameter temporarily... 890
Chapter 16. Monitoring HCL Workload Automation 891
Exposing metrics to monitor your workload..... 891
Frequently asked questions about metrics and monitoring..... 894
The BmEvents configuration file..... 896

Logged events....899
Enabling observability with OpenTelemetry....903
Leveraging Instana for advanced observability....905
Chapter 17. Extending HCL Workload Automation capabilities....910
Remote command jobs....910
Creating custom plug-ins....914
Chapter 18. Using utility commands....915
Command descriptions....915
at and batch....918
Cpuinfo....921
Dataexport....924
Dataimport....925
Datecalc....926
da_test_connection....932
DELETE....933
Evtdef....934
Evtsize....938
Filemonitor....941
Jobinfo....947
Jobstdl....949
Maestro....952
Makecal....953
Metronome....955
Morestdl....955
Parms....958
Release....961
Rmstdlist....965
Sendevent....967
Showexec....969
Shutdown....970
ShutDownLwa....971
StartUp....971
StartUpLwa....972
version....973
wa_pull_info....974
Unsupported commands....977
Chapter 19. Using utility commands in the dynamic environment....978
Command-line configuration file....979
cleanuserjobs....983
exportserverdata....984
importserverdata....986
jobprop....987
movehistorydata....988
param....990
resource....993
Using the resource command from an agent...1003
sendevent....1005
twstrace....1007
Chapter 20. Getting reports and statistics....1010
Setup for using report commands....1010
Changing the date format....1011
Command descriptions....1011
Rep1 - rep4b....1012
Rep7....1014
Rep8....1015
Rep11....1017
Reptr....1018
Xref....1020
Sample report outputs....1021
Report 01 - Job Details Listing....1021
Report 02 - Prompt Listing....1024
Report 03 - Calendar Listing....1024
Report 04A - Parameter Listing....1025
Report 04B - Resource Listing....1025
Report 07 - Job History Listing....1026
Report 08 - Job Histogram....1027
Report 9B - Planned Production Detail....1027
Report 10B - Actual Production Detail....1028
Report 11 - Planned Production Schedule....1029
Report 12 - Cross Reference Report....1031
Report extract programs....1033
Jbxtract....1034
Prxtract....1036
caxtract....1037
Paxtract....1038
Rextract....1039
R11xtr....1040
Xrxtrct....1042
Running reports and batch reports....1048
Historical reports....1051
Production reports....1056
Running batch reports from the command line interface....1057
Chapter 21. Managing time zones....1064
Enabling time zone management....1064
How HCL Workload Automation manages time zones....1065
Moving to daylight saving time on....1067
Moving to daylight saving time off....1068
General rules....1068
Chapter 22. Integrating with SAP and PeopleSoft....1070
Access method interface....1071
Method command line syntax....1071
Method response messages....1073

Method options file....1074
Running methods....1077
Launch job task (LJ)....1077
Manage job task (MJ)....1078
Check file task (CF)....1079
Get status task (GS)....1079
Cpuinfo command for extended agents only...1081
Access methods....1081
Installing and configuring the access methods....1081
Access method for PeopleSoft....1096
Common serviceability for the access methods....1112
Integration with SAP 1124
Introducing HCL Workload Automation for SAP....1125
Access method for SAP....1128
Scheduling jobs on HCL Workload Automation from SAP Solution Manager....1309
Troubleshooting....1319
Job standard list error messages....1319
Method not executable....1319
Console Manager messages for extended agents only....1319
Composer and compiler messages for extended agents only....1320
Jobman messages for extended agents only..1320
Chapter 23. Managing internetwork dependencies.....1322
Internetwork dependencies overview....1322
Understanding how an internetwork dependency is shown....1323
Configuring a network agent....1324
A sample network agent definition....1325
Defining an internetwork dependency....1326
Managing internetwork dependencies in the plan...1327
States of jobs defined in the EXTERNAL job stream....1327
Working with jobs defined in the EXTERNAL job stream....1328
Sample internetwork dependency management scenarios....1329
Internetwork dependencies in a mixed environment....1331
Chapter 24. Applying conditional branching logic.....1333
Setting up conditional dependencies....1336
Joining or combining conditional dependencies.....1339
Scheduling and submitting conditional dependencies....1340
Evaluating and processing a conditional dependency flow....1341
Monitoring conditional dependencies....1347
Plan handling of conditional dependencies....1351
Chapter 25. Defining and managing cross dependencies....1352
An introduction to cross dependencies....1352
Processing flow across the distributed scheduling environment....1354
Defining a cross dependency....1357
Monitoring a cross dependency resolution in the production plan....1359
How the shadow job status changes until a bind is established....1359
How the shadow job status changes after the bind is established....1365
How to see why the shadow job status is FAIL....1367
Shadow job status during the remote job recovery or rerun....1367
How carry forward applies to shadow jobs....1367
Managing shadow jobs in the production plan....1368
Chapter 26. Managing an IBM i dynamic environment. 1369
Defining agents on IBM i systems....1369
Defining IBM i jobs....1369
Managing agents on IBM i systems....1370
Starting and stopping agents on IBM i systems....1370
Using utility commands on agents on IBM i systems....1370
Scheduling jobs on IBM i systems....1371
The agent joblog and TWSASPOOLS environment variable....1373
Child job monitoring on IBM i agents....1374
The agent return code retrieval....1378
Controlling the job environment with the user return code....1378
Alternative method to set the user return code....1379
Appendix A. Event-driven workload automation event and action definitions....1381
Event providers and definitions....1381
TWSObjectsMonitor events....1381
FileMonitor events....1385
TWSApplicationMonitor events....1395
DatasetMonitor events....1396
Action providers and definitions....1397
GenericAction actions....1398
MailSender actions....1399
MessageLogger actions....1399
ServiceNow actions....1399
TWSAction actions....1400
TWSForZosAction actions....1400
Appendix B. Quick reference for commands....1403
Managing the plan....1403
Managing objects in the database....1405

General purpose commands....1405
Scheduling objects....1405
Composer commands....1413
Managing objects in the plan....1420
Conman commands....1420
Utility commands....1428
Report commands....1432
Appendix C. Accessibility....1435
Notices....mcdxxxvi
Index....1440

List of Figures

Figure 1: Single-domain network.....37

Figure 2: Multiple-domain network.....38

Figure 3: Process tree in UNIX®..... 62

Figure 4: Process tree in Windows®.....63

Figure 5: Inter-process communication on the master domain manager..... 69

Figure 6: Inter-process communication on the master domain manager and fault-tolerant agent.....70

Figure 7: Sameday matching criteria..... 101

Figure 8: Closest preceding matching criteria.....102

Figure 9: Within a relative interval matching criteria..... 103

Figure 10: Within an absolute interval matching criteria... 104

Figure 11: Closest preceding predecessor job 105

Figure 12: Pending predecessor instance..... 106

Figure 13: Sameday matching criteria - Step 1: at Start of Day (SOD) on a Thursday.....107

Figure 14: Sameday matching criteria - Step 2: at 9:00 108

Figure 15: Sameday matching criteria - Step 3: at 15:00... 108

Figure 16: Closest preceding matching criteria - Step 1: before 08:00.....109

Figure 17: Closest preceding matching criteria - Step 2: at 08:00 on weekdays except Thursdays and Fridays.....110

Figure 18: Closest preceding matching criteria - Step 3: at 09:00 on Thursdays and Fridays.....110

Figure 19: Closest preceding matching criteria - Step 4: at 15:00 on every day 111

Figure 20: Relative Interval matching criteria - at start of day on Thursday..... 113

Figure 21: Absolute interval matching criteria - at start of day on Thursday..... 114

Figure 22: Critical path.....160

Figure 23: Condition-based workload automation..... 171

Figure 24: User definition.....285

Figure 25: Example when start of day conversion is not applied.....1066

Figure 26: Example when start of day conversion is applied.....1067

Figure 27: Defining an Extended Agent workstation..... 1092

Figure 28: Defining an Extended Agent job for end-to-end scheduling.....1095

Figure 29: Command syntax..... 1171

Figure 30: The Variant List panel..... 1176

Figure 31: The Variant Information page of the Variant List panel.....1177

Figure 32: Job definition syntax.....1181

Figure 33: The Raise Event panel.....1196

Figure 34: Job definition syntax.....1201

Figure 35: The Table Criteria panel..... 1228

Figure 36: The Table Criteria panel..... 1231

Figure 37: Job definition syntax.....1237

Figure 38: Dynamic Workload Console - Table of results. 1242

Figure 39: Dynamic Workload Console - Details of a process chain job..... 1243

Figure 40: Command syntax..... 1258

Figure 41: Command syntax..... 1265

Figure 42: Managing high priority IDocs overview..... 1271

Figure 43: A monitor and its MTEs - © SAP AG 2009. All rights reserved.....1282

Figure 44: Name and description of an MTE - © SAP AG 2009. All rights reserved..... 1285

Figure 45: Command syntax..... 1290

Figure 46: Command syntax..... 1292

Figure 47: Local and remote networks..... 1325

Figure 48: A follows dependency on the ABSENCES job..1341

Figure 49: Two different conditional dependencies on SUCC and ABEND statuses on the ABSENCES job..... 1342

Figure 50: Conditional dependencies on output conditions on the ABSENCES job..... 1343

Figure 51: A join dependency containing three dependencies on SUCC status..... 1344

Figure 52: Status conditional dependency on a job with **recovery stop** setting..... 1345

Figure 53: ABEND status conditional dependency..... 1345

Figure 54: STATUS_OK output condition..... 1346

Figure 55: ERROR output condition..... 1346

Figure 56: STATUS_OK output condition..... 1347

Figure 57: Cross dependency logic..... 1354

Figure 58: Shadow job status transition until the bind is established..... 1360

Figure 59: Instance to be bound if the shadow job scheduled time is included in the CP interval..... 1363

Figure 60: Instance to be bound if the instance that most closely precedes the shadow job scheduled time exists in the LTP but was canceled from the CP..... 1364

Figure 61: The scheduled time of the shadow job is included in the CP but no instance to bind exists..... 1364

Figure 62: The instance to be bound exists but it is not yet included in the CP..... 1365

Figure 63: The LTP interval still does not contain the shadow job scheduled time..... 1365

Figure 64: Shadow job status transition chain after the bind was established..... 1366

List of Tables

Table 1: Command syntax....xvii
Table 2: iCalendar RRule generator parameters....23
Table 3: Recurrence parameters compatibility....24
Table 4: Scenario 1. No time restriction in the run cycle group....28
Table 5: Scenario 2. Time restriction in the run cycle group without offset....29
Table 6: Scenario 3. Time restriction in the run cycle group with offset (+1 12:00)....29
Table 7: Starting and stopping HCL Workload Automation on a workstation....64
Table 8: Starting and stopping the agent 66
Table 9: Job environment variables for Windows®....75
Table 10: Job environment variables for UNIX®....79
Table 11: Variables defined by default in the jobmanrc file....83
Table 12: Variables defined by default in the jobmanrc.cmd file....89
Table 13: Carry forward global options settings....115
Table 14: Resulting carry forward settings....116
Table 15: Workload service assurance global options....154
Table 16: Workload service assurance local options....156
Table 17: The relationship between variable tables and their enclosed variables in the HCL Workload Automation security file....166
Table 18: conman commands for managing monitoring engines....178
Table 19: conman commands for managing the event processing server....179
Table 20: Interfaces and commands for managing event-driven workload automation....180
Table 21: List of supported scheduling object keywords..197
Table 22: List of supported security object keywords....198
Table 23: List of reserved words when defining jobs and job streams....198
Table 24: List of reserved words when defining workstations....199
Table 25: List of reserved words when defining users....200
Table 26: Attribute settings for management workstation types....202
Table 27: Attribute settings for target workstation types..204
Table 28: Type of communication depending on the security level value....219
Table 29: Examples: renaming the job definition....230
Table 30: Comparison operators....235
Table 31: Logical operators....236
Table 32: Recovery options and actions....240
Table 33: Built-in HCL Workload Automation variables....248
Table 34: Required and optional attributes for the definition of an IBM i job....251
Table 35: Required and optional attributes for the definition of an executable job....255
Table 36: Required and optional attributes for the definition of an access method job....256
Table 37: Supported HCL Workload Automation variables in JSDL definitions....268
Table 38: Properties for shadow jobs....273
Table 39: Properties for IBM Sterling Connect:Direct jobs....274
Table 40: Properties for file transfer jobs....274
Table 41: Properties for JSR 352 Java Batch jobs 276
Table 42: Properties for Job Management jobs....277
Table 43: Properties for Job Stream Submission jobs....277
Table 44: Properties for database jobs....277
Table 45: Folder commands....288

Table 46: How to handle a backslash in variable substitution..... 291

Table 47: Keywords that can take local parameters in submit commands..... 292

Table 48: Required access keyword on variable table in Security file (vartable object) and allowed actions..... 297

Table 49: List of scheduling keywords..... 317

Table 50: Explanation of the notation defining the number of occurrences for a language element..... 395

Table 51: `TWSObjectsMonitor` events..... 401

Table 52: `TWSApplicationMonitor` events..... 403

Table 53: `FileMonitor` events..... 403

Table 54: `DatasetMonitor` events..... 403

Table 55: Action types by action provider..... 406

Table 56: Security object types..... 420

Table 57: Actions that users or groups can perform on the different objects..... 421

Table 58: Actions that users or groups can perform when designing and monitoring the workload..... 423

Table 59: Actions that users or groups can perform when modifying current plan..... 423

Table 60: Actions that users or groups can perform when submitting workload 424

Table 61: Actions that users or groups can perform when managing the workload environment..... 425

Table 62: Actions that users or groups can perform when managing event rules..... 426

Table 63: Administrative tasks that users or groups can perform 427

Table 64: Actions that users or groups can perform on workload reports..... 427

Table 65: Actions that users or groups can perform on folders..... 428

Table 66: Attributes for security object types..... 429

Table 67: Scheduling objects filtering criteria 443

Table 68: Delimeters and special characters for composer..... 446

Table 69: List of composer commands..... 448

Table 70: Object identifiers for each type of object defined in the database..... 450

Table 71: Object definition update upon deletion of referenced object..... 450

Table 72: Referential integrity check when deleting an object from the database..... 451

Table 73: Objects extracted during the export process.... 530

Table 74: Resolving the mapping file..... 534

Table 75: Delimiters and special characters for conman.. 554

Table 76: List of conman commands..... 583

Table 77: State change after confirm command..... 603

Table 78: Recovery options retrieval criteria..... 636

Table 79: Successors status..... 640

Table 80: OQL syntax keywords..... 733

Table 81: OQL syntax fields for workstation..... 734

Table 82: OQL syntax fields for job stream..... 737

Table 83: OQL syntax fields for jobs..... 745

Table 84: OQL syntax fields for resources..... 753

Table 85: Mandatory fields..... 758

Table 86: Optional fields..... 758

Table 87: Scheduling items..... 777

Table 88: Output format for displaying items..... 779

Table 89: Scheduling items..... 782

Table 90: Scheduling items..... 786

Table 91: Output format for displaying items..... 788

Table 92: Scheduling items..... 790

Table 93: Scheduling items..... 794

Table 94: Scheduling items..... 797

Table 95: Scheduling items..... 802

Table 96: Scheduling items..... 806

Table 97: Change in status after confirm command.....820
Table 98: 834
Table 99: Special characters as delimiters.....883
Table 100: 884
Table 101: Workload Automation exposed metrics..... 891
Table 102: Events filtered by CHSCHED..... 897
Table 103: Logged events.....899
Table 104: Required and optional attributes for the definition of a remote command job.....911
Table 105: List of utility commands..... 915
Table 106: Additional properties that can be used for defininig custom events..... 936
Table 107: List of utility commands for dynamic workstations..... 978
Table 108: Date formats.....1011
Table 109: List of report commands..... 1011
Table 110: Report extract programs..... 1033
Table 111: Jbxtract output fields..... 1035
Table 112: Prxtract output fields.....1037
Table 113: Caxtract output fields.....1038
Table 114: Paxtract output fields.....1039
Table 115: Rextract output fields..... 1040
Table 116: R11xtr output fields.....1041
Table 117: Xdep_job output fields.....1043
Table 118: Xdep_job output fields (continued).....1043
Table 119: Xdep_sched output fields.....1044
Table 120: Xfile output fields.....1045
Table 121: Xjob output fields..... 1045
Table 122: Xprompts output fields.....1046
Table 123: Xresource output fields.....1046
Table 124: Xsched output fields.....1047
Table 125: Xwhen output fields.....1048
Table 126: Supported report output formats.....1050
Table 127: Summary of historical reports..... 1051
Table 128: 1057
Table 129: Method command task options.....1072
Table 130: Launch job task (LJ) messages.....1078
Table 131: Check file task (CF) messages..... 1079
Table 132: Get status task (GS) messages.....1080
Table 133: How to complete the extended agents definition..... 1087
Table 134: Roles and responsibilities in Access method for PeopleSoft.....1097
Table 135: Psagent access method options..... 1099
Table 136: Task string parameters for PeopleSoft jobs.. 1108
Table 137: Relationship between the run status, the distribution status, and the HCL Workload Automation job status.....1111
Table 138: Relationship between the run status and the HCL Workload Automation job status..... 1112
Table 139: Job states and return codes for the PeopleSoft access method.....1116
Table 140: HCL Workload Automation for SAP features..1126
Table 141: Roles and responsibilities in HCL Workload Automation for SAP.....1129
Table 142: Access keywords for activities with SAP scheduling objects..... 1131
Table 143: ABAP/4 modules installed 1139
Table 144: ABAP/4 modules contents.....1140
Table 145: r3batch global configuration options..... 1148
Table 146: r3batch local configuration options.....1150
Table 147: r3batch common configuration options.....1154
Table 148: Placeholders and counters for extended variants.....1178
Table 149: Task string parameters for SAP jobs.....1181
Table 150: Status transitions in HCL Workload Automation (internal status) and the corresponding SAP R/3 status. 1193

Table 151: Task string parameters for SAP jobs (dynamic definition)..... 1201

Table 152: Supported attributes for ABAP step definition..... 1211

Table 153: Supported attributes for external programs and external commands step definition..... 1214

Table 154: Placeholders for job interception template files..... 1232

Table 155: Task string parameters for SAP jobs..... 1238

Table 156: Actions performed when you rerun a process chain job 1244

Table 157: Parameters to define an SAP internetwork dependency 1263

Table 158: Internetwork dependency definition and possible resolution..... 1264

Table 159: History table of the SAP events raised..... 1268

Table 160: SAP event matching with the event rule defined..... 1268

Table 161: History table of the SAP events raised..... 1269

Table 162: SAP events matching with the event rule defined..... 1269

Table 163: HCL Workload Automation fields used to define event rules based on IDocs..... 1272

Table 164: HCL Workload Automation fields used to define correlation rules for IDoc events..... 1273

Table 165: Parameters of IDOCEventGenerated event type..... 1275

Table 166: Standard outbound IDoc statuses..... 1276

Table 167: Standard inbound IDoc statuses..... 1278

Table 168: Mapping between root context MTE name and HCL Workload Automation fields..... 1286

Table 169: Mapping between summary context MTE name and HCL Workload Automation fields..... 1286

Table 170: Mapping between object MTE name and HCL Workload Automation fields..... 1287

Table 171: Mapping between attribute MTE name and HCL Workload Automation fields..... 1288

Table 172: Alert properties for correlations..... 1288

Table 173: SAP supported code pages..... 1296

Table 174: Miscellaneous troubleshooting items..... 1297

Table 175: Properties for the `smseadapter.properties` file..... 1311

Table 176: Internetwork dependencies in a mixed environment..... 1331

Table 177: Shadow job status transition..... 1354

Table 178: Matching criteria for distributed shadow jobs..... 1358

Table 179: Regular expression syntax..... 1386

Table 180: Regular expression examples..... 1388

Table 181: `SMF` events..... 1396

Table 182: Parameters of ReadCompleted and ModificationCompleted event types..... 1396

Table 183: Commands used against the plan..... 1403

Table 184: General purpose commands..... 1405

Table 185: Composer commands..... 1414

Table 186: Commands that can be run from conman.... 1421

Table 187: Utility commands available for both UNIX® and Windows®..... 1429

Table 188: Utility commands available for UNIX® only... 1432

Table 189: Utility commands available for Windows® only..... 1432

Table 190: Report commands..... 1433

Table 191: Report extract programs..... 1434

About this publication

HCL Workload Automation simplifies systems management across distributed environments by integrating systems management functions. HCL Workload Automation plans, automates, and controls the processing of your enterprise's entire production workload. The *HCL Workload Automation User's Guide and Reference* provides detailed information about the command line interface, scheduling language, and utility commands for HCL Workload Automation.

What is new in this release

Learn what is new in this release.

For information about the new or changed functions in this release, see *Overview*, section *Summary of enhancements*.

New or changed content is marked with revision bars.

What is new in this publication

Learn what is new in this publication.

APARs and defects have been fixed. All changes are marked with revision bars.

Conventions used in this publication

Learn what conventions are used in this publication.

This publication uses several conventions for special terms and actions, operating system-dependent commands and paths, command syntax, and margin graphics.

Typeface conventions

This publication uses the following typeface conventions:

Bold

- Lowercase commands and mixed case commands that are otherwise difficult to distinguish from surrounding text
- Interface controls (check boxes, push buttons, radio buttons, spin buttons, fields, folders, icons, list boxes, items inside list boxes, multicolumn lists, containers, menu choices, menu names, tabs, property sheets), labels (such as **Tip:**, and **Operating system considerations**:)
- Keywords and parameters in text

Italic

- Words defined in text
- Emphasis of words (words as words)

- New terms in text (except in a definition list)
- Variables and values you must provide

Monospace

- Examples and code examples
- File names, programming keywords, and other elements that are difficult to distinguish from surrounding text
- Message text and prompts addressed to the user
- Text that the user must type
- Values for arguments or command options

Operating system-dependent variables and paths

This publication uses the UNIX® convention for specifying environment variables and for directory notation, except where the context or the example path is specifically Windows.

When using the Windows® command line, replace **$***variable* with **%** *variable***%** for environment variables and replace each forward slash (**/**) with a backslash (****) in directory paths. The names of environment variables are not always the same in Windows® and UNIX® environments. For example, %TEMP% in Windows® is equivalent to $tmp in UNIX® environments.

Note: If you are using the bash shell on a Windows® system, you can use the UNIX® conventions.

Command syntax

This publication uses the following syntax wherever it describes commands:

Table 1. Command syntax

Syntax convention	Description	Example
Name of command	The first word or set of consecutive characters.	**conman**
Brackets ([ ])	The information enclosed in brackets ([ ]) is optional. Anything not enclosed in brackets must be specified.	[**-file** *definition_file*]
Braces ({ })	Braces ({ }) identify a set of mutually exclusive options, when one option is required.	{**-prompts** | **-prompt** *prompt_name* }
Underscore (_)	An underscore (_) connects multiple words in a variable.	*prompt_name*
Vertical bar (|)	Mutually exclusive options are separated by a vertical bar (|).	{**-prompts** | **-prompt** *prompt_name* }

Table 1. Command syntax (continued)

Syntax convention	Description	Example
	You can enter one of the options separated by the vertical bar, but you cannot enter multiple options in a single use of the command.	
Bold	**Bold** text designates literal information that must be entered on the command line exactly as shown. This applies to command names and non-variable options.	**composer add** *file_name*
Italic	Italic text is variable and must be replaced by whatever it represents. In the example to the right, the user would replace file_name with the name of the specific file.	*file_name*
Ellipsis (…)	An ellipsis (…) indicates that the previous option can be repeated multiple times with different values. It can be used inside or outside of brackets.	[**–x** *file_name*]… An ellipsis outside the brackets indicates that **–x** *file_name* is optional and may be repeated as follows: ***–x*** *file_name1* ***–x*** *file_name2****–x*** *file_name3* [**–x** *file_name*…] An ellipsis inside the brackets indicates that **–x** *file_name* is optional, and the file variable can be repeated as follows: ***–x*** *file_name1 file_name2 file_name3* **–x** *file_name* [**–x** *file_name*]… An ellipsis used with this syntax indicates that you must specify **–x** *file_name* at least once.

Chapter 1. HCL Workload Automation overview

HCL Workload Automation provides you with the ability to manage your production environment and automate many operator activities. HCL Workload Automation manages job processing, resolves interdependencies, and launches and tracks jobs. Because jobs start as soon as their dependencies are satisfied, idle time is minimized and throughput is significantly improved. If a job fails, HCL Workload Automation manages the recovery process with little or no operator intervention.

This chapter is divided into the following sections:

- Understanding basic concepts on page 19
- HCL Workload Automation user interfaces on page 52
- Starting production on page 54

Understanding basic concepts

This section describes the basic concepts of HCL Workload Automation and is divided into the following sections:

- HCL Workload Automation database objects on page 19
- The HCL Workload Automation network on page 40
- Configuring your HCL Workload Automation runtime environment on page 41
- Defining scheduling activities using HCL Workload Automation on page 42
- Managing production scheduling activities with HCL Workload Automation on page 50

HCL Workload Automation database objects

This section introduces the HCL Workload Automation database objects that you work with. The following database objects are described:

- Job, see Job on page 20
- Job stream, see Job stream on page 20
- Workload application, see Workload application on page 21
- Run cycle, see Run cycle on page 22
- Run cycle group, see Run cycle group on page 25
- Calendar, see Calendar on page 30
- Prompt, see Prompt on page 31
- Workstation, see Workstation on page 31
- Workstation class, see Workstation class on page 36
- Domain, see Domain on page 37
- Event rule, see Event rule on page 39
- Resource, see Resource on page 39
- Parameter, see Parameter on page 39
- Variable table, see Variable table on page 40

Job

A *job* is a unit of work specifying an action, such as a weekly data backup, to be performed on specific workstations in the HCL Workload Automation network. In a HCL Workload Automation distributed environment, jobs can be defined either independently from job streams or within a job stream definition. smart workflow folders.

Job types can be divided between existing HCL Workload Automation jobs and job types with advanced options. The existing job types are standard jobs with generic scripts or commands you customize according to your needs. The job types with advanced options are jobs designed to perform specific operations, such as database, file transfer, Java, and web service operations. You schedule these jobs types only on dynamic agents, pools and dynamic pools.

If you want to leverage the dynamic capability when scheduling job types with advanced options, you schedule them on pools and dynamic pools, which assign the job dynamically to the best available resource. If you are interested only in defining job types with advanced options, without using the dynamic scheduling capability, you schedule these jobs on a specific agent on which the job runs statically.

Regardless whether the HCL Workload Automation engine is distributed or z/OS based, you can define locally a *shadow job* to map a remote job instance running on a different HCL Workload Automation engine.

For information about how to define jobs, see Job definition on page 228.

For information about how to define workstations, see Workstation definition on page 201.

Once job definitions have been submitted into the production plan, you still have the opportunity to make one-off changes to the definitions before they run, or after they have run. You can update the definition of a job that has already run and then rerun it. The job definition in the database remains unchanged.

Job stream

A *job stream* is a sequence of jobs to be run, together with times, priorities, and other dependencies that determine the order of processing. Each job stream is assigned a time to run, represented by run cycle with type calendar, set of dates, or repetition rates. Job streams can be defined in a specific smart workflow folder if you want to organize them by line of business or some other custom category.

Dependencies in a distributed environment:

You can have dependencies between both jobs and job streams. They can be:

Internal dependencies

These are dependencies established between jobs belonging to the same job stream.

External dependencies

These are dependencies between job streams, or between job streams and jobs belonging to other job streams, or between jobs belonging to different job streams.

Internetwork dependencies

These are dependencies on jobs or job streams running in another HCL Workload Automation network. Internetwork dependencies require a network agent workstation to communicate with the external HCL Workload Automation network.

Dependencies on resources are supported by HCL Workload Automation both in the distributed and in the z/OS environments.

For information about how to define job streams, see Job stream definition on page 314.

Workload application

A workload application is one or more job streams together with all the referenced jobs that can be shared with other HCL Workload Automation environments through an easy deployment process.

A workload application is an HCL Workload Automation database object that acts as a container for one or more job streams. You can use workload applications to standardize a workload automation solution so that the solution can be reused in one or more HCL Workload Automation environments thereby automating business processes.

You can prepare a workload application template in a source HCL Workload Automation environment and then export it so that it can be deployed in a target environment. The export process extracts from the source environment all of the elements necessary to reproduce the solution in another environment. It produces a compressed file containing a number of files required to import the workload application into the target environment. These files contain a definition of the objects in the source environment extracted from the HCL Workload Automation database. For those elements that depend on the topology of the target environment, some manual configuration is required. For example, the definitions extracted from the source environment contain references to workstations that do not exist in the target environment. For this reason, before proceeding with the import, a mapping of some of the elements must be made associating the name of the object in the target environment.

The exported workload application template contains definitions or references for all of the following objects:

- Job streams
- Jobs
- Workstations, workstation classes
- Calendars
- Prompts
- Run cycles
- Run cycle groups
- Resources
- Internetwork dependencies
- External dependencies
- Event rules

For information about how to define workload application templates, see "Defining workload application" in the *User's Guide and Reference*.

Run cycle

A ***run cycle*** specifies the days that a job stream is scheduled to run. A cycle is defined for a specific job stream and cannot be used by multiple job streams.

Run cycles can be either inclusive or exclusive; that is:

inclusive

A run cycle that specifies the days and times that a job stream is scheduled to run. Inclusive run cycles give precedence to exclusive run cycles.

exclusive

A run cycle that specifies the days and times that a job stream cannot be run. Exclusive run cycles take precedence over inclusive run cycles.

Run cycles

When defining a run cycle, you need to specify the following properties:

Rule

Run cycle rules are based on lists of ordinal numbers, types of days, and common calendar intervals. For example, the first Monday of every month. Rule-based run cycles are based on conventional periods, such as calendar months, weeks of the year, and days of the week. There are four different types of available rules:

Simple

By selecting a simple rule type, you can specify the run cycle days directly from the calendar widget.

Calendar

By selecting a calendar rule type, you can reference an existing calendar that is saved in the database and the days that are defined on that calendar are used as a base for the run cycle.

iCalendar RRule

By selecting an iCalendar RRule type, you can enter an iCalendar Recurrence Rule expression and use it as base for the run cycle. From the Graphical Designer, a **RRule** generator is available; for more information, see iCalendar RRule generation on page 23

Run cycle group

A run cycle group is a single, reusable item that contains one or more run cycles. For more information, see Run cycle group definition on page 301.

Time restrictions

You can use time restrictions to define the time properties of the run cycle and the actions to take if the latest start time defined for the job stream.

Exclusive run cycles

When defining an exclusive run cycle, you need to specify the following properties:

Rule

Exclusive run cycle rules are based on lists of ordinal numbers, types of days, and common calendar intervals. For example, the first Monday of every month. Rule-based exclusive run cycles are based on conventional periods, such as calendar months, weeks of the year, and days of the week. There are four different types of available rules:

Simple

By selecting a calendar rule type, you can reference an existing calendar that is saved in the database and the days that are defined on that calendar are used as a base for the exclusive run cycle.

Calendar

By selecting a calendar rule type, you can reference an existing calendar that is saved in the database and the days that are defined on that calendar are used as a base for the exclusive run cycle.

iCalendar RRule

By selecting an **iCalendar RRule** type, you can enter an iCalendar Recurrence Rule expression and use it as base for the run cycle. From the Graphical Designer, a **RRule** generator is available; for more information, see iCalendar RRule generation on page 23

Run cycle group

A run cycle group is a single, reusable item that contains one or more exclusive run cycles. For more information, see Run cycle group definition on page 301.

Time restrictions

You can use time restrictions to define the time properties of the exclusive run cycle and the actions to take if the latest start time defined for the job stream.

iCalendar RRule generation

In the **Rule** section of the trigger **Properties** panel, you can select the **iCalendar RRule** type and then enter an expression or generate one by clicking **Generate RRule**. The **Create Recurring Event** panel appears, and you can define the following parameters:

Table 2. iCalendar RRule generator parameters

Basic parameters	Meaning	Description
`FREQ`	Frequency	Defines the core interval of time for the recurrence. The available frequency options are:

Table 2. iCalendar RRule generator parameters (continued)

Basic parameters	Meaning	Description
		• Daily • Daily by freedays • Daily by working days • Weekly • Monthly • Yearly **Note:** If you set the frequency to `DAILY; BY FREEDAY` or `DAILY; BY WORKING DAY`, the only other parameter that you can define for the recurring event is `INTERVAL`.
`INTERVAL`	Interval	Defines how often the `FREQ` repeats (from 0 to 2,147,483,647).
`BYDAY`	Day of the week	Specifies one or more days of the week, for example: `MO`, `TU`.
`BYMONTH`	Month of the year	Specifies one or more months of the year, for example: `1` for January.
`BYMONTHDAY`	Day of the month	Specifies the specific day of the month (from -31 to 31). For example: `BYMONTHDAY=1` is the first day of the month.

The check marks (✓) in the following table outline the supported combinations for recurrence parameters:

Table 3. Recurrence parameters compatibility

Frequency (`FREQ`)	`BYDAY`	`BYFREEDAY`	`BYWORKINGDAY`	`BYMONTH`	`BYMONTHDAY`	`BYDAY`+`BYMONTHDAY`
`DAILY`		✓	✓	✓		
`WEEKLY`	✓			✓		
`MONTHLY`	✓			✓	✓	
`YEARLY`	✓			✓		

After having defined the necessary parameters, click **Set rule** to add the generated **RRule** to the **Expression** field.

Run cycle group

You can optionally define a run cycle group for your job stream instead of, or in addition to, a number of single run cycles.

A run cycle group is a list of run cycles that are combined together to produce a set of run dates.

By using run cycle groups, you can benefit from the following advantages:

A run cycle group is a distinct database object

It is defined by itself and can be matched with one or more job streams. It is not defined as part of a specific job stream like single run cycles.

The same run cycle group can be used on different job streams

This improves the overall usability of the run cycles, because you can specify the same run cycle group in multiple job streams, avoiding the need to have multiple run cycle definitions for the same scheduling rules.

Run cycle groups enhance the use of exclusive run cycles

Exclusive (or negative) run cycles are used to generate negative occurrences, which identify the days when a job stream would normally be scheduled but is not required. The sum of the exclusive run cycles are subtracted from the inclusive ones. A negative occurrence always cancels any matching positive occurrences and you can specify a negative occurrence only if the positive equivalent already exists. An exact matching of the days, as well as any time restrictions, is required between the exclusive and inclusive run cycles for the cancellation to occur. Run cycle groups add much flexibility by allowing users to apply exclusive run cycles to a subset of the positive ones rather than to all of them. Group your run cycles into *subsets* so that the exclusive run cycles can be applied only to the positive occurrences generated by the run cycles belonging to the same set.

Run cycles must be organized into *subsets* within a run cycle group. The subsets are always in a logical **OR** relationship with each other. The result of the run cycle group is always a date or set of dates; it cannot be negative.

For example, you might want your job stream to run every day of the month except the last day of the month. But, you also want the it to be scheduled on the last day of the year (the last day of December). You can define a run cycle group using subsets, as follows:

Subset 1

- **Run cycle 1** - Inclusive run cycle every day of the month
- **Run cycle 2** - Exclusive run cycle on the last day of the month

Subset 2

- **Run cycle 3** - Inclusive run cycle on December 31st

where, run cycle 2 cancels the last day of each month in Subset 1, while run cycle 3 generates December 31st as a separate date and therefore you can schedule the job stream on Dec 31st.

Run cycle groups allow the use of a logical AND between individual run cycles in the subset

By default, the run cycles within a subset are in a logical **OR** relationship but you can change this to a logical **AND**, if the run cycle group result is a positive date or set of dates (Inclusive). For each run cycle, you can specify either operator (**AND** ,**OR**), obtaining the following behavior:

1. All the run cycles of the group that are in *AND* relationship are calculated first. The result of this calculation is a date or a set of dates.
2. Then, all the run cycles in an *OR* relationship are added to the result of the previous step.

A similar behavior is applied to inclusive and exclusive run cycles to determine the final date or set of dates of a group.

Inclusive (A)

Rule-based run cycle. Select days when the job stream is to be run if they belong to all A types of the set of run cycles.

Exclusive (D)

Exclusion rule-based run cycle. Select days when the job stream is NOT to be run if they belong to all D types of the set of run cycles.

For example, you can add two conditions together:

```
Run on Wednesday AND the 8th workday of the month.
```

In this way, the only scheduled dates are any 8th work day of the month that falls on a Wednesday.

Full compatibility with *traditional* run cycles

The *traditional* run cycles specified in the job stream definition can reference run cycle groups, with the possibility to specify shift or offsets on them (as with periods for z/OS or calendars for distributed systems).

A set of dates (interval starts) is created automatically either at run cycle level directly (inclusively or exclusively with offsets, or in the rule. This is a two-step process with run cycles:

1. Define the key "business event", such as, Month End, using run cycles and free day rules
2. Define rules that use the dates of the "business event" as the intervals against which the other batch run can be scheduled relative to.

For example, you have a *Month End process* that runs on the Last Friday of a month, but that moves forward to the next working day, except in December when it runs on the 3rd Friday of the month. This scheduling rule can be defined with a few rules, run cycles, and free day rules.

Two working days before Month End you need to run a pre-validation process to allow problems to be addressed before the run. You cannot choose the last Wednesday of the month, because in some months this might occur after the last Friday. Similarly, if the last Friday was a free day, the last Wednesday will not be 2

working days before it, because the Free Day rule applies ONLY to the day the rule falls on, it cannot look at anything else.

Many other batch runs might also need to be run on a certain number of days before or after the Month End, but the same restrictions apply.

You can now define work to run relative to something defined by a combination of run cycles and free day rules.

Use of calendars with run cycles within a run cycle group

Optionally, you can specify more than one calendar to calculate the working and non-working days definition for a run cycle. The primary calendar is used to calculate which working days are valid, and a secondary calendar is used to calculate specific non-working dates. If the dates calculated according to the secondary calendar match with the working days in the primary calendar, the job is scheduled; if they do not match, the job is not scheduled.

For example, a global company that runs workload in the United States for many other countries needs many calendar combinations to ensure that the batch jobs only run on a day that is a working day both in the United States and the other country. The calendar can be defined at job stream level and, if not specified, a default calendar is used. However, the calendar at run cycle level, whenever defined, can be used as secondary calendar and the job stream (or default) calendar can be used as the primary calendar.

For example, Primary calendar can be *WORKDAYS*, which is defined as MONDAY to FRIDAY excluding US holiday dates. You might also need to calculate the job runs based on calendar *HKWORK*, which is defined as Monday to Friday excluding Hong Kong holiday dates. The job might have several schedules:

- Run on working days, but not the last working day and not Mondays
- Run on Mondays, but not on the last working day
- Run on the last working day

Because each schedule is calculated against the *WORKHK* calendar it is also checked against the *WORKDAYS* calendar to ensure that it is scheduled on a US working day.

The use of time restrictions with run cycle groups

You can specify time constraints to define the time when processing must start or the time after which processing must no longer start. To do this, you can associate *time restrictions* to job, job streams, run cycles, and run cycle groups. When you define a time restriction, you basically obtain a *time*. Because you can associate time restrictions to multiple objects, the following hierarchy shows the order by which the different time restrictions are taken into consideration to actually define when to start the processing:

1. Time restriction defined in the run cycle into the job stream
2. Time restriction defined in the job stream
3. Time restriction defined in the run cycle contained in the run cycle group associated to the job stream.

4. Time restriction defined in the run cycle group associated to the job stream.
5. Start of Day

This means that:

Time restrictions in the job stream

Override and take precedence *over any other time restrictions* defined in the run cycles or run cycle groups associated to the job stream.

No time restrictions in the job stream nor in the run cycle group

The group generates only a date that is the *Start Of Day*. If offsets and free day rules are to be calculated, the calculation always starts from the *Start Of Day*.

Time restrictions in the run cycle group (not in the job stream)

Time restrictions (and possible offset) are calculated starting from the *Start Of Day* and the resulting date and time indicate the start of processing.

Examples

Table 4. Scenario 1. No time restriction in the run cycle group

Run cycle group	Scheduled date	Earliest Start
Run cycle group	10/24	10/24
Run cycle group with offset (+ 3 days)	10/27 (Saturday)	10/27/ (Saturday)
Run cycle group with free day rule	10/29/ (Monday)	0/29/ (Monday)
Run cycle in the job stream with time restrictions		
Run cycle in the job stream with + 4 working days shift	11/02 (Friday)	11/02 (Friday)
Run cycle in the job stream with free day rule	11/02 (Friday)	11/02 (Friday)
Run cycle in the job stream with earliest start +1 1pm	11/02 (Friday)	11/03 (Saturday) 1pm
Run cycle in the job stream without time restrictions		
Run cycle in the job stream with + 4 working days shift	11/02 (Friday)	11/02 (Friday) Start of Day
Run cycle in the job stream with free day rule	11/02 (Friday)	11/02 (Friday) Start of Day

Table 5. Scenario 2. Time restriction in the run cycle group without offset

Run cycle group	Scheduled date	Earliest Start
Run cycle group	10/24	10/24
Run cycle group with calendar offset (+ 3 days)	10/27/ (Saturday)	10/27/ (Saturday)
Run cycle group with free day rule	10/29/ (Monday)	0/29/ (Monday)
Run cycle in the job stream with time restrictions		
Run cycle in the job stream with + 4 working days shift	11/02 (Friday)	11/02 (Friday)
Run cycle in the job stream with free day rule	11/02 (Friday)	11/02 (Friday)
Run cycle in the job stream with earliest start +1 1pm	11/02 (Friday)	11/03 (Saturday) 1pm
Run cycle in the job stream without time restrictions		
Run cycle in the job stream with + 4 working days shift	11/02 (Friday)	11/02 (Friday) Start of Day
Run cycle in the job stream with free day rule	11/02 (Friday)	11/02 (Friday) Start of Day

Table 6. Scenario 3. Time restriction in the run cycle group with offset (+1 12:00)

Run cycle group	Scheduled date	Earliest Start
Run cycle group	10/24	10/24
Run cycle group with calendar offset (+ 3 days)	10/27/ (Saturday)	10/27/ (Saturday)
Run cycle group with free day rule	10/29/ (Monday)	10/29/ (Monday)
Run cycle group with offset +1 12:00	10/29/ (Monday)	10/30 12:00 (Tuesday)
Run cycle in the job stream with time restrictions		
Run cycle in the job stream with + 4 working days shift	11/02 (Friday)	11/02 (Friday)
Run cycle in the job stream with free day rule	11/02 (Friday)	11/02 (Friday)
Run cycle in the job stream with earliest start +1 1pm	11/02 (Friday)	11/03 (Saturday) 1pm

Table 6. Scenario 3. Time restriction in the run cycle group with offset (+1 12:00) (continued)

Run cycle group	Scheduled date	Earliest Start
Run cycle in the job stream without time restrictions		
Run cycle in the job stream with + 4 working days shift	11/02 (Friday)	11/03 12:00 (Saturday)
Run cycle in the job stream with free day rule	11/02 (Friday)	11/03 12:00 (Saturday)

Availability of the GENDAYS command at run cycle group level

Using GENDAYS, you can check the result of the combination of all the run cycles in the group.

Visualizing run cycle groups combinations

Run cycle groups can be managed from the Orchestration CLI and from the Dynamic Workload Console. When defining a run cycle group from the Dynamic Workload Console, the Run Cycle Preview helps you understand their impact on the run cycle group by highlighting when instances are added or removed on specific days. For more information, see Run cycle preview.

Related information

Run cycle preview

Run cycle

Folder

A *folder* is used to organize jobs and job streams.

A smart workflow folder is a container of jobs, job streams, and other folders. Use smart workflow folders to organize your jobs and job streams according to your lines of business or other custom categories. A folder can contain one or more jobs or job streams, while each job stream can be associated to one folder. If no folder is defined for a job stream, the root folder (/) is used by default.

You can move and rename scheduling objects into folders in batch mode using the composer rename command. Parts of the object names are used to name the folders.

For information about how to define folders, see Folder definition on page 287.

Calendar

A *calendar* is a list of dates which define if and when a job stream runs.

A calendar can also be designated for use as a *non-working days* calendar in a job stream. A non-working days calendar is a calendar that is assigned to a job stream to represent the days when the job stream and its jobs do not run. It can also

be used to designate Saturdays or Sundays, or both, as workdays. By convention many users define a non-working days calendar named *holidays*, where habitually Saturday and Sunday are specified as non-working days.

For information about how to define calendars, see Calendar definition on page 286.

Prompt

A *prompt* identifies a textual message that is displayed to the operator and halts processing of the job or job stream until an affirmative answer is received (either manually from the operator or automatically by an event rule action). After the prompt is replied to, processing continues. You can use prompts as dependencies in jobs and job streams. You can also use prompts to alert an operator that a specific task was performed. In this case, an operator response is not required.

Workstation

Read this section for information about the use of workstations for scheduling jobs and job streams. If, instead, you want to learn about workstations because you are planning your network, you can find the information you need in the *HCL Workload Automation: Planning and Installation*.

The computer system where you run your jobs and job streams is called a *workstation*. When you define a job or job stream in the HCL Workload Automation database you identify the workstation definitions for the physical or virtual computer systems on which your job is scheduled to run. Workstations can be grouped logically into *workstation classes* and organized hierarchically into *domains*, managed by *domain managers*.

When creating a workstation, you can define it in a folder. If no folder path is specified, then the workstation is created in the current folder. By default, the current folder is the root (\) folder, but you can customize it to a different folder path. You can also use the composer rename command to move and rename workstation in batch mode that use a naming convention to a different folder using part of the workstation name to name the folder.

For more information about workstation classes, see Workstation class on page 36, and for domains, see Domain on page 37.

When you create a workstation definition for a system in your network you define a set of characteristics that uniquely identify the system and affect the way jobs run on it. Some examples of these characteristics are the IP address of the workstation, if it is positioned behind a firewall, the secure or unsecure communication, the time zone where the workstation is located, and the identity of its domain manager.

Workstations in the HCL Workload Automation scheduling network perform job and job stream processing, but can also have other roles. When your network was designed, these roles were assigned to these workstations to suit the specific needs of your business. The following list describes all the workstation roles:

Master domain manager

A workstation acting as the management hub for the network. It manages all your scheduling objects. This workstation is registered in the HCL Workload Automation database as **master**.

Backup master domain manager

A workstation which can act as a backup for the master domain manager, when problems occur. It is effectively a master domain manager, waiting to be activated. Its use is optional. Learn more about switching to a backup master domain manager in the *HCL Workload Automation: Administration Guide*. This workstation must be installed as "master domain manager configured as backup". This workstation is registered in the HCL Workload Automation database as **fta**.

Domain manager

A workstation that controls a domain and shares management responsibilities for part of the HCL Workload Automation network. It is installed as an agent, and then configured as a domain manager workstation when you define the workstation in the database. This workstation is registered in the HCL Workload Automation database as **manager**.

Dynamic domain manager

An installed component in a distributed HCL Workload Automation network that is the management hub in a domain. All communication to and from the agents in the domain is routed through the dynamic domain manager. When you install a dynamic domain manager the following workstation types are created in the database:

fta

Fault-tolerant agent component manually configured as domain manager

broker

Broker server component

agent

Dynamic agent component

Backup dynamic domain manager

A workstation that can act as a backup for the dynamic domain manager when problems occur. It is effectively a dynamic domain manager, waiting to be activated. Its use is optional. Learn more about switching to a backup dynamic domain manager in the *HCL Workload Automation: Administration Guide*. When you install a dynamic domain manager the following workstation types are created in the database:

fta

Fault-tolerant agent component.

broker

Broker server component

agent

Dynamic agent component

Fault-tolerant agent

A workstation that receives and runs jobs. If there are communication problems with its domain manager, it can run jobs locally. It is installed as an agent, and then configured as a fault-tolerant agent workstation when you define the workstation in the database. This workstation is registered in the HCL Workload Automation database as **fta**.

Standard agent

A workstation that receives and runs jobs only under the control of its domain manager. It is installed as an agent, and then configured as a standard agent workstation when you define the workstation in the database. This workstation is registered in the HCL Workload Automation database as **s-agent**.

Extended agent

A workstation on which an HCL Workload Automation access method has been installed as a bridge so that you can schedule jobs in the SAP R/3, Oracle E-Business Suite, PeopleSoft, z/OS, or custom applications. It must be physically hosted by a master domain manager, domain manager, standard agent, or a fault-tolerant agent (up to 255 extended agents per fault-tolerant agent) and then defined as an extended agent in the database. For more information, see the *HCL Workload Automation User's Guide*. This workstation is registered in the HCL Workload Automation database as **x-agent**.

Workload broker

A workstation that runs both existing job types and job types with advanced options. It is the broker server installed with the master domain manager and the dynamic domain manager. It can host one or more of the following workstations:

- Extended agent
- Remote engine
- Pool

- Dynamic pool
- Agent. This definition includes the following agents:
  - Agent
  - HCL Workload Automation Agent
  - Agent for z/OS

  For more information about the agent and HCL Workload Automation Agent, see *HCL Workload Automation: Scheduling Workload Dynamically*. For more information about the agent for z/OS, see *HCL Workload Automation: Scheduling with the Agent for z/OS*.

This workstation is registered in the HCL Workload Automation database as **broker**.

Dynamic agent

A workstation that manages a wide variety of job types, for example, specific database or FTP jobs, in addition to existing job types. This workstation is automatically created and registered in the HCL Workload Automation database when you install the agent. The agent is hosted by the workload broker workstation. Because the installation and registration processes are performed automatically, when you view the agent in the Dynamic Workload Console, it results as updated by the Resource Advisor Agent. You can group agents in pools and dynamic pools. This workstation is registered in the HCL Workload Automation database as **agent**.

In a simple configuration, dynamic agents connect directly to a master domain manager or to a dynamic domain manager. However, in more complex network topologies, if the network configuration prevents the master domain manager or the dynamic domain manager from directly communicating with the dynamic agent, then you can configure your dynamic agents to use a local or remote gateway.

Note: If you have the `enAddWorkstation` global option set to `"yes"`, the dynamic agent workstation definition is automatically added to the Plan after the installation process creates the dynamic agent workstation in the database.

Pool

A logical workstation that groups a set of agents with similar hardware or software characteristics to which to submit jobs. HCL Workload Automation balances the jobs among the agents within the pool and automatically reassigns jobs to available agents if an agent is no longer available. To create a pool of agents in your HCL Workload Automation environment, define a workstation of type **pool** hosted by the workload broker

workstation, then select the agents you want to add to the pool. You can define the pool using the Dynamic Workload Console or the **composer** command.

You can also register an agent with a pool by directly editing the `pools.properties` file located in `<TWS_home>/ITA/cpa/config`. See the topic about automatically registering agents to a pool in the Planning and Installation.

This workstation is registered in the HCL Workload Automation database as **pool**. When you create a pool in your HCL Workload Automation environment, a logical resource with the same name is automatically created in the Dynamic Workload Broker. This logical resource is used to correlate and group together the agents belonging to the same pool, and as a requirement for the jobs scheduled in the HCL Workload Automation pool. Consider that these database objects are two different objects. If you rename the HCL Workload Automation pool, this change is not made to the Dynamic Workload Broker logical resource.

Dynamic pool

A logical workstation that groups a set of agents,which is dynamically defined based on the resource requirements you specify and hosted by the workload broker workstation. For example, if you require a workstation with low CPU usage and Windows installed to run your job, you specify these requirements using the Dynamic Workload Console or the **composer** command. When you save the set of requirements, a new workstation is automatically created in the HCL Workload Automation database. This workstation maps all the agents in your environment that meet the requirements you specified. The resulting pool is dynamically updated whenever a new suitable agent becomes available. Jobs scheduled on this workstation automatically inherit the requirements defined for the workstation. This workstation is hosted by the workload broker workstationand registered in the HCL Workload Automation database as **d-pool**.

The dynamic agents, pools, and dynamic pools leverage the dynamic functionality built into HCL Workload Automation and provide the possibility at run time to dynamically associate your submitted workload (or part of it) to the best available resources. You can enable dynamic scheduling capabilities to workstations at installation time. For more information about installing the dynamic agents, see Installing agents.

You can use dynamic agents, pools and dynamic pools to schedule job types with advanced options. The job types with advanced options include both those supplied with the product and the additional types implemented through the custom plug-ins. Both job types run only on dynamic agents, pools, and dynamic pools. For more information about how to create custom plug-ins, see Automation Hub.

You can also use dynamic agents, pools, and dynamic pools to run the jobs you created for the existing HCL Workload Automation workstation types. To run these jobs on the dynamic workstation types, you only have to change the specification of the workstation where you want the job to run.

If you want to leverage the dynamic capability when scheduling job types with advanced options, you schedule them on pools and dynamic pools, which assign dynamically the job to the best available resource. If you are interested only in defining job types with advanced options, without using the dynamic scheduling capability, you schedule these jobs on a specific dynamic agent, on which the job runs statically.

Remote engine

A workstation that manages the exchange of information about cross dependencies resolution between your environment and a remote HCL Workload Automation engine (controller) or an HCL Workload Automation engine (master domain manager or backup master domain manager). This workstation is hosted by the workload broker workstation and registered in the HCL Workload Automation database as **rem-eng**.

Note: If you plan to change the workstation types, consider the following rules:

- You can change fault-tolerant agent, standard agent, extended agent, domain manager and dynamic workload broker workstations to any workstation type, with the exception of dynamic agent, pool, dynamic pool, and remote engine.
- You cannot change the type of dynamic agent, pool, dynamic pool, and remote engine.

For information about how to define workstations, see Workstation definition on page 201.

Workstation class

Workstations can be grouped into classes. A *workstation class* is a group of workstations with similar job scheduling characteristics. Any number of workstations can be grouped in a class, and a workstation can be in many classes. Jobs and job streams can be assigned to run on a specific workstation class and this makes the running of jobs and job streams across multiple workstations easier.

For example, you can set up the following types of workstation classes:

- Workstation classes that group workstations according to your internal departmental structure, so that you can define a job to run on all the workstations in a department
- Workstation classes that group workstations according to the software installed on them, so that you can define a job to run on all the workstations that have a particular application installed
- Workstation classes that group workstations according to the role of the user, so that you can define a job to run on all the workstations belonging to, for example, managers

In this example, an individual workstation can be in one workstation class for its department, another for its user, and several for the software installed on it.

When creating a workstation class, you can define it in a folder. If no folder path is specified, then the workstation class is created in the current folder. By default, the current folder is the root (/) folder, but you can customize it to a different folder path. You can also use the composer rename command to move and rename workstation classes in batch mode that use a naming convention to a different folder using part of the workstation class name to name the folder.

Workstations can also be grouped into domains. This is done when your network is set up. The domain name is not one of the selection criteria when choosing where to run a job, so you might need to mirror your domain structure with workstation classes if you want to schedule a job to run on all workstations in a domain.

Workstation classes can be defined in a specific smart workflow folder if you want to organize them by line of business or some other custom category.

For more information about domains, see Domain on page 37

For more information about how to define workstation classes, see Workstation class definition on page 224.

Domain

All workstations in a distributed HCL Workload Automation network are organized in one or more *domains*, each of which consists of one or more agents and a domain manager acting as the management hub. Most communication to and from the agents in the domain is routed through the domain manager.

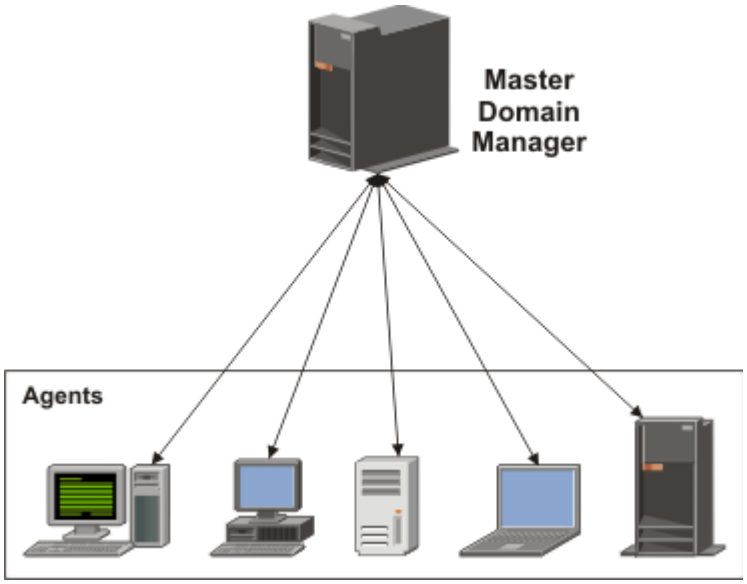
All networks have a master domain where the domain manager is the master domain manager. It maintains the database of all scheduling objects in the domain and the central configuration files. The master domain manager generates the plan and creates and distributes the Symphony file. In addition, logs and reports for the network are maintained on the master domain manager.

You can organize all agents in your network in a single domain, or in multiple domains.

Single-domain networks

A single domain network consists of a master domain manager and any number of agents. The following shows an example of a single domain network. A single domain network is well suited to companies that have few locations and business functions. All communication in the network is routed through the master domain manager. With a single location, you are concerned only with the reliability of your local network and the amount of traffic it can handle.

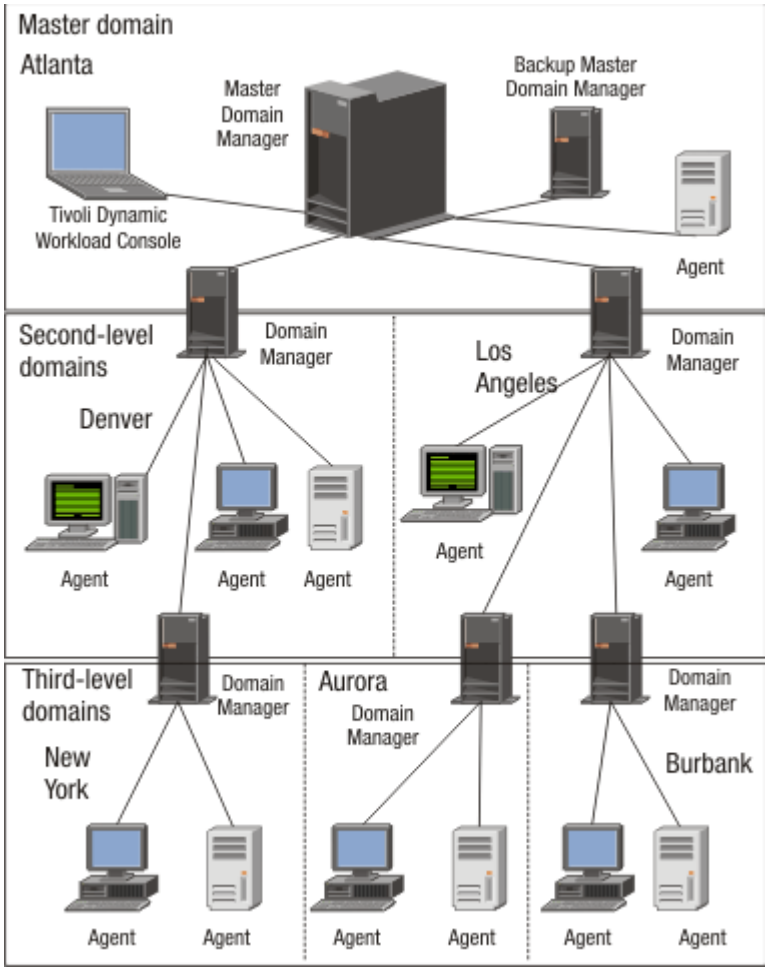
Figure 1. Single-domain network



Multiple-domain network

Multiple domain networks are especially suited to companies that span multiple locations, departments, or business functions. A multiple domain network consists of a master domain manager, any number of lower tier domain managers, and any number of agents in each domain. Agents communicate only with their domain managers, and domain managers communicate with their parent domain managers. The hierarchy of domains can go down to any number of levels.

Figure 2. Multiple-domain network



In this example, the master domain manager is located in Atlanta. The master domain manager contains the database files used to document the scheduling objects, and distributes the Symphony file to its agents and the domain managers in Denver and Los Angeles. The Denver and Los Angeles domain managers then distribute the Symphony file to their agents and subordinate domain managers in New York, Aurora, and Burbank. The master domain manager in Atlanta is responsible for broadcasting inter-domain information throughout the network.

All communication to and from the Boulder domain manager is routed through its parent domain manager in Denver. If there are schedules or jobs in the Boulder domain that are dependent on schedules or jobs in the Aurora domain, those dependencies are resolved by the Denver domain manager. Most inter-agent dependencies are handled locally by the lower tier domain managers, greatly reducing traffic on the network.

You can change the domain infrastructure dynamically as you develop your network. To move a workstation to a different domain, just requires you to change the domain name in its database definition.

Note: You cannot schedule jobs or job streams to run on all workstations in a domain by identifying the domain in the job or job stream definition. To achieve this, create a *workstation class* that contains all workstations in the domain.

For more information about workstation classes, see Workstation class on page 36

For information about how to define domains, see Domain definition on page 227.

Event rule

An *event rule* defines a set of actions that run when specific event conditions occur. An event rule definition correlates events and trigger actions.

For information about how to define event rules, see Defining event rules on page 182.

Resource

A *resource* is either a physical or logical system resource that you use as a dependency for jobs and job streams. A job or job stream with a resource dependency cannot start to run until the required quantity of the defined resource is available.

For information about how to define resources, see Resource definition on page 300.

Parameter

A *parameter* is an object to which you assign different values to be substituted in jobs and job streams, either from values in the database or at run time. Parameters are useful when you have values that change depending on your job or job stream. Job and job stream definitions that use parameters are updated automatically with the value at the start of the production cycle. Use parameters as substitutes for repetitive values when defining jobs and job streams. For example, using parameters for user logon and script file names in job definitions and for file and prompt dependencies allows the use of values that can be maintained centrally in the database on the master.

For more information about how to define parameters, see Variable and parameter definition on page 289.

User

A *User* is the user name used as the logon value for several operating system job definition. Users must be defined in the database.

If you schedule a job on an agent, on a pool, or on a dynamic pool, the job runs with the user defined on the pool or dynamic pool. However, the user must exist on all workstations in the pool or dynamic pool where you plan to run the job.

Note: If you have the `enAddUser` global option set to `"yes"`, the user definition is automatically added to the plan after you create or modify the user definition in the database.

Variable table

A *variable table* is a table containing multiple variables and their values. All global parameters, now called *variables*, are contained in at least one variable table.

You are not required to create variable tables to be able to use variables, because the scheduler provides a default variable table.

However, you might want to define a variable with the same name, but different values, depending on when and where it is used. You do this by assigning different values to the same variable in different variable tables. You can then use the same variable name in different job definitions and when defining prompts and file dependencies. Variable tables can be assigned at run cycle, job stream, and workstation level.

Variable tables can be particularly useful in job definitions when a job definition is used as a template for a job that belongs to more than one job stream. For example, you can assign different values to the same variable and reuse the same job definition in different job streams.

For information about how to define variable tables, see Variable table definition on page 295.

The HCL Workload Automation network

An HCL Workload Automation network consists of a set of linked ***workstations*** on which you perform batch job processing using HCL Workload Automation management capabilities.

Workstations communicate using TCP/IP links, and a store-and-forward technology to maintain consistency and fault-tolerance across the network. This means that if a workstation is not linked, all the information is stored in the messages file and only sent when the link is reestablished.

The HCL Workload Automation network consists of one or more domains, each having a ***domain manager*** workstation acting as a management hub, and one or more ***agent*** workstations.

There are four types of agent: ***standard***, ***fault-tolerant***, ***extended***, and ***workload broker***. Standard and fault-tolerant agents can be defined on UNIX® and Windows® computers. Extended agents are logical definitions, each hosted by a physical workstation, and are used to perform job processing where an agent is not installed. For example, extended agents are available for Peoplesoft, SAP R/3, z/OS®, CA-7, JES, OPC, Oracle EBS, and VMS but you can also install them on UNIX® and Windows® systems. Workload broker agents are workstations that manage the lifecycle of HCL Workload Automation Workload Broker type jobs in dynamic workload broker.

Another type of workstation that you can define in your network is a ***remote engine workstation***. This kind of workstation is used to manage the communication with a remote HCL Workload Automation engine, either distributed or z/OS based, to manage dependencies for local jobs from jobs defined on the remote engine. For more information, see Defining and managing cross dependencies on page 1352.

For information about workstations, see Workstation definition on page 201.

In the hierarchical HCL Workload Automation topology, the ***master domain manager*** is the domain manager of the topmost domain. All production setup tasks and the generation of the ***production plan*** are performed on the master domain manager. A production plan contains all job management activities to be performed across the HCL Workload Automation network during a specific time frame. A copy of the production plan is distributed from the master domain manager to the other workstations. On each workstation HCL Workload Automation launches and tracks its own jobs, and sends job processing status to the master domain manager.

For more information about HCL Workload Automation plan management capabilities, refer to Managing the production cycle on page 96.

Configuring your HCL Workload Automation runtime environment

About this task

This section gives you a high level overview of how you can configure your HCL Workload Automation runtime environment.

Configuring properties

About this task

You can set two types of properties to configure your HCL Workload Automation runtime environment, properties that are set on the master domain manager and affect processing on all workstations in the HCL Workload Automation network, and properties that are set locally on a workstation and affect processing on that workstation only. The former are managed using the HCL Workload Automation command line program named **optman**, and the latter you define locally on the workstation by customizing the configuration files `useropts`, `localopts`, and `jobmanrc`.

For more information on how to use the **optman** command line to manage global options and about local options defined in the `localopts` file, refer to *Administration Guide*.

For more information about the local options defined in the `useropts` file, refer to Setting up options for using the user interfaces on page 93.

Configuring security

About this task

Each time you run an HCL Workload Automation program, or invoke an HCL Workload Automation command, security information is read from a special file, the ***Security file***, to determine your user capabilities. This file contains one or more *user definitions*. A user definition is a group of one or more users who are either allowed or denied to perform specific actions against specific scheduling object types.

The main HCL Workload Automation user, ***TWS_user***, is defined at installation time in the security file. That user ID can be used to complete the setup procedure, to set properties, and to manage user definitions inside the security file. You can modify the security file at any time to meet your system requirements.

For more information about managing user authorizations, refer to *HCL Workload Automation: Administration Guide*.

Defining scheduling activities using HCL Workload Automation

About this task

To perform scheduling activities using HCL Workload Automation you must first define the environment you want to manage in terms of *scheduling objects* and in terms of rules to be applied when scheduling operations to run on these objects. This information is stored by HCL Workload Automation in a DB2® or other supported database, herein referred to as ***database***.

In addition to the definitions of scheduling objects, such as jobs, job streams, resources, workstations, and so on, the database also contains statistics about processed jobs and job streams, as well as information about the user who created an object and when an object was last modified. You can manage the scheduling object definitions in the database using either the HCL Workload Automation command-line program named **composer** or the graphical user interfaces, the **Dynamic Workload Console**. You can retrieve statistics or history information about processed jobs and job streams in the database using:

- The HCL Workload Automation **report utilities** from the command line.
- The Dynamic Workload Console.
- The database views.

For more information about how to define scheduling objects, see Defining objects in the database on page 196.

For more information about report utility commands, refer to Getting reports and statistics on page 1010.

For more information about the Dynamic Workload Console, refer to the corresponding documentation.

For more information on database views, refer to *Database Views*.

Controlling job and job stream processing

About this task

You can control how HCL Workload Automation processes jobs and job streams by setting one or more of these rules:

- Defining dependencies on page 42
- Setting time constraints on page 45
- Setting job priority and workstation fence on page 46
- Setting limits on page 46
- Defining resources on page 47
- Asking for job confirmation on page 47
- Defining job rerun and recovery actions on page 47

Defining dependencies

When you define job streams and manage your plan, you can enforce a precise processing order across your environment by configuring **dependencies**.

A **dependency** functions as a prerequisite condition that the scheduling environment must completely satisfy before a specific job or job stream can proceed with processing.

To build a reliable and automated workflow, you can classify these prerequisites into distinct local, cross-boundary, and cross-platform architectural scopes—including internal, external, internetwork, and cross dependencies.

Depending on your specific automation requirements, the system evaluates and triggers these connections based on diverse dependency types, such as:

- Standard completion dependencies on page 43
- Conditional dependencies on page 43
- Resource dependencies on page 44
- File dependencies on page 44
- Prompt dependencies on page 44

HCL Workload Automation then resolves these relationships at run time by identifying the correct predecessor instance in the plan using exact resolution criteria, such as closest preceding times, matching scheduled calendar dates, or defined absolute and relative time intervals. The sections below describe each of these dimensions in deeper technical detail to help you map, configure, and manage your workload path.

Dependency types

Within your HCL Workload Automation distributed scheduling environment you can choose from the following different types of dependencies:

- ***On successful completion of jobs and job streams***: a job or a job stream, named *successor*, must not begin processing until other jobs and job streams, named *predecessor*, have completed successfully. For more information, see follows on page 344.
- ***On satisfaction of specific conditions by jobs and job streams***: a job or a job stream, named *successor*, must not begin processing until other jobs and job streams, named *predecessor*, have met one, all, or a subset of specific conditions that can be related to the status of the job or job stream, the return code, output variables, or job log content. When the conditions are not met by the predecessor, then any successor jobs with a conditional dependency associated to them are put in suppress state. Successor jobs with a standard dependency are evaluated as usual.

  You can also join or aggregate conditional dependencies related to different predecessors into a single join dependency. A join contains multiple dependencies, but you decide how many of those dependencies must be satisfied for the join to be considered satisfied. You can define an unlimited number of conditional dependencies, standard dependencies, or both in a join.

  Note: This dependency type is not supported on Limited Fault-Tolerant Agent IBM i.

  For more information, see the topic about applying conditional branching logic in *Dynamic Workload Console User's Guide*, and the topics about the **follows** and **joins** keywords in *User's Guide and Reference*.

- ***Resource***: a job or a job stream needs one or more resources available before it can begin to run. For more information, refer to needs on page 362.
- ***File***: a job or a job stream needs to have access to one or more files before it can begin to run. For more information, refer to opens on page 374.
- ***Prompt***: a job or a job stream needs to wait for an affirmative response to a prompt before it can begin processing. For more information, refer to Prompt definition on page 298 and prompt on page 378.

You can define up to 40 dependencies for a job or job stream. If you need to define more than 40 dependencies, you can group them in a join dependency. In this case, the join is used simply as a container of standard dependencies and therefore any standard dependencies in it that are not met are processed as usual and do not cause the join dependency to be considered as suppressed. For more information about join dependencies, see Joining or combining conditional dependencies on page 1339 and join on page 352.

Scope of dependencies

You can classify dependencies in a distributed or z/OS environment into different categories.

When the predecessor and successor run within the same local environment, you can choose between two categories:

- **Internal dependencies:** Dependencies that you establish between jobs that belong to the same job stream.
- **External dependencies:** Dependencies that you establish between different job streams, between job streams and individual jobs in other job streams, or between jobs in separate job streams.

In an HCL Workload Automation network, dependencies can cross workstation boundaries. For example, you can make `job1`, which runs on your HCL Workload Automation local environment `site1`, dependent on the successful completion of `job2`, which runs on a remote HCL Workload Automation environment `site2`. The remote scheduling environment can be either HCL Workload Automation engines (controller) or another HCL Workload Automation engines (master domain manager). Two types of dependencies implement such requirement:

- **Internetwork dependencies:** Dependencies on jobs or job streams that run in an entirely separate HCL Workload Automation network. Use this type of dependency when:
  - The local HCL Workload Automation environment is distributed.
  - You want to search for a remote predecessor job instance only in the plan currently running (production plan) on the remote environment.
  - You need to match a predecessor instance in the remote plan, not *that* specific predecessor instance.
  - You can wait for the polling interval to expire before being updated about the remote job status transition.
  - You do not mind using different syntaxes and configurations based on whether the remote HCL Workload Automation environment is distributed rather than z/OS.
  - You do not mind using a proprietary connection protocol for communicating with the remote engine.

  For more information, see Managing internetwork dependencies on page 1322.
- **Cross dependencies:** Dependencies that you use to integrate your workload across different engines, such as between distributed engines and z/OS controllers. Use this type of dependency when:

- Your local HCL Workload Automation environment can be either distributed or z/OS.
- You want to search for the remote predecessor instance also among the scheduled instances that are not yet included in the plan currently running on the remote engine.
- You want to match a precise remote predecessor instance in the remote engine plan. To do this you can use different out-of-the-box matching criteria.
- You want your dependency to be updated as soon as the remote job instance changes status. To do this the product uses an asynchronous notifications from the remote engine to the local engine.
- You want to use the same syntax and configuration regardless of whether the local HCL Workload Automation environment is distributed or z/OS.
- You want to use HTTP or HTTPS connections for communicating with the remote engine.

For more information, see Defining and managing cross dependencies on page 1352.

Resolution criteria

HCL Workload Automation uses these specific evaluation criteria to identify the correct predecessor instance in the plan:

- **Closest preceding:** HCL Workload Automation targets the closest instance in time that precedes the scheduled start time of the successor. If no preceding instance exists, the system targets the closest instance that starts after the successor.
- **Same scheduled date:** HCL Workload Automation targets the predecessor instance planned to run on the exact same calendar day.
- **Within a relative interval:** HCL Workload Automation searches for the closest preceding instance within a relative time window that you specify. If the plan does not contain a preceding instance, the system evaluates the closest following instance within that same relative window.
- **Within an absolute interval:** HCL Workload Automation evaluates instances within a fixed, absolute time window using the same preceding-to-following fallback logic.

Setting time constraints

Defining the time limits for running jobs and job streams

About this task

Time constraints can be specified for both jobs and job streams. For a specific run cycle, you can specify the time that processing begins, by using the keyword **at**, or the time after which processing is no longer started, by using the keyword **until**. By specifying both, you define a time window within which a job or job stream runs. Both **at** and **until** represent time dependencies.

As an alternative to the **until** keyword, you can specify the **jsuntil** keyword. The **jsuntil** keyword also defines the latest start time of a job stream. It also determines the behavior of the jobs in the job stream when the job stream is approaching its latest start time. Use the **jsuntil** keyword to avoid that the job stream is either suppressed, canceled, or set to continue (depending on the action specified in the **onuntil** keyword) if it starts before its latest start time. For example, if you have a job stream with **jsuntil** set to 10:00 am, and one of the jobs starts running at 9:59 am, the job and its successors run as scheduled.

There is also a major difference with between the **until** and **jsuntil** keywords:

If you specify the until keyword in your job stream definition

This keyword is evaluated also after the job stream has started. As a result, if the latest start time expires before the job stream completes successfully, the action specified in the related **onuntil** keyword is performed on the job stream and on its jobs, which have not yet started.

If you specify the jsuntil keyword in your job stream definition

This keyword is evaluated only once, as soon as all dependencies of the job stream are satisfied and the job stream state changes to READY. If the latest start time defined using the **jsuntil** keyword has not expired at this time, it is no longer evaluated and the job stream runs independently of it. However, to prevent the job stream from remaining in READY state indefinitely, two days after the time specified in the **jsuntil** keyword has expired, the job stream is suppressed by default.

Another time setting that can be specified is the **schedtime**; it indicates the time that is referred to when calculating jobs and job streams dependencies. You can also specify a ***repetition rate***; for example, you can have HCL Workload Automation launches the same job every 30 minutes between 8:30 a.m. and 1:30 p.m.

You can also specify a **maximum duration** or a **minimum duration** for a job defined within a job stream. If a job is running and the maximum duration time has been exceeded, then the job can either be killed or can continue to run. If a job does not run long enough to reach the minimum duration time specified, then the job can be set to Abend status, to Confirm status awaiting user confirmation, or it can continue running.

Setting job priority and workstation fence

About this task

HCL Workload Automation has its own queuing system, consisting of levels of ***priority***. Assigning a priority to jobs and job streams gives you added control over their precedence and order of running.

The ***job fence*** provides another type of control over job processing on a workstation. When it is set to a priority level, it only allows jobs and job streams whose priority exceeds the job fence to run on that workstation. Setting the fence to 40, for example, prevents jobs with priorities of 40 or less from being launched.

For more information, refer to FENCE on page 615 and priority on page 377.

Setting limits

About this task

The ***limit*** provides a means of setting the highest number of jobs that HCL Workload Automation is allowed to launch. You can set a limit:

- In the job stream definition using the ***job limit*** argument
- In the workstation definition using the ***limit cpu*** command

Setting the limit on a workstation to 25, for example, allows HCL Workload Automation to have no more than 25 jobs running concurrently on that workstation.

For more information, refer to LIMIT CPU on page 619, and LIMIT SCHED on page 620.

Defining resources

About this task

You can define ***resources*** to represent physical or logical assets on your system. Each resource is represented by a name and a number of available units. If you have three tape units, for example, you can define a resource named `tapes` with three available units. A job that uses two units of the `tapes` resource would then prevent other jobs requiring more than the one remaining unit from being launched. However because a resource is not strictly linked to an asset, you can use a mock resource as a dependency to control job processing.

For more information, refer to Resource definition on page 300.

Asking for job confirmation

About this task

There might be scenarios where the completion status of a job cannot be determined until you have performed some tasks. You might want to check the results printed in a report, for example. In this case, you can set in the job definition that the job requires *confirmation*, and HCL Workload Automation waits for your response before marking the job as successful or failed.

For more information, refer to CONFIRM on page 602.

Defining job rerun and recovery actions

About this task

You have several options when defining recovery actions for your jobs, both when creating the job definition in the database and when monitoring the job execution in the plan.

When you create a job definition, either in the composer command line or in the Graphical Designer, you can specify the type of recovery you want performed by HCL Workload Automation if the job fails. The predefined recovery options are:

stop

If the job ends abnormally, do not continue with the next job.

You can also stop the processing sequence after a prompt is issued which requires a response from the operator.

continue

If the job ends abnormally, continue with the next job.

You can also continue with the next job after a prompt is issued which requires a response from the operator.

rerun

If the job ends abnormally, rerun the job.

You can add flexibility to the rerun option by defining a rerun sequence with specific properties. The options described below are mutually exclusive.

repeatevery *hhmm* for *number* attempts

You can specify how often you want HCL Workload Automation to rerun the failed job and the maximum number of rerun attempts to be performed. If any rerun in the sequence completes successfully, the remaining rerun sequence is ignored and any job dependencies are released.

rerun after prompt

HCL Workload Automation reruns the failed job after the operator has replied to a prompt.

same_workstation

If the parent job ran on a workstation that is part of a pool or a dynamic pool, you can decide whether it must rerun on the same workstation or on a different one. This is because the workload on pools and dynamic pools is assigned dynamically based on a number of criteria and the job might be rerun on a different workstation.

You can also decide to rerun the job in the HCL Workload Automation plan. In this case, you have the option of rerunning the job, or rerunning the job with its successors, either all successors in the same job stream, or all successors overall, both in the same job stream and in other job streams, if any. From the **conman** command line, use the LISTSUCC on page 626 command to identify the job's successors and the RERUNSUCC on page 639 command to rerun them.

The rerun option is especially useful when managing long and complex job streams. In case of a job completing in error, you can rerun the job with all its successors. You can easily identify the job successors, both in the same job stream and in any external job streams from the **conman** LISTSUCC on page 626 and RERUNSUCC on page 639 commands or the Dynamic Workload Console. From the Dynamic Workload Console, you can easily view the list of all job successors before rerunning them from the **Monitor Workload** view, by selecting the job and clicking **More Actions> Rerun with successors**. You can also choose whether you want to run all successors of the parent job or only the successors in the same job stream as the parent job. To manage the rerun option for parent job successors, see RERUNSUCC on page 639 and LISTSUCC on page 626.

When you decide to rerun the job in the HCL Workload Automation plan, you have the option to modify the previous job definition. From the **conman** RERUN on page 635 command, you can specify that the job is rerun under a new user name in place of the original user name. Also, you can specify the new command or script that the rerun job must run in place of the original command or script. From the Dynamic Workload Console, this is done from the **Monitor Workload** view, by selecting the job and clicking **Rerun>Edit Definition**.

recovery job

If the job ends abnormally, run a recovery job you have previously defined to try and solve the error condition. For example, you know that a job which requires a database connection might fail because the database

happens to be unreachable. To solve this error condition, you might define a recovery job which restarts the database.

You can combine the rerun sequence with the recovery job, so that if the parent job fails, a recovery job is started. When the recovery job completes successfully, the parent job is restarted after the specified interval, if any, for a specific number of times, with or without its successors.

For example, if you define a rerun sequence in which a parent job is associated with a recovery job and the parent job is scheduled to be rerun for three times after waiting for one minute for the recovery job to complete, the rerun sequence unfolds as follows:

1. The parent job runs and ends abnormally.
2. The recovery job starts and completes successfully.
3. The parent job waits for the specified interval after the recovery job completion before restarting, then restarts.
4. If it completes successfully, the remaining rerun sequence is ignored and any job dependencies are released. If the parent job completes in error again, steps 2 and 3 are repeated for three times, unless one of the reruns completes successfully.
5. If all reruns end abnormally, the job stream fails or remains in STUCK state.

For more information, refer to Job definition on page 228.

Modifying job instances in the plan to control job processing

After your jobs have been submitted into the current plan, you can control their processing by modifying the job definition directly from the **Orchestration Monitor** or **Job Stream View**, without having to go back to the original job definition in the database by using the following actions:

- Set hold for definition
- Remove hold for definition
- Set `No operation` for definition
- Remove `No operation` for definition

Note: These actions do not apply to the selected job-in-job-stream instance, but to the future instances of the job.

You can find the description of what each action does below:

Set no operation for definition

Sets the **No operation** property of the selected job-in-job-stream definition in the database, so that any future instance of the job is created with this property, which sets the job instances status to **success** without the job actually running.

Remove no operation for definition

Removes the **No operation** property of the selected job-in-job-stream definition in the database, so that any future instance of the job is created without this property and it can run and perform its function as designed.

Set hold for definition

Sets the priority of the selected job-in-job-stream definition in the database to 0, so that any future instance of the job is created with this priority.

Remove hold for definition

Sets the priority of the selected job-in-job-stream definition in the database to 10, so that any future instance of the job is created with this priority.

Important: To perform these actions in the database, your user profile must be granted specific permissions in the security file:

- **NOOPDB**: Required to use the **Set no operation for definition** and **Remove no operation for definition** actions.
- **HOLDDB**: Required to use the **Set hold for definition** and **Remove hold for definition** actions.

To perform these actions in the plan, your user profile must be granted the **USE** permission.

To modify job definitions starting from a job instance, perform the following steps:

From the Orchestration Monitor

1. **Monitoring and Reporting > Orchestration Monitor**.
2. Run a query on your jobs.
3. From the resulting list of jobs, select the job instance that you want to modify and click **More Actions**.
4. From the drop-down menu, select the desired action.

From the Job Stream View

1. **Monitoring and Reporting > Orchestration Monitor**
2. Run a query on your job streams.
3. From the resulting list of job streams, select a job stream and click **More Actions > Job Stream View**.
4. Select the block corresponding to the job instance that you want to modify and click the **... More Actions** button.
5. From the drop-down menu, select the desired action.

Managing production scheduling activities with HCL Workload Automation

About this task

Each time a new production plan is generated, HCL Workload Automation selects the job streams that run in the time window specified for the plan, and carries forward uncompleted job streams from the previous production plan. All the required information is written in a file, named ***Symphony***, which is continually updated while processing to indicate work completed, work in progress, and work to be done. The HCL Workload Automation **conman** (Console Manager) command-line program is used to manage the information in the `Symphony` file. The **conman** command-line program can be used to:

- Start and stop HCL Workload Automation control processes.
- Display the status of jobs and job streams.
- Alter priorities and dependencies.
- Alter the job fence and job limits.
- Rerun jobs.
- Cancel jobs and job streams.
- Submit new jobs and job streams.
- Reply to prompts.
- Link and unlink workstations in the HCL Workload Automation network.
- Modify the number of available resources.

Starting with version 9.1, all of the plan information written to the `Symphony` file is then replicated to the database. Various monitoring operations requested from the Dynamic Workload Console access the database, rather than the Symphony file, resulting in quicker response times and overall performance. The following operations requested from the Dynamic Workload Console access information from the database:

- Monitoring jobs and job streams
- Refreshing job and job stream monitoring views
- Monitoring workstations
- Monitoring resources, files, prompts
- Running baseline reports
- Displaying the plan in a graphical view
- Displaying the job stream in a graphical view

Note: This feature does not work with DB2 JDBC driver type = 2. HCL Workload Automation is supplied with JDBC driver type 4

Automating workload using event rules

About this task

Beside doing plan-based job scheduling, you can automate workload based on demand with the aid of event rules. The objective of event rules is to carry out a predefined set of actions in response to specific events affecting HCL Workload Automation and non-HCL Workload Automation objects.

With respect to HCL Workload Automation objects, the product provides a plug-in that you can use to detect the following events:

- A specific job or job stream:
  - Changes status
  - Is beyond its latest start time
  - Is submitted
  - Is cancelled

    - Is restarted
    - Becomes late
- A certain workstation:
    - Changes status
    - Changes its link status from its parent workstation
    - Changes its link status from its child workstation
- A specific prompt is displayed or replied to
- The application server on a certain workstation is started or stopped

When any of these events takes place, any of the following actions can be triggered:

- Submit a job stream, a job, or a task
- Reply to a prompt
- Run non-HCL Workload Automation commands
- Log an operator message
- Notify users via email

You can also define and run event rules that act either on the detection of one or more of these events or on a sequence or set of these events not completing within a specific length of time.

More information is available on Running event-driven workload automation on page 174.

HCL Workload Automation user interfaces

A combination of graphical and command-line and API interface programs are provided to work with HCL Workload Automation. In particular, the command-line interface is available for certain advanced features which are not available in the graphical user interface. The available HCL Workload Automation user interface programs are:

Single Entry Point

Single Entry Point is a web-based page to access all the HCL Workload Automation user interfaces:

- Dynamic Workload Console
- Self Service UIs (Self-Service Catalog and Self-Service Dashboards)

Single Entry Point is a role-based interface that you can access from any computer in your environment by using a web browser through the secure HTTPS.

Single Entry Point provides quick links to the most important Dynamic Workload Console tasks: connect your engines, design your workload, monitor your workload, and dashboard. With Single Entry Point you can access the mobile applications, the Self-Service Catalog and the Self-Service Dashboards, through the link or the Qrcode.

Dynamic Workload Console

A Web-based user interface available for viewing and controlling scheduling activities in production on both the HCL Workload Automation distributed and z/OS environments. With the Dynamic Workload Console you can use any supported browser to access the HCL Workload Automation environment from any location in your network.

You can use the Dynamic Workload Console to:

- Define scheduling objects in the HCL Workload Automation database
- Browse and manage scheduling objects involved in current plan activities
- Create and control connections to HCL Workload Automation environments
- Submit jobs and job streams in production
- Set user preferences
- Create and manage event rules
- Define and manage mission-critical jobs

Dynamic Workload Console must be installed on a server that can reach the HCL Workload Automation nodes using network connections. See the *HCL Workload Automation: Planning and Installation* for information.

composer

A command-line program used to define and manage scheduling objects in the database. This interface program and its use are described in Defining scheduling objects on page 196 and Managing objects in the database - composer on page 434.

conman

A command-line program used to monitor and control the HCL Workload Automation production plan processing. This interface program is described in Managing objects in the plan - conman on page 545.

Java™ API and plug-ins

A set of available classes and methods running in a JAVA environment that you use to create your custom interface to manage scheduling objects in the database and in the plan. This API cannot be used to create your custom interface to set global options. In addition, you can use and modify a set of plug-ins that perform specific tasks, or create your own plug-ins. The API is available through a Software Development Kit, which is part of the product.

optman

A command-line program used to manage the settings that affect the entire HCL Workload Automation environment. These settings, also called global options, are stored in the database. This interface program is described in the section about setting up global options in *Administration Guide*.

planman

A command-line program used to manage the HCL Workload Automation planning capability. This interface program is described in Planman command line on page 129.

You must install the HCL Workload Automation Command Line Client feature on fault-tolerant agents and systems outside the HCL Workload Automation network to use the **composer** and **optman** command-line programs and to run **planman showinfo** and **planman unlock** commands.

For information on how to set the options needed to allow a user to access the command-line interfaces, refer to .

Starting production

About this task

This section provides you with a step-by-step path of basic operations you can perform quickly implement HCL Workload Automation in your environment using the command-line interface. It is assumed that:

- These steps are performed on the master domain manager immediately after successfully installing the product on the systems where you want to perform your scheduling activities.
- The user ID used to perform the operations is the same as the one used for installing the product.

If you are not familiar with HCL Workload Automation you can follow the non-optional steps to define a limited number of scheduling objects, and add more as you become familiar with the product. You might start, for example, with two or three of your most frequent applications, defining scheduling objects to meet their requirements only.

Alternatively, you can use the Dynamic Workload Console to perform both the modeling and the operational tasks. Refer to the corresponding product documentation for more information.

The first activity you must perform is to access the HCL Workload Automation database and to define the environment where you want to perform your scheduling activities using the HCL Workload Automation scheduling object types. To do this perform the following steps:

1. **Set up the HCL Workload Automation environment variables**

   Run one of the following scripts:

   `. ./TWS_home/tws_env.sh` for Bourne and Korn shells in UNIX®

   `. ./TWS_home/tws_env.csh` for C shells in UNIX®

   `TWS_home\tws_env.cmd` in Windows®

in a system shell to set the *PATH* and *TWS_TISDIR* variables.

2. **Connect to the HCL Workload Automation database**

   You can use the following syntax to connect to the master domain manager as *TWS_user*:

   ```
   composer -user <TWS_user> -password <TWS_user_password>
   ```

   where *TWS_user* is the user ID you specified at installation time.

   Note: If you want to perform this step and the following ones from a system other than the master domain manager you must specify the connection parameters when starting **composer** as described in Setting up options for using the user interfaces on page 93.

3. **Optionally add in the database the definitions to describe the topology of your scheduling environment in terms of:**

   - **Domains**

     Use this step if you want to create a hierarchical tree of the path through the environment. Using multiple domains decreases the network traffic by reducing the communications between the master domain manager and the other workstations. For additional information, refer to Domain definition on page 227.

   - **Workstations**

     Define a workstation for each machine belonging to your scheduling environment with the exception of the master domain manager which is automatically defined during the HCL Workload Automation installation. For additional information, refer to Workstation definition on page 201. The master domain manager is automatically defined in the database at installation time.

4. **Optionally define the users allowed to run jobs on Windows® workstations**

   Define any user allowed to run jobs using HCL Workload Automation by specifying user name and password. For additional information, refer to User definition on page 281.

5. **Optionally define calendars**

   Calendars allow you to determine if and when a job or a job stream has to run. You can use them to include or exclude days and times for processing. Calendars are not strictly required to define scheduling days for the job streams (`simple` or `rule` run cycles may be used as well); their main goal is to define *global* sets of dates that can be reused in multiple job streams. For additional information refer to Calendar definition on page 286.

6. **Optionally define parameters, prompts, and resources**

   For additional information refer to Variable and parameter definition on page 289, Prompt definition on page 298, and Resource definition on page 300.

7. **Define jobs and job streams**

   For additional information refer to Job on page 1407, and to Job stream definition on page 314.

8. **Optionally define restrictions and settings to control when jobs and job streams run.**

You can define dependencies for jobs and job streams. There can be up to 40 dependencies for a job stream. If you need to define more than 40 dependencies, you can group them in a join dependency. In this case, the join is used simply as a container of standard dependencies and therefore any standard dependencies in it that are not met are processed as usual and do not cause the join dependency to be considered as suppressed. For more information about join dependencies, see Joining or combining conditional dependencies on page 1339 and join on page 352.They can be:

- Resource dependencies
- File dependencies
- Job and job stream follow dependencies, both on successful completion of jobs and job streams and on satisfaction of specific conditions by jobs and job streams
- Prompt dependencies

You can define time settings for jobs and job streams to run in terms of:

- Run cycles
- Time constraints

You can tailor the way jobs run concurrently either on a workstation or within a job stream by setting:

- Limit
- Priority

9. **Automate the plan extension at the end of the current production term**

   Add the `final` job stream to the database to perform automatic production plan extension at the end of each current production term by running the following command:

   ```
   add Sfinal
   ```

   For additional information, refer to Automating production plan processing on page 150.

10. **Generate the plan**

    Run the **JnextPlan** command to generate the production plan. This command starts the processing of the scheduling information stored in the database and creates the production plan for the time frame specified in the **JnextPlan** command. The default time frame is 24 hours. If you automated the plan generation as described in the previous step, you only need to run the **JnextPlan** command the first time.

When you complete this step-by-step process, your scheduling environment is up and running, with batch processing of an ordered sequence of jobs and job streams being performed against resources defined on a set of workstations, if defined. By default, the first time you run the **JnextPlan** command, the number of jobs that can run simultaneously on a workstation is zero, so make sure that you increase this value by changing the `limit cpu` to allow job execution on that workstation, see the section LIMIT CPU on page 619 for more details.

If you want to modify anything while the production plan is already in process, use the **conman** program. While the production plan is processing across the network you can still continue to define or modify jobs and job streams in the database. Consider however that these modifications will only be used if you submit the modified jobs or job streams, using the command **sbj** for jobs or **sbs** for job streams, on a workstation which has already received the plan, or after a new production plan is generated using **JnextPlan**. See Managing objects in the plan - conman on page 545 for more details about the **conman** program and the operations you can perform on the production plan in process.

Chapter 2. Understanding basic processes and commands

In a multi-tier HCL Workload Automation network, locally on each workstation a group of specialized scheduling processes performs job management and sends back the information about job processing throughout the hierarchical tree until the master domain manager is reached. Using the information received from the workstations, the master domain manager then updates both its copy of the `Symphony` file and the replicated plan in the database, which contain the records describing the job processing activities to be performed across the HCL Workload Automation network during the current production plan, and sends the updates on the activities to be performed to the workstations involved.

Issuing commands on Windows operating systems

On Windows operating systems, ensure that you are issuing the HCL Workload Automation commands from a command prompt with the **Run as administrator** privilege level.

If you install an HCL Workload Automation component, either agent or server, in the `C:\` or `C:\Program Files\` directories, the **twsuser** or any user other than the Administrator might encounter issues when running commands from the command-line interface due to security settings. For example, the resource command-line interface might fail to run commands, and the conman command might display error or warning messages. To prevent these issues, open the command prompt with the **Run as administrator** option to run commands.

HCL Workload Automation workstation processes

The management of communication between workstations and local job processing, together with the notification of state updates, are performed on each HCL Workload Automation workstation by a series of management processes that are active while the engine is running. On fault-tolerant agents and domain managers these processes are based on the WebSphere Application Server Liberty infrastructure. This infrastructure is automatically installed with the workstation and allows HCL Workload Automation to:

- Communicate across the HCL Workload Automation network.
- Manage authentication mechanisms for remote clients, such as command-line programs, connecting to the master domain manager using the HTTP or HTTPS protocols.

For information on how to start and stop both the WebSphere Application Server Liberty infrastructure and the HCL Workload Automation processes on a workstation refer to Starting and stopping processes on a workstation on page 63. Except for manually starting and stopping the WebSphere Application Server Liberty and managing connection parameters when communicating across the HCL Workload Automation network, the WebSphere Application Server Liberty infrastructure is transparent when using HCL Workload Automation.

In this guide *HCL Workload Automation processes* or *workstation processes* are used to identify the following processes:

netman
monman
writer
mailman
batchman
jobman

With the exception of standard agents, these processes are started in the following order on the HCL Workload Automation workstations:

netman

Netman is the Network Management process. It is started by the **Startup** command and it behaves like a network listener program which receives start, stop, link, or unlink requests from the network. **Netman** examines each incoming request and creates a local HCL Workload Automation process.

monman

Monman is a process started by **netman** and is used in event management. It starts monitoring and **ssmagent** services that have the task of detecting the events defined in the event rules deployed and activated on the specific workstation. When these services catch any such events, after a preliminary filtering action, they send them to the event processing server that runs usually in the master domain manager. If no event rule configurations are downloaded to the workstation, the monitoring services stay idle.

Ssmagent services are used only for file monitoring event types. For more information, see paragraph *"File monitoring events"* in the section *"Event management"* in chapter *"HCL Workload Automation Concepts"* of the *Dynamic Workload Console User's Guide*.

The communication process between the monitoring agents and the event processing server is independent of the HCL Workload Automation network topology. It is based directly on the EIF port number of the event processor and the event information flows directly from the monitoring agents without passing through intermediate domain managers. A degree of fault-tolerance is guaranteed by local cache memories that

temporarily store the event occurrences on the agents in case communication with the event processor is down.

writer

Writer is a process started by **netman** to pass incoming messages to the local **mailman** process. The **writer** processes (there might be more than one on a domain manager workstation) are started by link requests (see LINK on page 621) and are stopped by unlink requests (see UNLINK on page 729) or when the communicating **mailman** ends.

mailman

Mailman is the Mail Management process. It routes messages to either local or remote workstations. On a domain manager, additional **mailman** processes can be created to divide the load on **mailman** due to the initialization of agents and to improve the timeliness of messages. When the domain manager starts, it creates a separate **mailman** process instance for each *ServerID* specified in the workstation definitions of the fault-tolerant agents and standard agents it manages. Each workstation is contacted by its own *ServerID* on the domain manager. For additional information, refer to Workstation definition on page 201.

batchman

Batchman is the Production Control process. It interacts directly with the copy of the `Symphony` file distributed to the workstations at the beginning of the production period and updates it. **Batchman** performs several functions:

- Manages locally plan processing and updating.
- Resolves dependencies of jobs and job streams.
- Selects jobs to be run.
- Updates the plan with the results of job processing.

Batchman is the only process that can update the `Symphony` file.

jobman

Jobman is the Job Management process. It launches jobs under the direction of **batchman** and reports job status back to **mailman**. It is responsible for tracking job states and for setting the environment as defined in the `jobmanrc` and `.jobmanrc` scripts when requesting to launch jobs. For information about these scripts, see Configuring the job environment on page 74. When the **jobman** process receives a launch job message from **batchman**, it creates a job monitor process. The maximum number of job monitor processes that can be created on a workstation is set by using the **limit cpu** command from the **conman** command line prompt (see LIMIT CPU on page 619).

job monitor (jobman on UNIX®, JOBMON.exe and joblnch.exe on Windows® operating system)

The job monitor process first performs a set of actions that set the environment before the job is launched, and then launches the job by running the script file or command specified in the job definition. For additional details on how to specify the script file or the command launched with the job, refer to Job on page 1407.

The setup activities consist of launching the standard configuration file (*`TWS_home`*`/jobmanrc` in UNIX® and *`TWS_home`*`/jobmanrc.cmd` in Windows®) which contains settings that apply to all jobs running on the workstation. In addition, on UNIX® workstations a local configuration script *`TWS_user`*`/.jobmanrc` is launched, if it exists in the home directory of the user launching the job. This local configuration file contains settings that apply only to jobs launched by the specific user.

If any of these steps fails, the job ends in the FAIL state.

Attention: If, on Windows systems, a system variable called `TEMP` exists, user *TWS_user* must be authorized to create files in the directory to which the variable is set. If this requirement is not met, the **JOBMON.exe** binary file fails to start successfully.

All processes, except **jobman**, run as the *TWS_user*. **Jobman** runs as root.

On standard agent workstations, the **batchman** process is not launched because this type of workstation does not manage job scheduling. These workstations only launch jobs under the direction of their domain manager. Locally on the workstation the management processes wait for a request to launch a job from the domain manager in LISTEN mode. When the request is received the job is launched locally and the result is sent back to the domain manager. For additional information on standard agent workstations refer to *HCL Workload Automation: Planning and Installation Guide*.

Figure 3: Process tree in UNIX on page 62 shows the process tree on HCL Workload Automation workstations, other than standard agents, installed on UNIX®:

Figure 3. Process tree in UNIX®

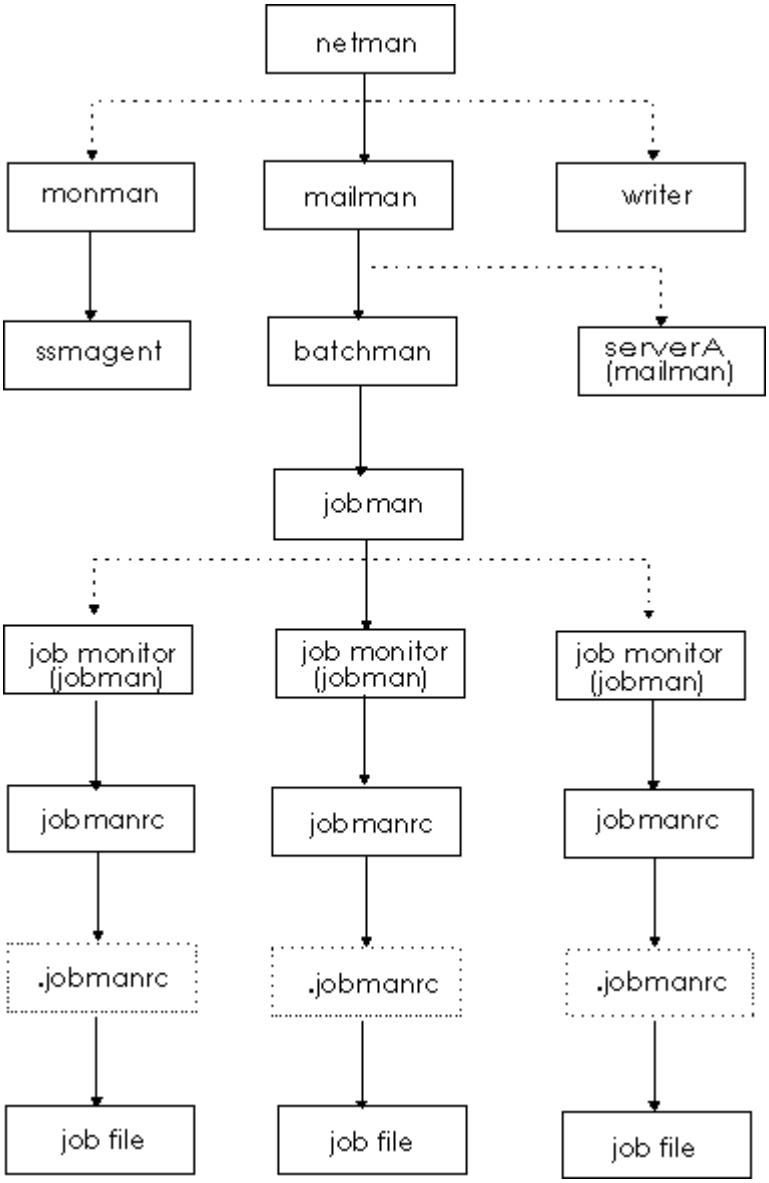
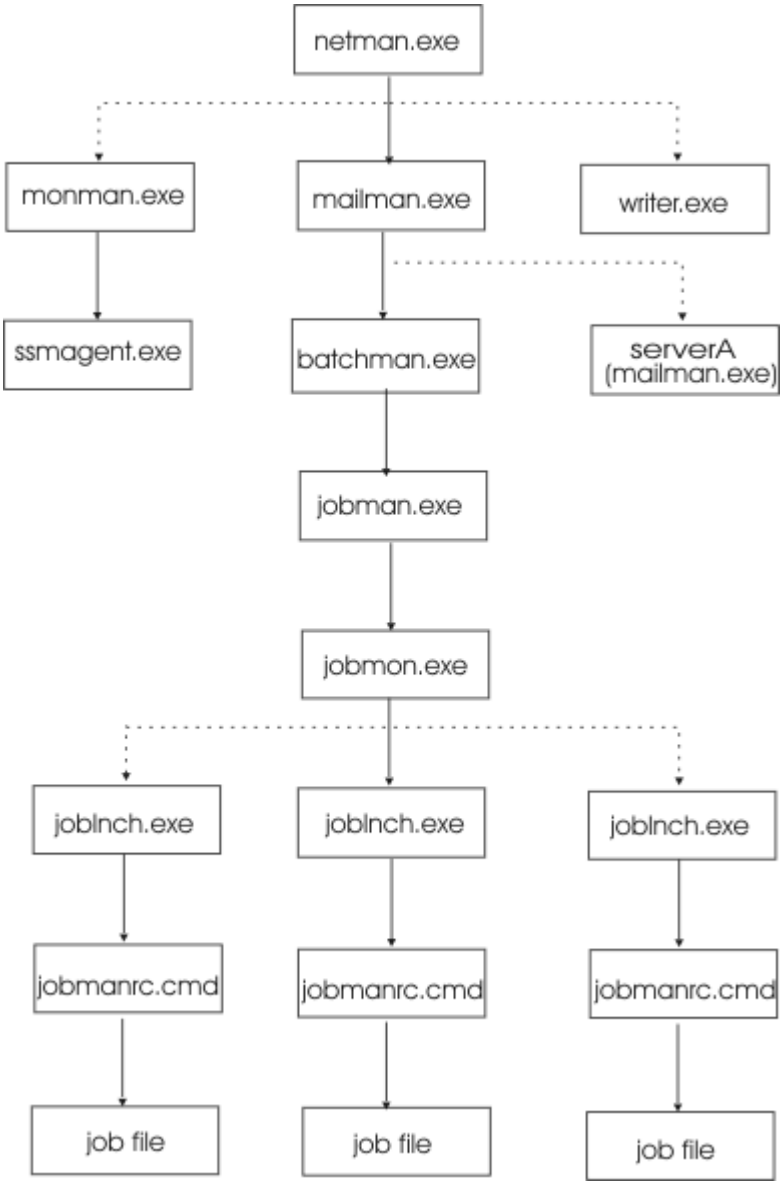


Figure 4: Process tree in Windows on page 63 shows the process tree on HCL Workload Automation workstations, other than standard agents, installed on Windows®:

Figure 4. Process tree in Windows®



On Windows® platforms there is an additional service, the `Tivoli Token Service`, which enables HCL Workload Automation processes to be launched as if they were issued by the HCL Workload Automation user.

Starting and stopping processes on a workstation

About this task

The type of operating system installed on the workstation determines how HCL Workload Automation processes can be started from the command line. Table 7: Starting and stopping HCL Workload Automation on a workstation on page 64 explains how you can start and stop both the WebSphere Application Server Liberty infrastructure and HCL Workload Automation processes on a workstation based on the operating system installed.

Table 7. Starting and stopping HCL Workload Automation on a workstation

Action	Commands used on UNIX® platform	Commands used on Windows® platform
Start all HCL Workload Automation processes including WebSphere Application Server Liberty and the event monitoring engine.	conman start conman startappserver conman startmon	conman start conman startappserver conman startmon
Start netman and WebSphere Application Server Liberty. On Windows starts also the `Tivoli Token Service`	./StartUp.sh	StartUp
Stop all HCL Workload Automation processes and WebSphere Application Server Liberty.	conman shutdown stopAppServer	conman shutdown -appsrv shutdown -appsrv
Stop all HCL Workload Automation processes with the exception of WebSphere Application Server Liberty.	conman shutdown	conman shutdown shutdown
Start all HCL Workload Automation processes with the exception of WebSphere Application Server Liberty and the event monitoring engine.	conman start	conman start
Stop all HCL Workload Automation processes but **netman**, **monman**, **writer**, and **appservman**.	conman stop	conman stop
Stop all HCL Workload Automation processes (including **netman**).	conman shutdown	conman shutdown shutdown
Start WebSphere Application Server Liberty	conman startappserver	conman startappserver
Stop WebSphere Application Server Liberty	conman stopappserver	conman stopappserver
Start the event monitoring engine	conman startmon	conman startmon
Stop the event monitoring engine	conman stopmon	conman stopmon
Start the agent locally	./StartUpLwa.sh	startuplwa

Table 7. Starting and stopping HCL Workload Automation on a workstation (continued)

Action	Commands used on UNIX® platform	Commands used on Windows® platform
	Note: can be run by *TWS_user* or root user only.	
Stop the agent locally	./ShutDownLwa.sh	shutdownlwa
	Note: can be run by *TWS_user* or root user only.	

Note: On Windows systems refrain from using Windows services to stop WebSphere Application Server Liberty. Use one of the commands listed in this table instead. If you use Windows services to stop WebSphere Application Server Liberty, the *appserverman* process, which continues to run, will start it again.

Refer to StartUp on page 971 for more information about the **StartUp** utility command.

Refer to shutdown on page 970 for more information about the **shutdown** utility command.

Refer to the section about starting and stopping the application server in *Administration Guide* for more information about starting and stopping WebSphere Application Server Liberty.

Refer to START on page 697 for more information about the **conman start** command.

Refer to STOP on page 702 for more information about the **conman stop** command.

Refer to SHUTDOWN on page 696 for more information about the **conman shutdown** command.

Refer to STARTAPPSERVER on page 699 for more information about the **conman startappserver** command.

Refer to STOPAPPSERVER on page 704 for more information about the **conman stopappserver** command.

Refer to STARTMON on page 700 for more information about the **conman startmon** command.

Refer to STOPMON on page 706 for more information about the **conman stopmon** command.

If the agent is installed on a Windows® system, WebSphere Application Server Liberty and the **netman** processes are automatically started at start time as services together with the `Tivoli Token Service`.

If the agent is installed on a UNIX® system, WebSphere Application Server Liberty and the **netman** processes can be automatically started at start time by adding a statement invoking **Startup** in the `/etc/inittab` file.

Starting and stopping the agent

About this task

The type of operating system installed on the workstation determines how agents can be started from the command line.

Table 8. Starting and stopping the agent

Action	Commands used on UNIX® platform	Commands used on Windows® platform
Start the agent locally	./StartUpLwa.sh **Note:** can be run by *TWS_user* or root user only.	startuplwa
Stop the agent locally	./ShutDownLwa.sh **Note:** can be run by *TWS_user* or root user only.	shutdownlwa

For more infomation about stopping and starting the agent, see ShutDownLwa - Stop the agent on page 971 and StartUpLwa - Start the agent on page 972.

Workstation inter-process communication

HCL Workload Automation uses message queues for local inter-process communication. There are 11 message files, which reside in the *TWS_home* directory:

Appserverbox.msg

This message file is written by the **conman** and **mailman** processes and read by the **appservman** process. It receives messages such as WebSphere Application Server Liberty START and STOP.

auditbox.msg

This message file is written by the **mailman** process and read by the WebSphere Application Server Liberty process. It receives audit messages to be stored in the database.

Courier.msg

This message file is written by the **batchman** process and read by the **jobman** process.

Intercom.msg

This message file is read by the **batchman** process and contains instructions sent by the local **mailman** process.

Mailbox.msg

This message file is read by the **mailman** process. It receives messages, through the graphical user interface (Dynamic Workload Console) or the console manager (**conman**), incoming from the local **batchman** and **jobman** processes and from other HCL Workload Automation workstations in the network.

mirrorbox.msg

This message file is written by the **mailman** process and read by the WebSphere Application Server Liberty process. It receives any **batchman** incoming messages.

Monbox.msg

This message file is written by the **mailman**, **batchman**, and **appservman** processes and read by the **monman** process. It receives messages such as QUEUE_EVENT.

Moncmd.msg

This message file is written by the **conman**, **batchman**, **mailman** processes and by the Dynamic Workload Console, and read by the **monman** process. It receives messages such as STOP, DEPLOY_CONFIG, UPGRADE_WORKSTATION

NetReq.msg

This message file is read by the **netman** process for local commands. It receives messages such as START, STOP, LINK, and UNLINK.

PlanBox.msg

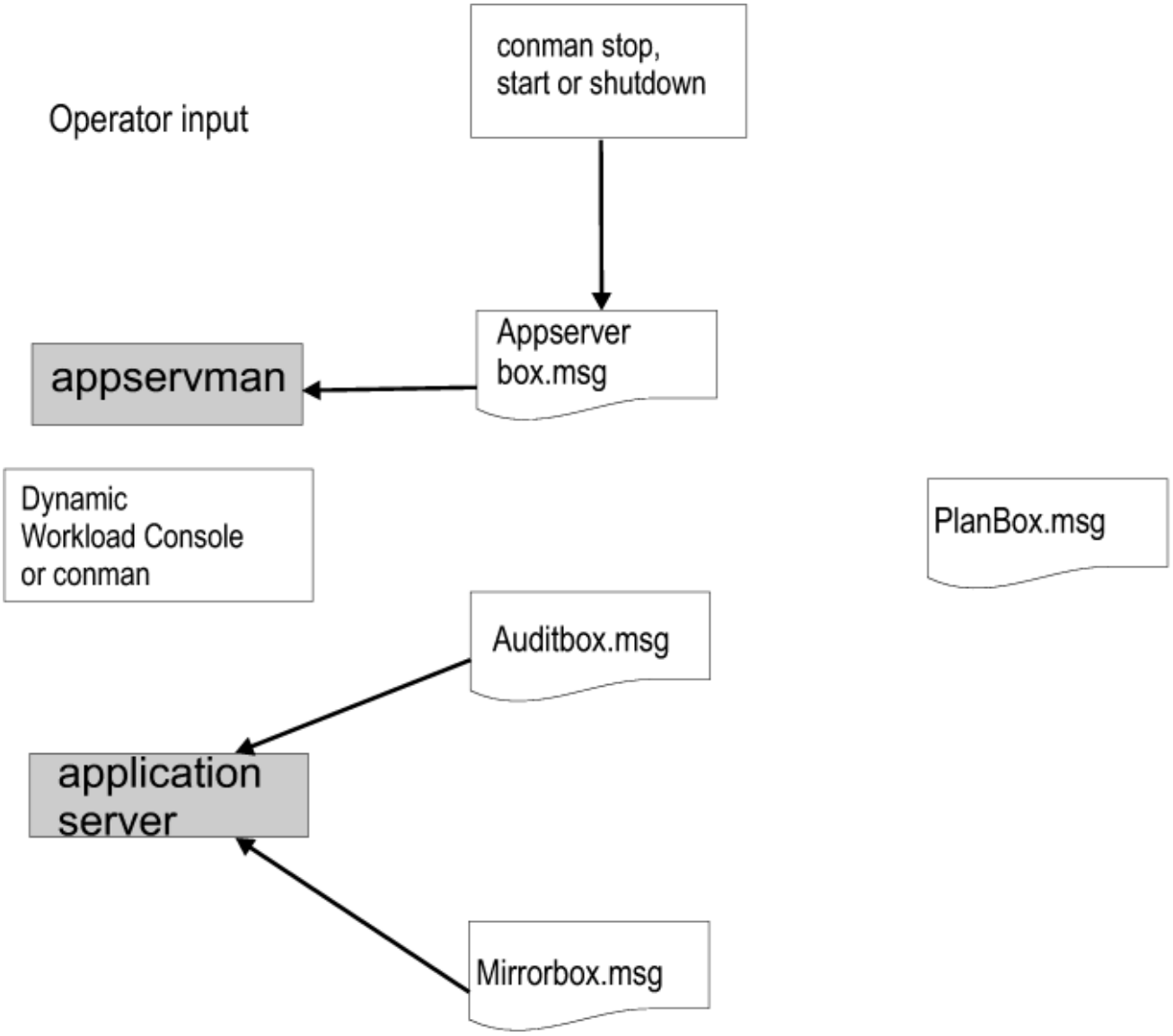
This message file is written by the **batchman** process and read by the engine.

Server.msg

This message file is written by the **batchman** process and read by the engine.

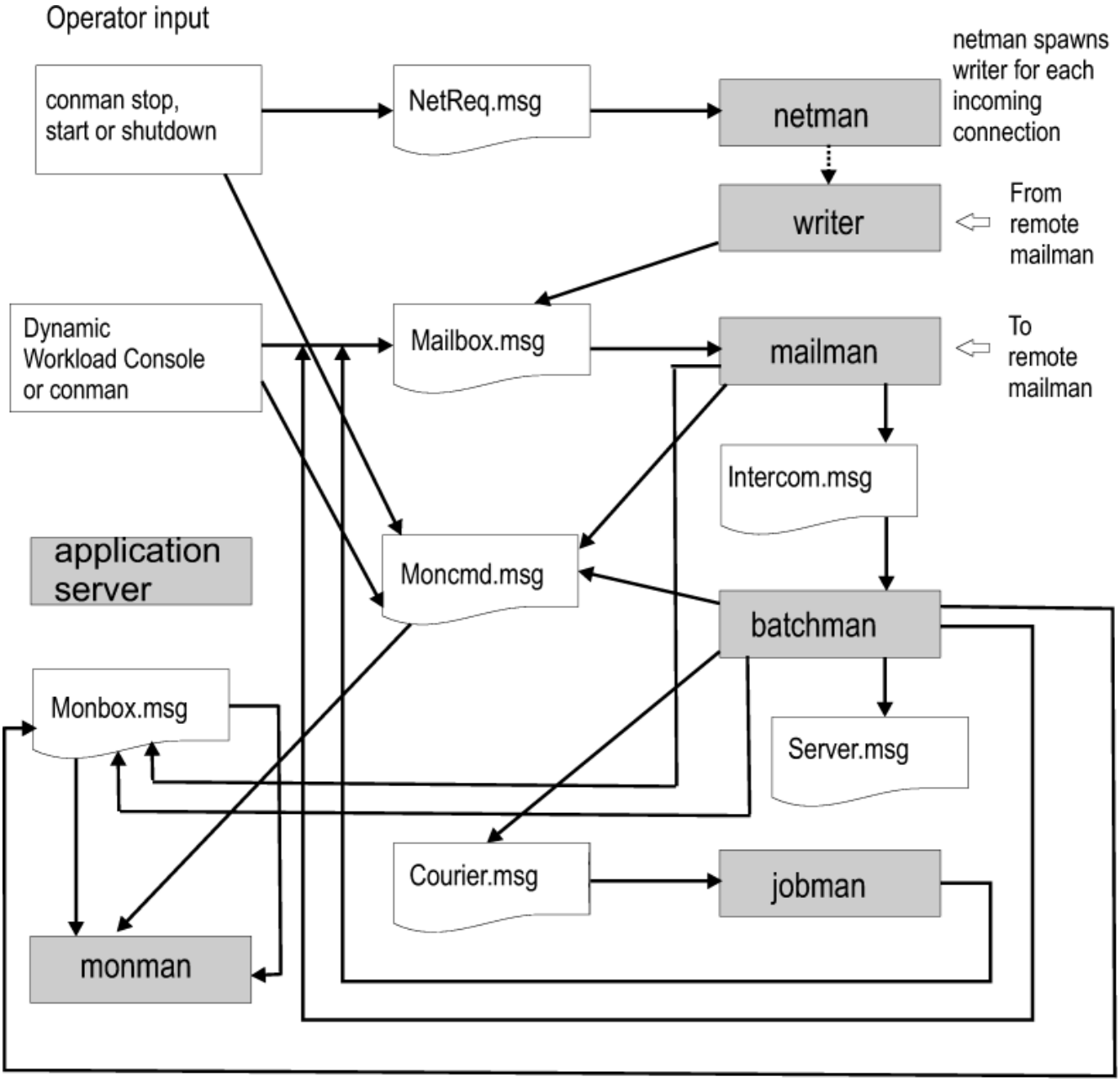
This figure illustrates the inter-process communication on the master domain manager.

Figure 5. Inter-process communication on the master domain manager



This figure illustrates the inter-process communication on the master domain manager and fault-tolerant agent.

Figure 6. Inter-process communication on the master domain manager and fault-tolerant agent



These files have a default maximum size of 10MB. The maximum size can be changed using the **evtsize** utility (see evtsize on page 938).

HCL Workload Automation network communication

HCL Workload Automation uses the TCP/IP protocol for network communication. The node name and the port number used to establish the TCP/IP connection are set for each workstation in its workstation definition. Refer to Workstation definition on page 201 for additional details.

A *store-and-forward* technology is used by HCL Workload Automation to maintain consistency and fault-tolerance at all times across the network by queuing messages in message files while the connection is not active. When TCP/IP communication is established between systems, HCL Workload Automation provides bi-directional communication between workstations using links. Links are controlled by the **autolink** flag set in the Workstation definition on page 201, and by the LINK on page 621 and UNLINK on page 729 commands issued from the **conman** command-line program.

When a link is opened, messages are passed between two workstations. When a link is closed, the sending workstation stores messages in a local message file and sends them to the destination workstation as soon as the link is re-opened.

There are basically two types of communication that take place in the HCL Workload Automation environment, connection initialization and scheduling event delivery in the form of change of state messages during the processing period. These two types of communication are now explained in detail.

Connection initialization and two-ways communication setup

These are the steps involved in the establishment of a two-ways HCL Workload Automation link between a domain manager and a remote fault-tolerant agent:

1. On the domain manager, the **mailman** process reads the host name, TCP/IP address, and port number of the fault-tolerant agent from the `Symphony` file.
2. The **mailman** process on the domain manager establishes a TCP/IP connection to the **netman** process on the fault-tolerant agent using the information obtained from the `Symphony` file.
3. The **netman** process on the fault-tolerant agent determines that the request is coming from the **mailman** process on the domain manager, and creates a new **writer** process to handle the incoming connection.
4. The **mailman** process on the domain manager is now connected to the **writer** process on the fault-tolerant agent. The **writer** process on the fault-tolerant agent communicates the current run number of its copy of the `Symphony` file to the **mailman** process on the domain manager. This run number is the identifier used by HCL Workload Automation to recognize each `Symphony` file generated by **JnextPlan**. This step is necessary for the domain manager to check if the current plan has already been sent to the fault-tolerant agent.
5. The **mailman** process on the domain manager compares its `Symphony` file run number with the run number of the `Symphony` file on the fault-tolerant agent. If the run numbers are different, the **mailman** process on the domain manager sends to the **writer** process on the fault-tolerant agent the latest copy of the `Symphony` file.
6. When the current `Symphony` file is in place on the fault-tolerant agent, the **mailman** process on the domain manager sends a `start` command to the fault-tolerant agent.
7. The **netman** process on the fault-tolerant agent starts the local **mailman** process. At this point a one-way communication link is established from the domain manager to the fault-tolerant agent.

8. The **mailman** process on the fault-tolerant agent reads the host name, TCP/IP address, and port number of the domain manager from the `Symphony` file and uses them to establish the uplink back to the **netman** process on the domain manager.
9. The **netman** process on the domain manager determines that the request is coming from the **mailman** process on the fault-tolerant agent, and creates a new **writer** process to handle the incoming connection. The **mailman** process on the fault-tolerant agent is now connected to the **writer** on the domain manager and a full two-way communication link is established. As a result of this, the **writer** process on the domain manager writes messages received from the fault-tolerant agent into the `Mailbox.msg` file on the domain manager, and the **writer** process on the fault-tolerant agent writes messages from the domain manager into the `Mailbox.msg` file on the fault-tolerant agent.

Job processing and scheduling event delivery in the form of change of state messages during the processing day performed locally by the fault-tolerant agent

During the production period, the `Symphony` file present on the fault-tolerant agent is read and updated with the state change information about jobs that are run locally by the HCL Workload Automation workstation processes. These are the steps that are performed locally on the fault-tolerant agent to read and update the `Symphony` file and to process jobs:

1. The **batchman** process reads a record in the `Symphony` file that states that `job1` is to be launched on the workstation.
2. The **batchman** process writes in the `Courier.msg` file that `job1` has to start.
3. The **jobman** process reads this information in the `Courier.msg` file, starts `job1`, and writes in the `Mailbox.msg` file that `job1` started with its *process_id* and *timestamp*.
4. The **mailman** process reads this information in its `Mailbox.msg` file, and sends a message that `job1` started with its *process_id* and *timestamp*, to both the `Mailbox.msg` file on the domain manager and the local `Intercom.msg` file on the fault-tolerant agent.
5. The **batchman** process on the fault-tolerant agent reads the message in the `Intercom.msg` file and updates the local copy of the Symphony file.
6. When job `job1` completes processing, the **jobman** process updates the `Mailbox.msg` file with the information that says that `job1` completed.
7. The **mailman** process reads the information in the `Mailbox.msg` file, and sends a message that `job1` completed to both the `Mailbox.msg` file on the domain manager and the local `Intercom.msg` file on the fault-tolerant agent.
8. The **batchman** process on the fault-tolerant agent reads the message in the `Intercom.msg` file, updates the local copy of the `Symphony` file, and determines the next job that has to be run.

For information on how to tune job processing on a workstation, refer to the *HCL Workload Automation: Administration Guide*.

Support for Internet Protocol version 6

HCL Workload Automation supports Internet Protocol version 6 (IPv6) in addition to the legacy IPv4. To assist customers in staging the transition from an IPv4 environment to a complete IPv6 environment, HCL Workload Automation provides IP dual-stack support. In other terms, the product is designed to communicate using both IPv4 and IPv6 addresses simultaneously with other applications using IPv4 or IPv6.

To this end, the `gethostbyname` and the `gethostbyaddr` functions were dropped from HCL Workload Automation as they exclusively support IPv4. They are replaced by the new `getaddrinfo` API that makes the client-server mechanism entirely protocol independent.

The `getaddrinfo` function handles both name-to-address and service-to-port translation, and returns `sockaddr` structures instead of a list of addresses These `sockaddr` structures can then be used by the socket functions directly. In this way, `getaddrinfo` hides all the protocol dependencies in the library function, which is where they belong. The application deals only with the socket address structures that are filled in by `getaddrinfo`.

Chapter 3. Configuring the job environment

This chapter describes how to customize the way job management is performed on a workstation. This customization is made by assigning locally on each workstation values to variables that have an impact on the processing of **jobman**. This chapter includes the following sections:

- Job environment overview on page 74
- Environment variables exported by jobman on page 75
- Customizing job processing on a UNIX workstation - jobmanrc on page 83
- Customizing job processing for a user on UNIX workstations - .jobmanrc on page 86
- Customizing job processing on a Windows workstation - jobmanrc.cmd on page 88
- Customizing job processing on a Windows workstation - djobmanrc.cmd on page 91

Job environment overview

On each workstation, jobs are launched by the **batchman** production control process. The **batchman** process resolves all job dependencies to ensure the correct order of job processing, and then queues a job launch message to the **jobman** process.

Each of the processes launched by **jobman**, including the configuration scripts and the jobs, retains the user name recorded with the logon of the job. Submitted jobs (jobs, files, or commands submitted not through a scheduled plan, but *manually* by a user) retain the submitting user's name.

The **jobman** process starts a job monitor process that begins by setting a group of environment variables, and then runs a standard configuration script named *TWS_home*/**jobmanrc** which can be customized. The **jobmanrc** script sets variables that are used to configure locally on the workstation the way jobs are launched, regardless of the user.

On UNIX® workstations, if the user is allowed to use a local configuration script, and the script *USER_HOME*/**.jobmanrc** exists, the local configuration script **.jobmanrc** is *also* run. The job is then run either by the standard configuration script, or by the local one. The results of job processing are reported to **jobman** which, in turn, updates the `Mailbox.msg` file with the information on job completion status. To have jobs run with the user's environment, add the following instruction in the local configuration script:

```
. $USER_home/.profile
```

Note: Before adding the `.profile` in the `.jobmanrc` file, make sure that it does not contain any *stty* setting or any step that requires user manual intervention. In case it does, add in the `.jobmanrc` file only the necessary steps contained in the `.profile`.

On Windows workstations the local configuration script djobmanrc.cmd is run if it exists in the user's Documents and Settings directory which is represented by the environment variable %USERPROFILE% and depends on the Windows language installation. The djobmanrc.cmd script will be ran by jobmanrc.cmd script.

Environment variables exported by jobman

The variables listed in Table 9: Job environment variables for Windows on page 75 are set locally on the workstation and exported by **jobman** on Windows® operating systems:

Table 9. Job environment variables for Windows®

Variable Name	Value
COMPUTERNAME	The value of the *COMPUTERNAME* set in the user environment.
HOME	The path where the HCL Workload Automation instance was installed.
HOMEDRIVE	The value of the *HOMEDRIVE* set in the user environment.
HOMEPATH	The value of the *HOMEPATH* set in the user environment.
LANG	The value of the *LANG* set in the user environment. If not set, its value is set to C.
LOGNAME	The login user's name.
MAESTRO_OUTPUT_STYLE	The setting for output style for long object names.
SystemDrive	The value of the *SYSTEMDRIVE* set in the user environment.

Table 9. Job environment variables for Windows® (continued)

Variable Name	Value
SystemRoot	The value of the *SYSTEMROOT* set in the user environment.
TEMP	The value of the *TEMP* set in the user environment. If not specified, its value is set to `c:\temp`.
TIVOLI_JOB_DATE	The scheduled date for a job.
TMPTEMP	The value of the *TMP* set in the user environment. If not specified, its value is set to `c:\temp`.
TMPDIR	The value of the *TMPDIR* set in the user environment. If not specified, its value is set to `c:\temp`.
TWS_PROMOTED_JOB	Applies to the Workload Service Assurance functions. Can be `YES` or `No`. When the value is `YES`, it means that the job (a critical job or one of its predecessors) was promoted.
TZ	The time zone, if it was set in the operating system environment.
UNISON_CPU	The name of this workstation.
UNISON_DIR	The value of the *UNISON_DIR* set in the user environment.

Table 9. Job environment variables for Windows® (continued)

Variable Name	Value
UNISON_EXEC_PATH	The `jobmanrc` fully qualified path.
UNISONHOME	The path where the HCL Workload Automation instance was installed.
UNISON_HOST	The name of the host CPU.
UNISON_JOB	The absolute job identifier: `worktation#sched_id.job`.
UNISON_JOBNUM	The job number.
UNISON_MASTER	The name of the master workstation.
UNISON_RUN	The HCL Workload Automation current production run number.
UNISON_SCHED	The job stream name.

Table 9. Job environment variables for Windows® (continued)

Variable Name	Value
UNISON_SCHED_DATE	The HCL Workload Automation production date (yymmdd) reported in the header of the `Symphony` file.
UNISON_SCHED_ID	The *jobstreamID* of the job stream containing the job in process.
UNISON_SCHED_IA	The date when the job stream has been added to the plan.
UNISON_SCHED_EPOCH	The HCL Workload Automation production date expressed in epoch form.
UNISON_SHELL	The login shell of the user running the job.
UNISON_STDLIST	The path name of the standard list file of the job.
UNISON_SYM	The `Symphony` record number of the launched job.
USERDOMAIN	The value of the *USERDOMAIN* set in the user environment.
USERNAME	The value of the *USERNAME* set in the user environment.

Table 9. Job environment variables for Windows® (continued)

Variable Name	Value
USERPROFILE	The value of the *USERPROFILE* set in the user environment.

The variables listed in Table 10: Job environment variables for UNIX on page 79 are set locally on the workstation and exported by **jobman** on UNIX® operating systems:

Table 10. Job environment variables for UNIX®

Variable Name	Value
HOME	The home directory of the user.
LANG	The value of the *LANG* set in the user environment.
LD_LIBRARY_PATH	The value of the *LD_LIBRARY_PATH* set in the user environment.
LD_RUN_PATH	The value of the *LD_RUN_PATH* set in the user environment.
LOGNAME	The login user name.
MAESTRO_OUTPUT_STYLE	The setting for output style for long object names.

Table 10. Job environment variables for UNIX® (continued)

Variable Name	Value
PATH	`/bin:/usr/bin`
TIVOLI_JOB_DATE	The scheduled date for a job.
TWS_PROMOTED_JOB	Applies to the Workload Service Assurance functions. Can be `YES` or `No`. When the value is `YES`, it means that the job (a critical job or one of its predecessors) was promoted.
TWS_TISDIR	The value of the *TWS_TISDIR* set in the user environment.
TZ	The time zone set.
UNISON_CPU	The name of this workstation.
UNISON_DIR	The value of the *UNISON_DIR* set in the user environment.
UNISON_EXEC_PATH	The `.jobmanrc` fully qualified path.

Table 10. Job environment variables for UNIX® (continued)

Variable Name	Value
UNISONHOME	The path where the HCL Workload Automation instance was installed.
UNISON_HOST	The name of the host CPU.
UNISON_JOB	The absolute job identifier: `worktation#sched_id.job`.
UNISON_JOB NUM	The job number.
UNISON_MAS TER	The name of the master workstation.
UNISON_RUN	The HCL Workload Automation current production run number.
UNISON_SCHED	The job stream name.
UNISON_SCHED _DATE	The HCL Workload Automation production date (yymmdd) reported in the header of the `Symphony` file.

Table 10. Job environment variables for UNIX® (continued)

Variable Name	Value
UNISON_SCHED_ID	The *jobstreamID* of the job stream containing the job in process.
UNISON_SCHED_IA	The date when the job stream has been added to the plan.
UNISON_SCHED_EPOCH	The HCL Workload Automation production date, expressed in epoch form.
UNISON_SHELL	The login shell of the user running the job.
UNISON_STDLIST	The path name of the standard list file of the job.
UNISON_SYM	The `Symphony` record number of the launched job.

Customizing date formatting in the stdlist

About this task

You can use an environment variable named ***UNISON_DATE_FORMAT*** to specify the date format that is used for the date in the header and in the footer of the `stdlist` file. This variable can be set on both UNIX® and Windows® workstations and must be set before starting HCL Workload Automation processes on that workstation to become effective. To set this variable, follow these steps:

On UNIX® workstations

1. Add the statement to export the *UNISON_DATE_FORMAT* variable in the root `.profile` file.
2. Run the `.profile` file.
3. Run **conman shutdown** and then **./StartUp.sh**.

On Windows® workstations

1. From the System Properties set the *UNISON_DATE_FORMAT* in the System Variable.
2. Run **conman shutdown** and then **StartUp**.

These are some examples of the settings used to display the year format in the date field in the header and footer of the `stdlist` file. The setting:

```
UNISON_DATE_FORMAT = "%a %x %X %Z %Y"
```

produces an output with the following format:

```
Fri 13/10/23 11:05:24 AM GMT 2023
```

The setting:

```
UNISON_DATE_FORMAT = "%a %x %X %Z"
```

produces an output with the following format:

```
Fri 13/10/23 11:05:24 AM GMT
```

Set this variable locally on every workstation for which you want to display the 4-digit year format. If omitted, the standard 2-digit format is used.

Customizing job processing on a UNIX workstation - jobmanrc

About this task

A standard configuration script template named *TWS_home*`/config/`**`jobmanrc`** is supplied with HCL Workload Automation. It is installed automatically as *TWS_home*`/jobmanrc`. This script can be used by the system administrator to set the required environment before each job is run. To alter the script, make your modifications in the working copy (*TWS_home*`/jobmanrc`), leaving the template file unchanged. The file contains variables which can be configured, and comments to help you understand the methodology. Table 11: Variables defined by default in the jobmanrc file on page 83 describes the `jobmanrc` variables.

Table 11. Variables defined by default in the jobmanrc file

Variable Name	Value
UNISON_JCL	The path name of the job's script file.

Table 11. Variables defined by default in the jobmanrc file (continued)

Variable Name	Value
UNISON_STDLIST	The path name of the job's standard list file.
UNISON_EXIT	**yes** | **no** If set to **yes**, the job ends immediately if any command returns a nonzero exit code. If set to **no**, the job continues to run if a command returns a nonzero exit code. Any other setting is interpreted as **no**.
LOCAL_RC_OK	**yes** | **no** If set to **yes**, the user's local configuration script is run (if it exists), passing *$UNISON_JCL* as the first argument. The user might be allowed or denied this option. See Customizing job processing for a user on UNIX workstations - .jobmanrc on page 86 for more information. If set to **no**, the presence of a local configuration script is ignored, and *$UNISON_JCL* is run. Any other setting is interpreted as **no**.
MAIL_ON_ABEND	**yes** | **no** For UNIX® operating systems: If set to **yes**, a message is mailed to the login user's mailbox if the job ends with a non zero exit code. This can also be set to one or more user names, separated by spaces so that a message is mailed to each user. For example, "root mis sam mary". If set to **no**, no messages are mailed if the job abends. Abend messages have the following format: *cpu#sched.job* *jcl-file* `failed with` *exit-code* `Please review` *standard-list-filename* You can change the wording of the message or translate the message into another language. For an explanation of how to do this, see Customizing the MAIL_ON_ABEND section of jobmanrc on page 85.

Table 11. Variables defined by default in the jobmanrc file (continued)

Variable Name	Value
SHELL_TYPE	**standard** | **user** | **script** If set to **standard**, the first line of the JCL file is read to determine which shell to use to run the job. If the first line does not start with **#!**, then `/bin/sh` is used to run the local configuration script or *$UNISON_JCL*. Commands are echoed to the job's standard list file. If set to user, the local configuration script or *$UNISON_JCL* is run by the user's login shell (*$UNISON_SHELL*). Commands are echoed to the job's standard list file. If set to **script**, the local configuration script or *$UNISON_JCL* is run directly, and commands are not echoed unless the local configuration script or *$UNISON_JCL* contains a **set -x** command. Any other setting is interpreted as **standard**.
USE_EXEC	**yes** | **no** If set to **yes**, the job, or the user's local configuration script is run using the **exec** command, thus eliminating an extra process. This option is overridden if *MAIL_ON_ABEND* is also set to **yes**. Any other setting is interpreted as **no**, in which case the job or local configuration script is run by another shell process.

Customizing the MAIL_ON_ABEND section of jobmanrc

About this task

You can modify the wording used in the message sent to the users specified in the ***MAIL_ON_ABEND*** field of the *TWS_home*`/jobmanrc` configuration file by accessing that file and changing the wording in the parts highlighted in bold:

```
# Mail a message to user or to root if the job fails.

if [ "$MAIL_ON_ABEND" = "YES" ]
then
  if [ $UNISON_RETURN -ne 0 ]
  then
  mail $LOGNAME <<-!
        $UNISON_JOB
        \'$UNISON_JCL\' failed with $UNISON_RETURN
        Please review $UNISON_STDLIST
!
  fi
elif [ "$MAIL_ON_ABEND" = "ROOT" ]
then
```

```
  if [ $UNISON_RETURN -ne 0 ]
  then
  mail root <<-!
        $UNISON_JOB
        \'$UNISON_JCL\' failed with $UNISON_RETURN
        Please review $UNISON_STDLIST
!
  fi
elif [ "$MAIL_ON_ABEND" != "NO" ]
then
  if [ $UNISON_RETURN -ne 0 ]
  then
  mail $MAIL_ON_ABEND <<-!
        $UNISON_JOB
        \'$UNISON_JCL\' failed with $UNISON_RETURN
        Please review $UNISON_STDLIST
!
  fi
fi
```

Customizing job processing for a user on UNIX® workstations - .jobmanrc

About this task

On UNIX® workstations, the local configuration script .jobmanrc permits users to establish a required environment when processing their own jobs. Unlike the jobmanrc script, the .jobmanrc script can be customized to perform different actions for different users. Each user defined as *tws_user* can customize in the home directory the .jobmanrc script to perform pre- and post-processing actions. The .jobmanrc script is an extra step that occurs before the job is actually launched.

The .jobmanrc script runs only under the following conditions:

- The standard configuration script, `jobmanrc`, is installed, and the environment variable ***LOCAL_RC_OK*** is set to **yes** (see Table 11: Variables defined by default in the jobmanrc file on page 83).
- If the file *TWS_home*`/localrc.allow` exists, the user's name must appear in the file. If the *TWS_home*`/localrc.allow` file does not exist, the user's name must not appear in the file, *TWS_home*`/localrc.deny`. If neither of these files exists, the user is permitted to use a local configuration script.
- The local configuration script is installed in the user's home directory (*USER_home*`/.jobmanrc`), and it has execute permission.

Jobs are not automatically run, the command or script must be launched from inside the **.jobmanrc**. Depending on the type of process activity you want to perform, the command or script is launched differently. Follow these general rules when launching scripts from inside **.jobmanrc**:

- Use **eval** if you want to launch a command.
- Use either **eval** or **exec** if you want to launch a script that does not need post processing activities.
- Use **eval** if you want to launch a script that requires post processing activities.

If you intend to use a local configuration script, it must, at a minimum, run the job's script file (*$UNISON_JCL*). HCL Workload Automation provides you with a standard configuration script, `jobmanrc`, which runs your local configuration script as follows:

```
$EXECIT $USE_SHELL $USER_home/.jobmanrc "$UNISON_JCL" $IS_COMMAND
```

where:

- The value of *USE_SHELL* is set to the value of the `jobmanrc` ***SHELL_TYPE*** variable (see Table 11: Variables defined by default in the jobmanrc file on page 83).
- *IS_COMMAND* is set to **yes** if the job was scheduled or submitted in production using **submit docommand**.
- *EXECIT* is set to **exec** if the variable ***USE_EXEC*** is set to **yes** (see Table 11: Variables defined by default in the jobmanrc file on page 83), otherwise it is null.

All the variables exported into jobmanrc are available in the .jobmanrc shell, however, variables that are defined, but not exported, are not available.

The following example shows how to run a job's script file, or command, in your local configuration script:

```
#!/bin/ksh
PATH=TWS_home:TWS_home/bin:$PATH
export PATH
/bin/sh -c "$UNISON_JCL"
```

The following is an example of a `.jobmanrc` that does processing based on the exit code of the user's job:

```
#!/bin/sh
#
PATH=TWS_home:TWS_home/bin:$PATH
export PATH
/bin/sh -c "$UNISON_JCL"
#or use eval "$UNISON_JCL" and the quotes are required
RETVAL=$?
if [ $RETVAL -eq 1 ]
then
echo "Exit code 1 - Non Fatal Error"
exit 0
elif [ $RETVAL -gt 1 -a $RETVAL -lt 100 ]
then
conman "tellop This is a database error - page the dba"
elif [ $RETVAL -ge 100 ]
then
conman "tellop Job aborted. Please page the admin"
fi
```

If jobs on dynamic agents fail with exit code `126` displaying error "**script.sh**: cannot execute `[Text file busy]`", and `.jobmanrc` is used, add the following:

```
eval $JCL

stat=$?

if [ $stat -eq 126 ]
```

```
then

echo "JOB ERROR: The command $JCL ended with error 126."

echo "JOB ERROR: try the workaround"

echo running ksh $JCL

ksh $JCL

stat=$?

if [ $stat -eq 126 ]

then

echo "#!/bin/ksh" > $HOME/tmp.$$.sh

echo $JCL >> $HOME/tmp.$$.sh

chmod +x $HOME/tmp.$$.sh

echo running ksh $HOME/tmp.$$.sh

ksh $HOME/tmp.$$.sh

stat=$?

rm $HOME/tmp.$$.sh

else

echo "JOB OK: using workaround 1, exit code=$stat"

fi

fi

echo rc $stat

exit $stat
```

Customizing job processing on a Windows workstation - jobmanrc.cmd

About this task

A standard configuration script template named *TWS_home*\config\jobmanrc.cmd is supplied with HCL Workload Automation. It is installed automatically as *TWS_home*\jobmanrc.cmd. You can use this command file to set the required environment before each job is run. To alter the file, make your modifications in the working copy (*TWS_home*\jobmanrc.cmd), leaving the template

file unchanged. The file contains variables which can be configured, and comments to help you understand the methodology. Table 12: Variables defined by default in the jobmanrc.cmd file on page 89 describes the `jobmanrc.cmd` variables.

Table 12. Variables defined by default in the jobmanrc.cmd file

Variable Name	Value
HOME	The path to the *TWS_home* directory
POSIXHOME	The path to the *TWS_home* directory in a POSIX complaint format
LOCAL_RC_OK	• If set to **yes**, the user's local configuration script is run, if existing. • If set to **no**, the presence of a local configuration script is ignored. Any other setting is interpreted as **no**.
MAIL_ON_ABEND	• If set to **YES**, an email is sent to the email ID defined in the *email_ID* variable, if the job ends in error. • If set to any value other than **YES** or **NO**, an email is sent to the email ID specified in this variable if the job ends in error. • If set to **NO**, no messages are sent if the job ends in error. For more details, see Customizing the MAIL_ON_ABEND section of jobmanrc.cmd on page 91.

Running multiple Windows commands in a single job

By default, if you define a job to run two or more Windows commands or `.CMD` files separated by the ampersand (`&`) character, only the first command runs and subsequent commands are ignored.

To run a chain of commands (for example, `cmd1.cmd & cmd2.cmd`), you must modify the `jobmanrc.cmd` file as follows:

1. **Modify the jobmanrc.cmd script**

   Update the following functions in your `jobmanrc.cmd` file:

- **Update the `:getargs` function:**

  Ensure the function is defined exactly as shown below:

  ```
  :getargs
  set ARGS=%*
  ```

- **Update the `:launch_args` function:**

  Modify the logic to handle the command chain and include the `no_local_rc` logic for string parsing:

  ```
  :launch_args
  REM Launch job based on ARGS (JCL)
  if /I "%LOCAL_RC_EXISTS%"=="YES" (goto :yes_local_rc) else (goto :no_local_rc)

  :yes_local_rc
  "%USERPROFILE%\jobmanrc.cmd"
  goto :continue

  :no_local_rc
  set doublequote=%ARGS:~0,1%
  set TEST=%doublequote:"=X%

  if "%TEST%" == "X " (goto :remove_and_launch) else (goto :launch)

  :launch
  call %ARGS%
  goto :continue

  :remove_and_launch
  call %ARGS:~1,-1%
  goto :continue

  :continue
  ```

  Note: Ensure there are two spaces after the *X* in the line `if "%TEST%" == "X "`.

2. **Update the Job Definition**

   To prevent `jobmanrc.cmd` from interpreting the ampersand (`&`) as a script separator during its own internal logic, you must wrap the command in an additional set of double quotes within the job definition. The script logic added in Step 1 (`call %ARGS:~1,-1%`) automatically removes these extra quotes just before the command starts.

Example: Updating a composer definition

Change a standard definition:

```
WIN-V91#CHAIN
DOCOMMAND "dir & echo hi"
STREAMLOGON mdm91
DESCRIPTION "Added via composer."
TASKTYPE WINDOWS
RECOVERY STOP
```

To the modified version with escaped double quotes:

```
WIN-V91#CHAIN
DOCOMMAND "\"dir & echo hi\""
STREAMLOGON mdm91
DESCRIPTION "Added via composer."
TASKTYPE WINDOWS
RECOVERY STOP
```

Customizing the MAIL_ON_ABEND section of jobmanrc.cmd

About this task

You can modify the wording used in the message sent to the users specified in the ***MAIL_ON_ABEND*** field of the *TWS_home*/jobmanrc.cmd configuration file by accessing that file and changing the wording in the parts highlighted in bold. To clarify how to generate the email message, a sample mail program with name bmail.exe is used.

```
if /I "%MAIL_ON_ABEND%"=="NO" (goto :out) else (goto :mail_on_abend)

:mail_on_abend
REM ******email, task or other action inserted here ********************
if /I "%MAIL_ON_ABEND%"=="YES" (goto :email) else (goto :email_spec)

:email
c:\"Program Files"\utils\bmail.exe -s smtp.yourcompany.com -t %EMAIL_ID%
-f %COMPUTERNAME%@yourcompany.com -h -a "Subject: Job %UNISON_JOB% abended"
-b "Job %UNISON_JOB% Job Number %UNISON_JOBNUM% abended"
goto :out

:email_spec
REM set > c:\tmp\abended_jobs\%UNISON_JOB%.j%UNISON_JOBNUM%
c:\"Program Files"\utils\bmail.exe -s smtp.yourcompany.com -t %MAIL_ON_ABEND%
-f %COMPUTERNAME%@yourcompany.com -h -a "Subject: Job %UNISON_JOB% abended"
-b "Job %UNISON_JOB% Job Number %UNISON_JOBNUM% abended“
```

Customizing job processing on a Windows workstation - djobmanrc.cmd

About this task

On Windows workstations, you can use the local configuration script djobmanrc.cmd to establish a specific environment when processing your custom jobs. Unlike the jobmanrc.cmd script, you can customize the djobmanrc.cmd script to perform different actions for different users.

The following conditions apply:

- The script must contain all environment application variables or paths necessary for HCL Workload Automation to launch correctly.
- The script must exist if a user-specific environment for running job is required or if an email must be sent to the job logon user when the HCL Workload Automation job ends in error.

To create a custom djobmanrc.cmd script, perform the following steps:

1. Logon as the user who defines environment variables for launching HCL Workload Automation jobs.
2. Open a DOS command prompt.
3. Type the **set** command redirecting standard output to a flat file named `user_env`.
4. Create a file named djobmanrc.cmd in the user's Documents and Settings directory with the following default text at the beginning:

```
@ECHO OFF
    echo Invoking %USERNAME% DJOBMANRC.CMD V.1
    set USERPROFILE=%USERPROFILE%
    ::Setup User Environment Phase
```

5. Edit the `user_env` file created in step 3.
6. Insert the **set** command on each line before each environment variable.
7. Add the changes to the PATH variable at the end of the djobmanrc.cmd in a string similar to the following:

```
set PATH=<TWSHOME>;<TWSHOME>\bin;%PATH%
```

8. Add the following text at the end of the `user_env` file and replace the string *user email id* with the email ID of the user that receives the email notification if the job ends in error.

```
set EMAIL_ID=<user email id>
    ::Launch Operation Phase
    %ARGS%
    ::Post Operations Phase
    :out
```

9. Add the updated `user_env` file to the end of the djobmanrc.cmd file. The edited djobmanrc.cmd file should look like the following example:

```
@ECHO OFF
    echo Invoking %USERNAME% DJOBMANRC.CMD V.1
    set USERPROFILE=%USERPROFILE%
    ::Setup User Environment Phase
    set ALLUSERSPROFILE=C:\Documents and Settings\All Users
    set APPDATA=C:\Documents and Settings\petes\Application Data
    set CommonProgramFiles=C:\Program Files\Common Files
    set COMPUTERNAME=PSOTOJ
    set ComSpec=C:\WINDOWS\system32\cmd.exe
    set CURDRIVE=C
    set FP_NO_HOST_CHECK=NOset
    set HOMEDRIVE=c:
    set HOMEPATH=\docs
    set LOGONSERVER=\\PSOTOJ
    set NEWVAR=c:\tmp\tmp\mlist1
    set NUMBER_OF_PROCESSORS=1
    set OPC_CLIENT_ROOT=C:\opc\Client
    set OS=Windows_NT
    set Path=C:\Program Files\utils;C:\PROGRAM
    FILES\THINKPAD\UTILITIES;C:\WINDOWS\system32;C:\WINDOWS;C:\WINDOWS\System32\Wbem;C:\Program
    Files\IBM\Infoprint Select;C:\Utilities;C:\Notes;C:\Program Files\IBM\Trace Facility\;C:\Program
    Files\IBM\Personal Communications\;C:\Program Files\XLView\;C:\lotus\compnent\;C:\WINDOWS\Downloaded
    Program Files;C:\Program Files\Symantec\pcAnywhere\;"C:\Program Files\Symantec\Norton Ghost
    2003\";C:\Infoprint;
    set PATHEXT=.COM;.EXE;.BAT;.CMD;.VBS;.VBE;.JS;.JSE;.WSF;.WSH
    set PCOMM_Root=C:\Program Files\IBM\Personal Communications\
    set PDBASE=C:\Program Files\IBM\Infoprint Select
    set PDHOST=
    set PD_SOCKET=6874
    set PROCESSOR_ARCHITECTURE=x86
    set PROCESSOR_IDENTIFIER=x86 Family 6 Model 9 Stepping 5, GenuineIntel
    set PROCESSOR_LEVEL=6
    set PROCESSOR_REVISION=0905
    set ProgramFiles=C:\Program Files
    set PROMPT=$P$G
    set SESSIONNAME=Console
```

```
set SystemDrive=C:
set SystemRoot=C:\WINDOWS
set TEMP=C:\DOCUME~1\petes\LOCALS~1\Temp
set TMP=C:\DOCUME~1\petes\LOCALS~1\Temp
set tvdebugflags=0x260
set tvlogsessioncount=5000
set TWS4APPS_JDKHOME=c:\win32app\TWS\pete\methods\_tools
set USERDOMAIN=PSOTOJ
set USERNAME=petes
set USERPROFILE=C:\Documents and Settings\petes
set windir=C:\WINDOWSPATH=c:\win32app\tws\twsuser:c:\win32app\tws\twsuser\bin:%PATH%
set PATH=c:\win32app\TWS\twsuser;c:\win32app\TWS\twsuser\bin;%PATH%
set EMAIL_ID=johndoe@yourcompany.com
::Launch Operation Phase
%ARGS%
::Post Operations Phase
:out
```

The Launch Operations Phase in the script is where script, binary or command defined for job is completed. The “%ARGS%? text is required.

The Post Operations Phase in the script is where a job exit code might be re-adjusted from ABEND to SUCC state, changing a non-zero exit code to a zero exit code. Some applications might have exit codes that might be warnings. HCL Workload Automation evaluates exit codes as either zero or non-zero. Zero exit codes indicate a job in “SUCC? state. All other codes indicate a job in ABEND state. Specific non-zero exit codes can be adjusted if necessary. The following example shows what might be included in the Post Operations Phase. The example retrieves the exit code of the defined job to determine how to handle itbased on the `If` statements:

```
set EMAIL_ID=johndoe@yourcompany.com
    ::Launch Operation Phase
    %ARGS%
    ::Post Operations Phase
    set RETVAL=%ERRORLEVEL%
    if “%RETVAL%?==“0? goto out
    if “%RETVAL%?==“1? set RETVAL=0
    if “%RETVAL%?==“6? set RETVAL=0
    :out
    exit %RETVAL%
```

Setting up options for using the user interfaces

About this task

To use the Dynamic Workload Console, the connection parameters are supplied within the console and saved as part of its configuration.

To use the HCL Workload Automation command line client, you need to provide the following setup information (called the ***connection_parameters***) to connect to the master domain manager via HTTP/HTTPS using the WebSphere Application Server Liberty infrastructure:

hostname

The hostname of the master domain manager.

port number

The port number used when establishing the connection with the master domain manager.

JSON Web Token

Specify the JWT to be used for authentication between the master domain manager and agents. You can retrieve the token from the Dynamic Workload Console. This parameter is mutually exclusive with the **username** and **password** parameters. The JWT authentication applies to the commands you launch in the shell where you specify the JWT.

user name, password

The credentials, user name and password, of the *TWS_user*. This parameter is mutually exclusive with the JSON Web Token parameter.

proxy hostname

The proxy hostname used in the connection with the HTTP protocol. This parameter is mutually exclusive with the JSON Web Token parameter.

proxy port number

The proxy port number used in the connection with the HTTP protocol.

protocol

The protocol used during the communication. This can be HTTP with basic authentication, or HTTPS with certificate authentication.

timeout

The timeout indicating the maximum time the connecting user interface program can wait for the master domain manager response before considering the communication request as failed.

default workstation

The workstation name of the master domain manager you want to connect to.

SSL parameters

If you have configured your network to use SSL to communicate between the interfaces and the master domain manager, you need also to supply the appropriate set of SSL parameters (which depends on how your SSL is configured).

In the case of the command line client installed on the master domain manager, this configuration is performed automatically at installation.

For the command line client installed on other workstations, this information can be supplied either by storing it in properties files on those workstations, or by supplying the information as part of the command string of the commands you use.

The properties files referred to are the `localopts` and `useropts` files:

localopts

This contains a set of parameters applicable to the local workstation for a specific instance of the installed product. See the localopts details in the *Administration Guide*

useropts

This contains a subset of those localopts parameters that have custom values for a specific user. The path of this file is within the user's home directory, which maintains the privacy of this information. See the useropts details in the *Administration Guide*

Because HCL Workload Automation supports multiple product instances installed on the same machine, there can be more than one `useropts` file instance of each user. The possibility to have more `useropts` files, having a different name each, provides the ability to specify different sets of connection settings for users defined on more than one instance of the product installed on the same machine.

In the `localopts` file of each instance of the installed product the option named ***useropts*** identifies the file name of the `useropts` file that has to be accessed to connect to that installation instance.

This means that, if two HCL Workload Automation instances are installed on a machine and a system user named `operator` is defined as user in both instances, then in the `localopts` file of the first instance the local option `useropts = useropts1` identifies the `useropts1` file containing the connection parameters settings that user `operator` needs to use to connect to that HCL Workload Automation instance. On the other hand, in the `localopts` file of the second HCL Workload Automation instance the local option `useropts = useropts2` identifies the `useropts2` file containing the connection parameters settings that user `operator` needs to use to connect to that HCL Workload Automation instance.

Full details of how to configure this access are given in the *Administration Guide*, in the topic entitled "Configuring command-line client access authentication".

Chapter 4. Managing the production cycle

The core part of a job management and scheduling solution is the creation and management of the *production plan*. The production plan is the to-do list that contains the actions to be performed in a stated interval of time on the workstations of the scheduling network using the available resources and preserving the defined relationships and restrictions.

This chapter describes how HCL Workload Automation manages plans.

The chapter is divided into the following sections:

- Plan management basic concepts on page 96
- Customizing plan management using global options on page 119
- Creating and extending the production plan on page 125
- Planman command line on page 129
- Starting production plan processing on page 149
- Automating production plan processing on page 150

Plan management basic concepts

The *production plan* contains information about the jobs to run, on which fault-tolerant agent, and what dependencies must be satisfied before each job is launched. HCL Workload Automation creates the production plan starting from the modeling data stored in the database and from an intermediate plan called the *preproduction plan*. The preproduction plan is automatically created and managed by the product. To avoid problems, the database is locked during the generation of the plan, and is unlocked when the generation completes or if an error condition occurs. The preproduction plan is used to identify in advance the job stream instances and the external follows job stream dependencies involved in a specified time-window.

You use the **JnextPlan** script on the master domain manager to generate the production plan and distribute it across the HCL Workload Automation network.

You can run the JnextPlan command from a command prompt shell on the master domain manager if you are one of the following users:

- The TWS_user user for which you installed the product on that machine, if not disabled by the settings that are defined in the security file.
- Root on UNIX operating systems or Administrator on Windows operating systems, if not disabled by the settings that are defined in the security file.

For additional information on the **JnextPlan** script, refer to Creating and extending the production plan on page 125.

To generate and start a new production plan HCL Workload Automation performs the following steps:

1. Updates the preproduction plan with the objects defined in the database that were added or updated since the last time the plan was created or extended.
2. Retrieves from the preproduction plan the information about the job streams to run in the specified time period and saves it in an intermediate production plan.
3. Includes in the new production plan the uncompleted job streams from the previous production plan.
4. Creates the new production plan and stores it in a file named `Symphony`. The plan data is also replicated in the database.
5. Distributes a copy of the `Symphony` file to the workstations involved in the new product plan processing.
6. Logs all the statistics of the previous production plan into an archive
7. Updates the job stream states in the preproduction plan.

The copy of the newly generated `Symphony` file is deployed starting from the top domain's fault-tolerant agents and domain managers of the child domains and down the tree to all subordinate domains.

Each fault-tolerant agent and domain manager that receives the new `Symphony` file, archives the previous `Symphony` to `Symphony.last` in the path `<TWA_home>/TWS/`, so that a backup copy is maintained. This permits viewing of the previous `Symphony` data in case there were any message updates on the job and job stream states that were lost between the agent and its master domain manager.

Each fault-tolerant agent that receives the production plan can continue processing even if the network connection to its domain manager goes down.

At each destination fault-tolerant agent the HCL Workload Automation processes perform the following actions to manage job processing:

1. Access the copy of the `Symphony` file and read the instructions about which jobs to run.
2. Make calls to the operating system to launch jobs as required.
3. Update its copy of the `Symphony` file with the job processing results and send notification back to the master domain manager and to all full status fault-tolerant agents. The original copy of the `Symphony` file stored on the master domain manager and the copies stored on the backup master domain managers, if defined, are updated accordingly.

This means that during job processing, each fault-tolerant agent has its own copy of the `Symphony` file updated with the information about the jobs it is running (or that are running in its domain and child domains if the fault-tolerant agent is full-status or a domain manager). Also the master domain manager (and backup master domain manager if defined) has the copy of the `Symphony` file that contains all updates coming from all fault-tolerant agents. In this way the `Symphony` file on the master domain manager is kept up-to-date with the jobs that must be run, those that are running, and those that have completed.

The processing that occurs on each workstation involved in the current production plan activities is described in more detail in HCL Workload Automation workstation processes on page 57.

Note: While the current production plan is in process, any changes you make to the plan using **conman** do not affect the definitions in the database, but the replica of the plan data in the database is updated with the changes. Subsequent updates to job instances in the plan are supported, but do not affect the job definitions in the database.

Changes to the objects in the database do not affect the plan until the production plan is extended or created again using the **JnextPlan** script or **planman** command-line interface. Updates to objects in the database do not affect instances of those objects already in the production plan.

Preproduction plan

The preproduction plan is used to identify in advance the job stream instances and the job stream dependencies involved in a specified time period.

This improves performance when generating the production plan by preparing in advance a high-level schedule of the anticipated production workload.

The preproduction plan contains:

- The job stream instances to be run during the covered time interval.
- The external follows dependencies that exist between the job streams and jobs included in different job streams.

A job or job stream that cannot start before another specific external job or job stream is successfully completed is named *successor*. An external job or job stream that must complete successfully before the successor job or job stream can start is named *predecessor*.

HCL Workload Automation automatically generates, expands, and updates, if necessary, the preproduction plan by performing the following steps:

- Removes the job stream instances in COMPLETE and CANCEL states.
- Selects all the job streams scheduled after the end of the current production plan and generates their instances.
- Resolves all job and job stream dependencies, including external follows dependencies, according to the defined matching criteria.

To avoid any conflicts the database is locked during the generation of the preproduction plan and unlocked when the generation completes or if an error condition occurs.

At this stage only the job streams with the time they are scheduled to start and their dependencies are highlighted. All the remaining information about the job streams and the other scheduling objects (calendars, prompts, domains, workstations, resources, files, and users) that will be involved in the production plan for that time period are not included, but are retrieved from the database as soon as the production plan is generated.

When the production plan is extended, old job stream instances are automatically removed. The criteria used in removing these instances takes into account this information:

- The first job stream instance that is not in COMPLETE state at the time the new plan is generated (FNCJSI). This job stream instance can be both a planned instance, that is an instance added to the plan when the production plan is generated, and a job stream instance submitted from the command line during production using the **conman sbs** command.
- The time period between the time FNCJSI is planned to start and the end time of the old production plan.

Assuming **T** is this time period, the algorithm used to calculate which job stream instances are removed from the preproduction plan is the following:

if T < 7

All job stream instances older than 7 days from the start time of the new production plan are removed from the preproduction plan; all job stream instances closer than 7 days to the start time of the new production plan are kept regardless of their states.

if T > 7

All job stream instances older than FNCJSI are removed from the preproduction plan; all job stream instances younger than FNCJSI are kept.

This algorithm is used to ensure that the preproduction plan size does not increase continuously and, at the same time, to ensure that no job stream instance that is a potential predecessor of a job stream newly added to the new preproduction plan is deleted.

For more information about how you can open the preproduction plan in view mode from the Dynamic Workload Console, see the Dynamic Workload Console User's Guide, section about View preproduction plan.

Note: In the HCL Workload Automation for Z terminology the concept that corresponds to the preproduction plan is *long term plan* (LTP).

Identifying job stream instances in the plan

In earlier versions than 8.3 the plan had a fixed duration of one day. Since version 8.3 the plan can cover a period lasting several days or less than one day. This change has added the possibility to have in the same plan more than one instance of the same job stream with the same name, and also the need to define a new convention to uniquely identify each job stream instance in the plan. Each job stream instance is identified in the plan by the following values:

workstation

Specifies the name of the workstation on which the job stream is scheduled to run.

jobstreamname

Corresponds to the job stream name used in earlier versions of HCL Workload Automation.

scheddateandtime

Represents when the job stream instance is planned to start in the preproduction plan. It corresponds to the day specified in the run cycle set in the job stream definition by an **on** clause and the time set in the job stream definition by an **at** or **schedtime** keyword. If set, the **schedtime** keyword is used only to order chronologically the job stream instances in the preproduction plan while, if set, the **at** keyword also represents a dependency for the job stream. For more information about these keywords refer to on on page 364, at on page 325 and schedtime on page 379.

Together with these two values that you can set in the job stream definition, HCL Workload Automation generates and assigns a unique alphanumeric identifier to each job stream instance, the ***jobstream_id***, for its internal processing. For more information on the format of the *jobstream_id* refer to SHOWJOBS on page 659.

You can use any of the two types of identifiers, *workstation#jobstreamname* and *scheddateandtime* instead of *workstation#jobstream_id*, to uniquely identify a job stream instance when managing job streams in the plan using the **conman** command-line program. The default convention used to identify a job stream instance, both in this guide and in the command-line interfaces of the product, is the one that uses *workstation#jobstreamname* and *scheddateandtime*. For more information on how to specify a job stream instance in a command using **conman**, refer to JSSELECT Selecting job streams in commands on page 571.

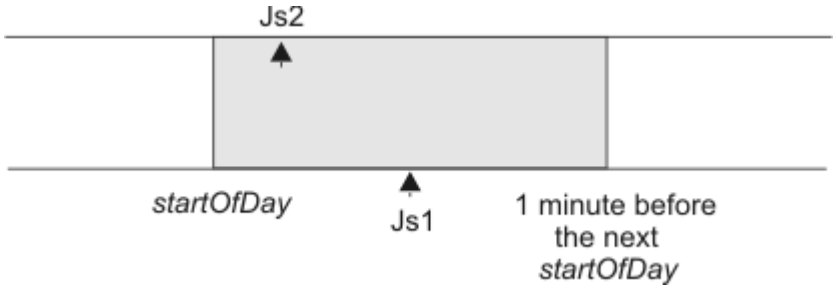
Managing external follows dependencies for jobs and job streams

During the creation of the preproduction plan, all external follows dependencies to job streams and jobs are resolved using four different possible *matching criteria*:

Same day

Considering the job or job stream instances planned to run on the same day. In this case you set the clause **follows**...**sameday** in the object definition. Figure 7: Sameday matching criteria on page 101 shows a job stream named `Js1` which has an external follows dependency on the instance of the job stream `Js2` that is scheduled to start on the same day.

Figure 7. Sameday matching criteria



Below is an example of how to define the involved job streams.

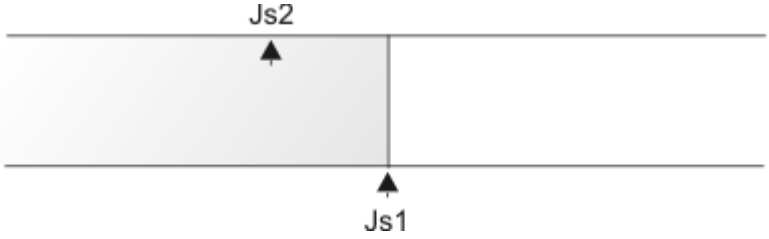
schedule Js2	schedule Js1
on everyday	on everyday
at 0700	at 1000
:job2	follows wk1#Js2 sameday
end	:job1
	end

The job stream `Js1` in not launched until the job stream instance of `Js2` on the workstation `wk1` completes successfully.

Closest preceding

Using the closest job or job stream instance (earlier or same time). The job or job stream instance that HCL Workload Automation uses to resolve the dependency is the closest in time before the instance that includes the dependency. In this case you set the **follows ... previous** clause in the object definition. Figure 8: Closest preceding matching criteria on page 102 shows a job stream named `Js1` which has an external follows dependency on the closest earlier instance of job stream `Js2`. The time frame where the predecessor is searched is greyed out in the figure.

Figure 8. Closest preceding matching criteria



Below is an example of how to define the involved job streams.

schedule Js2	schedule Js1
on Th	on Fr
at 0700	at 1000
:job2	follows wk1#Js2 previous
end	:job1
	end

The job stream `Js1` in not launched until the closest preceding job stream instance of `Js2` on the workstation `wk1` completes successfully.

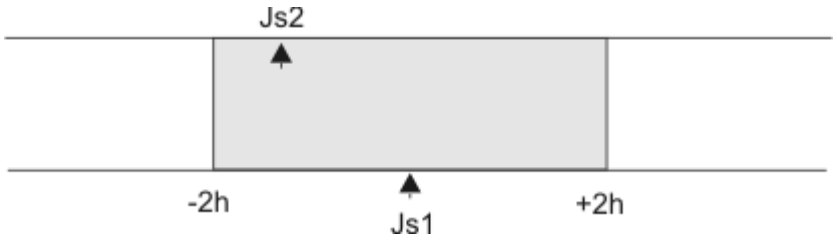
Consider the impact of Daylight Saving Time (DST) "spring forward" transitions. If the scheduled start time (AT time) of the predecessor falls within the skipped hour (for example, between 02:00 and 03:00), its evaluated time is shifted forward. If this shift causes the predecessor to be scheduled after the successor, the PREVIOUS matching criteria does no longer apply. As a result, the dependency will fail to resolve, leaving the successor indefinitely in HOLD status.

Within a relative interval

Considering the job or job stream instances defined in a range with an offset relative to the start time of the dependent job or job stream. For example, from 25 hours before the dependent job stream start time to 5 hours

after the dependent job stream start time. In this case you set the **follows ... relative from ... to ...** clause in the object definition. Figure 9: Within a relative interval matching criteria on page 103 shows a job stream named `Js1` which has an external follows dependency on the job stream instance of `Js2` that starts with an offset of 2 hours with respect to `Js1`. The job or job stream instance that HCL Workload Automation considers to resolve the dependency is the closest one within the relative time interval you chose.

Figure 9. Within a relative interval matching criteria



Below is an example of how to define the involved job streams.

schedule Js2	schedule Js1
on everyday	on everyday
at 0900	at 1000
:job2	follows wk1#Js2 relative from 0200 to 0200
end	:job1
	end

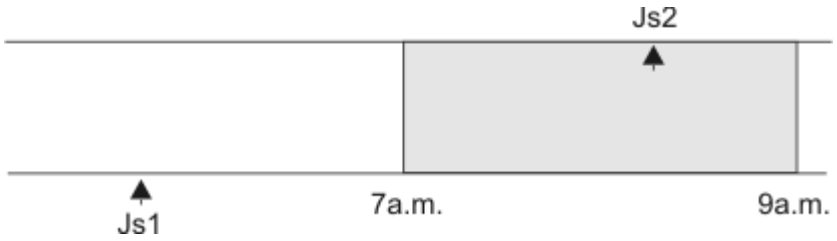
The job stream `Js1` in not launched until the job stream instance of `Js2` on the workstation `wk1` that runs in the 08:00 to 12:00 time frame completes successfully.

Within an absolute interval

Using only the job or job stream instances defined in a range. For example from today at 6:00 a.m. to the day after tomorrow at 5:59 a.m. In this case you set the **follows ... from ... to ...** clause in the object definition. Figure 10: Within an absolute interval matching criteria on page 104 shows a job stream named `Js2` which

has an external follows dependency on the instance of job stream `Js1` that is positioned in the preproduction plan between 7 a.m. and 9 a.m. The job or job stream instance that HCL Workload Automation considers to resolve the dependency is the closest one within the absolute time interval you chose. The time interval specifies the time of the day on which the interval starts and ends, either on the same day as the instance that include the dependency or on a day defined relative to that day.

Figure 10. Within an absolute interval matching criteria



Below is an example of how to define the involved job streams.

schedule Js1	schedule Js2
on everyday	on everyday
at 0800	at 1000
:job1	follows wk1#Js1 from 0700 to 0900
end	:job2
	end

The job stream `Js2` in not launched until the job stream instance of `Js1` on the workstation `wk1` that runs in the 07:00 to 09:00 time frame on the same day completes successfully.

Regardless of which matching criteria are used, if multiple instances of potential predecessor job streams exist in the specified time interval, the rule used by the product to identify the correct predecessor instance is the following:

1. HCL Workload Automation searches for the closest instance that precedes the depending job or job stream start time. If such an instance exists, this is the predecessor instance.
2. If there is no preceding instance, HCL Workload Automation considers the correct predecessor instance as the closest instance that starts after the depending job or job stream start time.

This behavior applies for external follows dependencies between job streams. For external follows dependencies of a job stream or job from another job the criteria are matched by considering the start time of the job stream hosting the predecessor job instead of the start time of the predecessor job itself. Figure 11: Closest preceding predecessor job on page 105 shows in bold the instances of `job1` the successor job or job stream is dependent on.

Figure 11. Closest preceding predecessor job



External follows dependencies are identified between jobs and job streams stored in the database whose instances are added to the preproduction plan when the preproduction plan is automatically created or extended. Job and job stream instances submitted in production from the **conman** command line are written in the preproduction plan but they are not used to recalculate predecessors of external follows dependencies already resolved in the preproduction plan.

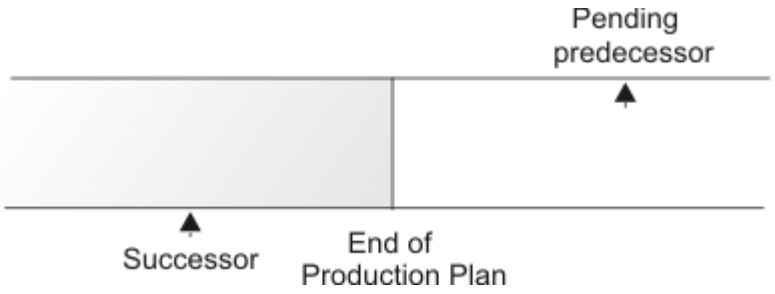
The scheduler classifies follows dependencies as *internal* when they are specified only by their job name within the job stream. It classifies them as *external* when they are specified in the *jobStreamName.workstationName.jobName* format.

When a job stream includes a job with a follows dependency that shares the same job stream name (for example, job stream `schedA` includes a job named `job6` that has a follows dependency on `schedA.job2`), the dependency is added to the plan as an *external* follows dependency. Because the scheduler uses the `sameday` matching criteria to resolve external dependencies, dependencies originated in this way are never added the first time the object is submitted.

A job or job stream not yet included in the production plan, but that can be a potential predecessor of instances of jobs and job streams added to the production plan as the production plan is extended, is called a ***pending predecessor***. A pending

predecessor is like a dummy occurrence created by the planning process to honor a dependency that has been resolved in the preproduction plan, but that cannot be resolved in the current production plan because the predecessor's start time is not within the current production plan end time. Figure 12: Pending predecessor instance on page 106 shows how a pending predecessor and its successor are positioned in the preproduction plan.

Figure 12. Pending predecessor instance



The way in which pending predecessors are managed is strictly linked to whether or not the successor job or job stream is carried forward:

- If the successor is carried forward when the production plan is extended, the predecessor is included in the new production plan and the dependency becomes current. A pending predecessor job or job stream is marked with a `[P]` in the `Dependencies` column in the output of the conman showjobs on page 659 and conman showschedules on page 690 commands.
- If the predecessor is not carried forward when the production plan is extended, the successor is included in the new production plan, but the dependency becomes ***orphaned***. This can happen, for example, if, when extending the production plan, the successor is carried forward and the pending predecessor is not added to the plan because it was flagged as *draft* in the database. The orphaned dependencies are marked with a `[O]` in the `Dependencies` column in the output of the conman showjobs on page 659 command. When dealing with an orphaned dependency you must verify if it can be released and, if so, cancel it.

When the product evaluates matching criteria to resolve external follows dependencies it compares the start times using local time if both job stream instances use the same timezone, or it uses UTC in case they use different timezones. However, if you set the EnLegacyStartOfDayEvaluation to `yes`, the product compares the local times of the job stream instances, regardless of whether the instances are defined on the same timezone or on different timezones.

External follows dependency resolution and status transition examples

This section includes examples for each of the four matching criteria described in the previous paragraphs. In all the examples, the start of day time (SOD) is set to 06:00 AM.

Same day

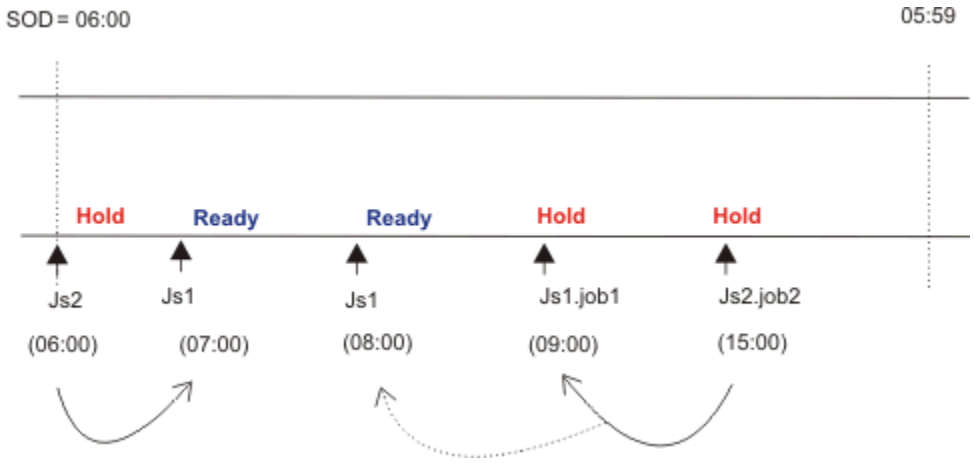
The job or job stream instance to be considered in resolving the dependency is the closest one on the same day in which the instance that includes the dependency is scheduled to run. In this example, two job streams, `Js1` and `Js2`, each have one job. Job stream `Js1` is scheduled to run every day at 08:00 and on Thursdays also at 07:00. `Js1.Job1` runs at 09:00. Job stream `Js2` has no time restrictions and is scheduled by default at the defined start of day time. `Js2.Job2` is scheduled to run at 15:00 and has an external follows dependency on the closest earlier instance of the job stream `Js1` running on the same day. The two job streams are defined in this way:

```
SCHEDULE MY_MASTER#JS1
ON RUNCYCLE RULE1 "FREQ=WEEKLY;BYDAY=TH"
(AT 0700)
ON RUNCYCLE RULE2 "FREQ=DAILY"
(AT 0800)
:
MY_MASTER#JOB1
AT 0900
END

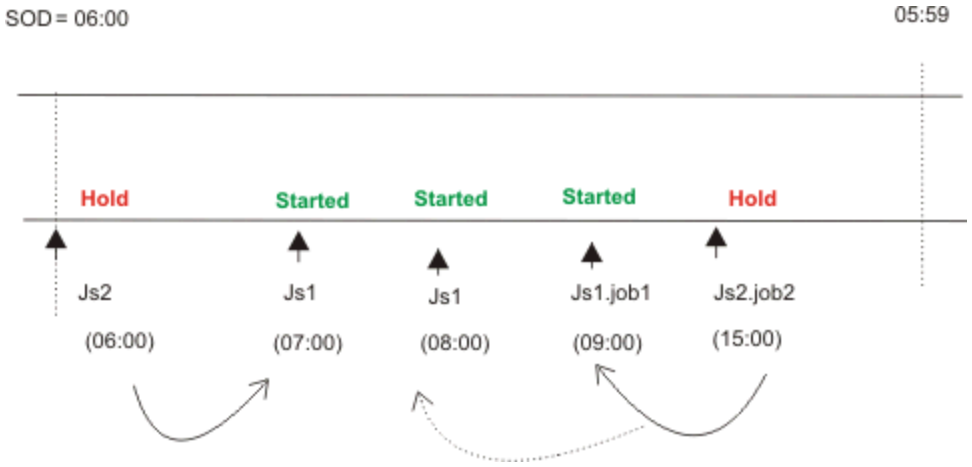
SCHEDULE MY_MASTER#JS2
ON RUNCYCLE RULE2 "FREQ=DAILY;"
FOLLOWS MY_MASTER#JS1.@ SAMEDAY
:
MY_MASTER#JOB2
AT 1500
END
```

When the schedules are included in the plan, the sequence of graphics illustrate how the dependency is resolved:

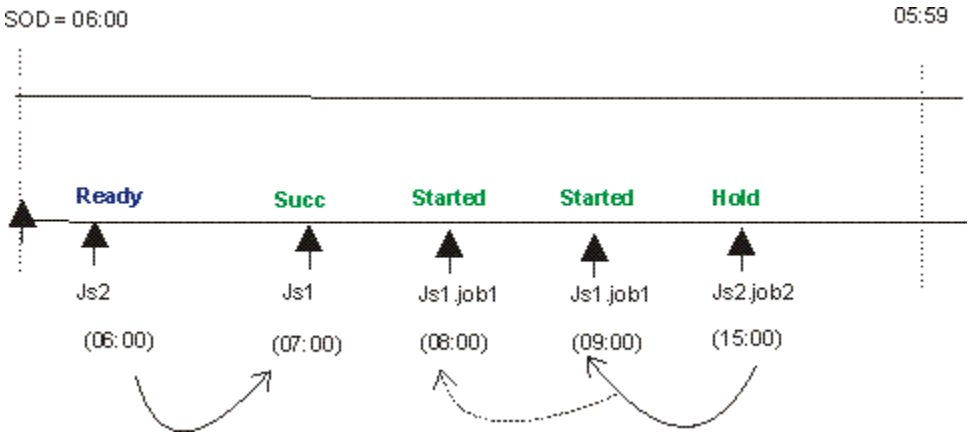
1. On Thursdays, the instance of `Js2` scheduled at 06:00 depends on the instance of `Js1` scheduled to run at 07:00. On any other day of the week, `Js2` has a dependency on the instance of `Js1` scheduled at 08:00. Figure 13: Sameday matching criteria - Step 1: at Start of Day (SOD) on a Thursday on page 107 shows the status of the two job streams in the plan at 06:00 (SOD) on Thursday:
   Figure 13. Sameday matching criteria - Step 1: at Start of Day (SOD) on a Thursday



2. At 09:00, `Js1.job1` starts and `Js1` changes status. `Js2.job2` is held until its scheduled time.Figure 14: Sameday matching criteria - Step 2: at 9:00 on page 108 shows the status of the job streams in the plan at 09:00.
   Figure 14. Sameday matching criteria - Step 2: at 9:00



3. On Thursdays at 15:00, `Js2` changes to ready status and `Js2.job2` starts. Figure 15: Sameday matching criteria - Step 3: at 15:00 on page 108 shows the status of the two job streams in the plan at 15:00.
   Figure 15. Sameday matching criteria - Step 3: at 15:00



Closest preceding

- In this example, two job streams, `Js1` and `Js2`, each have one job. The job in `Js2` has an external follows dependency on the closest preceding instance of the job in `Js1`. The two job streams are defined in this way:

```
SCHEDULE MY_MASTER#JS1
ON RUNCYCLE RULE1 "FREQ=DAILY;"
(AT 0800)
ON RUNCYCLE RULE2 "FREQ=WEEKLY;BYDAY=TH,FR"
(AT 0900)
:
MY_MASTER#JOB1
END
```

```
SCHEDULE MY_MASTER#JS2
ON RUNCYCLE RULE1 "FREQ=DAILY;"
(AT 1200)
FOLLOWS MY_MASTER#JS1.@ PREVIOUS
:
MY_MASTER#JOB2
AT 1500
END
```

Job stream `Js1` runs every day at 0800 and on Thursdays and Fridays also at 0900. Job stream `Js2` runs every day at 1200, and has an external dependency on the closest preceding instance of `Js1`. When the job streams are included in the plan, the sequence of graphics illustrates how the dependency is resolved:

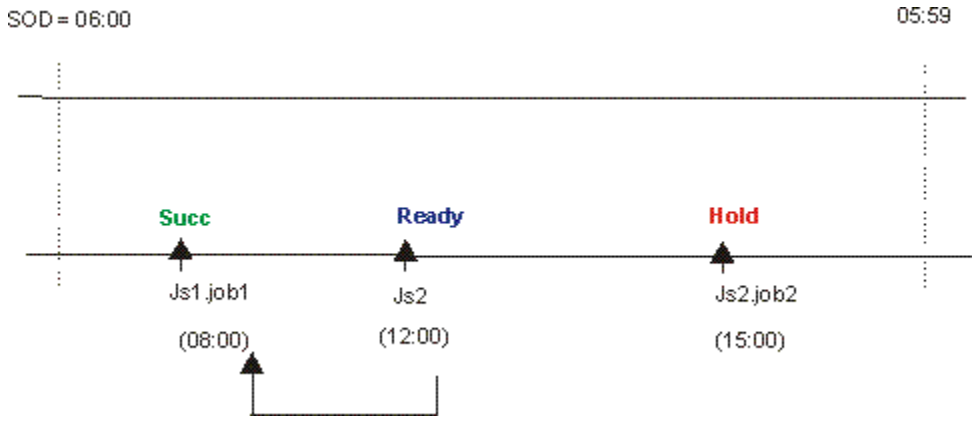
1. Before 12:00 on Thursdays and Fridays, there are two instances of `Js1.Job1`. Job stream `Js2` has a dependency on the instance of `Js1.Job1` that is scheduled to run at 09:00, because it is the closest preceding in terms of time. Figure 16: Closest preceding matching criteria - Step 1: before 08:00 on page 109 shows the status of the two job streams in the plan on Thursdays and Fridays.

   Figure 16. Closest preceding matching criteria - Step 1: before 08:00

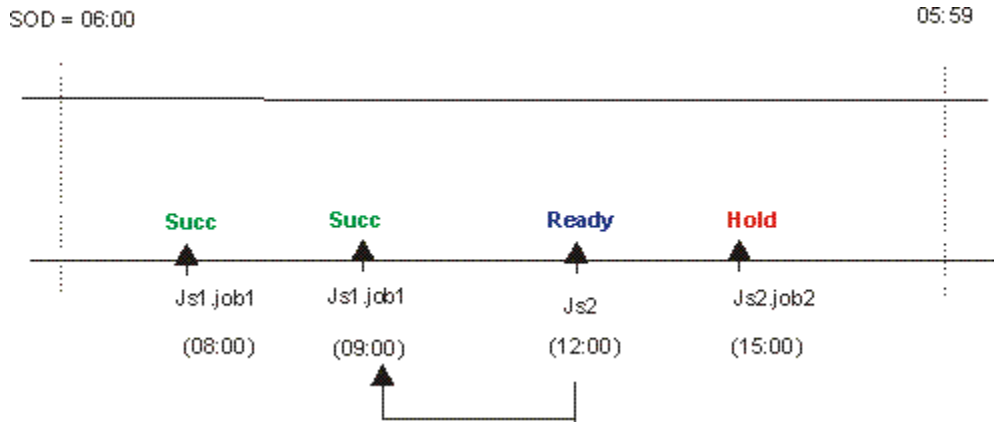
2. On any other day of the week, the only instance of `Js1.Job1` in plan, is the one scheduled to run at 08:00. In this case, `Js2` has a dependency on this instance. When `Job1` completes successfully, the status of `Js2` becomes **Ready**. Figure 17: Closest preceding matching criteria - Step 2: at 08:00 on weekdays except Thursdays and Fridays on page 110 shows the status of the two job streams in the plan on any other weekday except Thursdays and Fridays.

Figure 17. Closest preceding matching criteria - Step 2: at 08:00 on weekdays except Thursdays and Fridays

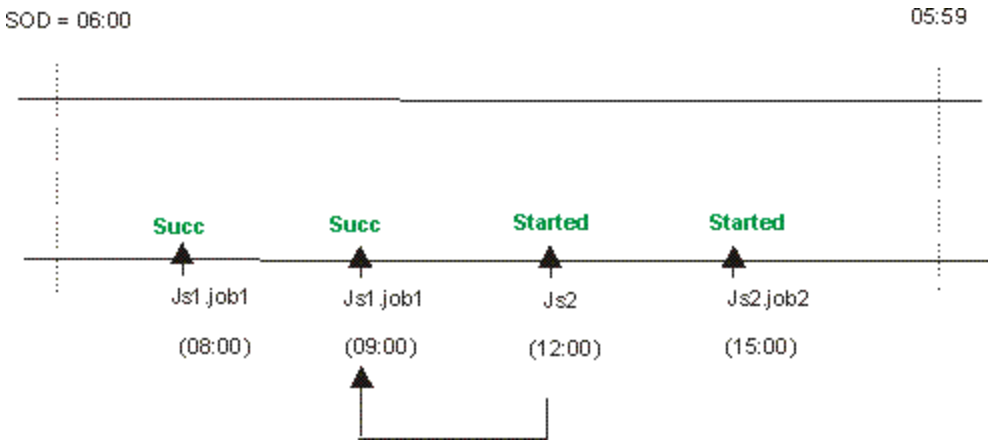


3. On Thursdays and Fridays at 09:00, the second instance of `Js1.Job1` completes successfully. Job stream `Js2` changes to **Ready**. `Js2.Job2` is held until its scheduled start time. Figure 18: Closest preceding matching criteria - Step 3: at 09:00 on Thursdays and Fridays on page 110 shows the status of the two job streams in the plan.

   Figure 18. Closest preceding matching criteria - Step 3: at 09:00 on Thursdays and Fridays



4. At 15:00 the time dependency of `Js2.Job2` is satisfied and `Job2` starts. Figure 19: Closest preceding matching criteria - Step 4: at 15:00 on every day on page 111 shows the status of the two job streams in the plan at 15:00.
   Figure 19. Closest preceding matching criteria - Step 4: at 15:00 on every day



In the job stream definition, run cycle *Rule1* can be substituted by the keywords `ON EVERYDAY`.

- In this second example, the difference between the use of `sameday` and `closest preceding` matching criteria in a plan is described. Job stream `Js1` runs every Friday at 0900, while job stream `Js2` and `Js3` run every Saturday at 0900. The three job streams are defined in this way:

```
SCHEDULE ACCOUNTING#JS1
ON RUNCYCLE RULE1 "FREQ=WEEKLY;BYDAY=FR"
:
ACCOUNTING#JOB1
 AT 0900
END

SCHEDULE ACCOUNTING#JS2
ON RUNCYCLE RULE2 "FREQ=WEEKLY;BYDAY=SA"
FOLLOWS ACCOUNTING#JS1.@ PREVIOUS
:
ACCOUNTING#JOB1
 AT 0900
END

SCHEDULE ACCOUNTING#JS3
ON RUNCYCLE RULE2 "FREQ=WEEKLY;BYDAY=SA"
FOLLOWS ACCOUNTING#JS1.@
:
ACCOUNTING#JOB1
 AT 0900
END
```

Job stream `Js2` has an external dependency on the closest preceding instance of `Js1`, which is resolved as described in the previous example. Job stream `Js3` is defined with `sameday` matching criteria, so it does not have any dependency on job stream `Js1`, because `Js1` is not defined to run on the same day as `Js2`.

Within a relative interval

In this example, the job or job stream instance considered to resolve the dependency is the closest one in a time interval of your choice, which is defined relatively to the time when the instance that includes the dependency is scheduled to run. Job stream `Js1` is scheduled to run every day at 15:00 and on Thursdays also at 08:00. `Js2` is scheduled to run every day at 13:00 and on Thursdays also at 06:00, because no specific time is defined in the run cycle, it is scheduled at start of day time. `Js2` uses the relative interval criteria (-04:00 to +04:00) to determine which instance is used to solve the dependency. The interval is based on the time the job stream enters the plan. The job streams are defined as follows:

```
SCHEDULE MY_MASTER#JS1
ON RUNCYCLE RULE1 "FREQ=WEEKLY;BYDAY=TH"
(AT 0800)
ON RUNCYCLE RULE2 "FREQ=DAILY"
(AT 1500)
:
MY_MASTER#JOB1
END

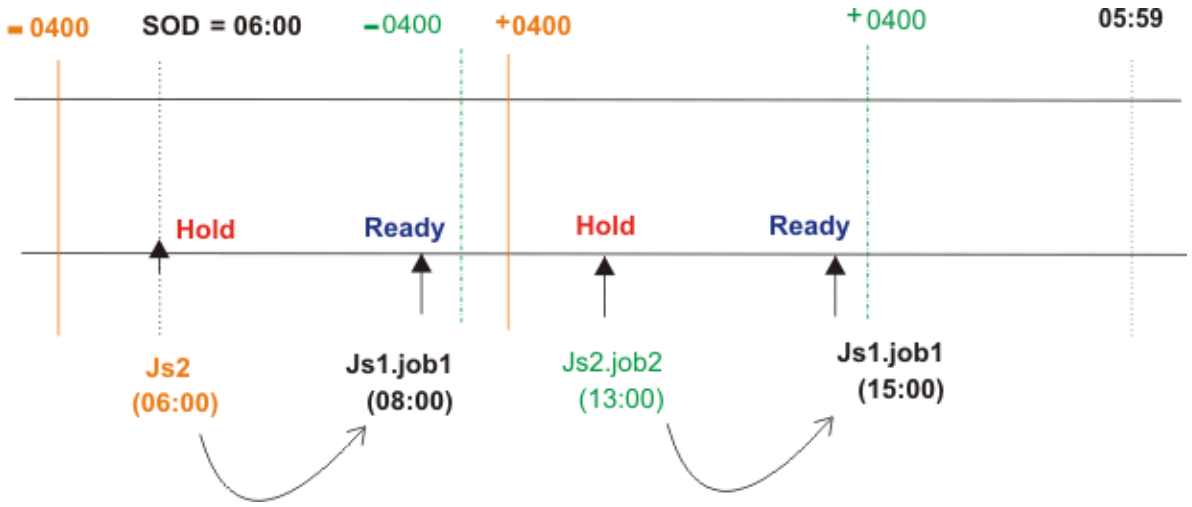
SCHEDULE MY_MASTER#JS2
ON RUNCYCLE RULE3 "FREQ=WEEKLY;BYDAY=TH"
ON RUNCYCLE RULE2 "FREQ=DAILY;"
(AT 1300)
FOLLOWS MY_MASTER#JS1.@
RELATIVE FROM -0400 TO 0400
:
MY_MASTER#JOB2
AT 1300
END
```

At plan creation time, **conman showjobs** produces the following output:

```
%sj @#@
                                            (Est)  (Est)
CPU      Schedule SchedTime  Job      State Pr Start  Elapse RetCode Deps
MY_MASTER#JS1      0800 11/13 ******** READY 10       (00:06)
                              JOB1     HOLD  10       (00:06)
MY_MASTER#JS1      1500 11/13 ******** READY 10       (00:06)
                             JOB1     HOLD  10       (00:06)
MY_MASTER#JS2      0600 11/13 ******** HOLD  10              JS1(0800 11/13/09).@
                              JOB2     HOLD  10(13:00)
MY_MASTER#JS2      1300 11/13 ******** HOLD  10(13:00)       JS1(1500 11/13/09).@
                              JOB2     HOLD  10(13:00)
```

Figure 20: Relative Interval matching criteria - at start of day on Thursday on page 113 shows the status of the job streams in the plan at start of day on Thursday.

Figure 20. Relative Interval matching criteria - at start of day on Thursday



The instance of `Js2` scheduled at 06:00 has a dependency on `Js1.job1` which is scheduled at 08:00, within the relative interval based on the scheduled time (06:00). `Js2.job2` depends on the instance of `Js1.job1` within the relative interval based on the scheduled time (13:00). When the instance of `Js1.job1` starts at 08:00, the status of`Js2` changes to Ready. From this point onwards, the sequence in which the job streams and jobs run follows the typical process.

Within an absolute interval

In this example, the job or job stream instance considered to resolve the dependency is the closest one in a fixed time interval of your choice. The time interval specifies the time of day on which the interval begins and the time of day on which it ends, either on the same day as the instance that includes the dependency, or on a day defined relatively to that date. `Js1` is scheduled to run every day at 08:00 and on Thursdays also at 07:00. Job `Js1.job1` is scheduled to run at 09:00. Job stream `Js2` is scheduled every day at 10:00 and on Thursdays also at start of day (06:00) and has a dependency on `Js1` based on the absolute interval occurring on the same day between 06:00 and 11:00. The job streams are defined as follows:

```
SCHEDULE MY_MASTER#JS1
ON RUNCYCLE RULE1 "FREQ=WEEKLY;BYDAY=TH"
(AT 0700)
ON RUNCYCLE RULE2 "FREQ=DAILY"
(AT 0800)
:
MY_MASTER#JOB1
AT 0900
END

SCHEDULE MY_MASTER#JS2
ON RUNCYCLE RULE3 "FREQ=WEEKLY;BYDAY=TH"
ON RUNCYCLE RULE2 "FREQ=DAILY;"
(AT 1000)
FOLLOWS MY_MASTER#JS1.@ FROM 0600 TO 1100
:
MY_MASTER#JOB2
```

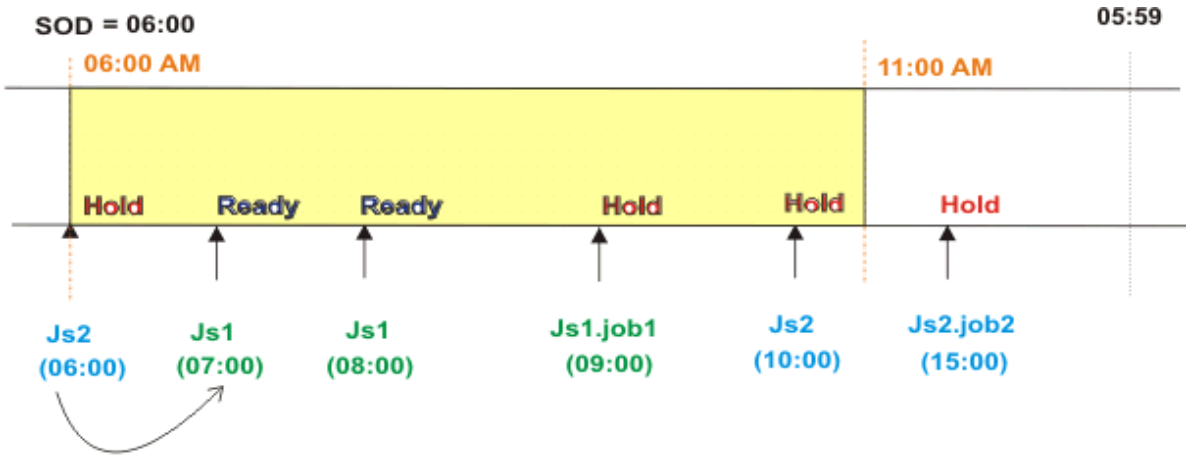
```
AT 1300
END
```

At plan creation time, **conman showjobs** produces the following output:

```
%sj @#@
                                          (Est)  (Est)
CPU      Schedule SchedTime  Job      State Pr Start  Elapse RetCode Deps
MY_MASTER#JS1      0700 11/13******** READY 10       (00:06)
                           JOB1     HOLD  10(09:00)(00:06)
MY_MASTER#JS1      0800 11/13 ******** READY 10       (00:06)
                           JOB1     HOLD  10(09:00)(00:06)
MY_MASTER#JS2      0600 11/13 ******** HOLD  10               JS1(0700 11/13/09).@
                           JOB2     HOLD  10(15:00)
MY_MASTER#JS2      1000 11/13 ******** HOLD  10(10:00)        JS1(0800 11/1309).@
                           JOB2     HOLD  10(15:00)
```

Figure 21: Absolute interval matching criteria - at start of day on Thursday on page 114 shows the status of the job streams in the plan at start of day on Thursday.

Figure 21. Absolute interval matching criteria - at start of day on Thursday



At 09:00, `Js1.job1` starts, and at 10:00 the dependency is released and `Js2` becomes ready. From this point onwards, the sequence is the same as described in the previous matching criteria.

Production plan

After having created or updated the preproduction plan, HCL Workload Automation completes the information stored in the preproduction plan with the information stored in the database about the operations to be performed in the selected time period and the other scheduling objects involved when processing the plan and copies it in a new `Symphony` file. It also adds in this file the cross-plan dependencies, such as job streams carried forward from the production plan already processed, and archives the old `Symphony` file in the `schedlog` directory.

At the end of this process, the new `Symphony` file contains all the information implementing the new production plan, and in addition, all of the plan data is replicated in the database for easy querying from the Dynamic Workload Console and APIs.

A copy of the new `Symphony` file is distributed to all the workstations involved in running jobs or job streams for that production plan.

In the security file the user authorization required to generate the production plan is the *build* access keyword on the `prodsked` and `Symphony` files.

Note: To avoid running out of disk space, keep in mind that each job or job stream instance increases the size of the `Symphony` file by 512 bytes.

For information on how to generate the production plan refer to Creating and extending the production plan on page 125.

Understanding carry forward options

Job streams are carried forward when the production plan is generated. How the job stream is carried forward depends on:

- The **carryforward** keyword in the job stream. See carryforward on page 326.
- The **enCarryForward** global option. See the topic about global option in *HCL Workload Automation Administration Guide*.
- The **stageman -carryforward** command-line keyword. See The stageman command on page 143.
- The **carryStates** global option. See the topic about global option in *HCL Workload Automation Administration Guide*.

If a job is running when the production plan is generated, and the job stream that contains it is not carried forward, the job continues to run and is placed in a dedicated job stream called USERJOBS in the new production plan.

Table 13: Carry forward global options settings on page 115 shows how the carry forward global options work together.

Note: Job streams referenced by conditional dependencies in other scheduling objects are carried forward even when they have successfully completed. This occurs because, despite a job stream being in a successful state, it can still impact the execution of dependent scheduling objects.

Table 13. Carry forward global options settings

Global options	Carry forward operation
enCarryForward=all carryStates=()	Job streams are carried forward only if they did not complete. All jobs are carried forward with the job streams. This is the default setting.
enCarryForward=no	No job streams are carried forward. If this option is set to no, running jobs are moved to the USERJOBS job stream.
enCarryForward=yes carryStates=(*states***)**	Job streams are carried forward only if they have both jobs in the specified states and the **carryforward** keyword set in the job stream definition. Only the jobs in the specified states are carried forward with the job streams.
enCarryForward=yes carryStates=()	Job streams are carried forward only if they did not complete and have the **carryforward** keyword set in the job stream definition. All jobs are carried forward with the job streams.

Table 13. Carry forward global options settings (continued)

Global options	Carry forward operation
enCarryForward=all carryStates=(*states***)**	Job streams are carried forward only if they have jobs in the specified states. Only jobs in the specified states are carried forward with the job streams.

Table 14: Resulting carry forward settings on page 116 shows the result of the carry forward setting based on how the **enCarryForward** global option and the **stageman -carryforward** keywords are set.

Table 14. Resulting carry forward settings

enCarryForward	stageman -carryforward	Resulting carry forward setting
NO	YES	NO
NO	ALL	NO
YES	NO	NO
ALL	NO	NO
ALL	YES	ALL
YES	ALL	ALL
YES	YES	YES

The carry forward option set in the job stream definition is persistent. This means that an unsuccessful job stream that is marked as **carryforward**, continues to be carried forward until one of the following occurs:

- It ends in a SUCC state
- Its UNTIL time is reached
- It is cancelled.

Regardless of how carry forward options are set, job streams that do not contain jobs are not carried forward.

If you set carryStates=(succ) and either **enCarryForward=all** or **enCarryForward=yes**, then the next time you run **JnextPlan** there will be misalignment between the preproduction plan and the new `Symphony` file. This happens because, the preproduction plan does not contain the instances of job streams that ended successfully, but the new `Symphony` file does. The result of this misalignment is that dependencies are not resolved according to carried forward successful job stream instances because they no longer exist in the preproduction plan.

The decision to carry forward a repetitive job, that is a job that contains an **every** time setting in its definition, or a chain of rerun jobs is based on the state of its most recent run. Only the first job and the last job of the chain are carried forward.

Trial plan

A trial plan is a projection of what a production plan would be if it covered a longer period of time. For example, if you generate a production plan that covers two days, but you want to know what the plan would be if it covered three days you can generate a trial plan.

These are the characteristics of a trial plan:

- Its start date matches:
  - The preproduction plan start date.
  - The production plan end date.
- It is based on the static information stored in the current preproduction plan.
- It cannot be run to manage production.
- It can be managed by users with access *build* on **trialsked** file object type set in the security file on the master domain manager.
- It produces a file stored in the `schedlog` directory with these properties:
  - The same format as the `Symphony` file.
  - The file name starts with a leading T.

Trial plan generations may result in the extension of the preproduction plan end time. This depends on the settings of the minLen on page 119 and maxLen on page 119 global options. When this happens, the database is locked and only unlocked when the operation completes.

There is no restriction on the time period selected for a trial plan, but the size of the resulting file containing all trial plan information must be taken into account.

Because the trial plan is based on the static information stored in the preproduction plan, it does not take into account any dynamic updates made to the `Symphony` file while the production plan is being processed, so all the job streams it contains are in one of these two states:

HOLD

If they are dependent on other job streams or if their start time is later than the start of plan time.

READY

If they are free from dependencies and their start time has elapsed.

The operations that can be performed on a trial plan on the master domain manager are:

creation

Used to create a trial plan to have an overview of production when a production plan does not yet exist.

extension

Used to create a trial plan of the extension of the current production plan to have an overview of how production evolves in the future.

For information on how to create or extend a trial plan, refer to Planman command line on page 129.

Forecast plan

The *forecast plan* is a projection of what the production plan would be in a chosen time frame. For example, if you generated a production plan that covers two days and you want to know what the plan would be for the next week you can generate a forecast plan.

These are the characteristics of a forecast plan:

- It covers any time frame, in the future, in the past or even partially overlapping the time period covered by the current production plan.
- It is based on a sample preproduction plan covering the same time period selected for the forecast plan. This sample preproduction plan is deleted after the forecast plan is created.
- It cannot be run to manage production.
- It can be managed by users with access *build* on **trialsked** file object type set in the security file on the master domain manager.
- It produces a file stored in the `schedlog` directory with these properties:
  - The same format as the `Symphony` file.
  - The file name starts with a leading F.
- When workload service assurance is enabled, it can calculate the predicted start time of each job in the job stream. You can enable and disable this feature using the **enForecastStartTime** global option. HCL Workload Automation calculates the average run duration for each job based on all previous runs. For complex plans, enabling this feature could negatively impact the time taken to generate the forecast plan.

While creating a forecast plan the database is locked, and only unlocked when the operation completes.

There is no restriction on the time period selected to generate a forecast plan, but the size of the resulting file containing all forecast plan information must be taken into account.

Because the forecast plan is based on the static information stored in the database, it does not take into account any dynamic updates made to the `Symphony` file while the production plan is being processed or the preproduction plan, so all the job streams it contains are in one of these two states:

HOLD

If they are dependent on other job streams or if their start time is later than the start of plan time.

READY

If they are free from dependencies and their start time has elapsed.

The operation that can be performed on a forecast plan on the master domain manager is:

creation

It is used to create a forecast plan to have an overview of production in a chosen time frame.

For information on how to create a forecast plan, refer to Planman command line on page 129.

Customizing plan management using global options

About this task

You can customize some criteria for HCL Workload Automation to use when managing plans by setting specific options on the master domain manager using the **optman** command-line program. You need to generate the plan again to activate the new settings. The options you can customize are:

Properties impacting the generation of the preproduction plan:

minLen

It is the minimum length, calculated in days, of the preproduction plan which is left, as a buffer, after the end of the newly generated production plan. The value assigned to this option is used when the script **UpdateStats** is run from within **JnextPlan**. The value can be from 7 to 365 days. The default is 8 days.

maxLen

It is the maximum length, calculated in days, of the preproduction plan which is left, as a buffer, after the end of the newly generated production plan. The value can be from 8 to 365 days. The default is 14 days.

If the values of `minLen` and `maxLen` are equal, the preproduction plan is updated during the `MakePlan` phase. In general, the value of `maxLen` should exceed the value of `minLen` by at least 1 day, so that the preproduction plan can be updated during the `UpdateStats` phase.

Properties impacting the generation or extension of the production plan:

startOfDay

It represents the start time of the HCL Workload Automation processing day in 24-hour format: hhmm (0000-2359). The default setting is 0000.

enCarryForward

This is an option that affects how the **stageman** command carries forward job streams. Its setting determines whether or not job streams that did not complete are carried forward from the old to the new production plan. The available settings for *enCarryForward* are **yes, no,** and **all**. The default setting is **all.**

carryStates

This is an option that affects how the **stageman** command manages jobs in carried forward job streams. Its setting determines, based on their state, the jobs to be included in job streams that are carried forward. For example if :

```
carryStates='abend exec hold'
```

then all jobs that are in states `abend`, `exec`, or `hold` are included in carried forward job streams. The default setting is:

```
carryStates=null
```

that means that all jobs are included regardless of their states.

untilDays

If an **until** time (latest start time) has not been specified for a job stream, then the default **until** time is calculated adding the value of this option, expressed in number of days, to the scheduled time for the job stream. If the *enCarryForward* option is set to **all**, and the number of days specified for *untilDays* is reached, then any job stream instances in the plan that ended in error are automatically removed from the plan and not added to the new production plan. The default value is **0**. If the default value is used, then no default time is set for the **until** time (latest start time).

enCFInterNetworkDeps

This is an option that affects how the **stageman** command manages internetwork dependencies. Enter **yes** to have all `EXTERNAL` job streams carried forward. Enter **no** to completely disable the carry forward function for internetwork dependencies. The default setting is **yes**.

enCFResourceQuantity

This is an option that affects how the **stageman** command manages resources. When the production plan is extended, one of the following situations occurs:

- A resource not used by any of the job streams carried forward from the previous production plan is referenced by new job or job stream instances added to the new production plan. In this case the quantity of the resource is obtained from the resource definition stored in the database.
- A resource used by one or more job streams carried forward from the previous production plan is not referenced by job or job stream instances added to the new production plan. In this case the quantity of the resource is obtained from the old `Symphony` file.
- A resource used by one or more job streams carried forward from the previous production plan is referenced by job or job stream instances added to the new production plan. In this case the quantity of the resource that is taken into account is based on the value assigned to the *enCFResourceQuantity* option:

If *enCFResourceQuantity* is set to YES

The quantity of the resource is obtained from the old `Symphony` file.

If *enCFResourceQuantity* is set to NO

The quantity of the resource is obtained from the resource definition stored in the database.

The default setting is **yes**.

enEmptySchedsAreSucc

This option rules the behavior of job streams that do not contain jobs. The available settings are:

yes

The jobs streams that do not contain jobs are marked as SUCC as their dependencies are resolved.

no

The jobs streams that do not contain jobs remain in READY state.

enPreventStart

This is an option to manage, for multiple day production plan, any job streams without an **at** time constraint set. It is used to prevent job stream instances not time dependent from starting all at once as the production plan is created or extended. The available settings are:

yes

A job stream cannot start before 00:00 on the scheduled system date even if free from dependencies

no

A job stream can start immediately as the production plan starts if all its dependencies are resolved.

enLegacyId

Starting from version 9.5, this option is no longer supported. As a result, the job stream identifier *jobstream_id* is generated as described in SHOWJOBS on page 659. Carried forward job streams now keep their original names and identifiers, and they report between braces {} the date when they were carried forward. If you have defined in your environment any automated

procedures based on the name of carried forward job steams, you need to take this change into account.

logmanSmoothPolicy

This is an option that affects how the **logman** command handles statistics and history. It sets the weighting factor that favors the most recent job run when calculating the normal (average) run time for a job. This is expressed as a percentage. The default setting is **-1**.

logmanMinMaxPolicy

This option defines how the minimum and maximum job run times are logged and reported by **logman**. The available settings for the *logmanMinMaxPolicy* option are:

elapsedtime

The maximum and minimum run times and dates that are logged are based only on a job's elapsed time. Elapsed time, expressed in minutes, is greatly affected by system activity. It includes both the amount of time a job used the CPU and the time the job had to wait for other processes to release the CPU. In periods of high system activity, for example, a job might have a long elapsed time, and yet use no more CPU time than in periods of low system activity. The values are updated only if the latest job run has an elapsed time greater than the existing maximum, or less than the existing minimum.

cputime

The maximum and minimum run times and dates that are logged are based only on a job's CPU time. The CPU time is a measure, expressed in seconds, of the actual time a job used the CPU, and it does not include the intervals when the job was waiting. The values are updated only if the latest job run has a CPU time greater than the existing maximum, or less than the existing minimum.

both

The elapsed time and CPU time values are updated independently to indicate their maximum and minimum extremes, but the run dates correspond only to the elapsed

time values. No record is kept, in this case, of the run dates for maximum and minimum CPU times.

The default setting is **both**.

enTimeZone

Enables the time zone option.

Note: Starting from version 9.5, this option is deprecated and must not be modified. By default, its value is set to **yes**.

enLegacyStartOfDayEvaluation

This option affects the way the *startOfDay* variable is managed across the HCL Workload Automation network. This option requires the *enTimeZone* variable set to **yes** to become operational. The available settings for the *enLegacyStartOfDayEvaluation* option are:

no

The value assigned to the *startOfDay* option on the master domain manager is not converted to the local time zone set on each workstation across the network.

yes

The value assigned to the *startOfDay* option on the master domain manager is converted to the local time zone set on each workstation across the network.

Refer to How HCL Workload Automation manages time zones on page 1065 for more information about the *enLegacyStartOfDayEvaluation* variable.

For information on how to set options using the **optman** command-line program, refer to the *HCL Workload Automation Administration Guide*.

Creating and extending the production plan

The entire process of moving from an old to a new production plan, including its activation across the HCL Workload Automation network, is managed by the JnextPlan script. You can run JnextPlan at any time during the processing day. The new production plan that is generated is immediately activated on the target workstations regardless of the time set in the ***startOfDay*** variable. When the **JnextPlan** command is run, the *$MANAGER* variable is managed as follows:

- The variable is resolved if the workstation is a fault-tolerant agent of a version earlier than 8.6.
- The variable is left unresolved for fault-tolerant agent workstations version 8.6.

When you run the **JnextPlan** script, the workstation processes are stopped and restarted on all the workstations across the HCL Workload Automation network. For more information about workstation processes, see Understanding basic processes and commands on page 57.

The JnextPlan script can be run only from the master domain manager. It uses the default connection parameters defined in either the `localopts` or `useropts` files (see Setting up options for using the user interfaces on page 93). If you want to run JnextPlan using different connection parameter settings, you can edit the MakePlan script and modify the invocation to the planman statement as described in Planman command line on page 129.

The **JnextPlan** script is composed of the following sequence of commands and specialized scripts, each managing a specific aspect of the production plan generation:

conman startappserver

This command is invoked to start WebSphere Application Server Liberty if it is not already running.

MakePlan

This script inherits the flags and the values assigned to them from **JnextPlan**. Its syntax is:

MakePlan [**-from** *mm/dd/[yy]yy[hh[:]mm[**tz** | **timezone** tzname]]*] {**-to** *mm?dd?[yy]yy[hh[:]mm[**tz** | **timezone** tzname]]* | **-for** *[h]hh[:]mm* [**-days** *n*] | -**days** *n*}

MakePlan invokes internally the **planman** command line. **MakePlan** performs the following actions:

1. Creates a new plan or extends the current plan and stores the information in an intermediate production plan containing:
    - All the scheduling objects (jobs, job streams, calendars, prompts, resources, workstations, domains, files, users, dependencies) defined in the selected time period.
    - All dependencies between new instances of jobs and job streams and the jobs and job streams existing in the previous production plan.
    - All bind requests whose scheduled time is included in the selected time period.
2. Deletes all bind requests in final state.
3. Prints preproduction reports.

SwitchPlan

This script invokes internally the **stageman** command. For more information refer to The stageman command on page 143. **SwitchPlan** performs the following actions:

1. Stops HCL Workload Automation processes.
2. Generates the new `Symphony` file starting from the intermediate production plan created by **MakePlan**.
3. Archives the old plan file with the current date and time in the `schedlog` directory.
4. Creates a copy of the `Symphony` file to distribute to the workstations.
5. Restarts HCL Workload Automation processes which distribute the copy of the `Symphony` file to the workstation targets for running the jobs in plan.

Note: Make sure no **conman start** command is run while the production plan is been processed.

CreatePostReports

This script prints postproduction reports.

UpdateStats

This script invokes internally the **logman** command. For more information refer to The logman command on page 146. **UpdateStats** performs the following actions:

1. Logs job statistics.
2. Checks the policies and if necessary extends the preproduction plan.
3. Updates the preproduction plan reporting the job stream instance states.

For more information about how to use the JnextPlan script, see JnextPlan on page 127.

Note: Every time you extend the production plan, its run number increases by one. When the run number exceeds 32000, jobs submitted on dynamic agents remain in `intro` status indefinitely. To solve this problem, perform the steps described in Reset run number to 0.

JnextPlan

The JnextPlan script is used to manage the entire process of moving from an old to a new production plan (Symphony), including its activation across the HCL Workload Automation network. Every time you run JnextPlan all workstations are stopped and restarted.

When you run JnextPlan command a joblog file is created in the directory <*TWS_INST_DIR*>\TWS\stdlist\<*DATE*>, where <*TWS_INST_DIR*> is the HCL Workload Automation installation directory and <*DATE*> is the date when the script run.

Authorization

You can run the JnextPlan command from a command prompt shell on the master domain manager if you are one of the following users:

- The *TWS_user* user for which you installed the product on that machine, if not disabled by the settings that are defined in the security file.
- Root on UNIX operating systems or Administrator on Windows operating systems, if not disabled by the settings that are defined in the security file.

Syntax

JnextPlan
[**-V** | **-U**] |
[**-from** *mm/dd/[yy]yy[hh[:]mm*[**tz** | ***timezone*** *tzname]]*]
{**-to** *mm/dd/[yy]yy[hh[:]mm*[**tz** | ***timezone*** *tzname]]* |
-for *[h]hh[:]mm* [**-days** *n*] | **-days** *n*}
[**-noremove**]

Arguments

-V

Displays the command version and exits.

-U

Displays command usage information and exits.

-from

Sets the start time of the production plan. The format of the date is specified in the `localopts` file; where *hhmm* identifies the hours and the minutes and *tz* is the time zone. This flag is used only if a production plan does not exist. If the **-from** argument is not specified, the default value is `"today +startOfDay"`.

If the time zone is not specified, time zone GMT is used by default.

-to

Is the next start time of the HCL Workload Automation processing day, that is the value of the **startOfDay** global option. The format for the date is the same as that used for the **-from** argument. The **-to** argument is mutually exclusive with the **-for** and **-days** arguments. For more information about the **startOfDay** global option, see the topic about global options in *Administration Guide*.

If the time zone is not specified, time zone GMT is used by default.

-for

Is the plan extension expressed in time. The format is *hhhmm*, where *hhh* are the hours and *mm* are the minutes. The **-for** argument is mutually exclusive with **-to**.

-days *n*

Is the number of days you want to create or extend the production plan for. The **-days** parameter is mutually exclusive with the **-to** parameter.

-noremove

Ensures that the completed job stream instances are not removed from the new production plan.

If no **-to**, **-for**, or **-days** arguments are specified, then the default production plan length is one day.

JnextPlan -for 0000

The `JnextPlan -for 0000` command extends by 0 hours and 0 minutes the production plan and adds into the production plan (Symphony) the newly-created workstation, user, and calendar definitions in the database. It also removes all the successfully completed job stream instances.

If you use the `JnextPlan -for 0000 -noremove` command, all the successfully completed job stream instances in the Symphony are not removed.

The `enCarryForward` global option setting specifies if job streams that did not complete are carried forward from the old to the new production plan. Ensure that the `enCarryForward` option is set to `ALL` before running the command to have all incompleted job stream instances in the new production plan or use the `-noremove` option.

Example

Assuming that the value assigned to *startOfDay* is 00:00 a.m. and that the date format set in the `localopts` file is *mm/dd/yyyy*, if the values set are **-from** `07/05/2021` and **-to** `07/07/2021`, then the plan is created to span the time frame from 07/05/2021 at 00:00 a.m. to 07/06/2021 at 11:59 p.m. and not to 07/07/2021 at 11:59 p.m.

Planman command line

The **planman** command line is used to manage *intermediate* production plans, *trial* plans, and *forecast* plans. It is also used to have information about the currently active production plan, to unlock the database entries locked by the plan management processes, to deploy scheduling event rules, and to replicate plan data in the database. The command runs on the master domain manager. Use the following syntax when running **planman**:

planman -U

planman -V

planman [*connection_parameters*] *command*

where:

-U

Displays command usage information and exit.

-V

Displays the command version and exit.

connection_parameters

If you are using **planman** from the master domain manager, the connection parameters were configured at installation and do not need to be supplied, unless you do not want to use the default values.

If you are using **planman** from the command line client on another workstation, the connection parameters might be supplied by one or more of these methods:

- Stored in the `localopts` file
- Stored in the `useropts` file
- Supplied to the command in a parameter file
- Supplied to the command as part of the command string

For an overview of these options see Setting up options for using the user interfaces on page 93. For full details of the configuration parameters see the topic on configuring the command-line client access in the *HCL Workload Automation: Administration Guide*.

command

Represents the command you run to manage plans using the **planman** interface. These are the actions you can perform with plans:

- Creating an intermediate production plan on page 130
- Creating an intermediate plan for a plan extension on page 132
- Retrieving the production plan information on page 133
- Creating a trial plan on page 134
- Creating a trial plan of a production plan extension on page 135
- Creating a forecast plan on page 136
- Unlocking the production plan on page 139
- Removing the preproduction plan on page 140
- Resetting the production plan on page 139
- Replicating plan data in the database on page 140
- Monitoring the replication of plan data in the database on page 142

You can also use **planman** to deploy scheduling event rules on page 174. The command is explained in: Deploying rules on page 138. Refer to the related subsections for additional details on these commands.

Comments

If you receive the `AWSBEH021E` error message, check the WebSphere Application Server Liberty `message.log` file to obtain further details of the error. Note that the planman command is also run by JnextPlan and by the MakePlan job.

Creating an intermediate production plan

The **planman** with the **crt** option is invoked from within the **JnextPlan** command in one of these two situations:

- The first time the **JnextPlan** command is run after having installed the product.
- When generating a production plan after having reset the production plan using the **ResetPlan** command.

The result of running this command is the creation of a new intermediate production plan, named `Symnew`, covering the whole time the new production plan that is being generated will cover. The following syntax is used:

planman [*connection_parameters*] **crt**

[**-from** *mm/dd/*[*yy*]*yy* [*hh[:]mm* [**tz** | **timezone** *tzname*]]]

{**-to** *mm/dd/[yy]yy[hh[:]mm[**tz** | **timezone** tzname]]* |

-for *[h]hh[:]mm* [**-days** *n*] |

-**days** *n*}

where:

connection_parameters

Defines the settings to use when establishing the connection using HTTP or HTTPS through WebSphere Application Server Liberty to the master domain manager. For more information refer to Planman command line on page 129.

-from

Sets the start time of the new production plan.

If the **-from** argument is omitted, then:

- The default date is *today*.
- The default hour is the value set in the ***startOfDay*** global option using **optman** on the master domain manager.

-to

Is the new production plan end time. The format for the date is the same as that used for the **-from** argument. The **-to** argument is mutually exclusive with the **-for** and **-days** arguments.

-for

Is the plan extension expressed in time. The format is *hhhmm*, where *hhh* are the hours and *mm* are the minutes. The **-for** argument is mutually exclusive with **-to** argument.

-days *n*

Is the number of days you want to create the production plan for. The **-days** argument is mutually exclusive with the**-to** argument.

Note:

1. Make sure you run the **planman** command from within the **JnextPlan** command.
2. The format used for the date depends on the value assigned to the ***date format*** variable specified in the `localopts` file.

If no **-to**, **-for**, or **-days** arguments are specified then the default production plan length is one day.

These are some examples of using the **planman** command assuming the date format set in the `localopts` file is *mm/dd/yyyy*:

1. This command creates the production plan from 03/13/2024 at 23:07 (11:07 PM) to 03/14/2024 at 23:06 (11:06 PM) in the local time zone:

   ```
   planman crt –from 03/13/24 2307 –to 03/14/24 2306
   ```

2. This command creates the production plan from 03/13/2024 at 09:00 (9:00 AM) to 03/13/2024 at 15:00 (3:00 PM):

   ```
   planman crt –from 03/13/2024 0900 –for 0600
   ```

3. This command creates a production plan from 03/13/2024 at 18:05 to 03/24/2024 at 23:00 in the time zone of Europe\Paris:

   ```
   planman crt –from 03/13/2024 1805 tz Europe/Rome
               –to 03/24/2024 2300 tz Europe/Rome
   ```

4. This command creates a plan which runs for 6 hours:

```
planman crt -for 0600
```

Creating an intermediate plan for a plan extension

The **planman** command with the **ext** option is invoked from within the **JnextPlan** command when:

- **JnextPlan** is invoked.
- A production plan, represented by the `Symphony` file on the master domain manager, already exists.

The result of running this command is the creation of a new intermediate production plan, named `Symnew`, covering the extra time the new production plan that is being generated will span. The following syntax is used:

planman [*connection_parameters*] **ext**

{**-to** *mm/dd/[yy]yy[hh[:]mm*[***tz*** | ***timezone*** *tzname]]* |

-for *[h]hh[:]mm* [**-days** *n*] |

-**days** *n*}

where:

connection_parameters

Defines the settings to use when establishing the connection using HTTP or HTTPS through WebSphere Application Server Liberty to the master domain manager. For more information refer to Planman command line on page 129.

-to

Sets the end time of the extended production plan. The **-to** argument is mutually exclusive with the **-for** and **-days** arguments.

-for

Sets the length of the production plan extension. The format is *hhhmm*, where *hhh* are the hours and *mm* are the minutes. The **-for** argument is mutually exclusive with the **-to** argument.

-days *n*

Sets the number of days you want to extend the production plan for. The **-days** argument is mutually exclusive with the**-to** argument.

Note:

1. Make sure you run the **planman** command from within the **JnextPlan** command.
2. The format used for the date depends on the value assigned to the ***date format*** variable specified in the `localopts` file.
3. When the production plan is extended the numbers associated to prompts already present in the plan are modified.

If no **-to**, **-for**, or **-days** arguments are specified then the production plan is extended by one day.

Retrieving the production plan information

The following syntax is used to show information about the current production plan:

planman [*connection_parameters*] **showinfo**

where:

connection_parameters

Defines the settings to use when establishing the connection using HTTP or HTTPS through WebSphere Application Server Liberty to the master domain manager. For more information refer to Planman command line on page 129.

Note: You can install the HCL Workload Automation Command Line Client feature on fault-tolerant agents and systems outside the HCL Workload Automation network to issue from those systems the **planman showinfo** command.

The output of this command shows:

- The installation path.
- The start time of the production plan.
- The end time of the production plan.
- The duration of the production plan, after the last plan extension, if extended.
- The date and time of the last plan update, done either using **JnextPlan** or **planman**.
- The end time of the preproduction plan.
- The start time of the first not completed job stream instance.
- The *run number*, that is the total number of times the plan was generated.
- The *confirm run number*, that is the number of times the plan was successfully generated.

The start and end times of the production and preproduction plans are displayed using the format specified in the ***date format*** variable set in the `localopts` file and the time zone of the local machine.

A sample output of this command is the following:

```
# planman showinfoHCL Workload Automation
(UNIX)/PLANMAN 10.2.8
Licensed Materials - Property of IBM* and HCL**
```

```
5698-WSH
(c) Copyright IBM Corp. 1998, 2016 All rights reserved.
(c) Copyright HCL Technologies Ltd. 2016, 2026 All rights reserved.
* Trademark of International Business Machines
** Trademark of HCL Technologies Limited
Installed for user "aix61usr".
Locale LANG set to the following: "en"
Plan creation start time: 07/21/2023 06:00 TZ Europe/Rome
Production plan start time of last extension: 07/21/2023 06:00 TZ Europe/Rome
Production plan end time: 07/22/2023 05:59 TZ Europe/Rome
Production plan time extension: 024:00
Plan last update: 07/21/2023 10:05 TZ Europe/Rome
Preproduction plan end time: 08/05/2023 06:00 TZ Europe/Rome
Start time of first not complete preproduction plan job stream instance:
   07/21/2023 10:30 TZ Europe/Rome
Run number: 1
Confirm run number: 1
```

Creating a trial plan

The following syntax is used to create a trial plan:

planman [*connection_parameters*] **crttrial** *file_name*

[**-from** *mm/dd/[yy]yy* [*hh[:]mm* [**tz** | **timezone** *tzname*]]]

{**-to** *mm/dd/[yy]yy[hh[:]mm[**tz** | **timezone** tzname]]* |

-for *[h]hh[:]mm* [**-days** *n*] |

-**days** *n*}

where:

connection_parameters

Defines the settings to use when establishing the connection using HTTP or HTTPS through WebSphere Application Server Liberty to the master domain manager. For more information refer to Planman command line on page 129.

file_name

Assigns a name to the file to be created under the directory `TWS_home/schedTrial` and that contains the trial plan. The file name of the file containing the trial plan is **T*filename***. This means that if the value assigned to *file_name* is `myfile` then the file name that contains the generated trial plan is `Tmyfile`.

-from

Sets the start time of the trial plan.

If the **-from** argument is omitted, then:

- The default date is *today*.
- The default hour is the value set in the ***startOfDay*** global option using **optman** on the master domain manager.

-to

Sets the end time of the trial plan. The **-to** argument is mutually exclusive with the **-for** and **-days** arguments.

-for

Sets the length of the trial plan. The format is *hhhmm*, where *hhh* are the hours and *mm* are the minutes. The **-for** argument is mutually exclusive with the **-to** argument.

-days *n*

Sets the number of days you want the trial plan to last for. The **-days** argument is mutually exclusive with the **-to** argument.

Note: The format used for the date depends on the value assigned to the ***date format*** variable specified in the `localopts` file.

If no **-to**, **-for**, or **-days** arguments are specified then the default trial plan length is one day.

See also

From the Dynamic Workload Console you can perform the same task as described in:

the Dynamic Workload Console User's Guide, section Generating Trial and Forecast Plans.

Creating a trial plan of a production plan extension

The following syntax is used to create a trial plan with the extension of the current production plan:

planman [*connection_parameters*] **exttrial** *file_name*

{**-to** *mm/dd/[yy]yy[hh[:]mm[**tz** | **timezone** tzname]]* |

-for *[h]hh[:]mm* [**-days** *n*] |

-**days** *n*}

where:

connection_parameters

Defines the settings to use when establishing the connection using HTTP or HTTPS through WebSphere Application Server Liberty to the master domain manager. For more information refer to Planman command line on page 129.

file_name

Assigns a name to the file to be created under the directory `TWS_home/schedTrial` and that contains the trial plan. The file name of the file containing the trial plan is **T*filename***. This means that if the value assigned to *file_name* is `myfile` then the file name that contains the generated trial plan is `Tmyfile`.

-to

Sets the end time of the trial plan containing the production plan extension. The **-to** argument is mutually exclusive with the **-for** and **-days** arguments.

-for

Sets the length of the trial plan containing the production plan extension. The format is *hhhmm*, where *hhh* are the hours and *mm* are the minutes. The **-for** argument is mutually exclusive with the **-to** argument.

-days *n*

Sets the number of days you want the trial plan containing the production plan extension to last for. The **-days** argument is mutually exclusive with the **-to** argument.

Note: The format used for the date depends on the value assigned to the ***date format*** variable specified in the `localopts` file.

If no **-to**, **-for**, or **-days** arguments are specified then the default production plan extension contained in the trial plan is one day.

See also

From the Dynamic Workload Console you can perform the same task as described in:

the Dynamic Workload Console User's Guide, section Generating Trial and Forecast Plans.

Creating a forecast plan

The following syntax is used to create a forecast plan:

planman [*connection_parameters*] **crtfc** *file_name*

[**-from** *mm/dd/[yy]yy* [*hh[:]mm* [**tz** | **timezone** *tzname*]]]

{**-to** *mm/dd/[yy]yy[hh[:]mm[**tz** | **timezone** tzname]]* |

-for *[h]hh[:]mm* [**-days** *n*] |

-**days** *n*}

where:

connection_parameters

Defines the settings to use when establishing the connection using HTTP or HTTPS through WebSphere Application Server Liberty to the master domain manager. For more information refer to Planman command line on page 129.

file_name

Assigns a name to the file to be created under the directory `TWS_home/schedForecast` and that contains the forecast plan. The name of the file containing the forecast plan is **F*filename***. This means that if the value assigned to *file_name* is `myfile` then the file name that contains the generated forecast plan is `Fmyfile`.

The maximum length of *file_name* can be 148 characters.

-from

Sets the start time of the forecast plan. It includes the specified minute.

If the **-from** argument is omitted, then:

- The default date is *today*.
- The default hour is the value set in the ***startOfDay*** global option using **optman** on the master domain manager.

-to

Sets the end time of the forecast plan. It excludes the specified minute. The **-to** argument is mutually exclusive with the **-for** and **-days** arguments.

-for

Sets the length of the forecast plan. The format is *hhhmm*, where *hhh* are the hours and *mm* are the minutes. The **-for** argument is mutually exclusive with the **-to** argument.

-days *n*

Sets the number of days you want the forecast plan to last for. If the interval contains DST (daylight savings time), it is automatically included in the calculation. The **-days** argument is mutually exclusive with the **-to** argument.

Note: The format used for the date depends on the value assigned to the ***date format*** variable specified in the `localopts` file.

If no **-to**, **-for**, or **-days** arguments are specified then the default forecast plan length is one day.

See also

From the Dynamic Workload Console you can perform the same task as described in:

the Dynamic Workload Console User's Guide, section Generating Trial and Forecast Plans.

Deploying rules

The `planman deploy` command is used in event management on page 174. You can use it to manually deploy all rules that are not in draft state (the `isDraft` property is set to `NO` in their definition). The command operates as follows:

1. Selects all event rule definitions not in draft state from the HCL Workload Automation database.
2. Builds event rule configuration files.
3. Deploys the configuration files to the monitoring engines running on the HCL Workload Automation agents.

The new configuration files update the event rules running on each monitoring engine in terms of:

- New rules
- Changed rules
- Rules deleted or set back to `draft` state

You can use this command in addition to, or in replacement of, the `deploymentFrequency` (`df`) `optman` configuration option, which periodically checks event rule definitions for changes to deploy (see *Administration Guide* for details on this option).

The changes applied to the event rule definitions in the database become effective only after deployment has taken place.

The command syntax is:

planman [*connection_parameters*] **deploy** [**-scratch**]

where:

connection_parameters

Defines the settings to use when establishing the connection using HTTP or HTTPS through WebSphere Application Server Liberty to the master domain manager. For more information refer to Planman command line on page 129.

-scratch

Without this option, the command affects only the rules that have been added, changed, deleted, or set back to `draft` state .

With this option, the command deploys all the non-draft rules existing in the database, including the ones that are already in deployment and have not changed.

Note that the deployment time increases proportionally with the number of active rules to deploy. If you need to deploy a large number of new or changed rules, run `planman deploy` with this option to reduce the deployment time.

Use of this option results in a complete reset of the event processor and should be used with caution. The command may cause the loss of any rule instances in progress at the time you issue it. The typical case is a sequential rule that has been triggered and is waiting for additional events to take place: if you use the option at this time, the event rule environment is reset and the tracked events are lost.

To run this command, you need **build** access on the **prodsked** file.

Unlocking the production plan

When HCL Workload Automation starts to create the production plan, it locks the definitions of scheduling objects in the database and then unlocks them either when the creation of the production plan is finished or if an error condition occurs. The lock is applied to prevent object definitions from being modified when the production plan in generated or extended. If the processing ends abnormally the database entries might remain locked. Only users with **build** access on the **prodsked** file object type specified in the security file on the master domain manager are allowed to unlock the database. The command used to perform this action is:

planman [*connection_parameters*] **unlock**

where:

connection_parameters

Defines the settings to use when establishing the connection using HTTP or HTTPS through WebSphere Application Server Liberty to the master domain manager. For more information refer to Planman command line on page 129.

Note: You can install the HCL Workload Automation Command Line Client feature on fault-tolerant agents and systems outside the HCL Workload Automation network to issue from those systems the **planman unlock** command.

Resetting the production plan

The following script is used to either reset or scratch the production plan:

ResetPlan [*connection_parameters*] [**-scratch**]

where:

connection_parameters

Defines the settings to use when establishing the connection using HTTP or HTTPS through WebSphere Application Server Liberty to the master domain manager. For more information refer to Planman command line on page 129.

The difference between resetting and scratching the production plan is the following:

- If you *reset* the production plan, the preproduction plan is kept, it is updated with job statistics, and it is used later to generate a new production plan. This means that when you create a new production plan, it will contain all job stream instances which were not in COMPLETE state when you run the **ResetPlan**. The steps performed by the product when resetting the production plan are the following:
  1. The current `Symphony` file is archived.
  2. The job statistics are updated.
- If you *scratch* the production plan, the preproduction plan is scratched too. The preproduction plan will be created again based on the modeling information stored in the database when you later generate a new production plan. This

means that the new production plan will contain all job stream instances scheduled to run in the time frame covered by the plan regardless of whether or not they were already in COMPLETE state when the plan was scratched. The steps performed by the product when scratching the production plan are the following:

1. The current `Symphony` file is archived and the plan data replicated in the database is deleted.
2. The job statistics are updated.
3. The preproduction plan is scratched.

Note: If you use the **-scratch** option, make sure you run **dbrunstats** before the **JnextPlan** script. See the *Administration Guide* for details on `dbrunstats`.

When you run ResetPlan command a joblog file is created in the directory <*TWS_INST_DIR*>\TWS\stdlist\<*DATE*>, where <*TWS_INST_DIR*> is the HCL Workload Automation installation directory and <*DATE*> is the date when the script run.

Removing the preproduction plan

The following script is used to remove the preproduction plan, while maintaining the `Symphony` file:

Planman reset -scratch

When you run this command, the preproduction plan is scratched. The preproduction plan will be created again based on the modeling information stored in the database when you later generate a new production plan. This means that the new production plan will contain all job stream instances scheduled to run in the time frame covered by the plan regardless of whether or not they were already in COMPLETE state when the plan was scratched. The steps performed by the product when scratching the production plan are the following:

1. The `Symphony` file is maintained.
2. The job statistics are updated.
3. The preproduction plan is scratched.

Note: If you use the **-scratch** option, make sure you run **dbrunstats** before the **JnextPlan** script. See the *Administration Guide* for details on `dbrunstats`.

Replicating plan data in the database

Storing information about objects in the plan into a database makes accessing the plan data easier and faster. When large numbers of users are accessing the HCL Workload Automation backend environment concurrently, performance and reliability can be compromised. By replicating the plan in a relational database, users can access data quickly and reliably.

For versions earlier than 9.1, the following operations required access to the Symphony plan:

- Running baseline reports
- Displaying the plan in a graphical view

- Displaying the job stream graphical view
- Triggering actions
- Refreshing job and job stream monitoring views

All of these modelling or monitoring activities required access to the plan and, if you multiply these scenarios by the number of users who concurrently request access to the plan to perform one or more of these activities, the result is slow response times and overall performance.

To address this issue, HCL Workload Automation replicates the plan in a relational database where SQL statements are used to retrieve data rapidly and reliably. New message boxes, `mirrorbox.msg` and `mirrorbox<n>.msg`, are used to synchronize the database with the Symphony file. If the `mirrorbox.msg` file becomes full, for example, if the database remains unavailable for a long period of time, then the plan is automatically reloaded into the database using the information in the Symphony file.

In particular, the performance and response times of the UpdateStats script have greatly improved with this new way of managing plan data.

In addition, each time the plan is extended, the plan in the database is scratched and re-created, making the latest information available to all users. To manually replicate plan data from the Symphony file to the database, run the planman resync command.

The synchronization of the Symphony file with the database is enabled automatically when you add the Sfinal file to the database with the composer add Sfinal command. A new job, CHECKSYNC, has been added to the FINALPOSTREPORTS job stream contained in the Sfinal file that is responsible for monitoring the state of the process of replicating the Symphony file in the database. it sends the progress and status of this process to the job log. If the CHECKSYNC job should fail, then refer to the job log of the CHECKSYNC job, as well as the WebSphere Application Server Liberty log to determine the problem. After resolving the problem, run the planman resync command to reload the plan data from the Symphony file into the database.

To simplify integrations, a set of database views for a set of tables containing plan data in the HCL Workload Automation database are provided.

To see the database views containing information about HCL Workload Automation objects in the plan, refer to the views beginning with "PLAN_" in *HCL Workload Automation: Database Views*.

When running operations from the Dynamic Workload Console that retrieve current plan data, if you suspect the data is not up-to-date, you can run planman resync to update the plan data in the database with the latest information in the Symphony file. If the message box, `mirrorbox<n>.msg`, responsible for synchronizing the database with the Symphony file becomes full, for example, the database is unavailable for a long period of time, then a planman resync is automatically issued so that the plan is fully reloaded in the database.

The following syntax is used to replicate plan data in the database with the data in the Symphony file:

planman [*connection_parameters*] **resync**

where:

connection_parameters

Defines the settings to use when establishing the connection using HTTP or HTTPS through WebSphere Application Server Liberty to the master domain manager. For more information, see Planman command line on page 129.

Note:

- If you are upgrading your IBM Workload Scheduler environment to version 9.2, then there are a few manual steps you need to implement before your plan data is replicated in the database. See the section about customizing and submitting the optional final job stream in *Planning and Installation* and Automating production plan processing on page 150 for more information.
- Replicating plan data in the database requires that DB2 JDBC Driver Type 4 is installed. DB2 JDBC Driver Type 2 is not supported.
- Ensure that in the database configuration the CUR_COMMIT property is set to ENABLED. See DB2 documentation for more information about this setting.

For more information about how to optimize the process of plan replication in the database see the topic about tuning plan replication in the *Administration Guide*.

Monitoring the replication of plan data in the database

The following syntax is used to monitor the progress and outcome of replicating plan data in the database with the data in the Symphony file:

planman [*connection_parameters*] **checksync**

where:

connection_parameters

Defines the settings to use when establishing the connection using HTTP or HTTPS through WebSphere Application Server Liberty to the master domain manager. For more information refer to Planman command line on page 129.

Messages are written to standard output with the progress and status of the command. The `planman checksync` command is also defined in the job, CHECKSYNC, contained in the FINALPOSTREPORTS job stream contained in the Sfinal file. If the CHECKSYNC job should fail, then refer to the job log of the CHECKSYNC job, as well as the WebSphere Application Server Liberty log to determine the problem. After resolving the problem, run the `planman resync` command to reload the plan data from the Symphony file into the database.

For information about how to optimize the process of plan replication in the database see the topic about tuning plan replication in the *Administration Guide*.

The stageman command

The **stageman** command carries forward uncompleted job streams, archives the old production plan, and installs the new production plan. A copy of `Symphony`, is sent to domain managers and agents as part of the initialization process for the new production plan. When running **JnextPlan**, **stageman** is invoked from within the **SwitchPlan** script.

Syntax

stageman -V | -U | -parse

stageman
[**-carryforward**{**yes**|**no**|**all**}]
[**-log** *log_file*| **-nolog**]
[*symnew*]

Arguments

-V

Displays the command version and exits.

-U

Displays command usage information and exits.

-parse

Reads the global options from the database to check that job streams to be carried forward do not contain parsing errors.

-carryforward

Defines how uncompleted job streams are managed when moving to a new production plan. The available settings are:

no

Does not carry forward any job streams.

yes

Carries forward only the uncompleted job streams whose definition contains the keyword **carryforward**.

all

Carries forward all uncompleted job streams, regardless whether or not contain the keyword **carryforward** in the job stream definition.

If you omit this keyword, by default it is set to the value specified globally using **optman** for the *enCarryForward* option. Refer to Understanding carry forward options on page 115 to have information on the resulting carry forward setting when both the *enCarryForward* global option and the **-carryforward** keywords are set.

-log

Archives the old production plan in the directory *TWS_home*`/schedlog` with file name *log_file*. The archived production plans can then be listed and selected using the commands LISTSYM on page 623 and SETSYM on page 644. If neither **-log** nor **-nolog** keywords are specified, HCL Workload Automation archives the old production plan using the following naming convention:

```
Myyyymmddhhtt
```

where *yyyymmddhhtt* corresponds to the year, month, day, hour, minutes the old production plan is archived. If you generate the production plan using **JnextPlan** you can customize this naming convention in the **SwitchPlan** script.

Note: Be sure to monitor the disk space in the `schedlog` directory and remove older log files on a regular basis.

-nolog

Does not archive the old production plan.

symnew

The name assigned to the intermediate production plan file created by **planman**. If not specified, **stageman** uses the file name `Symnew`.

Comments

To allow carry forward procedures to work properly in a network, the master domain manager's production plan file, `Symphony`, must be updated with the latest job stream status from its agents and subordinate domain managers. Run the following command:

```
conman "link @"
```

before running **stageman**. This command links any unlinked workstations so that messages about job processing status queued in the `Mailbox.msg` file are sent back to the master domain manager to update the `Symphony` file.

Note: In UNIX® only, **stageman** also determines which executable files associated to jobs submitted using the **at** and **batch** utility commands can be deleted when HCL Workload Automation is started for the new production period. These jobs are not carried forward.

Example

Examples

Carry forward all uncompleted job streams (regardless of the status of the Carry Forward option), log the old `Symphony` file, and create the new `Symphony` file:

```
DATE='datecalc today pic YYYYMMDDHHTT'
stageman -carryforward all -log schedlog/M$DATE
```

Carry forward uncompleted job streams as defined by the ***carryforward*** global option, do not log the old `Symphony` file, and create an intermediate production plan named `mysym`:

```
stageman -nolog mysym
```

Managing concurrent accesses to the Symphony file

This section contains two sample scenarios describing how HCL Workload Automation manages possible concurrent accesses to the `Symphony` file when running **stageman**.

Scenario 1: Access to Symphony file locked by other HCL Workload Automation processes

If HCL Workload Automation processes are still active and accessing the `Symphony` file when **stageman** is run, the following message is displayed:

```
Unable to get exclusive access to Symphony.
Shutdown batchman and mailman.
```

To continue, stop HCL Workload Automation and rerun **stageman**. If **stageman** aborts for any reason, you must rerun both **planman** and **stageman**.

Scenario 2: Access to Symphony file locked by stageman

If you try to access the plan using the command-line interface while the `Symphony` is being switched, you get the following message:

```
Current Symphony file is old. Switching to new Symphony.
Schedule mm/dd/yyyy (nnnn) on cpu, Symphony switched.
```

Managing follows dependencies using carry forward prompt

To retain continuity when carrying forward job streams, **stageman** generates prompts for each job stream that is carried forwarded and has a `follows` dependency on another job stream which is not carried forward. These prompts are issued after the new processing period begins, when HCL Workload Automation checks to see if the job or job stream is ready to launch, and are replied to as standard prompts. The following is an example of a *carry forward prompt*:

```
INACT 1(SYS2#SKED2[(0600 01/11/06),(0AAAAAAAAAAAA2Y)]) follows
      SYS1#SKED1, satisfied?
```

This prompt indicates that a job stream, which is carried forward from the previous production plan, `(SYS2#SKED2[(0600 01/11/06),(0AAAAAAAAAAAA2Y)])`, has a follows dependency from a job stream named `SYS1#SKED1` which was not carried forward. For information on the syntax used to indicate the carried forward job stream refer to JSSELECT Selecting job streams in commands on page 571.

The state of the prompt, **INACT** in this case, defines the state of the corresponding `follows` dependency. The possible states are:

INACT

The prompt has not been issued and the dependency is not satisfied.

ASKED

The prompt has been issued, and is awaiting a reply. The dependency is not satisfied.

NO

Either a "no" reply was received, or it was determined before carry forward occurred that the followed job stream `SKED3` had not completed successfully. The dependency is not satisfied.

YES

Either a "yes" reply was received, or it was determined before carry forward occurred that the followed job stream `SKED3` had completed successfully. The dependency is satisfied.

The logman command

The **logman** command logs job statistics from a production plan log file.

Syntax

logman -V|-U

logman
[*connectionParameters*]

{**-prod** | *symphony-file*]
[**-smooth** *weighting*]
[**-minmax** {*elapsed* | *cpu*}]}

Arguments

-U

Displays command usage information and exits.

-V

Displays the command version and exits.

connectionParameters

Represents the set of parameters that control the interaction between the product interface, **logman** running on the master domain manager in this case, and the WebSphere Application Server Liberty infrastructure using **http** or **https**. Use this syntax to specify the settings for the connection parameters:

[**-host** *hostname*] [**-port** *port_number*] [**-protocol** *protocol_name*] [**-proxy** *proxy_name*] [**-proxyport** *proxy_port_number*] [**-password** *user_password*] [**-timeout** *timeout*] [**-username** *username*]

where:

hostname

The hostname of the master domain manager.

port_number

The port number used when establishing the connection with the master domain manager.

protocol_name

The protocol used during the communication. It can be **http** with basic authentication, or **https** with certificate authentication.

proxy_name

The proxy hostname used in the connection.

proxy_port_number

The proxy port number used in the connection.

user_password

The password of the user that is used to run **logman**.

Note:

On Windows workstations, when you specify a password that contains double quotation marks (") or other special characters, make sure that the character is escaped. For example, if your password is `tws11"tws`, write it as `"tws11\"tws"`.

timeout

The maximum time, expressed in seconds, the connecting command-line program can wait for the master domain manager response before considering the communication request as failed.

username

The name of the user running **logman**.

If any of these parameters is omitted when invoking **logman**, HCL Workload Automation searches for a value first in the `useropts` file and then in the `localopts` file. If a setting for the parameter is not found an error is displayed. Refer to Setting up options for using the user interfaces on page 93 for information on `useropts` and `localopts` files.

-prod

Updates the preproduction plan with the information on the job streams in COMPLETE state in production. By doing so the preproduction plan is kept up-to-date with the latest processing information. This avoids the possibility of the new production plan running again, job streams already completed in the previous production period.

-minmax {*elapsed* | *cpu*}

Defines how the minimum and maximum job run times are logged and reported. The available settings are:

elapsed

Base the minimum and maximum run times on elapsed time.

cpu

Base the minimum and maximum run times on CPU time.

This setting is used when the **logman** command is run from the command line and not by the **JnextPlan** script. When the **logman** command is run by **JnextPlan**, the setting used is the one specified in the *logmanMinMaxPolicy* global option.

-smooth *weighting*

Uses a weighting factor that favors the most recent job run when calculating the normal (average) run time for a job. This is expressed as a percentage. For example, **-smooth 40** applies a weighting factor of 40% to the most recent job run, and 60% to the existing average. The default is **-smooth -1**. This setting is used when the **logman** command is run from the command line, as you may need in case of job recovery if you want to replace the job statistics in the database. When the **logman** command is run by **JnextPlan**, the setting used is the one specified in the *logmanSmoothPolicy* global option. For more information about the *logmanSmoothPolicy* option, see the topic about global options in *Administration Guide*.

symphony-file

The name of an archived symphony file from which job statistics are extracted.

Comments

Jobs that have already been logged, cannot be logged again. Attempting to do so generates a `0 jobs logged` error message.

Example

Examples

Log job statistics from the log file M202403170935:

```
logman schedlog/M202403170935
```

Starting production plan processing

About this task

To start a production cycle, follow these steps:

1. Log in as *TWS_user* on the master domain manager.
2. At a command prompt, run the script command `. ./TWS_home/tws_env.sh` in UNIX® or `TWS_home\tws_env.cmd` in Windows® to set up the environment, then run the `JnextPlan` job by entering, for example on a UNIX® workstation, the following command:

   ```
   JnextPlan.sh -from 05/03/23 0400 tz Europe/Rome -to 06/06/23
   ```

   This creates a new production plan that starts on the third of May 2023 at 4:00 a.m. (Europe/Rome time) and stops on the sixth of June 2023 at 3:59 a.m. The processing day will start at the time specified on the master domain manager in the variable ***startOfDay***.
3. When the `JnextPlan` job completes, check the status of HCL Workload Automation:

   ```
   conman status
   ```

   If HCL Workload Automation started correctly, the status is

   ```
   Batchman=LIVES
   ```

   If mailman is still running a process on the remote workstation, you might see that the remote workstation does not initialize immediately. This happens because the workstation needs to complete any ongoing activities involving the mailman process before re-initializing. After the interval defined in the ***mm retry link*** parameter set in the `TWS_home/localopts` configuration file elapses, the domain manager tries again to initialize the workstation. As soon as the ongoing activities complete, the activities for the next day are initialized. For information about the `localopts` configuration file, see the topic about setting local options in *HCL Workload Automation Administration Guide* .
4. Increase the limit to allow jobs to run. The default job limit after installation is zero. This means no jobs will run.

   ```
   conman "limit;10"
   ```

Automating production plan processing

If you want to extend your production plan at a fixed time interval, for example every week, you have the option to automate the extension. This section explains how you can do this.

With IBM Workload Scheduler version 9.1 and later, the Sfinal file has been modified to include two sample job streams named `FINAL` and `FINALPOSTREPORTS` that help you automate plan management. A copy of these job streams are found in the `Sfinal` file in the *`TWS_home`* directory. In addition, a copy of the job scripts is also located in this same location.

You can use either this `Sfinal` file or create and customize a new one.

Important: In any case, to be able to run these job streams successfully, the *TWS_user* must have `Write` access to the `tmp` default temporary directory.

The `FINAL` job stream runs the sequence of script files described in `JnextPlan` to generate the new production plan. See Creating and extending the production plan on page 125 for reference.

The `FINALPOSTREPORTS` job stream, responsible for printing postproduction reports, follows the `FINAL` job stream and starts only when the last job listed in the `FINAL` job stream (SWITCHPLAN) has completed successfully. The FINALPOSTREPORTS job stream also includes a job named, CHECKSYNC, that monitors the progress and outcome of the planman resync command. The planman resync command loads the plan data from the Symphony file to the database.

With IBM Workload Scheduler version 9.1 and later, plan data is now fully replicated in the database. Each time the plan is extended, the plan in the database is recreated, making the latest information available to all users. If you are performing a fresh install of HCL Workload Automation, then this synchronization of the Symphony file with the database is enabled automatically when you add the Sfinal file to the database with the composer add Sfinal command. Since the Sfinal file is not overwritten during an upgrade, you must add the updated FINAL and FINALPOSTREPORTS job streams to the database by running the composesr add Sfinal command. This ensures that you have the CHECKSYNC job that is responsible for replicating plan data in the database. The updated Sfinal final can be found in `TWA_home/config/ directory`. You must then run JnextPlan to include the FINAL and FINALPOSTREPORTS job streams in the current production plan. See the section about customizing and submitting the optional final job stream in *Planning and Installation*

By default, the `FINAL` job stream is set to run once a day followed by the `FINALPOSTREPORTS` job stream. You can modify the time the job streams run by modifying two settings in the job stream definition. These are the details about the two steps you need to follow to do this, for example, to make the job streams run every three days:

- Schedule the job stream to run every three days by modifying the run cycle inside the job stream definition.
- In the statement that invokes **MakePlan** inside the `FINAL` job stream, set the production plan to last for three days by specifying **-for 72**.

Then you need to add the job streams to the database by performing the following steps:

1. Log in as *TWS_user*.
2. Run the **tws_env** script to set the HCL Workload Automation environment as follows:

- UNIX®: on C shells launch `<TWS_home/tws_env.csh`
- UNIX®: on Korn shells launch `<TWS_home/tws_env.sh`
- From a Windows® command line: launch `TWS_home\tws_env.cmd`

where *TWS_home* represents the product installation directory.

3. Add the `FINAL` and `FINALPOSTREPORTS` job stream definitions to the database by running the following command:

```
composer add Sfinal
```

If you did not use the `Sfinal` file provided with the product but you created a new one, use its name in place of `Sfinal`.

4. Start the production cycle by running the **JnextPlan** script. In this way the `FINAL` and `FINALPOSTREPORTS` job streams will be included in the current production plan.

Note: Even if you decided to automate the production plan extension you can still run **JnextPlan** at any time.

Detecting loops between jobs and job streams in current and trial plans

Ensure robust detection of loops involving nodes (that is, jobs and job streams) in both current and trial plans.

To detect loops in jobs and job streams, perform the following steps:

1. Stop WebSphere Application Server Liberty Base as described in the topic about starting and stopping the application server in *Administration Guide*.
2. Browse to the `TWSConfig.properties` file, located in the following paths, depending on your operating system:

   On Windows operating systems

   <TWA_home>\usr\servers\engineServer\resources\properties

   On UNIX operating systems

   <*TWA_DATA_DIR*>/usr/servers/engineServer/resources/properties

3. Set the `com.ibm.tws.planner.monitor.verboseLoopLogging` property to **true**.
4. Restart WebSphere Application Server Liberty Base as described in the topic about starting and stopping the application server in *Administration Guide*.
5. Monitor the `trace.log` file when generating the current plan, adding dependencies, generating a forecast plan, or performing other actions which might generate a loop. The file is located in the following paths, depending on your operating system:

   On Windows operating systems

   <*TWA_home*>\TWS\stdlist\appserver\engineServer\logs

   On UNIX operating systems

   <*TWA_DATA_DIR*>/stdlist/appserver/engineServer/logs

6. Messages similar to the following are displayed:

```
In plan "your_plan", the node "your_node" is inside a loop
```

```
In plan "your_plan", there are n plans inside a loop
```

To reduce verbosity, loop notifications are only logged when a change occurs—such as a job or job stream entering or exiting a detected loop.

Chapter 5. Using workload service assurance

Workload service assurance is an optional feature that provides the means to flag jobs as mission critical for your business and to ensure that they are processed in a timely manner. Using this function benefits your scheduling operations personnel by enhancing their ability to meet defined service levels.

When the workload service assurance feature is enabled, you can flag jobs as mission critical and ensure they have an associated completion deadline specified in their definition or at submission. Two additional threads of execution, Time Planner and Plan Monitor, that run within WebSphere Application Server Liberty, are thereafter engaged to make sure that the critical jobs are completed on time.

Defining a critical job and its deadline triggers the calculation of the start times of all the other jobs that are predecessors of the critical job. The set of predecessors of a critical job make up its *critical network*. This might include jobs from other job streams. Starting from the critical job's deadline and duration, Time Planner calculates its *critical start time*, which is the latest starting time for the job to keep up with its deadline. Moving backwards from the critical start time it calculates the latest time at which each predecessor within the critical network can start so that the critical job at the end of the chain can complete on time.

While the plan runs, Plan Monitor constantly checks the critical network to ensure that the deadline of the critical job can be met. When changes that have an impact on timings are made to the critical network, for example addition or removal of jobs or follows dependencies, Plan Monitor requests Time Planner to recalculate the critical start times. Also, when a critical network job completes, timings of jobs that follow it are recalculated to take account of the actual duration of the job.

Within a critical network, the set of predecessors that more directly risk delaying the critical start time is called *critical path*. The critical path is dynamically updated as predecessors complete or their risk of completing late changes.

The scheduler (batchman) acts automatically to remedy delays by prioritizing jobs that are actually or potentially putting the target deadline at risk, although some conditions that cause delays might require operator intervention. A series of specialized critical job views, available on the Dynamic Workload Console, allows operators to browse critical jobs, display their predecessors and the critical paths associated with them, identify jobs that are causing problems, and drill down to identify and remedy problems.

For detailed information, see:

- Enabling and configuring workload service assurance on page 153
- Planning critical jobs on page 157
- Processing and monitoring critical jobs on page 159
- Workload service assurance scenario on page 161

Enabling and configuring workload service assurance

A number of global and local options control the management of critical jobs. The HCL Workload Automation security file also must authorize users with proper access to all jobs, job streams, and workstations associated with critical jobs.

Global options

The workload service assurance feature is enabled and disabled by the global option `enWorkloadServiceAssurance`. It is enabled by default. Other global and local options are used to control different aspects of the processing of critical jobs and their predecessors.

Table 15. Workload service assurance global options (continued)

Option	Description
	considered to have a long duration and is added to the hot list that can be viewed on the Dynamic Workload Console.
`approachingLateOffset | al`	The critical start time of a job in the critical network is the latest time that the job can start without causing the critical job to end after the deadline. In most cases, a job will start well before the critical start time so that if the job runs longer than its estimated duration, the situation does not immediately become critical. Therefore, if a job has not started and the critical start time is only a few minutes away, the timely completion of the critical job is considered to be potentially at risk. The `approachingLateOffset` option allows you to determine the length of time before the critical start time of a job in the critical network at which you are to alerted to this potential risk. If a job has still not started the specified number of seconds before the critical start time, the job is added to a hot list that can be viewed on the Dynamic Workload Console. The default is 120 seconds. **Note:** The value of this parameter is checked regularly. JnextPlan does not need to be run for changes to take effect.
`deadlineOffset | do`	In general, a deadline should be specified for a job flagged as `critical`. If it is not, the scheduler uses the deadline defined for the job stream. The `deadlineOffset` option provides an offset used to calculate the critical start time in case the deadline is missing for both a critical job and its job stream. The plan end plus this offset is assumed as the critical job's deadline. The offset is expressed in minutes. The default is 2 minutes. **Important:** When the plan is extended, the start time of critical jobs whose deadline is calculated with this mechanism is automatically changed as a consequence

Table 15. Workload service assurance global options (continued)

Option	Description
	of the fact that it must now match the new plan finishing time.

For more information about global options, see *HCL Workload Automation Administration Guide*.

Local options

Workload service assurance uses local options to control the priority allocation of system resources to jobs in the critical network that must be promoted to maintain the critical deadline. Table 16: Workload service assurance local options on page 156 shows the local options used by the workload service assurance feature. To set local options, edit the *twshome*`\localopts` file on each workstation where critical jobs will be running. Run JnextPlan or restart the agent for changes to the local options to take effect.

Table 16. Workload service assurance local options

Option	Description
`jm promoted nice`	Sets the `nice` value to be assigned to critical jobs or critical job predecessors that need to be promoted on UNIX and Linux operating systems, so that they are assigned more resources and processed ahead of other jobs. Specific values vary for the different platforms, but in general, the setting must be a negative integer. The default is -1 and lower numbers represent higher priorities. If you specify a positive integer, the default value is used. The **jm nice** local option has a similar role in prioritizing jobs that have been submitted by the root user. A critical job that has been submitted by the root user could be eligible for both prioritization mechanisms. In such a case, values would be added together. For example, if **jm promoted nice** is set to -4 and **jm nice** to -2, the critical job submitted by user root would have a priority of -6.

Table 16. Workload service assurance local options (continued)

Option	Description
`jm promoted priority`	Sets the priority value for critical jobs or critical job predecessors that need to be promoted so that Windows operating systems assign them more resources and process them ahead of other jobs. The possible values are: • High • AboveNormal • Normal • BelowNormal • Low or Idle The default is AboveNormal. Note that if you set a lower priority value than the one non-critical jobs might be assigned, no warning is given and no mechanism such as the one available for jm promoted nice sets it back to the default.

Security file requirements

It is mandatory that the users who own the HCL Workload Automation instances running critical jobs are authorized to work with all jobs, job streams, and workstations associated with these jobs. These users must therefore have DISPLAY, MODIFY, and LIST rights in the security file for all the JOB, SCHEDULE and CPU associated objects.

Planning critical jobs

Workload service assurance provides the means to identify critical jobs, define deadlines, and calculate timings for all jobs that must precede the critical job.

If it is critical that a job must be completed before a specific time, you can flag it as critical when you add it to a job stream using the Workload Designer functions on the Dynamic Workload Console. You can define the deadline either at job or job stream level.

Jobs can also be flagged as critical by including the **critical** keyword in the job statement when you create or modify a job stream using the **composer** command line.

When the **JnextPlan** command is run to include the new job in the production plan, all jobs that are direct or indirect predecessors of the critical job are identified. These jobs, together with the critical job itself, form a `critical network`.

Because timing of jobs in the critical network must be tightly controlled, Time Planner calculates the following timing benchmarks for each critical network job:

Critical start

It applies to distributed systems only and represents the latest time at which the job can start without causing the critical job to miss its deadline.

Critical start times are calculated starting with the deadline set for the critical job and working backwards using the estimated duration of each job to determine its critical start time. For example, if the critical job deadline is 19:00 and the estimated duration of the critical job is 30 minutes, the critical job will not finish by the deadline unless it has started by 18:30. If the immediate predecessor of the critical job has an estimated duration of 20 minutes, it must start at latest by 18.10.



Note: Only the deadline of the critical job is considered when calculating critical start times for jobs in the critical network. If other jobs have deadlines defined, their critical start times might be later than their deadlines.

Earliest start

Represents the earliest time at which a job in the critical network could start, taking into consideration all dependencies and resource requirements.

Estimated start and end times

Estimated start times are calculated starting with the earliest time at which the first job or jobs in the critical network could start and working forward using the estimated duration of each job to estimate the start time of the job that follows it.

Planned start and end times

For the initial calculations, these values are set to the estimated start and end times. They are subsequently recalculated to take into consideration any changes or delays in the plan.

Estimated duration

The estimated duration of a job is based on the statistics collected from previous runs of the job. Take this into account when considering the accuracy of calculated timings for the critical job networks that include jobs running for the first time. In the case of a shadow job, the estimated duration is always set to the default value of one minute. This applies to shadow jobs running for the first time, as well as any subsequent runs of the shadow job.

Confidence Factor

For each critical job, you are provided with a percentage that indicates the confidence with which the critical job will meet its deadline. When a job finishes running, the confidence factor is overwritten and set to 0% when the estimate deadline was exceeded and is set to 100% when the deadline was not exceeded.

The timings for each job in the critical network are added to the Symphony file that includes all the plan information and that is distributed to all workstations on which jobs are to be run.

As the plan is run, Plan Monitor monitors all critical networks: subsequent changes to the critical network that affect the timing of jobs will trigger the recalculation of the critical and estimated start times. Changes might include manual changes, for example releasing dependencies or rerunning jobs, and changes made automatically by the system in response to a potential or actual risk to the timely completion of the critical job.

Specific views for critical jobs and their predecessors, available from the Dynamic Workload Console, allow you to keep track of the processing of the critical network. The views can immediately identify problems in your planning of the critical job. For example, if the estimated start time of a job in the critical network is later than the critical start time, this is immediately signalled as a potential risk to the critical job.

Processing and monitoring critical jobs

Workload service assurance provides automatic tracking and prioritizing of critical network jobs and online functions that you use to monitor and intervene in the processing of critical network jobs.

Automatic tracking and prioritizing

To ensure that critical deadlines can be met, workload service assurance provides the following automated services to critical jobs and the predecessor jobs that form their critical networks:

Promotion

When the critical start time of a job is approaching and the job has not started, the promotion mechanism is used. A promoted job is assigned additional operating system resources and its submission is prioritized.

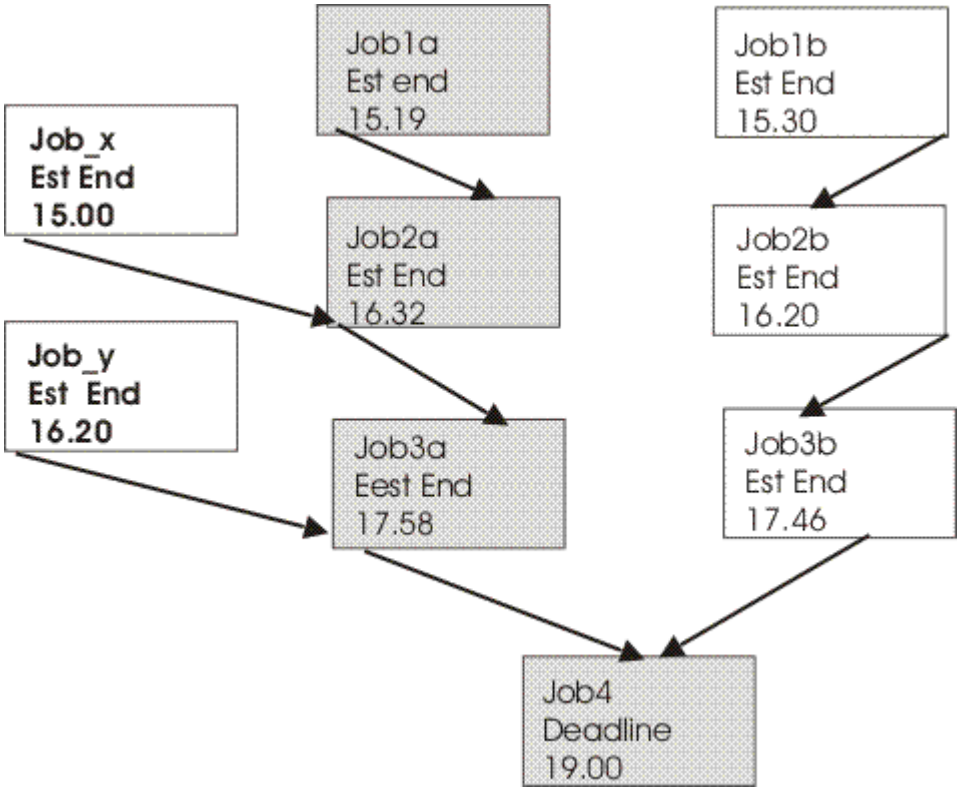
The timing of promotions is controlled by the global option `promotionoffset`. Promoted jobs are selected for submission after jobs that have priorities of "high" and "go", but before all other jobs. Prioritizing of operating system resources is controlled by the local options `jm promoted nice` (UNIX and Linux) and `jm promoted priority` (Windows).

Calculation of the critical path

The critical path is the chain of dependencies, leading to the critical job, that is most at risk of causing the deadline to be missed at any given time. The critical path is calculated using the estimated end times of the critical job predecessors. Working back from the critical job, the path is constructed by selecting the predecessor with the latest estimated end time. If the actual end time differs substantially from the estimated end time, the critical path is automatically recalculated.

Figure 22: Critical path on page 160 shows the critical path through a critical network at a specific moment during the processing of the plan.

Figure 22. Critical path



At this time, the critical path includes Job3a, Job2a, and Job1a. Job3a and Job3b are the immediate predecessors of the critical job, Job4, and Job3a has the later estimated end date. Job3a has two immediate predecessors, Job2a and Job_y. Job2a has the later estimated end time, and so on.

Addition of jobs to the hot list

Jobs that are part of the critical network are added to a hot list that is associated to the critical job itself. The hot list includes any critical network jobs that have a real or potential impact on the timely completion of the critical job. Jobs are added to the hot list for the one or more of the reasons listed next. Note that only the jobs beginning the current critical network, for which there is no predecessor, can be included in the hot list.

- The job has stopped with an error. The length of time before the critical start time is determined by the `approachingLateOffset` global option.
- The job has been running longer than estimated by a factor defined in the `longDurationThreshold` global option.
- The job has still not started, though all its follows dependencies have either been resolved or released, and at least one of the following conditions is true:
  - The critical start time has nearly been reached.
  - The job is scheduled to run on a workstation where the limit is set to zero.
  - The job belongs to a job stream for which the limit is set to zero.
  - The job or its job stream has been suppressed.
  - The job or its job stream currently has a priority that is lower than the fence or is set to zero.

Setting a high or potential risk status for the critical job

A risk status can be set for the critical job, as follows:

High risk

Calculated timings show that the critical job will finish after its deadline.

Potential risk

Critical predecessor jobs have been added to the hot list.

Online tracking of critical jobs

The Dynamic Workload Console provides specialized views for tracking the progress of critical jobs and their predecessors. You can access the views from the Dashboard or create a task to monitor critical tasks using Orchestration Monitor.

The initial view lists all critical jobs for the engine, showing the status: normal, potential risk, or high risk. From this view, you can navigate to see:

- The hot list of jobs that put the critical deadline at risk.
- The critical path.
- Details of all critical predecessors.
- Details of completed critical predecessors.
- Job logs of jobs that have already run.
- The confidence factor expressed as a percentage. The probability with which a critical job will meet its deadline.

Using the views, you can monitor the progress of the critical network, find out about current and potential problems, release dependencies, and rerun jobs.

Workload service assurance scenario

This scenario illustrates the use of workload service assurance in ensuring that important production deadlines can be met.

Fine Cola uses HCL Workload Automation to manage the timing and interdependencies of its production and supply process.

Fine Cola has service level agreements with many customers that support "just-in-time" restocking. This means that late starts on any of the delivery routes will almost certainly result in Fine Cola's products not being on the shelves.

The job that produces the loading orders for trucks must be completed at latest by 5.30 a.m. This job is dependent on the successful completion of other jobs. For example, although orders are processed ahead of time, last-minute changes often arrive when trucks return after completing a delivery route. Fine Cola also provides invoices with delivery notes, so changes to orders must also be reflected in the pricing and might trigger special-offer adjustments to prices.

Planning the critical job

Using the Workload Designer on the Dynamic Workload Console, the Fine Cola scheduler flags the loading order job as critical and sets the deadline for 5 a.m.

When **JnextPlan** is run, the critical start dates for this job and all the jobs that are identified as predecessors of the critical job are calculated.

Tracking the critical job

1. The HCL Workload Automation operator checks the dashboards and sees that there are critical jobs scheduled on one of the engines.
2. He sees that there is a critical job in potential risk. He clicks the potential risk link to get the list of critical jobs in this state.

   The loading orders job shows a status of "potential risk".
3. He selects the job and clicks **Hot List** to see the job or jobs that are putting the critical job at risk.

   The orders adjustment job is listed as being in error.
4. He selects the job and clicks **Job log**.

   The log shows that the job failed because of incorrect credentials for the orders database.
5. After discovering that the database password was changed that day, he changes the job definition in the symphony file and reruns the job.
6. When he returns to the dashboard, he sees that there are no longer any jobs in potential risk. Also, the critical jobs list that was opened when clicking on the potential risk link no longer shows the critical job after the job is rerun.
7. The job is now running and has been automatically promoted to give it higher priority for submission and system resources.
8. No further problems need fixing and the critical job finally completes at 4.45 a.m.

Promoting jobs scheduled on dynamic pools

This section explains how to promote a critical job scheduled on a dynamic pool. A promoted job can run on a larger number of dynamic agents in the dynamic pool than a non-promoted job. This ensures that an important job runs before other jobs that are less important.

To ensure that a critical job obtains the necessary resources and is processed in a timely manner, specify the following variables:

tws.job.promoted

This variable indicates if the job is promoted. Supported values are **YES** and **NO**. The value of this variable applies to all jobs submitted in the specified environment.

tws.job.resourcesForPromoted

This variable is defined in the dynamic pool definition and indicates the quantity of the required logical resources assigned on a dynamic pool to a promoted job. Values can be **1** if the job is promoted or **10** if the job is not promoted. The quantity is indicated with this notation: **${tws.job.resourcesForPromoted}**.

When a job is scheduled on the dynamic pool, the value of the **tws.job.promoted** variable in the job determines the behavior of the dynamic pool:

- If the value of the **tws.job.promoted** variable is **NO**, the value of the **tws.job.resourcesForPromoted** variable on the dynamic pool is 10, which means that few resources match the requirement.
- If the value of the **tws.job.promoted** variable is **YES**, the value of the **tws.job.resourcesForPromoted** variable on the dynamic pool is 1, which means that more resources match the requirement because the dynamic pool includes workstations with resource quantity equal to or greater than 1 and not only workstations with value equal or greater than 10.

For example, you can write a script that checks the value assigned to the **tws.job.promoted** variable in the job and performs different actions based on whether or not the job is promoted.

Chapter 6. Customizing your workload using variable tables

This chapter introduces the concept of variable tables to group global parameters, from now on called variables, to customize your workload.

Variable definitions are contained in variable tables. A variable table is an object that groups together multiple variables. Using variable tables you can assign different values to the same variable for use in job and job stream definitions in JCL, log on, prompts dependencies, file dependencies, and recovery prompts. This is particularly useful when a job definition is used as a template for a job that belongs to more than one job stream. For example, you can assign different values to the same variable and reuse the same job definition in different job streams saving therefore time and avoiding errors.

When you define a variable, you assign it to a variable table because the same variable can be defined in different variable tables with different values. Or, a better approach is to create one or more variable tables, specifying a list of variable names and values for each table. While doing this, you can add the same variable name with different values in different tables. When you request a list of variables you get *variabletable.variablename* pairs to easily identify to which variable table the variable belongs.

For example, the VAR1 variable defined in the REP1_TABLE1 variable table is shown as:

```
REP1_TABLE1.VAR1
```

You can assign variable tables to run cycles, job streams, and workstations.

Using variable tables you change the behavior of the workload according to when, why, and where you want to run your schedule giving you more flexibility in customizing your workload and meet your service level agreements. In detail:

When

To change the behavior of jobs and job streams based on when they are scheduled to run, that is, on which days they run. Using variable tables with run cycles.

Why

To change the behavior of jobs and job streams based on why they are scheduled to run, for example to create a job that runs different commands. Using variable tables with job streams.

Where

To change the behavior of jobs and job streams based on where they run, for example on different workstations. Using variable tables with workstations.

The default variable table

This topic describes the default variable table and how it works.

The default variable table is the table that contains all the variables that you defined without specifying any variable table name. The default name of the default variable table is **MAIN_TABLE**. You can modify this name at any time or set another variable table as the default variable table. You cannot delete the default variable table. When you set another variable table as the default, the original default variable table is no longer marked as default. You can work with the default variable table

in the same way as any other variable table. You can easily identify the default variable table on the user interface because it is marked with a **Y** in the **default** field.

Example

This example shows a list of variable tables

```
Variable Table Name                      Default  Updated On  Locked By
---------------------------------------  -------  ----------  ----------------
MAIN_TABLE                                Y       05/07/2026  -
VT_1                                         05/07/2026  -
VT_2                                         05/07/2026  -
VARTABLE3                                    05/07/2026  -
V4                                       05/07/2026  -
VT5                                          05/07/2026  -

AWSBIA291I Total objects: 6
```

Data integrity for variable tables

This topic explains how data integrity is guaranteed using variable tables.

In the same way as for other objects, HCL Workload Automation maintains variable table data integrity whenever you run commands that create, modify, rename, or delete the definition of a variable table.

When referencing a variable table from any run cycle, job stream, or workstation HCL Workload Automation checks that the variable table exists and preserves the link between it and the run cycle, job stream, and workstation. This means that you cannot delete a variable table definition while a reference from a run cycle, a job stream, or a workstation still exists.

Referential integrity is guaranteed at variable table level and not at variable level, that is, when a run cycle, a job stream, and a workstation references a variable table, HCL Workload Automation checks that the variable table exists, not that the referenced variable exists.

Locking mechanism for variable tables

This topic describes how the locking mechanism works for variable tables.

Locking is set at variable table level to ensure that definitions in the database are not overwritten by different users concurrently accessing the same variable table. This means that both when you lock a variable table and when you lock a variable you gain exclusive permissions for all the variables in that variable table. That is, you can perform any commands on the locked variable table and on all the variables in it. Any other user only has read-only access to this variable table and to the variables in it.

This prevents any other user from changing the same variable table that you are changing. If another user tries to lock a variable table or a variable that you have already locked, an error message is returned.

Variable table security

This topic describes how to define security settings for variable tables.

User access to variable tables must be authorized in the HCL Workload Automation security file. As for other objects, the connector verifies the existence of proper authorization before executing an action that requires access to a variable table. The following new keyword is available in the security file for this purpose:

```
vartable name=@ access=add,delete,display,modify,list,use,unlock
```

You need `use` access to be able to reference a variable table from other objects (job streams, run cycles and workstations). Security filters are based on the `name` attribute only, but your HCL Workload Automation administrator has the option to use the **$default** keyword to specify security permissions on the default table, regardless of its name.

Permission to work on a variable is no longer based on the individual variable but on the table enclosing it. Access to a variable is granted only if the corresponding action on the enclosing variable table is permitted. The following table shows the corresponding permissions for variables and variable tables:

Table 17. The relationship between variable tables and their enclosed variables in the HCL Workload Automation security file

Defined access to vartable	Allowed action on enclosed variables
modify	add
	delete
	modify
display	display
unlock	unlock

For information about the security file, see the topic about configuring user authorization (Security file) in *Administration Guide* .

Variable resolution

This topic describes how variables are resolved both when you generate a plan and when you submit a job or a job stream to be run.

The format used to specify a variable can also determine when a variable is resolved with a value. See Variable and parameter definition on page 289 for information about the formats you can use.

When you generate a plan, HCL Workload Automation analyzes the variable tables in the order shown below for variable resolution:

1. In the run cycle. The run cycle in the job stream is checked first, then the run cycle in a run cycle group, and finally the variable table defined at the run cycle group level.
2. In the job stream.
3. In the workstation. See Workstation considered for variable resolution on page 167.
4. In the default variable table, but only when the variables are specified in the ^*variablename*^ on page 291 format in the job definition. Variables specified in the ${*variablename*} format are not resolved.

At plan resolution time each level is analyzed in the order described above. If you specify a variable that is not contained in any variable table, including the default, a warning message containing the name of the unresolved variable is written in the `messages.log` log file and the variable name is left in the plan. The `messages.log` log file is located in

On Windows systems

`<TWA_home>\stdlist\appserver\engineServer\logs`

On UNIX systems

`TWA_DATA_DIR/stdlist/appserver/engineServer/logs`

When you submit a job stream, HCL Workload Automation resolves variables by analyzing the variable tables in the order shown below:

1. Specified during the submit operation.
2. In the job stream.
3. In the workstation. See Workstation considered for variable resolution on page 167.
4. In the default variable table, but only when the variables are specified in the ^*variablename*^ on page 291 format in the job definition. Variables specified in the ${*variablename*} format are not resolved.

When you submit a job, HCL Workload Automation resolves variables by analyzing the variable tables in the order shown below:

1. Specified during the submit operation, but only when the variables are specified in the ^*variablename*^ on page 291 format in the job definition. Variables specified in the ${*variablename*} format are not resolved.
2. In the workstation. See Workstation considered for variable resolution on page 167.
3. In the default variable table, but only when the variables are specified in the ^*variablename*^ on page 291 format in the job definition. Variables specified in the ${*variablename*} format are not resolved.

Workstation considered for variable resolution

When the variable is resolved by the variable table specified in the workstation, the workstation taken into consideration is:

For variable in file dependency

The workstation where the file resides.

For variable in job

The workstation where the job is defined.

For variable in prompt dependency

Global prompt

No workstation is taken into consideration. Variables in global prompts are resolved always using the default variable table. This is because global prompt are used by all jobs and job streams so just one value must be used for variable resolution.

Ad hoc prompt

The workstation where the job or the job stream that depends on the prompt dependency is defined.

Chapter 7. Condition-based workload automation

Condition-based workload automation provides a simple and immediate way to have your workflows start at just the right time. You can define in your job stream a start condition that, when met, releases the job stream to run as scheduled.

For example, if you have a job stream containing jobs which analyze one or more files, you can have the job stream start only after the file or files have been modified or created. Also, you can condition the job stream to the output condition of a specific job: the job stream starts only when the specified output condition is met. For example, if the job stream contains jobs which process the data in a database, you might want to have the job stream start after a new row has been written into the database.

You can start your workflow based on one of the following conditions:

- One or more files being created
- One or more files being modified
- A job completing with its output condition satisfied

For conditions based on files being created or modified, when you save the job stream, a monitoring job is automatically created to monitor the specified condition. This job becomes the first job in the job stream and remains in EXEC status until the condition it is monitoring is met or the job's deadline expires.

Note: If you do not specify a deadline, by default the value is defined using the **startConditionDeadlineOffset** optman option.

This job type is identified by the **+AUTOGEN+** label in the command line and by an icon with the **A** character in the Dynamic Workload Console.

For conditions based on the result of a specified job, when you save the job stream, the job becomes the first job in the job stream and restarts until the condition is satisfied, or the job's deadline expires.

Note: If you do not specify a deadline, by default the value is defined using the **startConditionDeadlineOffset** optman option.

This applies also if the job completes in Success status. This is the monitoring job. When you specify this condition type, HCL Workload Automation automatically defines a success output condition on the monitoring job. As a result, the monitoring job completes successfully when any of its output conditions is satisfied, including the condition on the monitoring job itself. You can apply this logic to the job stream or to specific jobs in the job stream. For more information about output conditions, see the section about Applying conditional branching logic in *User's Guide and Reference*.

When the condition is met, the job completes successfully and releases the remaining part of the job stream. When the job's deadline expires without the condition being met, the job stream is suppressed.

In the Workload Designer, you define condition-based workload automation in the **Start Condition** tab when creating a job stream. You can select the type of condition you want to monitor and specify the related settings. For more information,

see A business scenario on page 172 and the online help. From the **composer** command line, you can define the start condition in the job stream definition. For more information, see .

By default, HCL Workload Automation keeps monitoring the condition also after it is first met. This is accomplished by automatically creating a new Job Stream Submission job and adding it to the job stream as a successor of the monitoring job. This job type is identified by the **+AUTOGEN+** label in the command line and by an icon with the **A** character in the Dynamic Workload Console. To have HCL Workload Automation stop when the condition is first met, select the **Start once** check box in the Workload Designer, or omit the **rerun** keyword in the **composer** command line.

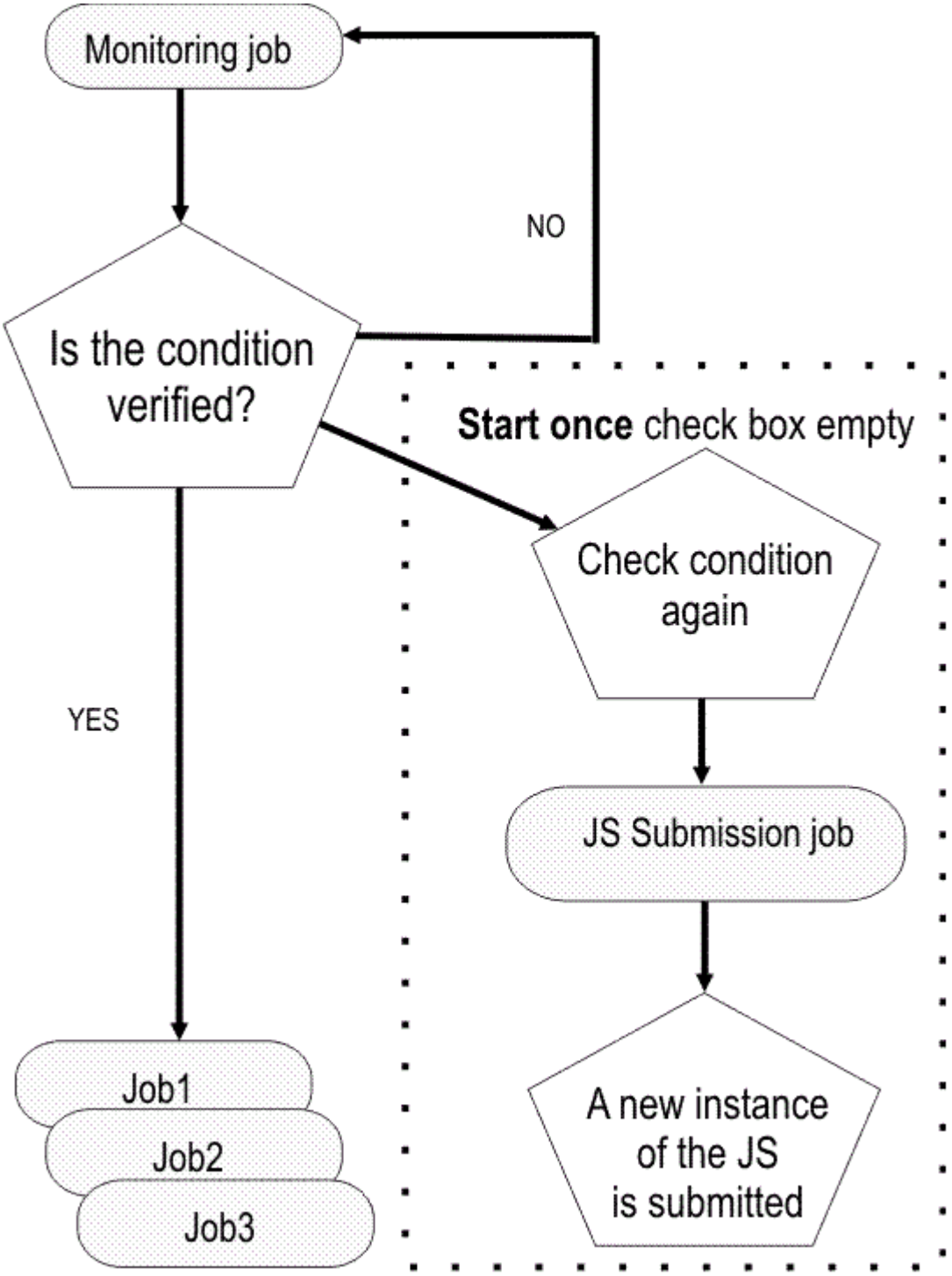
The Job Stream Submission job is defined on a specific pool workstation named MASTERAGENTS. This pool workstation contains the dynamic agent installed on the master domain manager and on the backup domain manager, if present. The dynamic agent installed on the master domain manager and backup domain manager (if present) are automatically added at installation time to this pool workstation. If you delete the MASTERAGENTS pool workstation and then recreate it, you must stop and restart the dynamic agent to add it back to the MASTERAGENTS pool workstation. See the topic about automatically registering agents to pools in the *Planning and Installation Guide*.

Note: The default name for the pool workstation, MASTERAGENTS, can be modified using the optman global option resubmitJobName. See the detailed description of the global options in the *Administration Guide* for details about this option.

The Job Stream Submission job creates a new instance of the job stream in which the start condition is defined. By default, the new job stream instance starts also if the previous instance is still running and the two instances run concurrently. To change this behavior, in the Workload Designer, switch to the **Scheduling options** tab and select **Queue the new instance** in the **Actions** section. From the **composer** command line, use the **onoverlap** keyword. For more information, see . The newly-generated instance is identical to the previous one it and is set to repeat the condition check, therefore a series of new instances is created until the job stream's deadline expires.

Figure 23: Condition-based workload automation on page 171 illustrates the details of the default process. If you select the **Start once** check box, the process stops after the condition is first met, therefore the steps within the dotted line in the graphic are not performed.

Figure 23. Condition-based workload automation



The monitoring job, the Job Stream Submission job, and the additional instances of the job stream, are not visible in the database, but are visible in the plan, so that you can check on the progress of the plan. In the Dynamic Workload Console,

you can check the progress of the plan by the **Monitor workload** view, and in the **conman** command line, you can use the **showjobs** command.

A business scenario

This scenario outlines one of the many ways in which condition-based workload automation can help you improve your workload automation

About this task

HCL Workload Automation offers a number of powerful solutions to help make your automation smarter. For example, using the iterative workflow automation feature, you can implement iterative processing of a sequence of jobs within a job stream.

In an online retailer company, this feature is very useful, because it iterates the processing of each item in each customer order, for each order in the database, by scheduling just a single job stream.

But it is also possible to take this scenario further and make the automation even more responsive to your needs. For example, you might want to have the orders processed as soon as they are placed in the database, to ensure a quicker response to customers' requests.

You can now create the job stream that processes the orders and add a start condition to it, so that the job stream verifies that an order is placed in the database before starting. As soon as the condition is met, the job stream starts.

In the following example, the ORDERS job stream starts running as soon as an order is created in the database.

To set up this type of job processing, complete the following steps:

1. Create the job stream definition.
2. Define the start condition.

To create the ORDERS job stream, complete the following steps:

1. In the Workload Designer, create the `ORDERS` job stream definition.
2. In the **Start Condition** tab, select **Job condition met**.
3. In the **Job definition** field, select the job whose output condition you want to monitor: the `QUERY_DB_ORDERS`, the job that queries the orders in the database. This is the monitoring job.
4. In the **Workstation** field, specify the workstation where the monitoring job is scheduled to run.
5. In the **Output condition value** field, specify the output condition that, when met, causes the job stream to start running.
   In the current scenario, the condition to be met is that the number of rows in the database is higher than zero. To monitor this condition, specify the following string: `${this.NumberOfRows} > 0`. As a result, the `ORDERS` job stream will start when the monitoring job completes with the specified output condition.
6. Leave the **Start Once** check box empty to keep monitoring the condition also after it is first met. This is the default behavior.
7. Complete the remaining tabs and fields as required and add the relevant jobs.
8. Save the job stream.

Results

The monitoring job starts checking the orders in the database repeatedly. As soon as an order is found, the monitoring job releases the remaining part of the job stream and then completes successfully.

A new instance of the job stream is automatically resubmitted, containing the same start condition, so that the cycle iterates and verifies for the next order to be inserted in the database. By default, the job stream instances run in parallel, processing each order as it is stored in the database, and iterating for each item in the order. If you do not want the instances to run in parallel, select **Queue the new instance** in the **Scheduling options** tab. By selecting this option, the new instance starts only after the previous instance has completed.

By combining two features, the automation of iterative workflows and condition-based workload automation, you can increase control over your workflow start conditions and process orders in real time.

Example

You can define the same scenario by using the composer command line as follows:

```
SCHEDULE NewYork#ORDERS
ON RUNCYCLE RC_DST "FREQ=DAILY;INTERVAL=1;BYWORKDAY"
STARTCOND JOB S_AGT#QUERY_DB_ORDERS OUTCOND "${this.NumberOfRows} > 0" INTERVAL 300
( RERUN )
:
Dallas#DBUpdate

Denver#OrderProcess

Lexington#WarehouseInfo
END
```

For more information about defining start conditions using the composer command line see Job stream definition on page 314.

Chapter 8. Running event-driven workload automation

Event-driven workload automation adds the capability to perform on-demand workload automation in addition to plan-based job scheduling. It provides the capability to define rules that can trigger on-demand workload automation.

The object of event-driven workload automation in HCL Workload Automation is to carry out a predefined set of actions in response to events that occur on nodes running HCL Workload Automation (but also on non-HCL Workload Automation ones, when you use the `sendevent` command line). This implies the capability to submit workload and run commands on the fly, notify users via email.

The main tasks of event-driven workload automation are:

- Trigger the execution of batch jobs and job streams based on the reception or combination of real time events.
- Reply to prompts
- Notify users when anomalous conditions occur in the HCL Workload Automation scheduling environment or batch scheduling activity.
- Invoke an external product when a particular event condition occurs.

Event-driven workload automation is based upon the concept of event rule. In HCL Workload Automation an event rule is a scheduling object that includes the following items:

- Events
- Event-correlating conditions
- Actions

When you define an event rule, you specify one or more events, a correlation rule, and the one or more actions that are triggered by those events. Moreover, you can specify validity dates, a daily time interval of activity, and a common time zone for all the time restrictions that are set.

The events that HCL Workload Automation can detect for action triggering can be:

Internal events

They are events involving HCL Workload Automation internal application status and changes in the status of HCL Workload Automation objects. Events of this category can be job or job stream status changes, critical jobs or job streams being late or canceled, and workstation status changes.

External events

They are events not directly involving HCL Workload Automation that may nonetheless impact workload submission. Events of this category can be messages written in log files, events sent by third party applications, or a file being created, updated, or deleted.

Within a rule, two or more events can be correlated through correlation attributes such as a common workstation or job. The correlation attributes provide a way to direct the rule to create a separate rule (or copy of itself) for each group of events that share common characteristics. Typically, each active rule has one copy that is running in the event processing server. However, sometimes the same rule is needed for different groups of events, which are often related to different groups of

resources. Using one or more correlation attributes is a method for directing a rule to create a separate rule copy for each group of events with common characteristics.

The actions that HCL Workload Automation can run when it detects any of these events can be:

Operational actions

They are actions that cause the change in the status of scheduling objects. Actions of this category are submitting a job, job stream, or command, or replying to a prompt.

Notification actions

They are actions that have no impact on the status of scheduling objects. Actions belonging to this category are sending an email, logging the event in an internal auditing database, or running a non-HCL Workload Automation command.

This classification of events and actions is conceptual. It has no impact on how they are handled by the event-driven mechanism.

Simple event rule scenarios

This section lists some simple scenarios involving the use of event rules. The corresponding XML coding is shown in Event rule examples on page 185.

Scenario 1: Send email notification

1. The administrator defines the following event rule:
   - When any of the `job123` jobs terminates in error and yields the following error message:

     ```
     AWSBHT001E The job "MYWORKSTATION#JOBS.JOB1234" in file "ls" has
     failed with the error: AWSBDW009E The following operating system
     error occurred retrieving the password structure for either the
     logon user...
     ```

     send an email to operator `john.smith@mycorp.com`. The subject of the email includes the names of the job instance and of the associated workstation.

     The event rule is valid from December 1st to December 31st in the 12:00-16:00 EST time window.
2. The administrator saves the rule as non-draft in the database and it is readily deployed by HCL Workload Automation.
3. The scheduler starts monitoring the jobs and every time one of them ends in error, John Smith is sent an email so that he can check the job and take corrective action.

Scenario 2: Monitor that workstation links back

1. The administrator defines the following event rule:
   - If workstation `CPU1` becomes unlinked and does not link back within 10 minutes, send a notification email to `chuck.derry@mycorp.com`.

2. The administrator saves the rule as non-draft in the database and it is readily deployed by HCL Workload Automation.
3. The scheduler starts monitoring `CPU1`.

   If the workstation status becomes unlinked, HCL Workload Automation starts the 10 minute timeout. If the `CPU1 linked` event is not received within 10 minutes, the scheduler sends the notification email to Chuck Derry.
4. Chuck Derry receives the email, queries the actions/rules that were triggered in the last 10 minutes, and from there navigates to the `CPU1` instance and runs a first problem analysis.

Scenario 3: Submit job stream when FTP has completed

1. The administrator defines the following event rule:
   - When file `daytransac*` is created in the `SFoperation` directory in workstation `system1`, and modifications to the file have terminated, submit the `calmonthlyrev` job stream.

     The event rule is valid year-round in the 18:00-22:00 EST time window.
2. The administrator saves the rule as non-draft in the database and it is readily deployed by HCL Workload Automation.
3. The scheduler starts monitoring the `SFoperation` directory. As soon as file `daytransac*` is created and is no longer in use, it submits job stream `calmonthlyrev`.
4. The operator can check the logs to find if the event rule or the job stream were run.

Scenario 4: Start long duration jobs based on timeout

1. The administrator defines the following event rule:
   - When the `job-x=exec` event and the `job-x=succ/abend` event are received in 5 minutes, the scheduler should reply `Yes` to `prompt-1` and start the `jobstream-z` job stream, otherwise it should send an email to `twsoper@mycompany.com` alerting that the job is late.
2. The administrator saves the event rule in draft status. After a few days the administrator edits the rule, changes the email recipient and saves it as non-draft. The rule is deployed.
3. Every time the status of `job-x` becomes `exec`, HCL Workload Automation starts the 5 minutes timeout.

   If the internal state of `job-x` does not change to `succ` or `abend` within 5 minutes and the corresponding event is not received, HCL Workload Automation sends the email, otherwise it replies `Yes` to the prompt and submits `jobstream-z`.

Scenario 5: Monitoring process status and running a batch script

The administrator creates a rule to monitor the status of HCL Workload Automation processes and run a batch script.

Scenario 6: Monitoring the Symphony file status and logging the occurrence of a corrupt record

The administrator creates a rule to monitor the status of the Symphony file in the HCL Workload Automation instance and logs the occurrence of a corrupt Symphony dependency record in the internal auditing database.

For a detailed example about how to set up an event rule that monitors the used disk space, see the section about Maintaining the file system in the *Administration Guide*.

The event rule management process

Event-driven workload automation is an ongoing process and can be reduced to the following steps:

1. An event rule definition is created or modified with the Dynamic Workload Console or with the composer command line and saved in the objects database. Rule definitions can be saved as draft on page 182 or non-draft on page 182.
2. All new and modified non-draft rules saved in the database are periodically (by default every five minutes) found, built, and deployed by an internal process named `rule builder`. At this time they become active. Meanwhile, an `event processing server`, which is normally located in the master domain manager, receives all events from the agents and processes them.
3. The updated monitoring configurations are downloaded to the HCL Workload Automation agents and activated. Each HCL Workload Automation agent runs a component named `monman` that manages two services named `monitoring engine` and `ssmagent` that are to catch the events occurring on the agent and perform a preliminary filtering action on them.
4. Each `monman` detects and sends its events to the event processing server.
5. The event processing server receives the events and checks if they match any deployed event rule.
6. If an event rule is matched, the event processing server calls an action helper to carry out the actions.
7. The action helper creates an event rule instance and logs the outcome of the action in the database.
8. The administrator or the operator reviews the status of event rule instances and actions in the database and logs.

The event-driven workload automation feature is automatically installed with the product. You can at any time change the value of the `enEventDrivenWorkloadAutomation` global option if you do not want to use it in your HCL Workload Automation network.

Event-driven workload automation is based on a number of services, subsystems, and internal mechanisms. The following ones are significant because they can be managed:

monman

Is installed on every HCL Workload Automation agent where it checks for all local events. All detected events are forwarded to the event processing server. The following `conman` commands are available to manage `monman`:

Table 18. conman commands for managing monitoring engines

Command	Purpose
deployconf	Updates the monitoring configuration file for the event monitoring engine on an agent. It is an optional command since the configuration is normally deployed automatically.
showcpus getmon	Returns the list of event rules defined for the monitor running on an agent. This command can be used remotely to get the information of the configuration file in another agent of the network
startmon	Starts `monman` on an agent. Can be issued from a different agent.
stopmon	Stops `monman` on an agent. Can be issued from a different agent.

`monman` starts automatically each time a new Symphony is activated. This is determined by the `autostart monman` local option that is set to `yes` by default (and that you can disable if you do not want to monitor events on a particular agent).

Following each rule deployment cycle, updated monitoring configurations are automatically distributed to the agents hosting rules that have been changed since the last deployment. Note that there might be some transitory situations while deployment is under course. For example, if a rule is pending deactivation, the agents might be sending events in the time fraction that the new configuration files have not been deployed yet, but the event processor already discards them.

If an agent is unable to send events to the `event processing server` for a specified period of time, the monitoring status of the agent is automatically turned off. The period of time can be customized (in seconds) with the `edwa connection timeout` parameter in the `localopts` file. By default, it is set to 300 seconds (5 minutes).

The following events can be configured in the `BMEvents.conf` file to post the monitoring status of an agent:

- TWS_Stop_Monitoring (261) : sent when the monitoring status of an agent is set to off (for stopmon command or because the agent is unable to send events to the `event processing server`).
- TWS_Start_Monitoring (262): sent when the monitoring status of an agent is set to on (for startmon command or because the agent has restarted to send events to the `event processing server`).

These events have the following positional fields:

1. Event number
2. Affected workstation
3. Reserved, currently always set to 1

Note: The domain manager hosting the workload broker workstation connected to a dynamic agent, either a master domain manager or a dynamic domain manager, deploys and manages events related to that dynamic agent.

The only event type that a dynamic agent deploys and manages locally is FileMonitor.

Event processing server

Can be installed on the master domain manager, the backup master, or on any fault-tolerant agent installed as a backup master. It runs in the application server. It can be active on only one node in the network. It builds the rules, creates configuration files for the agents, and notifies the agents to download the new configurations. It receives and correlates the events sent by the monitoring engines and runs the actions. The following `conman` commands are available to manage the event processing server:

Table 19. conman commands for managing the event processing server

Command	Purpose
starteventprocessor	Starts the event processing server
stopeventprocessor	Stops the event processing server
switcheventprocessor	Switches the event processing server from the master domain manager to the backup master or fault-tolerant agent installed as a backup master, or vice versa

The event processing server starts automatically with the master domain manager. Only one event processor may run in the network at any time. If you want to run the event processor installed on a workstation other than the master (that is, on the backup master or on any fault-tolerant agent installed as backup master), you must first use the switcheventprocessor on page 725 command to make it the active event processing server.

Note: If you set the ignore keyword on the workstation definition on page 201 of the agent (installed as backup master) that at the time hosts the active event processor, the first following JnextPlan occurrence acknowledges that this particular agent is out of the plan. As a consequence, it cannot restart the event processor hosted there. For this reason, the scheduler yields a warning message and starts the event processor hosted by the master domain manager.

Using the involved interfaces and commands

About this task

Running and managing event-driven workload automation calls for the following tasks:

- Edit configuration settings
- Model event rules
- Manually deploy or undeploy event rules
- Manage monitoring and event processing devices
- Monitor and manage event rule instances

You must be ready to use several HCL Workload Automation interfaces and commands to do them. Table 20: Interfaces and commands for managing event-driven workload automation on page 180 summarizes the ones you need:

Table 20. Interfaces and commands for managing event-driven workload automation

Interface or command	Use to...
optman	Change the default values of global options associated with event management. Global options are used to configure: • The frequency with which rule definitions are checked for updates (`deploymentFrequency`). Modified definitions are deployed in the HCL Workload Automation domain • The EIF port number where the event processing server receives events (`eventProcessorEIFPort`, or `eventProcessorEIFSSLPort` when SSL-protected). • Management of the cleanup policies of rule instance, action run, and message log data (`logCleanupFrequency`). • SMTP server properties if you deploy rules implementing actions that send emails via an SMTP server (`smtpServerName`, `smtpServerPort`, `smtpUseAuthentication`, `smtpUserName`, `smtpUserPassword`, `smtpUseSSL`, `smtpUseTLS`). • The possibility to disable the event rule management mechanism (`enEventDrivenWorkloadAutomation`) which is installed by default with the product. See the *Administration Guide* for a list of global options.
composer	Run modeling and management tasks of event rule definitions like add, create, delete, display, extract, list, lock, modify, new, print, unlock, validate. Event rules are defined in XML. Query the HCL Workload Automation relational database for: • event rule definitions filtered by: ◦ rule, event, and action properties ◦ jobs and job streams involved with the rule action • event rule instances, actions run, and message log records See Event rule definition on page 394 to learn how to define event rules. See Managing objects in the database - composer on page 434 for command reference.
Dynamic Workload Console	Have a graphical user interface to:

Table 20. Interfaces and commands for managing event-driven workload automation (continued)

Interface or command	Use to...
	• Model and manage event rule definitions (browse, create, delete, modify, query, unlock) • Query the HCL Workload Automation relational database for: ◦ event rule definitions filtered by: ▪ rule, event, and action properties ▪ jobs and job streams involved with the rule action ◦ event rule instances, actions run, and message log records • Manage the event processing server and monitoring engines, as described in tables Table 18: conman commands for managing monitoring engines on page 178 and Table 19: conman commands for managing the event processing server on page 179 See the Dynamic Workload Console documentation: Dynamic Workload Console User's Guide, section about Creating an event rule. Dynamic Workload Console User's Guide, section about Event management tasks.
conman	Manage the monitoring devices, namely the event processing server and monitoring engines, as described in tables Table 18: conman commands for managing monitoring engines on page 178 and Table 19: conman commands for managing the event processing server on page 179. See Managing objects in the plan - conman on page 545 for command reference.
utility commands	Create custom event definitions and manually send custom events to the event processing server. See evtdef on page 934 and sendevent on page 967 for details on these commands.
planman	Manually deploy new and changed rules. See Deploying rules on page 138 for details.
Security file	Set security authorizations to manage event rules, events, actions, and their instances. See the *HCL Workload Automation Administration Guide* for reference about configuring the HCL Workload Automation security file.

Important: If you use a security firewall, make sure that the ports defined in global option `eventProcessorEIFPort` and in the `nm port` local option on each agent are open for incoming and outgoing connections.

Defining event rules

About this task

When you define an event rule, you specify one or more events, a correlation rule, and one or more actions. To define event rules you can use:

- The `composer` command line
- The Dynamic Workload Console
- A set of APIs described in a separate document

  You can try out the REST API services and the operations available for each API on Swagger Docs by connecting to `https://MDM_IP_address:tdwbport/twsd/`. There, click **List Operations** to view the operations available with the service, and then click **Expand Operations** to view details such as, the implementation notes, parameters, and response messages for each method. At the end of the details you can find a **Try it out!** button to see the operation in action.

  You can also access some HCL Workload Automation REST API samples here: REST API samples.

In `composer`, you edit the rules with an XML editor of your choice (preferable but not mandatory) since event rule definitions are made using XML syntax.

The explanation of how you use `composer` to define event rules is in Event rule definition on page 394, while the explanation of how you use the Dynamic Workload Console can be found in: Dynamic Workload Console User's Guide, section about Creating an event rule.

Event rule definitions are saved in the HCL Workload Automation database like all other scheduling objects. You can save them as:

Draft

The rule is saved in the database but is not ready yet to be deployed and activated.

This state is determined by the `isDraft=yes` attribute.

Not draft

This rule is being deployed or is ready to be deployed in the scheduling environment.

This state is determined by the `isDraft=no` attribute.

Non-draft rules are ready to be activated. The rule builder calculates the status of each rule. The status on page 192 can be:

- active
- not active
- update pending
- update error
- activation pending
- activation error
- deactivation pending
- deactivation error

The scheduler periodically (every five minutes or in accordance with a time set in the `deploymentFrequency` global configuration option) scans the database for non-draft rules and builds rule configuration files for deployment. The new monitoring configurations are downloaded to the agents (each agent gets its own configuration file containing strictly the rules it is to run) only if there have been changes since the previous configuration files.

As an additional feature, a `planman deploy` command is available to deploy rules manually at any time.

The time required for deployment increases proportionally with the number of active rules to deploy. If you need to manually deploy a large number of new or changed rules, and you are concerned with keeping the deployment time low, run `planman deploy -scratch` to reduce the deployment time.

You can deploy and undeploy rules as you need by setting the `isDraft` attribute to `no` or to `yes` with `composer` or with the Dynamic Workload Console.

Based on their characteristics, rules are classified as:

filter

The rule is activated upon the detection of a single specific event.

sequence

The rule is activated when an ordered group of events is detected or fails to complete within a specific time interval.

set

The rule is activated when an unordered group of events is detected or fails to complete within a specific time interval.

Filter rules are based on the detection of one event such as a job being late, an HCL Workload Automation workstation going down, a file being modified, a job stream completing its run successfully, and so on.

Set and sequence rules are based on the detection of more events. Optionally, they can be based on a timeout condition. A rule times out when the first event(s) in a sequence or part of the events in a set are received, but not all the events are received within a specified time interval counted from when the first event was received.

Rule definitions may include attributes that specify a validity period and an activity time window within each day of validity. If you do not specify these attributes, the rule is active perpetually at all times once it is deployed and until it is changed back to draft status or deleted from the database.

You can use variable substitution. This means that when you define action parameters, you can use attributes of occurrences of events that trigger the event rule in either of these two forms:

- `${event.property}`

  Replaces the value as is. This is useful to pass the information to an action that works programmatically with that information, for example the `schedTime` of a job stream.

- `%{event.property}`

  Replaces the value formatted in human readable format. This is useful to pass the information to an action that forwards that information to a user, for example to format the `schedTime` of a job stream in the body of an email.

Where:

event

Is the name you set for the triggering `eventCondition`.

property

Is the `attributeFilter` name in the filtering predicate of the triggering event condition. The value taken by the attribute filter when the rule is triggered is replaced as a parameter value in the action definition before it is performed.

Note that you can use variable substitution also if no `attributeFilter` was specified for an attribute and also if the attribute is read-only.

You can see the use of variable substitution in some of the following sample definitions, where attribute filter values are replaced in email subject and body matters.

Security checks on event rules

Security checks on event rules enabled by default starting from version 9.5, Fix Pack 1.

Starting from product version 9.5, Fix Pack 1, security checks are enabled by default on event rules. The checks are enabled both for fresh installations and for upgrades from previous versions.

These checks verify that the user saving the event rule has DISPLAY permission on the relevant objects and thereby ensure a higher level of security when accessing event rules data.

To disable the security checks, set the `com.ibm.tws.conn.event.security.enabled` property to `false` in the `TWSConfig.properties` file. The `TWSConfig.properties` file is located in the following path:

On UNIX operating systems

TWA_DATA_DIR`/usr/servers/engineServer/resources/properties`

On Windows operating systems

`TWA_home\usr\servers\engineServer\resource\properties`

When you save the event rules, a security check is automatically performed to verify that you have DISPLAY permission for the following events:

FileMonitor

for all events that reference a workstation, DISPLAY permission is required on the specified workstation.

TWSObjectsMonitor

- for all job events, such as `JobStatusChanged`, DISPLAY permission is required on the specified job.
- for all job stream events, such as `Job Stream Status Changed`, DISPLAY permission is required on the specified job stream.
- for `Alert`, `Application Server` and `Workstation` events, DISPLAY permission is required on the specified workstation. If the event type is `Child Workstation Link Changed` or `Parent Workstation Link Changed`, DISPLAY permission is required on the specified child or parent workstation.
- for all prompt events, the following considerations apply:
  - if the prompt name refers to a global prompt, DISPLAY permission is required on the specified prompt.
  - if the prompt name refers to a local prompt, DISPLAY permission is required on the specified job and job stream. If no job nor job stream is specified, the `*` wildcard is assumed.
  - if the prompt name starts with the `*` wildcard, DISPLAY permission is required on the specified prompt, job, and job stream. If no job nor job stream is specified, the `*` wildcard is assumed.

If the user does not have the required permission, the event rule is not saved and an error message is displayed.

For more information about configuring security, see the section about configuring user authorization (Security file) in *Administration Guide*.

Event rule examples

The following are examples of event rule definitions that apply to the scenarios described in Simple event rule scenarios on page 175.

Event rule definition for scenario #1

When any of the `job123` jobs terminates in error and yields the following error message:

```
AWSBHT001E The job "MYWORKSTATION#JOBS.JOB1234" in file "ls" has failed with
the error: AWSBDW009E The following operating system error occurred retrieving
the password structure for either the logon user...
```

send an email to operator `john.smith@mycorp.com`. The subject of the email includes the names of the job instance and of the associated workstation.

The event rule is valid from December 1st to December 31st in the 12:00-16:00 EST time window.

```
<?xml version="1.0"?>
   <eventRuleSet xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
      xmlns="http://www.ibm.com/xmlns/prod/tws/1.0/event-management/rules"
      xsi:schemaLocation="http://www.ibm.com/xmlns/prod/tws/1.0/
                          event-management/rules/EventRules.xsd">
                          <eventRule name="scenario1_rule" ruleType="filter" isDraft="no">
                          <description>This is the definition for scenario1</description>
                          <timeZone>America/Indianapolis</timeZone>
   <validity from="2023-12-01" to="2023-12-31" />
   <activeTime start="12:00:00" end="16:00:00" />
   <eventCondition name="event1"
                     eventProvider="TWSObjectsMonitor"
                     eventType="JobStatusChanged">
                     <filteringPredicate>
                     <attributeFilter name="JobStreamWorkstation" operator="eq">
                     <value>*</value>
                     </attributeFilter>
                     <attributeFilter name="JobStreamName" operator="eq">
                     <value>*</value>
                     </attributeFilter>
                     <attributeFilter name="JobName" operator="eq">
                     <value>job123*</value>
                     </attributeFilter>
                     <attributeFilter name="Status" operator="eq">
                     <value>Error</value>
                     </attributeFilter>
                     <attributeFilter name="ErrorMessage" operator="eq">
                     <value>*AWSBDW009E*</value>
                     </attributeFilter>
                     </filteringPredicate>
                     </eventCondition>
                     <action actionProvider="MailSender"
              actionType="SendMail"
              responseType="onDetection">
              <description>Send email to John Smith including names of job
              and associated workstation</description>
   <parameter name="To">
   <value>john.smith@mycorp.com</value>
   </parameter>
   <parameter name="Subject">
   <value>Job %{event1.JobName} on agent %{event1.Workstation}
   ended in error</value>
   </parameter>
   </action>
   </eventRule>
   </eventRuleSet>
```

Important: The error message that explains why a job terminates in error can be found in the TWSMERGE log file. In this scenario, the TWSMERGE log file contains the following statement:

```
BATCHMAN:+
BATCHMAN:+ AWSBHT001E The job "MYWORKSTATION#JOBS.JOB1234" in file "ls"
```



```
has failed with the error: AWSBDW009E The following operating system
error occurred retrieving the password structure for either the logon
user, or the user who owns a file or external dependency
BATCHMAN:+
```

where the error message is everything that follows the string:

```
has failed with the error:
```

Event rule definition for scenario #2

If workstation `CPU1` becomes unlinked and does not link back within 1 hour, send a notification email to `chuck.derry@mycorp.com`.

```
<?xml version="1.0"?>
      <eventRuleSet xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
       xmlns="http://www.ibm.com/xmlns/prod/tws/1.0/event-management/rules"
       xsi:schemaLocation="http://www.ibm.com/xmlns/prod/tws/1.0/
              event-management/rules http://www.ibm.com/xmlns/prod/tws/1.0/
              event-management/rules/EventRules.xsd">
              <eventRule name="SCENARIO2_RULE" ruleType="sequence" isDraft="no">
              <description>This is the definition for scenario2</description>
              <timeZone>America/Anchorage</timeZone>
              <timeInterval amount="10" unit="minutes"/>
              <eventCondition name="WSevent"
                            eventProvider="TWSObjectsMonitor"
                            eventType="ChildWorkstationLinkChanged">
                            <scope>
       * TO CPU1
       </scope>
       <filteringPredicate>
       <attributeFilter name="Workstation" operator="eq">
       <value>CPU1</value>
       </attributeFilter>
       <attributeFilter name="LinkStatus" operator="eq">
       <value>Unlinked</value>
       </attributeFilter>
       </filteringPredicate>
       </eventCondition>
       <eventCondition name="childWksLnkChgEvt1"
                            eventProvider="TWSObjectsMonitor"
                            eventType="ChildWorkstationLinkChanged">
                            <scope>
    * TO CPU1
    </scope>
    <filteringPredicate>
    <attributeFilter name="Workstation" operator="eq">
    <value>CPU1</value>
    </attributeFilter>
    <attributeFilter name="LinkStatus" operator="eq">
    <value>Linked</value>
    </attributeFilter>
    </filteringPredicate>
    </eventCondition>
    <action actionProvider="MailSender" actionType="SendMail"
              responseType="onTimeOut">
              <scope>
```

```
      CHUCK.DERRY@MYCORP.COM : AGENT CPU1 HAS BEEN UNLINKED FOR AT LEAST 10 MINUTES
      </scope>
      <parameter name="Subject">
      <value>Agent CPU1 has been unlinked for at least 10 minutes</value>
      </parameter>
      <parameter name="To">
      <value>chuck.derry@mycorp.com</value>
      </parameter>
      <parameter name="Body">
      <value>The cause seems to be: %{WSevent.UnlinkReason}</value>
      </parameter>
      </action>
      </eventRule>
      </eventRuleSet>
```

Event rule definition for scenario #3

When file `daytransac` is created in the `SFoperation` directory in workstation `system1`, and modifications to the file have terminated, submit the `calmonthlyrev` job stream.

The event rule is valid year-round in the 18:00-22:00 EST time window.

```
<?xml version="1.0"?>
      <eventRuleSet xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
      xmlns="http://www.ibm.com/xmlns/prod/tws/1.0/event-management/rules"
      xsi:schemaLocation="http://www.ibm.com/xmlns/prod/tws/1.0/
                          event-management/rules/EventRules.xsd">
                          <eventRule name="scenario3_rule"
              ruleType="filter"
              isDraft="no">
              <description>This is the definition for scenario3</description>
              <timeZone>America/Louisville</timeZone>
      <validity from="2023-01-01" to="2023-12-31" />
      <activeTime start="18:00:00" end="22:00:00" />
      <eventCondition eventProvider="FileMonitor"
                      eventType="ModificationCompleted">
                      <filteringPredicate>
                      <attributeFilter name="FileName" operator="eq">
                      <value>daytransac</value>
                      </attributeFilter>
                      <attributeFilter name="Workstation" operator="eq">
                      <value>EVIAN1</value>
                      </attributeFilter>
                      </filteringPredicate>
                      </eventCondition>
                      <action actionProvider="TWSAction"
              actionType="sbs"
              responseType="onDetection">
              <description>Submit the calmonthlyrev job stream.</description>
              <parameter name="JobStreamName">
              <value>calmonthlyrev</value>
      </parameter>
      <parameter name="JobStreamWorkstationName">
      <value>act5cpu</value>
      </parameter>
      </action>
      </eventRule>
```

```
      </eventRuleSet>
```

Event rule definition for scenario #4

When the `job-x=exec` event and the `job-x=succ/abend` event are received in 500 seconds, the scheduler should reply `Yes` to `prompt-1` and start the `jobstream-z` job stream, otherwise it should send an email to `twsoper@mycompany.com` alerting that the job is late.

```
<?xml version="1.0"?>
      <eventRuleSet xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
      xmlns="http://www.ibm.com/xmlns/prod/tws/1.0/event-management/rules"
      xsi:schemaLocation="http://www.ibm.com/xmlns/prod/tws/1.0/
                          event-management/rules/EventRules.xsd">
                          <eventRule name="scenario4_rule"
              ruleType="sequence"
              isDraft="yes">
              <description>This is the definition for scenario4</description>
              <timeZone>America/Buenos_Aires</timeZone>
      <timeInterval amount="500" unit="seconds" />
      <eventCondition eventProvider="TWSObjectsMonitor"
                      eventType="JobStatusChanged">
                      <filteringPredicate>
                      <attributeFilter name="JobName" operator="eq">
                      <value>job-x</value>
      </attributeFilter>
      <attributeFilter name="InternalStatus" operator="eq">
      <value>EXEC</value>
      </attributeFilter>
      </filteringPredicate>
      </eventCondition>
      <eventCondition eventProvider="TWSObjectsMonitor"
                      eventType="JobStatusChanged">
                      <filteringPredicate>
                      <attributeFilter name="JobName" operator="eq">
                      <value>job-x</value>
                      </attributeFilter>
                      <attributeFilter name="InternalStatus" operator="eq">
                      <value>ABEND</value>
                      <value>SUCC</value>
                      </attributeFilter>
                      </filteringPredicate>
                      </eventCondition>
                      <action actionProvider="MailSender"
              actionType="SendMail"
              responseType="onTimeOut">
              <description>Send email to operator saying that job-x
              is late</description>
              <parameter name="To">
              <value>twsoper@mycorp.com</value>
      </parameter>
      <parameter name="Subject">
      <value>Job-x is late by at least 5 minutes</value>
      </parameter>
      </action>
      <action actionProvider="TWSAction"
              actionType="Reply"
              responseType="onDetection">
```

```
              <description>Reply Yes to prompt-1</description>
              <parameter name="PromptName">
         <value>prompt-1</value>
         </parameter>
         <parameter name="PromptAnswer">
         <value>Yes</value>
         </parameter>
         </action>
         <action actionProvider="TWSAction"
              actionType="sbs"
              responseType="onDetection">
              <description>Submit jobstream-z</description>
              <parameter name="JobStreamName">
              <value>jobstream-z</value>
              </parameter>
              <parameter name="JobStreamWorkstationName">
              <value>act23cpu</value>
              </parameter>
              </action>
              </eventRule>
              </eventRuleSet>
```

Event rule definition for scenario #5

Monitor the status of HCL Workload Automation processes listed in ProcessName and run the RUNCMDFM.BAT batch script located in E:\production\eventRules.

The TWSPATH keyword indicates the fully qualified path where the monitored HCL Workload Automation instance is installed, including the /TWS suffix.

On Windows operating systems, the event rule is triggered every time the agent is stopped using the ShutDownLwa command and every time the agent is stopped manually. On UNIX operating systems, the event rule is triggered when you stop the process manually, while it is not triggered by the ShutDownLwa command.

If you specify ProcessName=agent, the agent component is monitored, while the TWS JobManager process is not monitored.

```
<?xml version="1.0"?>
      <eventRuleSet xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
      xmlns="http://www.ibm.com/xmlns/prod/tws/1.0/event-management/rules"
      xsi:schemaLocation="http://www.ibm.com/xmlns/prod/tws/1.0/
                          event-management/rules/EventRules.xsd">
                          <eventRule name="scenario5rule" ruleType="filter" isDraft="no">
                          <eventCondition name="twsProcMonEvt1"
        eventProvider="TWSApplicationMonitor"
        eventType="TWSProcessMonitor">
        <scope>
                AGENT, NETMAN DOWN ON WIN86MAS
                </scope>
                <filteringPredicate>
                <attributeFilter name="ProcessName" operator="eq">
                <value>agent</value>
                <value>appservman</value>
                <value>batchman</value>
                <value>jobman</value>
```

```
                <value>mailman</value>
                <value>monman</value>
                <value>netman</value>
                </attributeFilter>
                <attributeFilter name="TWSPath" operator="eq">
                <value>E:\Program Files\HCL\TWA\TWS</value>
      </attributeFilter>
      <attributeFilter name="Workstation" operator="eq">
      <value>win86mas</value>
      </attributeFilter>
      <attributeFilter name="SampleInterval" operator="eq">
      <value>5</value>
      </attributeFilter>

</filteringPredicate>
</eventCondition>
<action actionProvider="GenericActionPlugin" actionType="RunCommand"
    responseType="onDetection">
    <scope>
                RUNCMDFM.BAT
                </scope>
                <parameter name="Command">
                <value>runCmdFM.bat</value>
      </parameter>
      <parameter name="WorkingDir">
      <value>E:\production\eventRules</value>
      </parameter>
      </action>
      </eventRule>
      </eventRuleSet>
```

Event rule definition for scenario #6

Monitor the Symphony file status in the *IT041866-9088* workstation and logs the occurrence of a corrupt Symphony record in the internal auditing log database.

In each workstation, the Batchman process monitors the Symphony file. When it detects a corrupt record, it send the corruption event to the Monman process message queue and then to Event Processor in the Master workstation.

The event rule is triggered every time the Batchman process finds a corrupt dependency record in the workstation specified in the event rule definition.

If you set the workstation value to IT041866-9088, the Symphony file on this workstation is monitored, and the event rule is triggered when the Batchman process on this workstation detects a corrupt record in the Symphony file.

The occurrence of the corrupt record is written to the Messagge logger audit file.

```
?xml version="1.0"?>
   <eventRuleSet xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
xmlns="http://www.ibm.com/xmlns/prod/tws/1.0/event-management/rules"
xsi:schemaLocation="http://www.ibm.com/xmlns/prod/tws/1.0/event-management/rules
http://www.ibm.com/xmlns/prod/tws/1.0/event-management/rules/EventRules.xsd">
<eventRule name="TEST1" ruleType="filter" isDraft="no">
<eventCondition name="productAlertEvt1" eventProvider="TWSObjectsMonitor"
eventType="ProductAlert">
```

```
<scope>
IT041866-9088
</scope>
   <filteringPredicate>
   <attributeFilter name="Workstation" operator="eq">
   <value>IT041866-9088</value>
   </attributeFilter>

</filteringPredicate>
</eventCondition>
<action actionProvider="MessageLogger" actionType="MSGLOG"
responseType="onDetection">
<scope>
OBJECT=%{PRODUCTALERTEVT1.WORKSTATION} MESSAGE=%{PRODUCTALERTEVT1.WORKSTATION}
corruption
</scope>
<parameter name="ObjectKey">
<value>%{productAlertEvt1.Workstation}</value>
   </parameter>
   <parameter name="Message">
   <value>%{productAlertEvt1.Workstation} corruption</value>
   </parameter>
   <parameter name="Severity">
   <value>Info</value>
   </parameter>
   </action>
   </eventRule>
   </eventRuleSet>
```

Rule operation notes

The following contains critical information on event rule behavior that might help you understand the reason for unexpected results:

Notes on rule status

- Depending on its `from` and `to` validity dates, the status of any rule changes as follows upon deployment:
  - If you create a rule with already expired `from` and `to` validity dates, the rule is saved in `activation pending` state. When the rule is deployed, it remains in `activation pending` status.
  - If you set the `to` validity field to a future date, the rule is deployed in the `active` state. If you reset it to a past date, the rule is redeployed in the `no active` state.
- Rule activity times (`start` and `end`) do not affect rule status. As long as a rule is within the right validity dates, the rule remains in the `active` state regardless whether it is within its defined activity times. If the scheduler receives a rule 's defined events outside its activity time, the events are discarded but the rule stays in the `active` status.

Event rule paths

Only on Windows operating systems, when you define a path for an event rule, you cannot use the slash `/` but only the single backslash `\` or double backslash `\\`.

Event rules constrained by files

When you have a file monitoring event rule constrained by the creation or existence of a file, if this file is deleted and then re-created in the same time interval between two consecutive checks, this file is unchanged for HCL Workload Automation and the event rule is not triggered. The time interval between checks can be reduced to a minimum of 5 seconds, but this involves a significant performance degradation. To avoid this issue, do not make file actions during this time interval. File checks are performed using the permissions of the user running ssmagent.

File monitoring event rules for files on Network File Systems

To activate a file monitoring event rule for files on Network File Systems, you must enable the property `MonitorRemoteFS` in the `init.cfg` file, which is located in:

dynamic agents

On UNIX operating systems

TWA_DATA_DIR`/EDWA/ssm/config/init.cfg`

On Windows operating systems

`tws_install_dir``\TWS\EDWA\ssm\Config\init.cfg`

fault-tolerant agents

On UNIX operating systems

TWA_DATA_DIR`/ssm/config/init.cfg`

On Windows operating systems

`tws_install_dir``\TWS\ssm\Config\init.cfg`

where *`tws_install_dir`* is the directory path where the HCL Workload Automation agent is installed. To activate this property, perform the following actions:

1. Edit the `init.cfg` file.
2. Change from `MonitorRemoteFS=off` to `MonitorRemoteFS=on`.
3. Stop the System Service Monitor (SSM) agent.
4. Start the SSM agent.

Note: On Windows operating systems, the remote workstation address must be included at the beginning of the full path of the file to be monitored. For example: `\\123.123.123.123\images\Automation\myFile.txt`. This is not needed on UNIX operating systems.

Lack of persistence in rule instance status

If the event processor is stopped or crashes, the status of rule instances that are only partially matched is lost.

Repeating `set` and `sequence` rules

Typically, each active rule has one, and only one, copy that runs in the event processing server.`Set` and `sequence` rules use the mechanism explained in the following example:

1. You define a `sequence` rule with two events, A and B.
2. When the first event that matches the sequence occurs (event A), it activates the rule and waits for the second event (event B).

   Once the rule is active, any additional event A events that may arrive are ignored. No additional rules are created for any newly detected event A events as long as the rule is waiting for event B.
3. Once event B occurs, the rule is completed and resets, waiting for event A to occur again.

The mechanism of `set` and `sequence` rules is such that any additional occurrences of an already detected event are ignored and discarded if the other defined events have not arrived.

You can prevent this problem by using correlation attributes. Using one or more correlation attributes is a method for directing a rule to create a separate rule copy when another event A arrives.

`Set` and `sequence` rule types on page 397 with shorter than 24 hours activity time on page 399 windows

Occurrences of `set` or `sequence` rules that were defined to be active for only a few hours of every day, are not purged when the activity period within each day expires if only part of the events arrive. They remain in an idle state, waiting to receive the remaining events in the following days.

For example, you define a `set` rule that includes two events. The rule is valid from January 1 to January 10, and is active daily from 6 to 10 AM.

If on January 1 the first event is received at 8 AM, the rule waits for the second event to take place, and stays *alive* beyond 10 AM if the event is not detected.

If the second event is received at 11 AM (which is out of the activity time window), it is discarded, but the rule keeps on staying *alive*. If the second event is received again at 7 AM of January 2, the rule is triggered and the actions are implemented.

If you don not want the rule to *carry over* to the next day, you should purge it.

Triggered rule elements

Every time the event conditions listed in a deployed event rule are matched, or timeout, an event rule instance is created. An event rule instance includes information like event rule name, date and time when it was matched, and the list of action instances, and their status, that were run as a result of the matching event conditions. The `RuleInstance` object is used to collect information about triggered rules in a history log of rule instances.

Actions carried out by a triggered rule are collected in a history log of action runs. The provided information includes action runs that have been completed with errors or warnings, as opposed to successful ones. If at least one action ends in error, then the whole rule instance will be reported in error. As part of the information tracked in action runs, rule fields are also maintained, and queries can be executed to look for action runs based on these fields (the rule instance identifier is also available).

Defining custom events

In addition to the already defined event types and event classes (known as providers) listed in detail in Event providers and definitions on page 1381, HCL Workload Automation supplies the template of a generic event provider named `GenericEventPlugIn` that programmers with specific application and XML programming skills can modify to define custom event types that might be of use to the organization.

The tools supplied to define custom event types are:

- The `GenericEventPlugIn` event provider in XML
- The evtdef on page 934 utility command with which a programmer can download the `GenericEventPlugIn` event provider as a local file to define the custom events
- The XML schema definition (XSD) files necessary to validate the modified generic event provider. They also contain online guidelines to aid in the programming task.
- The sendevent on page 967 utility command with which the custom events can be sent to the event processing server to trigger rules from any agent or any workstation running simply the HCL Workload Automation remote command line client.

This is the flow for defining and using custom events:

1. With the evtdef on page 934 command, the programmer:
    a. Downloads the generic event provider as a local file.
    b. Follows the schema definitions to add custom event types and to define their properties and attributes in the file with an XML editor.
    c. Uploads the local file as the modified generic event provider containing the new custom event type definitions. The modified generic event provider is saved in an XML file on the master domain manager.
2. The rule builder, or the administrator, defines, with either composer or the Dynamic Workload Console, the event rules that are to be triggered by these custom events, specifying:
    - The generic event provider as the event provider
    - The custom event types as the event types
    - The custom event type properties (or attributes) defined for the custom events in the generic event provider with the particular values that will trigger the rules.
3. Deploy the rules.
4. When the occurrence of a custom event takes place, it can be sent to the event processing server by the sendevent on page 967 command, run from a script or from the command line

    As soon as the event is received by the event processing server, it triggers the rule.



Note: When you perform an upgrade, custom events are not migrated. To add custom events follow the procedure described in the topic about migrating custom events in *HCL Workload Automation: Planning and Installation*.

Chapter 9. Defining objects in the database

HCL Workload Automation is based on a set of object definitions that are used to map your scheduling environment into the database. Scheduling objects are:

- calendars
- domains
- event rules
- jobs
- job streams
- prompts
- parameters
- resources
- run cycle groups
- folders
- variable tables
- workload applications
- workstations
- workstation classes
- users

A special set of scheduling objects is composed of the security objects. Security objects map the role-based security in your scheduling environment. Security objects are:

- access control lists
- security domains
- security roles

Defining scheduling objects

Scheduling objects are managed with the **composer** command-line program and are stored in the HCL Workload Automation database. The **composer** command-line program can be installed and used on any machine connected through TCP/IP to the system where the master domain manager is installed. It does not require the installation of an HCL Workload Automation workstation as a prerequisite. It communicates through HTTP/HTTPS with the master domain manager where the DB2® database is installed. The HTTP/HTTPS communication setup and the authentication check are managed by the WebSphere Application Server Liberty infrastructure. The **composer** program uses edit files to update the scheduling database. The format of the edit file used to define a specific object is described later in this chapter. For example, to create a new object, enter its definition in an edit file, and then use **composer** to add it to the database by specifying the edit file containing the definition. The **composer** command-line program checks for correct syntax inside the edit file, and, if correct, transforms the object definition into XML language and then sends the request through HTTP/HTTPS to the master domain manager.

On the master domain manager the XML definition is parsed, semantic and integrity checks are performed, and then the update is stored in the database.

All entries are managed individually and an object locking mechanism is used to manage concurrent access. Scheduling objects defined in the database are locked while accessed by a user to prevent concurrent accesses. This means that only the user locking the object has `write` permission to that object, and other users have `read only` access to that object. For additional information refer to LOCK on page 488 and UNLOCK on page 513.

You can use short and long keywords when issuing commands from **composer**, as shown in Table 21: List of supported scheduling object keywords on page 197 and Table 22: List of supported security object keywords on page 198. The first two columns on the left list the long and short keyword formats currently supported by HCL Workload Automation. The rightmost column lists the formats compatible with earlier versions that you want to use if your network includes pre-version 8.3 installations.

Table 21. List of supported scheduling object keywords

Long keywords	Short keywords	Keywords compatible with installations earlier than version 8.3
calendar	cal	calendars
domain	dom	cpu
eventrule	erule | er	—
jobdefinition	jd	jobs
jobstream	js	sched
parameter	parm	parms
prompt	prom	prompts
resource	res	resources
user	user	users
variabletable	vt	—
wat	wat	—
workstation	ws	cpu
workstationclass	wscl	cpu

Note: The **cpu** keyword is maintained to represent domains, workstations, and workstation classes for compatibility with earlier versions.

Table 22. List of supported security object keywords

Long keywords	Short keywords	Keywords compatible with installations earlier than version 8.3
accesscontrollist	acl	—
securitydomain	sdom	—
securityrole	srol	—

The **composer** program does not issue specific warnings if scheduling language keywords are used as names of scheduling objects. However, the use of such keywords can result in errors, so avoid using the keywords listed in Table 23: List of reserved words when defining jobs and job streams on page 198 when defining jobs and job streams:

Table 23. List of reserved words when defining jobs and job streams

Table 23. List of reserved words when defining jobs and job streams (continued)

List of reserved words:				
previous	priority	prompt	qualifier	rcconds ucc
recovery	relative	request	rerun	runcycle
sa	sameday	schedtime	schedule	scriptname
stop	streamlogon	su	succoutputcond	tasktype
timezone	to	token_in	until	validfrom
validto	vartable	vt	weekday(s)	workday(s)

Note: You cannot use the following keywords as <*property_name*> variables:

- RC
- returncode
- name
- status
- stdlist
- stduri

If you use these keywords to define custom output properties, they are not resolved and valued as expected by subsequent jobs. For example, if you specify `${job:PREVIOUSJOB.name}`, the successor job fails to resolve the variable.

Avoid using the keywords listed in Table 24: List of reserved words when defining workstations on page 199 when defining workstations, workstation classes, and domains:

Table 24. List of reserved words when defining workstations

List of reserved words:				
access	AIX	agent_type	autolink	behindfirewall
command	cpuclass	cpuname	description	domain

Table 24. List of reserved words when defining workstations (continued)

List of reserved words:				
enabled	end	extraneous	for	force
fta	fullstatus	host	hpux	ibm i
ignore	isdefault	linkto	maestro	manager
master	members	mpeix	mpev	mpexl
mpix	node	number	off	on
os	other	parent	posix	server
securea ddr	securityle vel	tcpaddr	timezone	type
tz	tzid	UNIX	using	vartable
wnt				

Avoid using the keywords listed in Table 25: List of reserved words when defining users on page 200 when defining users:

Table 25. List of reserved words when defining users

List of reserved words:		
username	passw ord	end

Using object definition templates

Scheduling object definition templates are available for your use in the `*TWS_home*\templates` directory. You can use the templates as a starting point when you define scheduling objects.

Note that the dates in the templates are in the format expressed in the `date format` local option.

Workstation definition

In an HCL Workload Automation network, a workstation is a scheduling object that runs jobs. You create and manage the scheduling object workstation in the HCL Workload Automation database by using a workstation definition. A workstation definition is required for every object that runs jobs. Typically, a workstation definition is used to represent a physical workstation but, in the case of extended agents for example, it represents a logical definition that must be hosted by a physical workstation.

The composer command cannot perform a full validation of the workstation definition. If you need a validation on the workstation, use the Dynamic Workload Console to create it.

Note: You can add workstation definitions to the database at any time, but you must run **JnextPlan -for 0000** again to be able to run jobs on newly created workstations. Every time you run **JnextPlan**, all workstations are stopped and restarted.

When creating a scheduling object, you can define it in a folder. If no folder path is specified, then the object definition is created in the current folder. By default, the current folder is the root (\) folder, but you can customize it to a different folder path. You can also use the composer rename command to move and rename objects in batch mode that use a naming convention to a different folder using part of the object name to assign a name to the object.

When the workstation or pool is renamed, the product creates a new name with a renamed value while preserving the old. The old name is no longer required. The workstation or pool can be deleted using the standard command.

When defining a new workstation, you can choose the licensing type to be applied to it. The licensing type determines the criteria by which the workload run by HCL Workload Automation on the workstation is accounted for and license consumption is evaluated. .

You can include multiple workstation definitions in the same text file, together with workstation class definitions and domain definitions. A workstation definition has the following syntax:

Syntax

cpuname *[folder/]workstation* [**description** "*description*"]
[**licensetype** *type*]
[**vartable** *table_name*]

os *os-type*
[**node** *hostname*] [**tcpaddr** *port*]
[**secureaddr** *port*] [**timezone**|**tz** *tzname*]
[**domain** *domainname*]
[**for maestro** [**host** *[folder/]workstation* [**access** *method* | **agentID** *agentID*]]
[**type fta** | **s-agent** | **x-agent** | **manager** | **broker** | **agent** | **rem-eng** |
pool | **d-pool**]
[**ignore**]
[**autolink on** | **off**]
[**behindfirewall on** | **off**]
[**securitylevel enabled** | **on** | **force** | **force_enabled**]
[**fullstatus on** | **off**]
[**server** *serverid*]]
[**protocol** `http | https`]
[**members** [*[folder/]workstation*] [...]]
[**requirements** *jsdl_definition*]]
end

[**cpuname** ...]

[**cpuclass** ...]

[**domain** ...]

Arguments

Table 26. Attribute settings for management workstation types

Attributes	Master domain manager	Domain manager	Backup domain manager
cpuname	The name of the workstation.		
description	A description for the workstation enclosed within double quotation marks. This attribute is optional.		
licensetype	Specifies the licensing type to be applied to the workstation.		
vartable	The name of a variable table associated with the workstation. Variables used with the workstation are defined in this table. This attribute is optional.		

os	The operating system installed on the system. Specify one of the following values:
	`UNIX`
	`WNT`
	`OTHER`
	`IBM_i`		

node	The system host name or IP address.		

Table 26. Attribute settings for management workstation types (continued)

Attributes	Master domain manager	Domain manager	Backup domain manager
tcpaddr	The value to be assigned to *nm port* in the `localopts` file. For multiple workstations on a system, enter an unused port number. The default value is 31111.		
secureaddr	The value to be assigned to *nm ssl port* in the `localopts` file. Specify it if **securitylevel** is set to `on`, `force`, `enabled`, or `force_enabled`.		
timezone | tz	The time zone in which the system is located. It is recommended that the value matches the value set on the operating system.		
domain	`MASTERDM`	The name of the managed domain.	
host	Not applicable		
access	Not applicable		
type	`manager`		`fta`
ignore	Use this attribute if you do not want this workstation to be included in the next production plan.		

autolink	It indicates if a link between workstations is automatically opened at startup time. Specify one of the following values:
	`ON`
	`OFF`
	This is an optional attribute. The default value is `ON`.		

behindfirewall	This setting is ignored.	It indicates if there is a firewall between the workstation and the master domain manager. Specify one of the following values:
		`ON`
		`OFF`
		The default value is `OFF`.	

securitylevel	The type of SSL authentication to use:
	`enabled`
	`on`
	`force`
	`force_enabled`		

fullstatus	`ON`		
server	Not applicable		This setting is ignored.
protocol	Not applicable		

Table 26. Attribute settings for management workstation types (continued)

Attributes	Master domain manager	Domain manager	Backup domain manager
members	Not applicable		
requirements	Not applicable		

Table 27: Attribute settings for target workstation types on page 204 describes the values that you set for each attribute for target workstation types. Following the table you find additional details about each attribute.

Table 27. Attribute settings for target workstation types

Attribute	Fault-tolerant agent and standard agent	Workload broker workstation	Extended agent	Agent	Remote engine workstation	Pool	Dynamic pool
cpuname	The name of the workstation.						
description	A description for the workstation enclosed within double quotation marks. This attribute is optional.						
licensetype	Specifies the licensing type to be applied to the workstation. This property applies to the following workstation types: • Fault-tolerant agent • Standard agent • Agent						
vartable	The name of a variable table associated with the workstation. Variables used with the workstation are defined in this table. This attribute is optional.						

os	The operating system installed on the system. Specify one of the following values:
	`UNIX`
	`WNT`
	`OTHER`
	`IBM_i`
	Specify `OTHER` for IBM i systems running as limited fault-tolerant agents.
		`OTHER`	The operating system installed on the machine. Specify one of the following values:
			`UNIX`
			`WNT`
			`OTHER`
			`IBM_i`
				This value setting is discovered on the system.	The operating system installed on the machine. Specify one of the following values:
					`UNIX`
					`WNT`
					`ZOS`
						The operating system installed on the machine. Specify one of the following values:
						`UNIX`
						`WNT`
						`OTHER`
						`IBM_i`	

node	The system host name or IP address.		The system host name or IP address. Specify `NULL` when **host** is set to `$MASTER`, or	Agent host name or IP address.	Remote engine host name or IP address.	Not applicable	

Table 27. Attribute settings for target workstation types (continued)

Attribute	Fault-tolerant agent and standard agent	Workload broker workstation	Extended agent	Agent	Remote engine workstation	Pool	Dynamic pool
			when defining an extended agent for PeopleSoft, SAP or Oracle.				
tcpaddr	The value to be assigned to *nm port* in the `localopts` file. When defining multiple workstations on a system, enter an unused port number. The default value is 31111. If you are upgrading a dynamic domain manager in SSL mode, ensure the **tcpaddr** and **secureaddr** attributes are set to the same value.	The value to be assigned to *nm port* in the `localopts` file. When defining multiple workstations on a system, enter an unused port number. The default value is 41114.	See the selected access method specifications.	The port number to communicate with the agent when the protocol is `http`.	The port number to communicate with the remote engine when the protocol is `http`.	Not applicable	
secureaddr	The value to be assigned to *nm ssl port* in the `localopts` file. Specify it if **securitylevel** is set to `on`, `force`, `enabled`.or `force_enabled`. If you are upgrading a dynamic domain manager in SSL mode, ensure the **tcpaddr** and **secureaddr** attributes are set to the same value.	Not Applicable	Not Applicable	The port number to communicate with the agent when the protocol is `https`.	The port number to communicate with the remote engine when the protocol is `https`.	Not applicable	
timezone | **tz**	The time zone in which the system is located. It is recommended that the value matches the value set on the operating system.		The time zone set on the workstation specified in the **host** attribute.	The time zone set on the agent.	The time zone set on the remote engine.	The time zone set on the pool agents.	The time zone set on the dynamic pool agents.
domain	Specify an existing domain. The default	Specify an existing domain.	This setting is needed only if the	Not applicable			

Table 27. Attribute settings for target workstation types (continued)

Attribute	Fault-tolerant agent and standard agent	Workload broker workstation	Extended agent	Agent	Remote engine workstation	Pool	Dynamic pool
	value for fault-tolerant agents is `MASTERDM`. This setting is mandatory for standard agents.	This setting is mandatory.	value assigned to **host** is `$MANAGER`.				
host	Not Applicable		The host workstation. It can be set to `$MASTER` or `$MANAGER`.	The broker workstation.			
access	Not Applicable			Select the appropriate access method file name.	Not Applicable		
agentID				The unique identifier of the agent			
type	`fta` `s-agent` The default value is `fta`. Specify `fta` for IBM isystems running as limited fault-tolerant agents.	`broker`	`x-agent`	`agent`	`rem-eng`	`pool`	`d-pool`
ignore	Use this attribute if you do not want this workstation to appear in the next production plan.						
autolink	It indicates if a link between workstations is automatically opened at startup. Specify one of the following values: `ON` `OFF` This is an optional attribute. The default value is `ON`.		`OFF`	Not applicable			
behindfirewall	It indicates if there is a firewall between the workstation and the master domain manager. Specify one of the following values:		`OFF`	Not applicable			

Table 27. Attribute settings for target workstation types (continued)

Attribute	Fault-tolerant agent and standard agent	Workload broker workstation	Extended agent	Agent	Remote engine workstation	Pool	Dynamic pool

	`ON`
	`OFF`
	The default value is `OFF`.						

securitylevel	The type of SSL authentication to use:
	`enabled`
	`on`
	`force` `force_enabled`
	Not applicable for IBM i systems running as limited fault-tolerant agents.
		Not Applicable					

fullstatus	It indicates if the workstation is updated about job processing status in its domain and subdomains. Specify one of the following values:
	`ON`
	`OFF`
	Specify `OFF` for standard agents.
		`OFF`		Not applicable			

server	0-9, A-Z. When specified, it requires the creation of a dedicated mailman processes on the parent workstation.		Not Applicable				

protocol	Not applicable			Specify one of the following values:
				`http`
				`https`
				This attribute is optional. When not specified, it is automatically determined from the settings specified for **tcpaddr** and **secureaddr**.
						Not applicable	

members	Not applicable					Required value	Not applicable

Table 27. Attribute settings for target workstation types (continued)

Attribute	Fault-tolerant agent and standard agent	Workload broker workstation	Extended agent	Agent	Remote engine workstation	Pool	Dynamic pool
requirements	Not applicable						Required value

The following list gives additional details for the workstation definition attributes:

cpuname *[folder/]workstation*

Specifies the name of the folder within which the workstation is defined, if any, and the workstation name. Workstation names must be unique and cannot be the same as workstation class names.

The name must start with a letter, and can contain alphanumeric characters, dashes, and underscores. It can contain up to 16 characters.

Do not use in this field any of the reserved words specified in Table 23: List of reserved words when defining jobs and job streams on page 198.

Note: Renaming any kind of domain manager, such as master domain manager, dynamic domain manager, or their backups is not a supported operation and might lead to unpredictable results.

description *"description"*

Provides a description of the workstation. The description can contain up to 120 alphanumeric characters. The text must be enclosed within double quotation marks.

licensetype *type*

Specifies the licensing type to be applied to the workstation. Use this keyword when you set the **licenseType** optman keyword to **byWorkstation**. The licensing type determines the criteria by which the workload run by HCL Workload Automation on the workstation is accounted for and license consumption is evaluated.. Supported values are as follows:

perJob

Your license consumption is evaluated based on the number of jobs that run on the workstation each month.

perServer

Your license consumption is evaluated based on the number of chargeable HCL Workload Automation components installed on the workstation.

If you install a dynamic agent and a fault-tolerant agent on the same workstation in the same path, the agents share the same license file. As a result, only the license type which is defined later in time is applied. For this reason, it is advisable to define the same license type for both agent workstations.

vartable ***table_name***

Specifies the name of the variable table that you want to associate with the workstation. Variables used with the workstation are defined in this table.

The name must start with a letter, and can contain alphanumeric characters, dashes, and underscores. It can contain up to 80 characters.

os ***os_type***

Specifies the operating system of the workstation. When used in remote engine workstation definitions it represents the operating system of the HCL Workload Automation remote engine.

Valid values are:

UNIX

For supported operating systems running on UNIX-based systems, including LINUX systems.

WNT

For supported Windows operating systems.

OTHER

Mandatory value for: dynamic workload broker workstations, and IBM i systems running as limited fault-tolerant agents. Optional value for other types of workstations.

ZOS

Used with remote engine workstations that are defined to communicate with HCL Workload Automation for Z remote engine.

IBM_i

For supported IBM i operating systems.

Note: For an up-to-date list of supported operating systems, see HCL Workload Automation Detailed System Requirements.

node *hostname*

Specify the host name or the TCP/IP address of the workstation. Fully-qualified domain names are accepted.

For host names, valid characters are alphanumeric, including dash (-). The maximum length is 51 characters.

Specify `NULL` when:

- Defining an extended agent for PeopleSoft, SAP or Oracle.
- **host** is set to `$MASTER`

If you are defining a remote engine workstation, specify the host name of the system where the remote engine is installed.

tcpaddr *port*

Specifies the **netman** TCP/IP port number that HCL Workload Automation uses for communicating between wo rkstations.

For workload broker workstations

Specify the value of the **TWS.Agent.Port** property of the `TWSAgentConfig.properties` file.

For remote engine workstations using HTTP protocol to communicate with the remote engine

Specify the HTTP port number of the remote engine.

For other types of workstations

Specify the value assigned in the `localopts` file for variable *nm port*.

The default value for this field is `31111`. Specify a value in the range from `1` to `65535`.

secureaddr

Defines the port used to listen for incoming SSL connections. This value is read when the **securitylevel** attribute is set.

For workload broker workstations

Ignore this attribute.

For remote engine workstations using HTTPS protocol to communicate with the remote engine

Specify the HTTPS port number of the remote engine.

For other types of workstations

Specify the value assigned in the `localopts` file for variable *nm ssl port*. The value must be different value from the value assigned to *nm port* variable in the `localopts` file.

If **securitylevel** is specified, but this attribute is missing, the default value for this field is `31113`. Specify a value in the range from `1` to `65535`.

See the *Administration Guide* for information about SSL authentication and local options set in the *`TWS_home`*`/localopts` configuration file.

timezone|**tz** ***tzname***

Specifies the time zone of the workstation. To ensure the accuracy of scheduling times, this time zone must be the same as the computer operating system time zone.

When used in remote engine workstation definitions it represents the time zone set on the HCL Workload Automation remote engine.

See Managing time zones on page 1064 for valid time zone names.

domain ***domainname***

Specifies the name of HCL Workload Automation domain the workstation belongs to. The default value for fault-tolerant workstation is `MASTERDM`.

HCL Workload Automation ignores domain setting when defined for extended agents, except when the **host** attribute is set to `$MASTER`.

This setting is mandatory for standard agent and dynamic workload broker workstations.

host ***host-workstation***

This attribute is mandatory for extended agents and remote engine workstations and specifies:

For remote engine workstations, agents, pools and dynamic pools:

The broker workstation hosting the workstation. This field cannot be updated after the remote engine workstation creation.

For extended agents

The workstation with which the extended agent communicates and where its access method is installed. The hosting workstation cannot be another extended agent.

If the hosting workstation is a domain manager for which you have defined a backup, you can specify one of the following values to ensure the extended agent is not isolated if the hosting workstation becomes unavailable:

$MASTER

To indicate that the host workstation for the extended agent is the master domain manager.

$MANAGER

To indicate that the host workstation for the extended agent is the domain manager. In this case you must specify the domain where the agent is located.

In this case make sure that the extended agent methods are installed also on the backup workstation. You can enable and disable the automatic resolution of the `$MASTER` key using the *mm resolve master* option in the `localopts` file.

For more information about the options available in the `localopts` file, see *HCL Workload Automation Administration Guide*.

access *method*

Specifies an access method for extended and network agents. It corresponds to the name of a file that is located in the *`TWS_home`*`/methods` directory on the hosting workstation.

Specify `NULL` when defining an extended agent for PeopleSoft, SAP, or Oracle.

agentID *agentID*

The unique identifier of the agent.

type

Specifies the type of the workstation. If you plan to change the workstation types, consider the following rules:

- You can change fault-tolerant agent, standard agent, extended agent, domain manager and dynamic workload broker workstations to any workstation type, with the exception of dynamic agent, pool, dynamic pool, and remote engine.
- You cannot change the type of dynamic agent, pool, dynamic pool, and remote engine.

Enter one of the following values:

fta

If you define a *fault-tolerant agent*, that is an agent workstation that launches jobs and resolves local dependencies without a domain manager. This is the default value for this attribute.

You must specify `fta` if you want to assign the workstation the role of backup domain manager or backup master domain manager.

Specify `fta` for IBM isystems running as limited fault-tolerant agents.

s-agent

If you define a *standard agent*, that is an agent workstation that launches jobs only under the direction of its domain manager.

x-agent

If you define an *extended agent*, that is an agent workstation that launches jobs only under the direction of its hosting workstation. Extended agents can be used to interface HCL Workload Automation with non-HCL systems and applications. To create an extended agent for SAP, define an **r3batch** access method. Consider the following example:

```
CPUNAME XA_R3
  OS OTHER
  NODE na TCPADDR 32111
  FOR MAESTRO HOST LONDON ACCESS "r3batch"
    TYPE X-AGENT
    AUTOLINK OFF
    BEHINDFIREWALL OFF
    FULLSTATUS OFF
END
```

For more information, see the topic about integrating with SAP and PeopleSoft in *Administration Guide*.

manager

If you define a *domain manager*, that is a workstation that manages a domain. When defining this type of workstation, specify:

Server

NULL

Domain

The name of the domain the workstation manages, if different from `MASTERDM` domain.

You specify that a workstation is a manager also in the *manager* field of the Domain definition on page 227. HCL Workload Automation automatically checks that the values specified in these fields are consistent.

broker

If you define a *dynamic workload broker* workstation, that is a workstation that runs both existing job types and job types with advanced options. It is the broker server installed with the master domain manager and the dynamic domain manager. It hosts the following workstations:

- Extended agent
- Remote engine

- Pool
- Dynamic pool
- Agent. This definition includes the following agents:
  - agent
  - HCL Workload Automation Agent
  - agent for z/OS

  For more information about the agent for z/OS, see *HCL Workload Automation: Scheduling with the Agent for z/OS*.

agent

If you define a *dynamic agent*, that it is a workstation that manages a wide variety of job types, for example, specific database or file transfer jobs, in addition to traditional job types. It is hosted by the workload broker workstation. The workstation definition is automatically created and registered when you install the dynamic agent component. In its definition you can edit only the following attributes:

- Description
- Vartable

Note: If you have the `enAddWorkstation` global option set to `"yes"`, the dynamic agent workstation definition is automatically added to the Plan after the installation process creates the dynamic agent workstation in the database.

rem-eng

If you define a *remote engine workstation*, that it is a workstation used to communicate with a remote engine when binding a locally defined job, named *shadow job*, to a specific job running on the remote engine, named *remote job*. When the two jobs are bound, the shadow job status

transition maps the remote job status transition. This mapping is useful also to define and monitor dependencies of local jobs on jobs running on the remote engine; such dependencies are called *cross dependencies*.

For more information about shadow jobs and cross dependencies, see Defining and managing cross dependencies on page 1352.

When defining this type of workstation, specify the following fields:

os

The operating system of the remote engine.

host

The name of the hosting broker workstation.

node

The hostname or the IP address of the remote engine.

When specifying the port number to use to communicate with the remote engine, use **secureaddr** if the protocol used is HTTPS, **tcpaddr** if the protocol used is HTTP. It is recommended that you specify in the **timezone** field the time zone set on the remote engine.

pool

If you define a *pool*, that is a set of dynamic agents with similar hardware or software characteristics to submit jobs to. This workstation is hosted by the workload broker workstation. In its definition you can edit only the following attributes:

- Description
- Vartable
- Members

d-pool

If you define a *dynamic pool*, that is a set of dynamic agents which is dynamically defined based on the requirements listed in the JSDL file specified in the **resources** attribute. This workstation is hosted by the workload broker workstation. In its definition you can edit only the following attributes:

- Description
- Vartable
- Requirements

ignore

Specifies that the workstation definition must not be added to the production plan. If you specify this setting, job streams scheduled to run on this workstation are not added to the production plan.

autolink

Specifies whether to open the link between workstations at startup. Depending on the type of the workstation, when you set its value to `on`:

On a fault-tolerant agent or on a standard agents

It means that the domain manager open the link to the agent when the domain manager is started.

On a domain manager

It means that its agents open links to the domain manager when they are started.

This setting is particularly useful when a new production plan is created on the master domain manager: As part of the production plan generation all workstations are stopped and then restarted. For each agent with **autolink** turned on, the domain manager automatically sends a copy of the new production plan and then starts the agent. If **autolink** is turned on also for the domain manager, the agent opens a link back to the domain manager.

If the value of **autolink** is `off` for an agent, you can open the link from its domain manager by running the **conman link** command on the agent's domain manager or the master domain manager.

behindfirewall

If set to `on`, it means there is a firewall between the workstation and the master domain manager. In this case only a direct connection between the workstation and its domain manager is allowed and the **start** *workstation*,

stop *workstation*, and **showjobs** commands are sent following the domain hierarchy, instead of making the master domain manager or the domain manager open a direct connection with the workstation.

Set this attribute to `off` if you are defining a workstation with type `broker`.

fullstatus

Specify this setting when defining a fault-tolerant agent workstation. For domain managers this keyword is automatically set to `on`. Specify one of the following values:

on

If you want the fault-tolerant agent workstation to operate in *full status* mode, meaning that the workstation is updated with the status of jobs and job streams running on all other workstations in its domain and subordinate domains, but not on peer or parent domains. The copy of the production plan on the agent is kept at the same level of detail as the copy of the production plan on its domain manager.

off

If you want the fault-tolerant agent workstation to be informed about the status of jobs and job streams only on other workstations that affect its own jobs and job streams. Specifying "off" can improve performance by reducing network activity.

securitylevel

Specifies the type of SSL authentication for the workstation. Do not specify this attribute for a workstation with type `broker`. It can have one of the following values:

enabled

The workstation uses SSL authentication only if its domain manager workstation or another fault-tolerant agent below it in the domain hierarchy requires it.

on

The workstation uses SSL authentication when it connects with its domain manager. The domain manager uses SSL authentication when it connects to its parent domain manager. The fault-tolerant agent refuses any incoming connection from its domain manager if it is not an SSL connection.

force

The workstation uses SSL authentication for all of its connections and accepts connections from both parent and subordinate domain managers.

force_enabled

The workstation uses SSL authentication for all of its connections to all target workstations which are set to this value. The workstation tries to establish a connection in FULLSSL mode and, if the attempt fails, it tries to establish an unsecure connection. If you plan to set different security levels between master domain manager and fault-tolerant agents, ensure all these components are at version 95 Fix Pack 4 or later. For versions earlier than 95 Fix Pack 4, the same security level is required for master domain manager and fault-tolerant agents.

If this attribute is omitted, the workstation is not configured for SSL connections and any value for **secureaddr** is ignored. Make sure, in this case, that the *nm ssl port* local option is set to 0 to ensure that **netman** process does not try to open the port specified in **secureaddr**. See the *Administration Guide* for information about SSL authentication and local options set in the `TWS_home/localopts` configuration file.

The following table describes the type of communication used for each type of **securitylevel** setting.

Table 28. Type of communication depending on the security level value

Value set on the Fault-tolerant Agent (or the Domain Manager)	Value set on its Domain Manager (or on its Parent Domain Manager)	Type of connection established
Not specified	Not specified	TCP/IP
Enabled	Not specified	TCP/IP
On	Not specified	No connection
Force	Not specified	No connection
Not specified	On	TCP/IP
Enabled	On	TCP/IP
On	On	SSL
Force	On	SSL
Not specified	Enabled	TCP/IP
Enabled	Enabled	TCP/IP
On	Enabled	SSL
Force	Enabled	SSL
Not specified	Force	No connection
Enabled	Force	SSL

Table 28. Type of communication depending on the security level value (continued)

Value set on the Fault-tolerant Agent (or the Domain Manager)	Value set on its Domain Manager (or on its Parent Domain Manager)	Type of connection established
On	Force	SSL
Force	Force	SSL
force_enabled	force_enabled	SSL

The value for **securitylevel** is not specified for dynamic workload broker workstations.

server ***ServerID***

Use the **server** attribute in the fault-tolerant agent workstation definition to reduce the time required to initialize agents and to improve the timeliness of messages. By default, communications with the fault-tolerant agent are handled by a **mailman** process running on the domain manager. The **server** attribute allows you to start a **mailman** process on the domain manager to handle communications with this fault-tolerant agent workstation only.

If you are defining a fault-tolerant agent that can work as a backup domain manager, the *ServerID* is used only when the workstation works as a fault-tolerant agent; the setting is ignored when the workstation works as a backup domain manager.

Within the *ServerID*, the ID is a single letter or a number (A-Z and 0-9). The IDs are unique to each domain manager, so you can use the same IDs in other domains without conflict. A specific *ServerID* can be dedicated to more than one fault-tolerant agent workstation.

As best practices:

- If more than 36 server IDs are required in a domain, consider to splitting the domain into two or more domains.
- If the same ID is used for multiple agents, a single server is created to handle their communications. Define extra servers to prevent a single server from handling more than eight agents.

If a *ServerID* is not specified, communications with the agent are handled by the main **mailman** process on the domain manager.

protocol `http | https`

Specifies the protocol to use to communicate with:

The broker workstation

If the workstation is an agent workstation.

The remote engine

If the workstation is a remote engine workstation.

members ***[workstation] [...]***

Use this value for a pool workstation to specify the agents that you want to add to the pool.

requirements ***jsdl_definition***

Use this value for a dynamic pool workstation to specify the requirements, in JSDL format; the agents that satisfy the requirement automatically belong to the dynamic pool. You use the following syntax:

```
<jsdl_definition:>
<jsdl:resources>
<jsdl:logicalResource subType="MyResourceType"/>
</jsdl:resources>
```

Example

Examples

The following example creates a master domain manager named `hdq-1`, and a fault-tolerant agent named `hdq-2` in the master domain. Note that a **domain** argument is optional in this example, because the master domain defaults to **masterdm**.

```
cpuname hdq-1 description “Headquarters master DM”
     os unix
     tz America/Los_Angeles
     node sultan.ibm.com
     domain masterdm
     for maestro type manager
          autolink on
          fullstatus on
end
```

```
cpuname hdq-2
     os wnt
     tz America/Los_Angeles
     node opera.ibm.com
     domain masterdm
     for maestro type fta
```

```
          autolink on
end
```

The following example creates a domain named `distr-a` with a domain manager named `distr-a1` and a standard agent named `distr-a2`:

```
domain distr-a
     manager distr-a1
     parent masterdm
end
```

```
cpuname distr-a1 description “District A domain mgr”
     os wnt
     tz America/New_York
     node pewter.ibm.com
     domain distr-a
     for maestro type manager
          autolink on
          fullstatus on
end
```

```
cpuname distr-a2
     os wnt
     node quatro.ibm.com
     tz America/New_York
     domain distr-a
     for maestro type s-agent
end
```

The following example creates a fault-tolerant workstation with SSL authentication. The **securitylevel** security definition specifies that the connection between the workstation and its domain manager can be only of the SSL type. Port 32222 is reserved for listening for incoming SSL connections.

```
cpuname ENNETI3
     os WNT
     node apollo
     tcpaddr 30112
     secureaddr 32222
     for maestro
          autolink off
          fullstatus on
          securitylevel on
end
```

The following example creates a broker workstation. This workstation manages the lifecycle of HCL Workload Automation workload broker type jobs in dynamic workload broker.

```
cpuname ITDWBAGENT
       vartable TABLE1
       os OTHER
      node itdwbtst11.ibm.com TCPADDR 41114
      timezone Europe/Rome
      domain MASTERDM
      for MAESTRO
          type  BROKER
          autolink  OFF
          behindfirewall OFF
```

```
          fullstatus OFF
end
```

The following example creates a remote engine workstation to use to manage cross dependencies and communicate with a remote engine installed on a system with hostname `London-hdq` using the default HTTPS port 31116. The remote engine workstation is hosted by the broker workstation ITDWBAGENT

```
cpuname REW_London
     description "Remote engine workstation to communicate with London-hdq"
     os WNT
     node London-hdq secureaddr 31116
     timezone Europe/London
     for maestro host ITDWBAGENT
        type rem-eng
        protocol HTTPS
end
```

The following example shows how to create a dynamic pool of agents. All agents in the dynamic pool must have the Linux operating system installed:

```
CPUNAME DPOOLUNIX
  DESCRIPTION "Sample Dynamic Pool Workstation"
  VARTABLE table1
  OS OTHER
  TIMEZONE Europe/Rome
  FOR MAESTRO HOST MAS86MAS_DWB
    TYPE D-POOL
  REQUIREMENTS
    <?xml version="1.0" encoding="UTF-8"?>
<jsdl:resourceRequirements
      xmlns:jsdl="http://www.ibm.com/xmlns/prod/scheduling/1.0/jsdl">
  <jsdl:resources>
    <jsdl:candidateOperatingSystems>
      <jsdl:operatingSystem type="LINUX"/>
    </jsdl:candidateOperatingSystems>
  </jsdl:resources>
</jsdl:resourceRequirements>
END
```

The following example shows how to create a dynamic pool of agents. All agents in the dynamic pool must have the Windows 2019 operating system installed:

```
CPUNAME DPOOLWIN
  DESCRIPTION "Sample Dynamic Pool Workstation"
  OS WNT
  TIMEZONE Europe/Rome
  FOR MAESTRO HOST MAS86MAS_DWB
    TYPE D-POOL
  REQUIREMENTS
 <?xml version="1.0" encoding="UTF-8"?>
<jsdl:resourceRequirements xmlns:jsdl="http://www.ibm.com/xmlns/prod/scheduling/1.0/jsdl">>
  <jsdl:resources>
    <jsdl:candidateOperatingSystems>
      <jsdl:operatingSystem type="Windows 2019"/>
    </jsdl:candidateOperatingSystems>
  </jsdl:resources>
```

```
</jsdl:resourceRequirements>
END
```

The following example shows how to create a pool of agents with name POOLUNIX and containing two agents: NC121105 and NC117248:

```
CPUNAME POOLUNIX
  DESCRIPTION "Sample Pool Workstation"
  OS OTHER
  TIMEZONE Europe/Rome
  FOR MAESTRO HOST MAS86MAS_DWB
    TYPE POOL
  MEMBERS
    NC121105
    NC117248
END
```

The following example shows how to create a standard agent on which license consumption is evaluated based on the number of jobs that run on the workstation each month:

```
CPUNAME SA1
  DESCRIPTION "Sample Standard Agent"
  LICENSETYPE PERJOB
  VARTABLE TABLE1
  OS UNIX
  NODE sa1.mycompany.com TCPADDR 31111
  SECUREADDR 31113
  TIMEZONE Europe/Rome
  DOMAIN MASTERDM
  FOR MAESTRO
    TYPE S-AGENT
    IGNORE
    AUTOLINK ON
    BEHINDFIREWALL ON
    SECURITYLEVEL FORCE
    FULLSTATUS OFF
    SERVER B
END
```

Workstation class definition

A workstation class is a group of workstations for which common jobs and job streams can be written. You can include multiple workstation class definitions in the same text file, along with workstation definitions and domain definitions.

When defining workstation classes, ensure that the workstations in the class support the job types you plan to run on them. The following rules apply:

- Job types with advanced options run only on dynamic agents, pools, and dynamic pools.
- Shadow jobs run only on remote engines.

When creating a scheduling object, you can define it in a folder. If no folder path is specified, then the object definition is created in the current folder. By default, the current folder is the root (\) folder, but you can customize it to a different folder path. You can also use the composer rename command to move and rename objects in batch mode that use a naming convention to a different folder using part of the object name to assign a name to the object.

Note: When defining a workstation class containing fault-tolerant agents at versions earlier than 9.3 Fix Pack 1, the following problems might be encountered: scheduled objects on the fault-tolerant agents are not correctly managed, the statuses of jobs and job streams are not reported consistently, and the number and statuses of unsatisfied dependencies related to conditional dependencies are reported incorrectly.

Each workstation class definition has the following format and arguments:

Syntax

cpuclass *[folder/]workstationclass*
  [**description** "*description*"]
  [**ignore**]
  members [*[folder/]workstation* | **@**] [...]
  end
[**cpuname** ...]
[**cpuclass** ...]
[**domain** ...]

Arguments

cpuclass *[folder/]workstationclass*

Specifies the name of the folder within which the workstation class is defined, if any, and the name of the workstation class. The name must start with a letter, and can contain alphanumeric characters, dashes, and underscores. It can contain up to 16 characters.

Note: You cannot use the same names for workstations and workstation classes.

description *"description"*

Provides a description of the workstation class. The description can contain up to 120 alphanumeric characters. The text must be enclosed within double quotes.

ignore

Specifies that HCL Workload Automation must ignore all workstations included in this workstation class when generating the production plan.

members ***[folder/]workstation***

Specifies a list of workstation names, preceded by the folder name within which the workstation class is defined, if any, separated by spaces, that are members of the class. The @ wildcard character means that the workstation class includes all workstations.

With the introduction of folders in which to define workstation classes, a change has taken place with respect to the use of wildcards in specifying workstation classes. While in the previous releases, wildcards applied to all workstation classes matching the specified requirements, starting from version 9.5, Fix Pack 2, wildcards apply to all workstation classes matching the specified requirements and stored in the same folder as the specified workstation class and all its subfolders.

To create or modify workstation classes, add the USE access method to CPU objects that are, or will be, members of a workstation class.

Example

Examples

The following example defines a workstation class named `backup`:

```
cpuclass backup
     members
          main
          site1
          site2
end
```

The following example defines a workstation class named `allcpus` that contains every workstation:

```
cpuclass allcpus
     members
          @
end
```

See also

For more information about how to perform the same task from the Dynamic Workload Console, see:

the Dynamic Workload Console User's Guide, "Designing your Workload" section.

Domain definition

A domain is a group of workstations consisting of one or more agents and a domain manager. The domain manager acts as the management hub for the agents in the domain. You can include multiple domain definitions in the same text file, along with workstation definitions and workstation class definitions. Each domain definition has the following format and arguments:

Syntax

domain *domainname*[**description** "*description*"]
  *** manager** *[folder/]workstationname*
  [**parent** *domainname* | **ismaster**]
end

[**cpuname** ...]

[**cpuclass** ...]

[**domain** ...]

Arguments

domain *domainname*

The name of the domain. It must start with a letter, and can contain alphanumeric characters, dashes, and underscores. It can contain up to 16 characters.

description "*description*"

Provides a description of the domain. The text must be enclosed within double quotation marks.

*** manager *[folder/]workstation***

This is a commented field used only to show, when displaying the domain definition, the name of the workstation that has the role of domain manager for that domain. Make sure this field remains commented. It is kept for compatibility with earlier versions. With IBM Workload Scheduler version 8.3, the information about whether a workstation is a domain manager is set in the **type** field in the Workstation definition on page 201.

parent *domainname*

The name of the parent domain to which the domain manager is linked. The default is the master domain, which does not require a domain definition. The master domain is defined by the global options **master** and **master domain**.

ismaster

If specified, indicates that the domain is the top domain of the HCL Workload Automation network. If set it cannot be removed later.

Example

Examples

The following example defines a domain named `east`, with the master domain as its parent, and two subordinate domains named `northeast` and `southeast`:

```
domain east
     description “The Eastern domain?
     * manager cyclops
end
domain northeast
     description “The Northeastern domain?
     * manager boxcar
     parent east
end
domain southeast
     description “The Southeastern domain?
     * manager sedan
     parent east
end
```

Job definition

A job is an executable file, program, or command that is scheduled and launched by HCL Workload Automation. You can write job definitions in edit files and then add them to the HCL Workload Automation database with the **composer** program. You can include multiple job definitions in a single edit file.

When creating a job, you can define it in a folder. If no folder path is specified, then the job definition is created in the current folder. By default, the current folder is the root (\) folder, but you can customize it to a different folder path. You can also use the composer rename command to move and rename jobs in batch mode that use a naming convention to a different folder using part of the job name to name the folder.

Two different job types are available: the standard HCL Workload Automation job is a generic executable file, program, or command that you can run statically, while the job types with advanced options are predefined jobs that you can use to run specific tasks, either statically or dynamically, such as file transfer operations or integration with other databases.

The job types with advanced options run only on dynamic agents, pools, dynamic pools, and remote engines.

To define standard jobs in the **composer** command line, you use the **script** and **docommand** arguments; to define job types with advanced options, you use the **task** argument.

For more information about job types with advanced options, see Extending HCL Workload Automation capabilities on page 910.

For information about how to pass variables between jobs in the same job stream instance, see Passing variables between jobs on page 269.

Note: The composer command line to create job definitions uses REST APIs. This means that when you create a job using composer, new APIs are used, which are not compatible with the APIs installed on masters with previous product versions. As a result, you cannot use a composer at version 9.4, Fix Pack1 level, to create a job definition on a master where a previous version of the product is installed.

Each job definition has the following format and arguments:

Syntax

$jobs
[*[folder/]workstation*#][*folder/]jobname*
  {**scriptname** *filename* **streamlogon** *username* |
   docommand "*command*" **streamlogon** *username* |
   task *job_definition* }
  [**description** "*description*"]
  [**tasktype** *tasktype*]
  [**interactive**]

  [**succoutputcond** *Condition_Name* "*Condition_Value*"]
  [**outputcond** *Condition_Name* "*Condition_Value*"]

[**recovery**
{**stop**
[**after** [*[folder/]workstation#*][*folder/]jobname*]
[**abendprompt** "*text*"]]
|**continue**
[**after** [*[folder/]workstation#*][*folder/]jobname*]
[**abendprompt** "*text*"]]
|**rerun [same_workstation]**
[[**repeatevery** *hhmm*] [**for** *number* **attempts**]]
[**after** [*[folder/]workstation#*][*folder/]jobname*]
|[**after** [*[folder/]workstation#*][*folder/]jobname*]
[**abendprompt** "*text*"]}

A job itself has no settings for dependencies, these must be added to the job when it is included in a job stream definition.

You can add or modify job definitions from within job stream definitions. Modifications to jobs definitions made in job streams definitions are reflected in the job definitions stored in the database. This means that if you modify the job definition of `job1` in job stream definition `js1` and `job1` is used also in job stream `js2`, also the definition of `job1` in `js2` definition is modified accordingly.

Note: Wrongly typed keywords used in job definitions lead to truncated job definitions stored in the database. In fact the wrong keyword is considered extraneous to the job definition and so it is interpreted as the job name of an additional job definition. Usually this misinterpretation causes also a syntax error or a non-existent job definition error for the additional job definition.

Special attention is required in the case where an alias has been assigned to a job. You can decide to use a different name to refer to a particular job instance within a job stream, but the alias name must not conflict with the job name of another job in the same job stream. If a job definition is renamed then jobs having the same name as the job definition modify the name in accordance with the job definition name. Here are some examples to understand the behavior of jobs when the job definition name is modified:

Table 29. Examples: renaming the job definition

Original job definition names in job stream	Rename job definition	Outcome
SCHEDULE *[folder/]*WKS#/APPS/DEV/JS : *[folder/]*FTA1#/APPS/DEV1/**A** *[folder/]*FTA1#/APPS/DEV1/**B as C** END	Rename job **A** to **D**	SCHEDULE *[folder/]*WKS#/APPS/DEV/JS : *[folder/]*FTA1#/APPS/DEV1/**D** *[folder/]*FTA1#/APPS/DEV1/**B as C** END
	Rename job **B** to **D**	SCHEDULE WKS#JS :

Table 29. Examples: renaming the job definition (continued)

Original job definition names in job stream	Rename job definition	Outcome
		FTA1#/APPS/DEV1/**A** FTA1#/APPS/DEV1/**D as C** END
	Rename job / APPS/DEV1/**A** to **C**	An error occurs when renaming job **A** to **C** because job C already exists as the alias for job **B**.

Note: Because a job in a job stream is identified only by its name, jobs with the same name require an alias even if their definitions have different workstations or folders.

Refer to section Job stream definition on page 314 for information on how to write job stream definitions.

Arguments

[folder/]workstation#

Specifies the name of the workstation or workstation class on which the job runs. The default is the workstation specified for *defaultws* when starting the **composer** session.

For more information on how to start a composer session refer to Running the composer program on page 437.

The pound sign (**#**) is a required delimiter. If you specify a workstation class, it must match the workstation class of any job stream in which the job is included.

If you are defining a job that manages a workload broker job, specify the name of the workstation where the workload broker workstation is installed. Using the workload broker workstation, HCL Workload Automation can submit job in the dynamic workload broker environment using the dynamic job submission.

[*folder/]jobname*

Specifies the name of the folder within which the job is defined and the job name. The job name must start with a letter, and can contain alphanumeric characters, dashes, and underscores. It can contain up to 40 characters. If you generally work from a precise folder, then you can use the chfolder command to navigate to folders and sub-folders. The chfolder command changes the working directory or current folder, which is set to root ("/") by default, so that you can use relative folder paths when submitting commands. If no folder path is specified, then the job definition is created in the current folder. If a relative path is specified, the path is relative to the current folder. See CHFOLDER on page 457 for more information about changing folder paths. See Folder definition on page 287 for specifications about folder names.

scriptname *filename*

Specifies the name of the file the job runs. Use **scriptname** for UNIX® and Windows® jobs. For an executable file, enter the file name and any options and arguments. The length of *filename* plus the length of *Success Condition* (of the **rccondsucc** keyword) must not exceed 4095 characters. You can also use HCL Workload Automation parameters.

docommand *command*

Specifies a command that the job runs. Enter a valid command and any options and arguments enclosed in double quotation marks ("). The length of *command* plus the length of *Success Condition* (of the **rccondsucc** keyword) must not exceed 4095 characters. You can also enter HCL Workload Automation parameters.

Use this argument to define standard HCL Workload Automation jobs.

The job `abends` if the file specified with the **docommand** option is not found or does not have execute permission.

See Using variables and parameters in job definitions on page 243 for more information.

task *job_definition*

Specifies the XML syntax for job types with advanced options and shadow jobs. The maximum length is 4095 characters.

To define standard job types, use the **docommand** or the **scriptname** arguments.

This argument applies only to workstations of the following types:

- agent
- pool
- d-pool
- rem-eng

The syntax of the job depends on the job type you define.

streamlogon *username*

The user name under which the job runs. This attribute is mandatory when **scriptname** or **docommand** are specified. The name can contain up to 47 characters. If the name contains special characters it must be enclosed in double quotation marks ("). Specify a user that can log on to the workstation on which the job runs. You can also enter HCL Workload Automation parameters.

See Using variables and parameters in job definitions on page 243 for more information.

For Windows® jobs, the user must also have a user definition.

See User definition on page 281 for user requirements.

If you are defining a job that manages a dynamic workload broker job, specify the name of the user you used to install dynamic workload broker.

The job `fails` if the user specified in the **streamlogon** option does not exist.

description "*description*"

Provides a description of the job. The text must be enclosed between double quotation marks.The maximum number of bytes allowed is 120.

tasktype *tasktype*

Specifies the job type. It can have one of the following values:

UNIX®

For jobs that run on UNIX® platforms.

WINDOWS

For jobs that run on Windows® operating systems.

OTHER

For jobs that run on extended agents. Refer to *HCL Workload Automation for Applications: User's Guide* for information about customized task types for supported vendor acquired applications.

BROKER

For jobs that manage the lifecycle of a dynamic workload broker job. Refer to *HCL Workload Automation: Scheduling Workload Dynamically* for information about how to use dynamic workload broker.

interactive

If you are defining a job that manages a dynamic workload broker job ignore this argument. Specifies that the job runs interactively on your desktop. This feature is available only on Windows® environments.

succoutputcond *Condition_Name* "*Condition_Value*"

A condition that when satisfied qualifies a job as having completed successfully and the job is set to the SUCC status. The condition is used when you need a successor job to start only after the successful completion of the predecessor job or job stream. They can also be used to specify alternative flows in a job stream starting from a predecessor job or job stream. The successor job is determined by which conditions the predecessor job or job stream satisfies.

When the predecessor is a job stream, the conditional dependency is only a status condition, as follows: `abend`, `succ`, and `suppr`. The successor job runs when the predecessor job stream status satisfies the job status specified using these arguments. You can specify one status, a combination of statuses, or all statuses. When specifying more than one status or condition name, separate the statuses or names by using the pipe (|) symbol.

You can specify any number of successful output conditions. The condition can be expressed as follows:

A return code

On fault-tolerant and dynamic agent workstations only, you can assign which return code signifies the successful completion of a job. Job return codes can be expressed in various ways:

Comparison expression

The syntax to use to specify a job return code: The syntax is:

```
(RC operator operand)
```

RC

The RC keyword.

operator

Comparison operator. It can have the following values:

Table 30. Comparison operators

Example	Operator	Description
RC<*value*	<	Less than
RC<=*value*	<=	Less than or equal to
RC>*value*	>	Greater than

Table 30. Comparison operators (continued)

Example	Operator	Description
RC>=*value*	>=	Greater than or equal to
RC=*value*	=	Equal to
RC!=*value*	!=	Not equal to
RC<>*value*	<>	Not equal to

operand

An integer between -2147483647 and 2147483647.

For example, you can define a successful job as a job that ends with a return code less than or equal to 3 as follows:

```
succoutputcond UPDATE_OK "(RC <= 3)"
```

Boolean expression

Specifies a logical combination of comparison expressions. The syntax is:

```
comparison_expression operator comparison_expression
```

comparison_expression

The expression is evaluated from left to right. You can use parentheses to assign a priority to the expression evaluation.

operator

Logical operator. It can have the following values:

Table 31. Logical operators

Example	Operator	Result
expr_a and expr_b	And	TRUE if both expr_a and expr_b are TRUE.
expr_a or expr_b	Or	TRUE if either expr_a or expr_b is TRUE.
Not expr_a	Not	TRUE if expr_a is not TRUE.

For example, you can define a successful job as a job that ends with a return code less than or equal to 3 or with a return code not equal to 5, and less than 10 as follows:

```
succoutputcond "(RC<=3) OR ((RC!=5) AND (RC<10))"
```

A job state

On fault-tolerant and dynamic agent workstations only, you can assign which job state signifies the successful completion of a job.

An output variable

On dynamic agent workstations only, qualify a job as having completed successfully using output variables.

- You can set a success condition or other condition for the job by analyzing the job properties.

  For example, for a file transfer job specifically, you enter the following expression:

  ```
  ${this.File.1.Size}>0
  ```

  if you want to qualify a file transfer job as successful when the size of the transferred file is greater than zero.
- You can also set a success or other condition based on the specific plug-in type of the job. For example, you enter the following expression: *${this.name}="executable"* if you want to qualify the job as successful only when its plug-in type is executable. Note that this property returns the plug-in type, and not the name of the job.
- You can set a success or other condition for the job by analyzing the job properties or the job output of another job in the same job stream.

  For example, for a file transfer job, you enter the following expression:

  ```
  ${this.NumberOfTransferredFiles}=
  ${job.DOWNLOAD.NumberOfTransferredFiles}
  ```

  If you want to qualify a file transfer job as successful when the number of uploaded files in the job is the same as the number of downloaded files in another job, named DOWNLOAD, in the same job stream.
- Similarly, you can set a condition based on the plug-in type of another job in the same job stream. For example, you enter the following expression: *${job.DOWNLOAD.name}*="`executable`" if you want to qualify the job as successful only when the specific plug-in type of the referenced job (DOWNLOAD) is executable.
- All Xpath (XML Path Language) functions and expressions are supported, for the above conditions, in the **succoutputcond** field:
  - String comparisons (contains, starts-with, matches, and so on)
  - String manipulations (concat, substring, uppercase, and so on)
  - Numeric comparison (=, !=, >, and so on)
  - Functions on numeric values (abs, floor, round, and so on)
  - Operators on numeric values (add, sum, div, and so on)
  - Boolean operators

Content in the job log

On dynamic agent workstations only, you can determine if a job is successful by analyzing the content of the job log.

Note: The `this.stdlist` variable is available only for jobs defined with advanced options (e.g. executable jobs).

To set a success or unsuccessful condition, you must check the `this.stdlist` variable. For example, you enter the following expression:

```
contains(${this.stdlist},"error")
```

The condition is satisfied if the word "error" is contained in the job output.

outputcond *Condition_Name* "*Condition_Value*"

An output condition that when satisfied determines which successor job runs. The condition is expressed as *Condition_Name* "*Condition_Value*". The format for the condition expression is the same as that for the **succoutputcond** conditions. The following are some examples of output conditions. For example, to create a condition that signifies that the predecessor job has completed with errors, you define the output condition as follows:

- `outputcond STATUS_ERR1 "RC=1"` to create a condition named STATUS_ERR1 that signifies that if the predecessor job completes with return code = 1, then the job completed with errors.
- `outputcond BACKUP_FLOW "RC != 5 and RC > 2"` to create a condition named BACKUP_FLOW. If the predecessor job satisfies the condition then the successor job connected to the predecessor with this conditional dependency runs.

recovery

Recovery options for the job. The default is **stop** with no recovery job and no recovery prompt. Enter one of the recovery options, **stop**, **continue**, or **rerun**. This can be followed by a recovery job, a recovery prompt, or both.

stop

If the job ends abnormally, do not continue with the next job.

continue

If the job ends abnormally, continue with the next job. The job is not listed as abended in the properties of the job stream. If no other problems occur, the job stream completes successfully.

rerun

If the job ends abnormally, rerun the job. You can use it in association with the after [folder/][workstation#][folder/]jobname on page 239 and repeatevery hhmm on page 239 options, or with the after [folder/][workstation#][folder/]jobname on page 239 and abendprompt "text" on page 240 options. You can optionally specify one or more of the following options to define a rerun sequence:

same_workstation

Specifies whether the job must be rerun on the same workstation as the parent job. This option is applicable only to pool and dynamic pool workstations.

repeatevery *hhmm*

Specifies how often HCL Workload Automation attempts to rerun the failed job. The default value is 0. The maximum supported value is 99 hours and 59 minutes. The countdown for starting the rerun attempts begins after the parent job, or the recovery job if any, has completed.

for *number* attempts

Specifies the maximum number of rerun attempts to be performed. The default value is 1. The maximum supported value is 10.000 attempts.

If you specify a recovery job and both the parent and recovery jobs fail, the dependencies of the parent job are not released and its successors, if any, are not run. If you have set the rerun option, the rerun is not performed. In this case, you must manually perform the following steps:

1. Manually confirm the recovery job is in SUCC state.
2. Clean up the environment by performing manually the operations that were to be performed by the recovery job.
3. Submit a rerun of the parent job.

after *[folder/][workstation#][folder/]jobname*

Specifies the name of a recovery job to run if the parent job ends abnormally. Recovery jobs are run only once for each abended instance of the parent job.

You can specify the recovery job's workstation if it is different from the parent job's workstation. The default is the parent job's workstation. Not all jobs are eligible to have recovery jobs run on a different workstation. Follow these guidelines:

- If the job has a recovery job in another workstation, not only the recovery job workstation needs to have the value **on** for **fullstatus** but also the parent job's workstation needs to have its value **on** for **fullstatus**.
- If either workstation is an extended agent, it must be hosted by a domain manager or a fault-tolerant agent with a value of **on** for **fullstatus**.

- The recovery job workstation can be in the same domain as the parent job workstation or in a higher domain.
- If the recovery job workstation is a fault-tolerant agent, it must have a value of **on** for **fullstatus**.

abendprompt "*text*"

Specifies the text of a recovery prompt, enclosed between double quotation marks, to be displayed if the job ends abnormally. The text can contain up to 64 characters. If the text begins with a colon (:), the prompt is displayed, but no reply is required to continue processing. If the text begins with an exclamation mark (!), the prompt is displayed, but it is not recorded in the log file. You can also use HCL Workload Automation parameters.

See Using variables and parameters in job definitions on page 243 for more information.

Table 32: Recovery options and actions on page 240 summarizes all possible combinations of recovery options and actions.

The table is based on the following criteria from a job stream called `sked1`:

- Job stream `sked1` has two jobs, `job1` and `job2`.
- If selected for `job1`, the recovery job is `jobr`.
- `job2` is dependent on `job1` and does not start until `job1` has completed.

Table 32. Recovery options and actions

	Stop	Continue	Rerun
Recovery prompt: No **Recovery job**: No	Intervention is required.	Run `job2`.	Rerun `job1`. If `job1` ends abnormally, issue a prompt. If reply is *yes*, repeat above. If `job1` is successful, run `job2`.
Recovery prompt: Yes **Recovery job**: No	Issue recovery prompt. Intervention is required.	Issue recovery prompt. If reply is *yes*, run `job2`.	Issue recovery prompt. If reply is *yes*, rerun `job1`. If `job1` ends abnormally, repeat above. If `job1` is successful, run `job2`.
Recovery prompt: No **Recovery job**: Yes	Run `jobr`. If it ends abnormally, intervention is required. If it is successful, run `job2`.	Run `jobr`. Run `job2`.	Run `jobr`. If `jobr` ends abnormally, intervention is required. If `jobr` is successful, rerun `job1`. If `job1` ends abnormally, issue a prompt. If reply is *yes*, repeat above. If `job1` is successful, run `job2`.

Table 32. Recovery options and actions (continued)

	Stop	Continue	Rerun
Recovery prompt: Yes **Recovery job**: Yes	Issue recovery prompt. If reply is *yes*, run `jobr`. If it ends abnormally, intervention is required. If it is successful, run `job2`.	Issue recovery prompt. If reply is *yes*, run `jobr`. Run `job2`.	Issue recovery prompt. If reply is *yes*, run `jobr`. If `jobr` ends abnormally, intervention is required. If `jobr` is successful, rerun `job1`. If `job1` ends abnormally, repeat above. If `job1` is successful, run `job2`.



Notes:

1. "Intervention is required" means that `job2` is not released from its dependency on `job1`, and therefore must be released by the operator.
2. The **continue** recovery option overrides the ends abnormally state, which might cause the job stream containing the ended abnormally job to be marked as successful. This prevents the job stream from being carried forward to the next production plan.
3. If you select the **rerun** option without supplying a recovery prompt, HCL Workload Automation generates its own prompt.
4. To reference a recovery job in **conman**, use the name of the original job (`job1` in the scenario above, not `jobr`). Only one recovery job is run for each abnormal end.

Example

Examples

The following is an example of a file containing two job definitions:

```
$jobs
cpu1#gl1
     scriptname "/usr/acct/scripts/gl1"
     streamlogon acct
     description "general ledger job1"
bkup
     scriptname "/usr/mis/scripts/bkup"
     streamlogon "^mis^"
     recovery continue after myfolder/recjob1
```

The following example shows how to define the HCL Workload Automation TWSJOB job stored in the APP/DEV folder that manages the workload broker broker_1 job that runs on the same workload broker agent where the TWSJOB2 ran:

```
ITDWBAGENT#APP/DEV/TWSJOB
SCRIPTNAME "broker_1 -var var1=name,var2=address
           -twsaffinity jobname=TWSJOB2"
STREAMLOGON brkuser
```

```
DESCRIPTION "Added by composer."
TASKTYPE BROKER
RECOVERY STOP
```

The following example shows how to define a job which is assigned to a dynamic pool of UNIX agents and runs the `df` script:

```
DPOOLUNIX#JOBDEF7
 TASK
 <?xml version="1.0" encoding="UTF-8"?>
    <jsdl:jobDefinition
          xmlns:jsdl="http://www.ibm.com/xmlns/prod/scheduling/1.0/jsdl"
          xmlns:jsdle="http://www.ibm.com/xmlns/prod/scheduling/1.0/jsdle">
    <jsdl:application name="executable">
    <jsdle:executable interactive="false">
    <jsdle:script>df</jsdle:script>
    </jsdle:executable>
    </jsdl:application>
    </jsdl:jobDefinition>
 DESCRIPTION "Added by composer."
 RECOVERY STOP
```

The following example shows how to define a job which is assigned to a dynamic pool of Windows agents and runs the `dir` script:

```
DPOOLWIN#JOBDEF6
 TASK
    <?xml version="1.0" encoding="UTF-8"?>
    <jsdl:jobDefinition
          xmlns:jsdl="http://www.ibm.com/xmlns/prod/scheduling/1.0/jsdl"
          xmlns:jsdle="http://www.ibm.com/xmlns/prod/scheduling/1.0/jsdle">
    <jsdl:application name="executable">
    <jsdle:executable interactive="false">
    jsdle:script>dir</jsdle:script>
    </jsdle:executable>
    </jsdl:application>
    </jsdl:jobDefinition>
 DESCRIPTION "Added by composer."
 RECOVERY STOP
```

The following example shows how to define a job which is assigned to the NC115084 agent and runs the `dir` script:

```
NC115084#JOBDEF3
 TASK
    <?xml version="1.0" encoding="UTF-8"?>
    <jsdl:jobDefinition
          xmlns:jsdl="http://www.ibm.com/xmlns/prod/scheduling/1.0/jsdl"
          xmlns:jsdle="http://www.ibm.com/xmlns/prod/scheduling/1.0/jsdle">
    <jsdl:application name="executable">
    <jsdle:executable interactive="false">
    jsdle:script>dir</jsdle:script>
    </jsdle:executable>
    </jsdl:application>
    </jsdl:jobDefinition>
 DESCRIPTION "Added by composer."
 RECOVERY STOP
```

The following example shows how to define a job which is assigned to a pool of UNIX agents and runs the script defined in the `script` tag:

```
POOLUNIX#JOBDEF5
 TASK
    <?xml version="1.0" encoding="UTF-8"?>
    <jsdl:jobDefinition
          xmlns:jsdl="http://www.ibm.com/xmlns/prod/scheduling/1.0/jsdl"
          xmlns:jsdle="http://www.ibm.com/xmlns/prod/scheduling/1.0</jsdle">
    <jsdl:application name="executable">
    <jsdle:executable interactive="false">
    <jsdle:script>#!/bin/sh
sleep 60
dir</jsdle:script>
    </jsdle:executable>
    </jsdl:application>
    </jsdl:jobDefinition>
 DESCRIPTION "Added by composer."
 RECOVERY STOP
```

The following example shows how to define a job which is assigned to a pool of Windows agents and runs the script defined in the `script` tag:

```
POOLWIN#JOBDEF4
 TASK
 <?xml version="1.0" encoding="UTF-8"?>
    <jsdl:jobDefinition
          xmlns:jsdl="http://www.ibm.com/xmlns/prod/scheduling/1.0/jsdl"
          xmlns:jsdle="http://www.ibm.com/xmlns/prod/scheduling/1.0</jsdle">
    <jsdl:application name="executable">
    <jsdle:executable interactive="false">
    <jsdle:script>ping -n 120 localhost</jsdle:script>
    </jsdle:executable>
    </jsdl:application>
    </jsdl:jobDefinition>
 DESCRIPTION "Added by composer."
 RECOVERY STOP
```

See also

From the Dynamic Workload Console you can perform the same task as described in

the *Dynamic Workload Console User's Guide*, section about Creating job definitions.

For more information about how to create and edit scheduling objects, see

the *Dynamic Workload Console User's Guide*, section about Designing your Workload.

Using variables and parameters in job definitions

A variable is a scheduling object that is part of a variable table and is defined in the HCL Workload Automation database. It can be used by all the agents in the domain as long as the users have proper authorization in the security file.

A parameter is defined and used locally on an agent (with the `parms` utility command).

Variables and parameters have the following uses and limitations in job definitions:

- Variables and parameters are allowed in the values for the **streamlogon**, **scriptname**, **docommand**, and **abendprompt** keywords.
- A variable or parameter can be used as an entire string or as part of it.
- Multiple variables and parameters are permitted in a single field.
- Enclose variable names in carets (^), and enclose the entire string in quotation marks. Ensure that caret characters are not preceded by a backslash in the string. If necessary, include the backslash in the definition of the variable or parameter.
- Enclose parameter names in single quotes (') in UNIX, and enclose the entire string in quotation marks.

In the following example a variable named **mis** is used in the **streamlogon** value:

```
$jobs
cpu1#bkup
     scriptname "/usr/mis/scripts/bkup"
     streamlogon "^mis^"
     recovery continue after recjob1
```

For additional examples, see Variable and parameter definition on page 289.

For information about how to pass variables between jobs in the same job stream instance, see Passing variables between jobs on page 269.

Nested variables for increased flexibility

You can define one or more variables whose value contains another variable to allow dynamic variable substitution at run time. This feature makes your definitions more flexible, maintainable, and reusable.

You can nest any number of variables: the variables are resolved at run time in a recursive pattern, until they are all resolved. If a variable is not set, it is ignored and HCL Workload Automation proceeds to the next variable, until the resolution process is completed.

For example, you can define the following variables in a variable table:

```
variable name = JOBCHAIN
variable value = PAYMENT_REPORTS

variable name = POOLNAME
variable value= DDS_${JOBCHAIN}
```

At run time, the value of the variable named `POOLNAME` dynamically resolves to `DDS_PAYMENT_REPORTS`, which is the value of the variable named `JOBCHAIN`.

Similarly, you can define a variable table with multiple nested and concatenated variables as follows:

```
VARTABLE TST_VT_1
MEMBERS
VAR1 "orig-value"
VAR2 "VAR2_${VAR1}"
VAR3 "VAR3_${VAR1}_${VAR2}"
VAR4 "VAR4_${VAR1}${VAR2}${VAR3}"
END
```

The first echo job returns the following output:

```
var1= orig-value
var2= VAR2_orig-value
var3= VAR3_orig-value_VAR2_orig-value
var4= VAR4_orig-value_VAR2_orig-valueVAR3_orig-value_VAR2_orig-value
```

If you update **VAR1** to **20260219**, the output from the second echo job is dynamically resolved with the new value at run time:

```
var1= 20260219
var2= VAR2_20260219
var3= VAR3_20260219_VAR2_20260219
var4= VAR4_20260219VAR2_20260219VAR3_20260219_VAR2_20260219
```

Consider a job stream containing a series of nested variables defined in the **CITY_ROUTES** variable table:

```
SCHEDULE EU-HWS-LNX104#NESTED_VARIABLES
    VARTABLE CITY_ROUTES
    :
    EU-HWS-LNX104_1#PRINT_ROUTES_JOB
    TASK
    <?xml version="1.0" encoding="UTF-8"?>
    <jsdl:jobDefinition xmlns:jsdl="http://www.ibm.com/xmlns/prod/scheduling/1.0/jsdl"
    xmlns:xs="http://www.w3.org/2001/XMLSchema"
 xmlns:jsdle="http://www.ibm.com/xmlns/prod/scheduling/1.0/jsdle"
    xs:text="resolveVariableTable" name="executable">
    <jsdl:application name="executable">
    <jsdle:executable interactive="false">
    <jsdle:script>echo "^VAR5^"
    echo "${VAR9}"</jsdle:script>
    </jsdle:executable>
    </jsdl:application>
    </jsdl:jobDefinition>
    RECOVERY STOP

    END
```

The **CITY_ROUTES** variable table is as follows:

```
VARTABLE CITY_ROUTES
    MEMBERS
    VAR1 "Rome"
    VAR2 "${VAR1}, Paris"
    VAR3 "${VAR2}, London"
    VAR4 "${VAR3}, Berlin"
    VAR5 "${VAR4}, Madrid"
    VAR6 "New York"
    VAR7 "${VAR6}, Toronto"
    VAR8 "${VAR7}, Tokyo"
    VAR9 "${VAR8}, Chicago"
    END
```

The variables are resolved in a recursive pattern until all nested values are substituted. Here is how the specific variables in the job script are resolved based on the syntaxes used:

- `^VAR5^`: Because it is enclosed in caret symbols (^), this variable is resolved when the production plan is generated or extended. It recursively resolves `VAR4`, `VAR3`, `VAR2`, and `VAR1`, outputting the final concatenated string: `Rome, Paris, London, Berlin, Madrid`.
- `${VAR9}`: Because it is enclosed using the dollar sign and braces (`${}`), this variable is dynamically resolved or overwritten when the job is submitted to run. It recursively resolves `VAR8`, `VAR7`, and `VAR6`, outputting the final concatenated string: `New York, Toronto, Tokyo, Chicago`. This resolution mode requires you set the **Variable resolution at runtime** property in the integration.

This example successfully follows the necessary syntax rules for nested variables:

- **Inside the variable table (`CITY_ROUTES`)**: The definitions correctly use the `${variablename}` syntax to nest other variables (for example, `"${VAR1}, Paris"`). You must use this syntax because the caret symbol (^) is not supported when defining nested variables within a variable table.
- **Inside the job definition (`PRINT_ROUTES_JOB`)**: Both the caret symbol (^) and the `${variablename}` syntax are fully supported when referencing those nested variables inside the script of an **integration**.

The job stream with the resolved variables looks as follows:

```
SCHEDULE EU-HWS-LNX104#NESTED_VARIABLES
    VARTABLE CITY_ROUTES
    :
    EU-HWS-LNX104_1#PRINT_ROUTES_JOB
    TASK
    <?xml version="1.0" encoding="UTF-8"?>
    <jsdl:jobDefinition xmlns:jsdl="http://www.ibm.com/xmlns/prod/scheduling/1.0/jsdl"
    xmlns:xs="http://www.w3.org/2001/XMLSchema"
 xmlns:jsdle="http://www.ibm.com/xmlns/prod/scheduling/1.0/jsdle"
    xs:text="resolveVariableTable" name="executable">
    <jsdl:application name="executable">
    <jsdle:executable interactive="false">
    <jsdle:script>echo "Rome, Paris, London, Berlin, Madrid"
    echo "New York, Toronto, Tokyo, Chicago"</jsdle:script>
    </jsdle:executable>
    </jsdl:application>
    </jsdl:jobDefinition>
    RECOVERY STOP

    END
```

In this resolved version, `^VAR5^` has been fully recursively resolved and replaced with `Rome, Paris, London, Berlin, Madrid`, and `${VAR9}` has been dynamically resolved and replaced with `New York, Toronto, Tokyo, Chicago` based on the hierarchy defined in the `CITY_ROUTES` variable table.

Note: The following limitations apply:

- This functionality is supported only for integrations running on dynamic agents. It is not supported on native job types.
- The caret symbol (^) is not supported if you define the variable within a variable table. When defining nested variables within a variable table, you must use the `${variablename}` syntax. However, both the caret symbol (^) and the **${variablename}** syntax are supported when you reference the variable in the job definition of an integration running on a dynamic agent.

Using variables in HCL Workload Automation jobs

This section explains how to add variables to jobs you plan to run with HCL Workload Automation.

When importing jobs from HCL Workload Automation, you can add variables to obtain higher flexibility for your job.

Table 33. Built-in HCL Workload Automation variables

Built-in variables	Description
tws.host.workstation	Name of the host workstation
tws.job.date	Date of the submitted job.
tws.job.fqname	Fully qualified name of the job (UNISON_JOB)
tws.job.ia	Input arrival time of the job
tws.job.interactive	Job is interactive. Values can be `true` or `false`. Applies only to backward-compatible jobs.
tws.job.logon	Credentials of the user who runs the job (LOGIN). Applies only to backward-compatible jobs.
tws.job.name	Name of the submitted job
this.name	The integration type of the job (for example, executable). Note that this variable returns theintegration type, not the name of the job.
job.<JOB_NAME>.name	The integration type of the specifically referenced job (for example, executable).
tws.job.num	UNISON_JOBNUM
tws.job.priority	Priority of the submitted job
tws.job.promoted	Job is promoted. Values can be `YES` or `No`. For more information about promotion for dynamic jobs, see the section about promoting jobs scheduled on dynamic pools in *HCL Workload Automation: User's Guide and Reference*.
tws.job.recnum	Record number of the job.
tws.job.resources	Quantity of the required logical resources assigned on a dynamic pool to a promoted job. Values can be `1` if the job is promoted or `10` if the job is not promoted. For more information about promotion for dynamic jobs, see

Table 33. Built-in HCL Workload Automation variables (continued)

Built-in variables	Description
ForPromoted	the section about promoting jobs scheduled on dynamic pools in *HCL Workload Automation: User's Guide and Reference*.
tws.job.taskstring	Task string of the submitted job. Applies only to backward-compatible jobs.
tws.job.workstation	Name of the workstation on which the job is defined
tws.jobstream.id	ID of the job stream that includes the job (UNISON_SCHED_ID)
tws.jobstream.name	Name of the job stream that includes the job (UNISON_SCHED)
tws.jobstream.workstation	Name of the workstation on which the job stream that includes the job is defined
tws.master.workstation	Name of the master domain manager (UNISON_MASTER)
tws.plan.date	Start date of the production plan (UNISON_SCHED_DATE)
tws.plan.date.epoch	Start date of the production plan, in epoch format (UNISON_SCHED_EPOCH)
tws.plan.runnumber	Run number of the production plan (UNISON_RUN)

The following example illustrates a JSDL file with several of the supported HCL Workload Automation variables defined:

```
…<jsdl:jobDefinition xmlns:jsdl="http://www.abc.com/xmlns/prod/scheduling/1.0/jsdl"
 xmlns:jsdle="http://www.abc.com/xmlns/prod/scheduling/1.0/jsdle" description="This jobs prints
 UNISON Variables received from TWS in standard OutPut " name="sampleUNISON_Variables">
  <jsdl:annotation>This jobs prints UNISON Variables received from TWS in standard OutPut
 </jsdl:annotation>
  <jsdl:variables>
```

```
    <jsdl:stringVariable name="tws.jobstream.name">none</jsdl:stringVariable>
    <jsdl:stringVariable name="tws.job.fqname">none</jsdl:stringVariable>
    <jsdl:stringVariable name="tws.master.workstation">none</jsdl:stringVariable>
    <jsdl:stringVariable name="tws.plan.runnumber">none</jsdl:stringVariable>
    <jsdl:stringVariable name="tws.plan.date">none</jsdl:stringVariable>
    <jsdl:stringVariable name="tws.plan.date.epoch">none</jsdl:stringVariable>
    <jsdl:stringVariable name="tws.job.logon">none</jsdl:stringVariable>
  </jsdl:variables>
  <jsdl:application name="executable">
    <jsdle:executable output="${tws.plan.runnumber}">
      <jsdle:environment>
        <jsdle:variable name="UNISON_SCHED">${tws.jobstream.name}</jsdle:variable>
        <jsdle:variable name="UNISON_JOB">${tws.job.fqname}</jsdle:variable>
        <jsdle:variable name="UNISON_MASTER">${tws.master.workstation}</jsdle:variable>
        <jsdle:variable name="UNISON_RUN">${tws.plan.runnumber}</jsdle:variable>
        <jsdle:variable name="UNISON_SCHED_DATE">${tws.plan.date}</jsdle:variable>
        <jsdle:variable name="UNISON_SCHED_EPOCH">${tws.plan.date.epoch}</jsdle:variable>
        <jsdle:variable name="LOGIN">${tws.job.logon}</jsdle:variable>
      </jsdle:environment>
…
```

The following example illustrates a job definition where variables are used to return both the name of the job and the plug-in type. For a job named **TEST2** of type executable, the variables in the script are resolved as follows:

- `${tws.job.name}` resolves to TEST2
- `${this.name}` resolves to executable
- `${job.TEST2.name}` resolves to executable

```
**$JOBS
WA_AGT#TEST2
 TASK
    <?xml version="1.0" encoding="UTF-8"?>
<jsdl:jobDefinition xmlns:jsdl="http://www.ibm.com/xmlns/prod/scheduling/1.0/jsdl"
xmlns:xs="http://www.w3.org/2001/XMLSchema" xmlns:jsdle="http://www.ibm.com/xmlns/prod/scheduling/1.0/jsdle"
xs:text="resolveVariableTable" name="executable">
    <jsdl:application name="executable">
        <jsdle:executable interactive="false">
            <jsdle:script>echo ${tws.job.name}
echo ${this.name}
echo ${job.TEST2.name}</jsdle:script>
        </jsdle:executable>
    </jsdl:application>
</jsdl:jobDefinition>
 RECOVERY STOP**
```

IBM i jobs

An IBM i job runs the command you specify on an IBM i system (formerly known as AS/400 and i5 OS).

This section describes the required and optional attributes for IBM i jobs. Each job definition has the following format and arguments:

Table 34. Required and optional attributes for the definition of an IBM i job.

Attribute	Description/value	Required
application name	IBM i	✓
user name	The user name authorized to run the job on the IBM i system.	
otherCommandType	The generic command to be run on the IBM i system.	Required for generic type command
SBMJOBType	The SBMJOB command to be run on the IBM i system.	Required for SBMJOB type command
jobName	The name of the job that is associated with the submitted job while it is being processed by the system.	
jobDescription	The job description used to submit jobs for batch processing.	
jobDescriptionLibrary	The library that qualifies the job description.	
jobQueue	The qualified name of the job queue on which the job is placed.	
jobQueueLibrary	The library that qualifies the job queue.	
jobPriority	The scheduling priority for the submitted job.	
outputPriority	The output priority for spooled files that are produced by the submitted job.	
outputQueue	The qualified name of the output queue used for spooled files.	
outputQueueLibrary	The library that qualifies the output queue.	
printDevice	The qualified name of the default printer device for the submitted job.	
systemLibraryList	The system portion of the initial library list that is used by the submitted job.	
currentLibrary	The name of the current library associated with the submitted job.	
initialLibraryList	The initial user part of the library list that is used to search for any object names that were specified without a library qualifier.	

Table 34. Required and optional attributes for the definition of an IBM i job. (continued)

Attribute	Description/value	Required
Child job options	The list of options to define if monitoring child jobs or not. Choose one of the following options: **Use agent settings** Depending on the variable defined on the IBM i agent, child jobs are monitored or not. **Follow child jobs** Child jobs are monitored. **Ignore child jobs** Child jobs are not monitored.	
LDA source (library name/name)	The name of the library and the name of the Local Data Area (LDA).	
msgReplyList	The list of messages for which you want to define an automated reply. For each message, specify: **msgReply** **msgId** The message identifier. **msgCmpDta** The message text. **msgRpy** The automated reply that you want to define. **Message Max Replies** The maximum number of automated replies accepted for the message. Valid range is from 0	

Table 34. Required and optional attributes for the definition of an IBM i job. (continued)

Attribute	Description/value	Required
	to 100. Default value is 10. If 0 is specified, the automated reply to the message is disabled.	

For more information about how to define the Submit Job (SBMJOB) command parameters, see IBM i product documentation.

The following example shows a job that issues a SBMJOB command with the related parameters:

```
$JOBS
IBMI72_94#IBMI_NEWDEF_TEST
</jsdlibmi:commandTypeGroup>
</jsdlibmi:Task>
<jsdlibmi:credential>
<jsdlibmi:userName>userName</jsdlibmi:userName>
</jsdlibmi:credential>
</jsdlibmi:IBMIParameters>
</jsdlibmi:ibmi>
(indent <jsdlibmi:credential> and </jsdlibmi:credential> at the same level of </jsdlibmi:Task>)
 TASK
 <?xml version="1.0" encoding="UTF-8"?>
<jsdl:jobDefinition xmlns:jsdl="http://www.ibm.com/xmlns/prod/scheduling/1.0/jsdl"
xmlns:jsdlibmi="http://www.ibm.com/xmlns/prod/scheduling/1.0/jsdlibmi" name="ibmi">
  <jsdl:application name="ibmi">
    <jsdlibmi:ibmi>
            <jsdlibmi:IBMIParameters>
                <jsdlibmi:Task>
                    <jsdlibmi:command>WRKSYSSTS</jsdlibmi:command>
                    <jsdlibmi:commandTypeGroup>
                        <jsdlibmi:SBMJOBType>
                            <jsdlibmi:jobName>TESTMEL</jsdlibmi:jobName>
                            <jsdlibmi:jobDescription>QDFTJOBD</jsdlibmi:jobDescription>
                            <jsdlibmi:jobDescriptionLibrary>QGPL</jsdlibmi:jobDescriptionLibrary>
                            <jsdlibmi:jobQueue/>
                            <jsdlibmi:jobQueueLibrary/>
                            <jsdlibmi:jobPriority>3</jsdlibmi:jobPriority>
                            <jsdlibmi:outputPriority>4</jsdlibmi:outputPriority>
                            <jsdlibmi:outputQueue>*DEV</jsdlibmi:outputQueue>
                            <jsdlibmi:outputQueueLibrary/>
                            <jsdlibmi:printDevice>PRT01</jsdlibmi:printDevice>
                            <jsdlibmi:systemLibraryList/>
                            <jsdlibmi:currentLibrary>*CRTDFT</jsdlibmi:currentLibrary>
                            <jsdlibmi:initialLibraryList>QGPL       QTEMP      QDEVELOP
                                QBLDSYS</jsdlibmi:initialLibraryList>
                        </jsdlibmi:SBMJOBType>
                    </jsdlibmi:commandTypeGroup>
                </jsdlibmi:Task>
            </jsdlibmi:IBMIParameters>
        </jsdlibmi:ibmi>
  </jsdl:application>
</jsdl:jobDefinition>
 RECOVERY STOP
```

Note: The user needs full access, that is possibility of creating files and directories and changing their ownership to the agent `stdlist` directory (*agent_data_dir*`/stdlist/JM`)

The following example shows a job that runs a command on an IBM i system and defines automated message replies, both for parent and child IBM i jobs. For more information about defining an automated reply for a message, see Scheduling and monitoring jobs on IBM i systems on page 1371.

```
$JOBS
AGTIBMI_MEL#IBMI_MSG_REPLY
 TASK
    <?xml version="1.0" encoding="UTF-8"?>
<jsdl:jobDefinition xmlns:jsdl="http://www.ibm.com/xmlns/prod/scheduling/1.0/jsdl"
xmlns:jsdlibmi="http://www.ibm.com/xmlns/prod/scheduling/1.0/jsdlibmi" name="ibmi">
  <jsdl:application name="ibmi">
    <jsdlibmi:ibmi>
            <jsdlibmi:IBMIParameters>
               <jsdlibmi:Task>
                    <jsdlibmi:command>SBMJOB CMD(CALL PGM(MINERMA/SENDMSGALL)) INQMSGRPY(*SYSRPYL)
                    </jsdlibmi:command>
                    <jsdlibmi:commandTypeGroup>
                        <jsdlibmi:otherCommandType/>
                    </jsdlibmi:commandTypeGroup>
                    <jsdlibmi:msgReplyList>
                        <jsdlibmi:msgReply>
                            <jsdlibmi:msgId>CPA2401/jsdlibmi:msgId>
                            <jsdlibmi:msgCmpDta>*/jsdlibmi:msgCmpDta>
                            <jsdlibmi:msgRpy>Y/jsdlibmi:msgRpy>
                            <jsdlibmi:msgMaxReplies>2/jsdlibmi:msgMaxReplies>
                        </jsdlibmi:msgReply>
                        <jsdlibmi:msgReply>
                            <jsdlibmi:msgId>CPA24*/jsdlibmi:msgId>
                            <jsdlibmi:msgCmpDta>*1*/jsdlibmi:msgCmpDta>
                            <jsdlibmi:msgRpy>Y/jsdlibmi:msgRpy>
                            <jsdlibmi:msgMaxReplies>54/jsdlibmi:msgMaxReplies>
                        </jsdlibmi:msgReply>
                    </jsdlibmi:msgReplyList>
                </jsdlibmi:Task>
            </jsdlibmi:IBMIParameters>
        </jsdlibmi:ibmi>
  </jsdl:application>
</jsdl:jobDefinition>
 RECOVERY STOP
```

See also

From the Dynamic Workload Console you can perform the same task as described in

the *Dynamic Workload Console User's Guide*, section about Creating job definitions.

For more information about how to create and edit scheduling objects, see

the *Dynamic Workload Console User's Guide*, section about Designing your Workload.

Executable jobs

An executable job runs scripts or commands with advanced options, such as redirecting standard input and standard output to a file.

A description of the job properties and valid values are detailed in the context-sensitive help in the Dynamic Workload Console by clicking the question mark (?) icon in the top-right corner of the properties pane.

This section describes the required and optional attributes for executable jobs. Each job definition has the following format and arguments:

Table 35. Required and optional attributes for the definition of an executable job.

Attribute	Description/value	Required
application name	executable	✓
interactive	Specify whether the job requires user intervention. This option applies only to jobs that run on Windows operating systems.	✓
value	Specify the name and value of one or more arguments.	
script	Type a script to be run by the job. The script is created and ran when the job runs. You can specify the arguments in this tag, or you can type them in the **value** tag and call them in the script.	✓
suffix	Specify the file name extension for the script to be run by the job. This option applies only to jobs that run on Windows operating systems. Do not insert the "." at the begin of the extension name.	

Example

The following example shows a job that pings two web sites. The address of the web sites is defined in the **value** tag and called in the **script** tag. This job has an affinity relationship with job `affine_test`, which means this job runs on the same workstation as `affine_test`:

```
$JOBS
AGENT#EXECUTABLE
TASK
<?xml version="1.0" encoding="UTF-8"?>
<jsdl:jobDefinition xmlns:jsdl="http://www.ibm.com/xmlns/prod/scheduling/1.0/jsdl"
xmlns:jsdle="http://www.ibm.com/xmlns/prod/scheduling/1.0/jsdle" name="executable">
  <jsdl:application name="executable">
    <jsdle:executable interactive="false" workingDirectory="c:\">
            <jsdle:arguments>
                <jsdle:value>www.mysite.com</jsdle:value>
                <jsdle:value>www.yoursite.com</jsdle:value>
            </jsdle:arguments>
            <jsdle:script>ping %1 ping %2</jsdle:script>
        </jsdle:executable>
  </jsdl:application>
</jsdl:jobDefinition>
DESCRIPTION "Defined using composer."
TWSAFFINITY "affine_test"
RECOVERY STOP
```

Example

The following example shows a job that runs a `vbs` script on Windows operating systems. The file name extension is defined in the `suffix` attribute of the `script` tag:

```
WIN_WKS1#VBS_NAT1
 TASK
    <?xml version="1.0" encoding="UTF-8"?>
<jsdl:jobDefinition xmlns:jsdl="http://www.ibm.com/xmlns/prod/scheduling/1.0/jsdl"
xmlns:jsdle="http://www.ibm.com/xmlns/prod/scheduling/1.0/jsdle" name="executable">
  <jsdl:application name="executable">
    <jsdle:executable interactive="true" workingDirectory="c:\tws">
            <jsdle:script suffix="vbs">Wscript.Echo "ciao"</jsdle:script>
        </jsdle:executable>
  </jsdl:application>
</jsdl:jobDefinition>
 RECOVERY STOP
```

See also

From the Dynamic Workload Console you can perform the same task as described in

the *Dynamic Workload Console User's Guide*, section about Creating job definitions.

For more information about how to create and edit scheduling objects, see

the *Dynamic Workload Console User's Guide*, section about Designing your Workload.

Access method jobs

Access method jobs extend HCL Workload Automation scheduling functions to other systems and applications using access methods. The access methods communicate with the external system to launch the job and return the status of the job.

A description of the job properties and valid values are detailed in the context-sensitive help in the Dynamic Workload Console by clicking the question mark (?) icon in the top-right corner of the properties pane.

This section describes the required and optional attributes for access method jobs. Each job definition has the following format and arguments:

Table 36. Required and optional attributes for the definition of an access method job

Attribute	Description/value	Required
application name	xajob	✓
accessMethod	The name of the access method used to communicate with the external system to start the job and return the status of the job.	✓
target	The name of an option file.	
taskString	Command to be interpreted by the selected method. The maximum line length is 8 KB.	✓
credentials	The name and password of the user running this job. As an alternative to hard-coding actual values, you can parametrize in one of the following ways:	

Table 36. Required and optional attributes for the definition of an access method job (continued)

Attribute	Description/value	Required

	• Enter a *username* specified in the database with the user definition (it is applicable to all operating systems on this job type) and key the statement:
	`<jsdl:password>${password:username}</jsdl:password>`
	The password is retrieved from the *username* user definition in the database and resolved at runtime. For further details, see Using user definitions on job types with advanced options on page 285.
	You can also specify the user of a different workstation and use the following syntax for the password:
	`<jsdl:password>${password:workstation#username}`
	`</jsdl:password>`
	• Enter a user and password defined with the `param` utility command locally on the dynamic agent that will run the job (if the job is to be submitted to a pool or to a dynamic pool, the definition must be present on all the agents of the pool). If you defined the user name with the variable *user* and a password, the corresponding credential statements is:
	`<jsdl:userName>${agent:user}</jsdl:userName>`
	`<jsdl:password>${agent:password.user}</jsdl:password>`
	The user and password variables are resolved on the agent at runtime. For further details, see Defining variables and passwords for local resolution on dynamic agents on page 258.	

The following example shows a job that creates a file in the `/methods` folder using a default access method job:

```
$JOBS
AGENT#XA_JOB
TASK
<?xml version="1.0" encoding="UTF-8"?>
<jsdl:jobDefinition xmlns:jsdl="http://www.ibm.com/xmlns/prod/scheduling/1.0/jsdl" xmlns:jsdlxa=
"http://www.ibm.com/xmlns/prod/scheduling/1.0/jsdlxa" name="xajob">
  <jsdl:application name="xajob">
    <jsdlxa:xajob accessMethod="unixlocl" target="optionFile">
       <jsdlxa:taskString>touch file</jsdlxa:taskString>
       <jsdlxa:credentials>
         <jsdlxa:userName>TestUser</jsdlxa:userName>
         <jsdlxa:password>{aes}IEr/DES8wRzQEij1ySQBfUR587QBxM0iwfQ1EWJaDds=</jsdlxa:password>
       </jsdlxa:credentials>
     </jsdlxa:xajob>
  </jsdl:application>
</jsdl:jobDefinition>
DESCRIPTION "Defined using composer."
RECOVERY STOP
```

See also

From the Dynamic Workload Console you can perform the same task as described in

the *Dynamic Workload Console User's Guide*, section about Creating job definitions.

For more information about how to create and edit scheduling objects, see

the *Dynamic Workload Console User's Guide*, section about Designing your Workload.

Defining variables and passwords for local resolution on dynamic agents

For job types with advanced options you have the possibility to let variables and passwords be defined and resolved locally on the dynamic agents (including pools and dynamic pools).

This is particularly useful in the case of passwords because you are not required to specify them in the job definition. The advantage is that, if the password has to change, you do not modify the job definition, but you change it with the param on page 990 command locally on the agents (or on the pool agents) that run or may run the job. If the job is to be submitted to a pool or dynamic pool, you can copy the file with the variable definitions to all the agents participating in that pool, so that the variables are resolved locally wherever the job will run.

This feature is not restricted to Windows workstations alone. You can use it also on UNIX, as long as you apply it on job types with advanced options.

To define a variable or a password locally on a dynamic agent, use the param on page 990 utility command. This command has the power to create, delete, and list local variables in dynamic agents. See the details on page 990 on this command to learn how to use it.

Specifying local variables and passwords in the job definitions

After defining a variable and its value with the `param` command, to add it within a job definition so that it is resolved locally on the agent at runtime, use the following syntax:

```
${agent:variable_name}
```

After defining a password with the `param` command, to add it within a job definition so that it is resolved locally on the agent at runtime, use the following syntax:

```
${agent:password.user_name}
```

You can nest variables within passwords. If you used a variable to define a user, enter the password as follows within the job definition:

```
${agent:password.${agent:variable_name}}
```

where *variable_name* was previously defined as having value *user_name* with the `param` command.

Example

An HCL Workload Automation administrator needs to define a file transfer job that downloads a file from a remote server to one of the dynamic agents making up a pool of agents. The administrator wants to parametrize in the job definition:

- The name with which the remote file will be saved locally
- The remote user name and its password

The administrator proceeds in the following way on one of the agents:

1. Defines a variable named `localfile`. The variable is given a value equal to `./npp.5.1.1.Installer.DW.exe` and is created in a new variables file named `FTPvars` (no section). The command to do this is:

   ```
   E:\HCL\TWA\TWS\CLI\bin>param -c FTPvars..localfile ./npp.5.1.1.Installer.DW.exe
   ```

2. Defines a variable named `remoteUser`. The variable is given a value equal to `FTPuser` and is created in the `FTPvars` file (no section). The command to do this is:

   ```
   E:\HCL\TWA\TWS\CLI\bin>param -c FTPvars..remoteUser FTPuser
   ```

3. Defines the password for `FTPuser`. The password value is `tdwb8nxt` and is created in the `password` section of the `FTPvars` file. The command to do this is:

   ```
   E:\HCL\TWA\TWS\CLI\bin>param -c FTPvars.password.FTPuser tdwb8nxt
   ```

4. With a text editor opens file

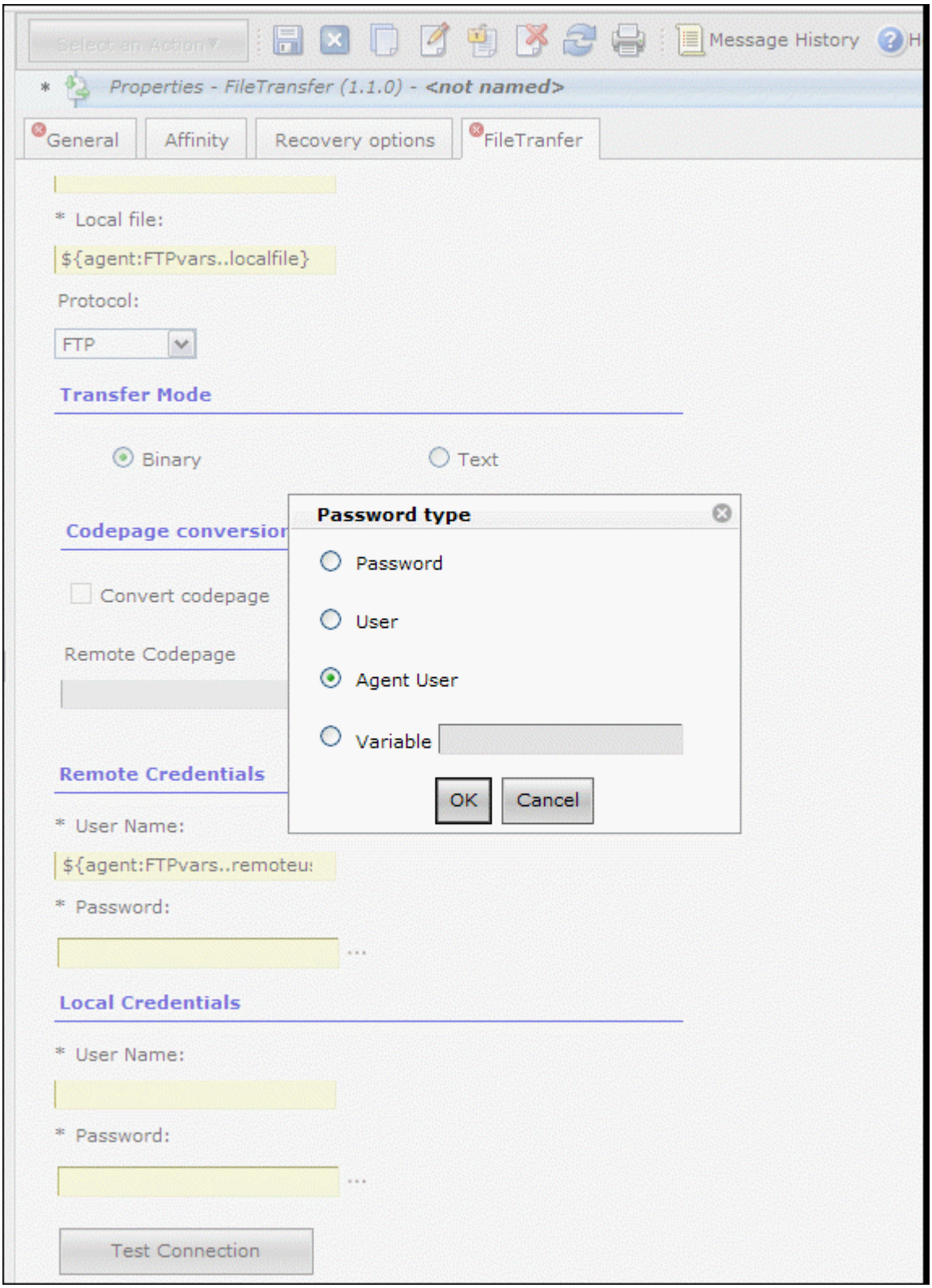
   ```
   E:\HCL\TWA\TWS\ITA\cpa\config\jm_variables_files\FTPvars
   ```

   and checks its contents:

   ```
   localfile = ./npp.5.1.1.Installer.DW.exe
   remoteuser = FTPuser

   [password]
   FTPuser = {aes}XMMYMY2zBHvDEDBo5DdZVmwOJao60pX1K6x2HhRcovA=
   ```

5. Copies file `FTPvars` in the *agent_installation_path*`\TWA\TWS\ITA\cpa\config\jm_variables_files>` path of every other agent defined in the pool.
6. Starts defining the new file transfer job in the Workload Designer panel of Dynamic Workload Console. In the `FileTransfer` window:
   a. Enters `${agent:FTPvars..localfile}` in the `Local file` field.
   b. Enters `${agent:FTPvars..remoteuser}` in the `Remote Credentials` →`User Name` field.
   c. Clicks the `...` button next to the `Remote Credentials` →`Password` field. The `Password type` window pops up and the administrator selects `Agent User`.



d. After the administrator clicks the `OK` button for confirmation in the popup window, the `Remote Credentials` →`Password` field is filled with the `${agent:password.${agent:FTPvars..remoteuser}}` value.

7. Fills in all the other fields to complete the job definition.

When the job is run, the entities and the password entered as variables are resolved with the values defined in the `FTPvars` file.

Obtaining passwords from password vaults

Define parameters in the job definition to retrieve passwords from password vaults.

In addition to resolving passwords locally on agents, as described in Specifying local variables and passwords in the job definitions on page 258, you can specify a string or a query within the job definition to retrieve the password from a password vault. A dedicated dynamic agent, specified in the string or query, interacts with the password vault and returns the password required to run the job. For more information about configuring agents to work with password vaults, see the topic about configuring the agent to work with a password vault in *Administration Guide*.

When creating the job definition, add the following syntax in the `<jsdl:password>...</jsdl:password>` section:

```
${vault:vault_wks#vault-profile-name:query-for-username}
```

where

vault_wks

The workstation in charge of interacting with the password vault. Supported workstation types are:

- dynamic agents
- pools
- dynamic pools
- z-centric agents (if configured to communicate with a dynamic domain manager)

vault-profile-name

The name of the profile to be applied when interacting with the password vault. This parameter is optional. For more information, see the topic about configuring the agent to work with a password vault in *Administration Guide*.

query-for-username

The name of the user whose password you want to retrieve. You can also specify a query in the format supported by the password vault if you have complex requirements for password management. For more information, see Query format on page 262. This query format applies to CyberArk. If you plan to use a different password vault, you have to use its specific syntax. If you specify a query, it overrides any other settings defined in the string.



Note: Make sure you write the query in a way that it retrieves only a single, unique password. If the query results in multiple passwords, the job fails with an error returned by the CyberArk server.

See also the examples and queries detailed in Examples of job definitions with password retrieval from CyberArk on page 263.

Query format

In addition to specifying the password request by filling in the fields described above, you can also use the syntax described in this section to create complex queries.

The syntax is based on combinations of the following parameters available in the **AppDescs** and **Query** sections of the `CyberArk.ini` file, as follows:

[CyberArk.AppDescs]

AppID

The unique ID of the application issuing the password request. This parameter is required.

[CyberArk.Query]

Safe

The name of the Safe where the password is stored.

Folder

The name of the folder where the password is stored.

Object

The name of the password object to retrieve.

Username

Defines search criteria according to the **UserName** account property.

Address

Defines search criteria according to the **Address** account property.

PolicyID

Defines the format that will be used in the **setPolicyID** method.

Database

Defines search criteria according to the **Database** account property.

```
<section_name>::<parameter_name>=parameter_value;<section_name>::<parameter_name>=parameter_value...
```

Consider the following example:

```
AppDescs::AppID=TestApplicationCert;Query::Safe=TestSafe;Query::Username=ITAuser;…;Query::PolicyID=UnixSSH
```

Note: Make sure you write the query in a way that it retrieves only a single, unique password. If the query results in multiple passwords, the job fails with an error returned by the CyberArk server.

For a full list of the parameters available in the `CyberArk.ini` file, see the topic about defining parameters in the CyberArk.ini configuration file in *Administration Guide*.

See also the examples and queries detailed in Examples of job definitions with password retrieval from CyberArk on page 263.

Examples of job definitions with password retrieval from CyberArk

Several examples in which a password vault is used tom retrieve the password for a specified user.

The job runs on the local agent named `HQ-PRD-NET2_B` and uses a profile named `VaultProfile`, stored locally on the `HQ-PRD-NET2_B` agent

```
$JOBS
/F3_DA/HQ-PRD-NET2_B#CYBERARK_JOB_135223229
TASK
<?xml version="1.0" encoding="UTF-8"?>
<jsdl:jobDefinition xmlns:jsdl="http://www.ibm.com/xmlns/prod/scheduling/1.0/jsdl"
xmlns:jsdlremotecommand="http://www.ibm.com/xmlns/prod/scheduling/1.0/jsdlremotecommand"
 name="REMOTECOMMAND">
    <jsdl:application name="remotecommand">
        <jsdlremotecommand:remotecommand>
            <jsdlremotecommand:RemoteCommandParameters>
                <jsdlremotecommand:taskPanel>
                    <jsdlremotecommand:command>netstat</jsdlremotecommand:command>
                                </jsdlremotecommand:taskPanel>
                <jsdlremotecommand:environmentPanel>
                    <jsdlremotecommand:standardOutput/>
                    <jsdlremotecommand:standardError/>
                                </jsdlremotecommand:environmentPanel>
                <jsdlremotecommand:serverPanel>
                    <jsdlremotecommand:serverInfo>
                        <jsdlremotecommand:serverName>HQ-PRD-NET212</jsdlremotecommand:serverName>
                        <jsdlremotecommand:port/>
                        <jsdlremotecommand:protocol>Auto</jsdlremotecommand:protocol>
                                        </jsdlremotecommand:serverInfo>
                    <jsdlremotecommand:credentials>
                        <jsdl:userName>rshLocalUser</jsdl:userName>

  <jsdl:password>${vault:/F3_DA/HQ-PRD-NET2_B#VaultProfile:rshLocalUser}</jsdl:password>
                                        </jsdlremotecommand:credentials>
                    <jsdlremotecommand:certificates>
                        <jsdlremotecommand:keystoreFilePath/>
                        <jsdlremotecommand:password/>
                                        </jsdlremotecommand:certificates>
                                </jsdlremotecommand:serverPanel>
                        </jsdlremotecommand:RemoteCommandParameters>
                </jsdlremotecommand:remotecommand>
    </jsdl:application>
</jsdl:jobDefinition>
 TWSAFFINITY ""
 DESCRIPTION ""
RECOVERY STOP
```

The job runs on a remote agent named `HQ-PRD-NET2_B`. Another agent, named `HQ-PRD-NET3_C`, interacts with the password vault using the default profile stored locally on the `HQ-PRD-NET3_C` agent.

```
$JOBS
/F3_DA/HQ-PRD-NET2_B#CYBERARK_JOB_135337329
TASK
```

```
<?xml version="1.0" encoding="UTF-8"?>
<jsdl:jobDefinition xmlns:jsdl="http://www.ibm.com/xmlns/prod/scheduling/1.0/jsdl"
xmlns:jsdlremotecommand="http://www.ibm.com/xmlns/prod/scheduling/1.0/jsdlremotecommand"
 name="REMOTECOMMAND">
    <jsdl:application name="remotecommand">
        <jsdlremotecommand:remotecommand>
            <jsdlremotecommand:RemoteCommandParameters>
                <jsdlremotecommand:taskPanel>
                    <jsdlremotecommand:command>netstat</jsdlremotecommand:command>
                            </jsdlremotecommand:taskPanel>
                <jsdlremotecommand:environmentPanel>
                    <jsdlremotecommand:standardOutput/>
                    <jsdlremotecommand:standardError/>
                            </jsdlremotecommand:environmentPanel>
                <jsdlremotecommand:serverPanel>
                    <jsdlremotecommand:serverInfo>
                        <jsdlremotecommand:serverName>HQ-PRD-NET212</jsdlremotecommand:serverName>
                        <jsdlremotecommand:port/>
                        <jsdlremotecommand:protocol>Auto</jsdlremotecommand:protocol>
                                    </jsdlremotecommand:serverInfo>
                    <jsdlremotecommand:credentials>
                        <jsdl:userName>rshLocalUser</jsdl:userName>
                        <jsdl:password>${vault:/F2_DA/HQ-PRD-NET3_C#rshLocalUser}</jsdl:password>
                                    </jsdlremotecommand:credentials>
                    <jsdlremotecommand:certificates>
                        <jsdlremotecommand:keystoreFilePath/>
                        <jsdlremotecommand:password/>
                                    </jsdlremotecommand:certificates>
                            </jsdlremotecommand:serverPanel>
                        </jsdlremotecommand:RemoteCommandParameters>
                </jsdlremotecommand:remotecommand>
    </jsdl:application>
</jsdl:jobDefinition>
 TWSAFFINITY ""
 DESCRIPTION ""
RECOVERY STOP
```

The job runs on a pool named `POOL4CYBERARK` and applies the profile named `VaultProfile` stored locally on the `POOL4CYBERARK` pool

```
$JOBS
POOL4CYBERARK#CYBERARK_JOB_135406395
TASK
<?xml version="1.0" encoding="UTF-8"?>
<jsdl:jobDefinition xmlns:jsdl="http://www.ibm.com/xmlns/prod/scheduling/1.0/jsdl"
xmlns:jsdlremotecommand="http://www.ibm.com/xmlns/prod/scheduling/1.0/jsdlremotecommand"
 name="REMOTECOMMAND">
    <jsdl:application name="remotecommand">
        <jsdlremotecommand:remotecommand>
            <jsdlremotecommand:RemoteCommandParameters>
                <jsdlremotecommand:taskPanel>
                    <jsdlremotecommand:command>netstat</jsdlremotecommand:command>
                            </jsdlremotecommand:taskPanel>
                <jsdlremotecommand:environmentPanel>
                    <jsdlremotecommand:standardOutput/>
                    <jsdlremotecommand:standardError/>
                            </jsdlremotecommand:environmentPanel>
                <jsdlremotecommand:serverPanel>
```

```
                    <jsdlremotecommand:serverInfo>
                        <jsdlremotecommand:serverName>HQ-PRD-NET212</jsdlremotecommand:serverName>
                        <jsdlremotecommand:port/>
                        <jsdlremotecommand:protocol>Auto</jsdlremotecommand:protocol>
                                    </jsdlremotecommand:serverInfo>
                    <jsdlremotecommand:credentials>
                        <jsdl:userName>rshLocalUser</jsdl:userName>

  <jsdl:password>${vault:POOL4CYBERARK#VaultProfile:rshLocalUser}</jsdl:password>
                                    </jsdlremotecommand:credentials>
                    <jsdlremotecommand:certificates>
                        <jsdlremotecommand:keystoreFilePath/>
                        <jsdlremotecommand:password/>
                                    </jsdlremotecommand:certificates>
                            </jsdlremotecommand:serverPanel>
                        </jsdlremotecommand:RemoteCommandParameters>
                </jsdlremotecommand:remotecommand>
    </jsdl:application>
</jsdl:jobDefinition>
 TWSAFFINITY ""
 DESCRIPTION ""
RECOVERY STOP
```

The job runs on a dynamic pool named `DPOOL4CYBERARK` and applies the default profile stored locally on the `DPOOL4CYBERARK` dynamic pool

```
$JOBS
DPOOL4CYBERARK#CYBERARK_JOB_135452149
TASK
<?xml version="1.0" encoding="UTF-8"?>
<jsdl:jobDefinition xmlns:jsdl="http://www.ibm.com/xmlns/prod/scheduling/1.0/jsdl"
xmlns:jsdlremotecommand="http://www.ibm.com/xmlns/prod/scheduling/1.0/jsdlremotecommand"
 name="REMOTECOMMAND">
    <jsdl:application name="remotecommand">
        <jsdlremotecommand:remotecommand>
            <jsdlremotecommand:RemoteCommandParameters>
                <jsdlremotecommand:taskPanel>
                    <jsdlremotecommand:command>netstat</jsdlremotecommand:command>
                                </jsdlremotecommand:taskPanel>
                <jsdlremotecommand:environmentPanel>
                    <jsdlremotecommand:standardOutput/>
                    <jsdlremotecommand:standardError/>
                                </jsdlremotecommand:environmentPanel>
                <jsdlremotecommand:serverPanel>
                    <jsdlremotecommand:serverInfo>
                        <jsdlremotecommand:serverName>HQ-PRD-NET212</jsdlremotecommand:serverName>
                        <jsdlremotecommand:port/>
                        <jsdlremotecommand:protocol>Auto</jsdlremotecommand:protocol>
                                    </jsdlremotecommand:serverInfo>
                    <jsdlremotecommand:credentials>
                        <jsdl:userName>rshLocalUser</jsdl:userName>
                        <jsdl:password>${vault:DPOOL4CYBERARK#rshLocalUser}</jsdl:password>
                                    </jsdlremotecommand:credentials>
                    <jsdlremotecommand:certificates>
                        <jsdlremotecommand:keystoreFilePath/>
                        <jsdlremotecommand:password/>
                                    </jsdlremotecommand:certificates>
                            </jsdlremotecommand:serverPanel>
```

```
                        </jsdlremotecommand:RemoteCommandParameters>
                </jsdlremotecommand:remotecommand>
    </jsdl:application>
</jsdl:jobDefinition>
 TWSAFFINITY ""
 DESCRIPTION ""
RECOVERY STOP
```

The job runs on the local agent named `HQ-PRD-NET2_B`, uses a CyberArk query to retrieve the password and applies a profile named `VaultProfile` and stored locally on the `HQ-PRD-NET2_B` agent

```
$JOBS
/F3_DA/HQ-PRD-NET2_B#CYBERARK_JOB_135514915
TASK
<?xml version="1.0" encoding="UTF-8"?>
<jsdl:jobDefinition xmlns:jsdl="http://www.ibm.com/xmlns/prod/scheduling/1.0/jsdl"
xmlns:jsdlremotecommand="http://www.ibm.com/xmlns/prod/scheduling/1.0/jsdlremotecommand"
 name="REMOTECOMMAND">
    <jsdl:application name="remotecommand">
        <jsdlremotecommand:remotecommand>
            <jsdlremotecommand:RemoteCommandParameters>
                <jsdlremotecommand:taskPanel>
                    <jsdlremotecommand:command>netstat</jsdlremotecommand:command>
                                </jsdlremotecommand:taskPanel>
                <jsdlremotecommand:environmentPanel>
                    <jsdlremotecommand:standardOutput/>
                    <jsdlremotecommand:standardError/>
                                </jsdlremotecommand:environmentPanel>
                <jsdlremotecommand:serverPanel>
                    <jsdlremotecommand:serverInfo>
                        <jsdlremotecommand:serverName>HQ-PRD-NET212</jsdlremotecommand:serverName>
                        <jsdlremotecommand:port/>
                        <jsdlremotecommand:protocol>Auto</jsdlremotecommand:protocol>
                                        </jsdlremotecommand:serverInfo>
                    <jsdlremotecommand:credentials>
                        <jsdl:userName>rshLocalUser</jsdl:userName>

  <jsdl:password>${vault:/F3_DA/HQ-
PRD-NET2_B#VaultProfile:AppDescs::AppID=TestApplicationCert;Query:
:Safe=TestSafe;Query::Folder=Root;Query::Object=rshLocalUser;Query::Username=rshLocalUser;Query::A
ddress=Address;Query::PolicyID=UnixSSH}</jsdl:password>
                                        </jsdlremotecommand:credentials>
                    <jsdlremotecommand:certificates>
                        <jsdlremotecommand:keystoreFilePath/>
                        <jsdlremotecommand:password/>
                                        </jsdlremotecommand:certificates>
                                </jsdlremotecommand:serverPanel>
                        </jsdlremotecommand:RemoteCommandParameters>
                </jsdlremotecommand:remotecommand>
    </jsdl:application>
</jsdl:jobDefinition>
 TWSAFFINITY ""
 DESCRIPTION ""
RECOVERY STOP
```

The job on the local agent named `HQ-PRD-NET2_B`, uses a CyberArk query to retrieve the password with no profile specified in the string. In this case, the default profile stored locally on the `HQ-PRD-NET2_B` agent is used.

```
$JOBS
/F3_DA/HQ-PRD-NET2_B#CYBERARK_JOB_135631034
TASK
<?xml version="1.0" encoding="UTF-8"?>
<jsdl:jobDefinition xmlns:jsdl="http://www.ibm.com/xmlns/prod/scheduling/1.0/jsdl"
xmlns:jsdlremotecommand="http://www.ibm.com/xmlns/prod/scheduling/1.0/jsdlremotecommand"
 name="REMOTECOMMAND">
    <jsdl:application name="remotecommand">
        <jsdlremotecommand:remotecommand>
            <jsdlremotecommand:RemoteCommandParameters>
                <jsdlremotecommand:taskPanel>
                    <jsdlremotecommand:command>netstat</jsdlremotecommand:command>
                                </jsdlremotecommand:taskPanel>
                <jsdlremotecommand:environmentPanel>
                    <jsdlremotecommand:standardOutput/>
                    <jsdlremotecommand:standardError/>
                                </jsdlremotecommand:environmentPanel>
                <jsdlremotecommand:serverPanel>
                    <jsdlremotecommand:serverInfo>
                        <jsdlremotecommand:serverName>HQ-PRD-NET212</jsdlremotecommand:serverName>
                        <jsdlremotecommand:port/>
                        <jsdlremotecommand:protocol>Auto</jsdlremotecommand:protocol>
                                        </jsdlremotecommand:serverInfo>
                    <jsdlremotecommand:credentials>
                        <jsdl:userName>rshLocalUser</jsdl:userName>

  <jsdl:password>${vault:/F3_DA/HQ-PRD-NET2_B#AppDescs::AppID=TestApplicationCert;Query:
:Safe=TestSafe;Query::Folder=Root;Query::Object=rshLocalUser;Query::Username=rshLocalUser;Query::A
ddress=Address;Query::PolicyID=UnixSSH}</jsdl:password>
                                        </jsdlremotecommand:credentials>
                    <jsdlremotecommand:certificates>
                        <jsdlremotecommand:keystoreFilePath/>
                        <jsdlremotecommand:password/>
                                        </jsdlremotecommand:certificates>
                                </jsdlremotecommand:serverPanel>
                        </jsdlremotecommand:RemoteCommandParameters>
                </jsdlremotecommand:remotecommand>
    </jsdl:application>
</jsdl:jobDefinition>
 TWSAFFINITY ""
 DESCRIPTION ""
RECOVERY STOP
```

Defining variables in dynamic workload broker jobs

This section explains how to add variables to jobs you plan to run with dynamic workload broker.

You can include variables in your job definition. The variables are resolved at submission time.

The supported variables are as follows:

Table 37. Supported HCL Workload Automation variables in JSDL definitions.

Variables that can be inserted in the dynamic workload broker job definition	Description
tws.host.workstation	Name of the host workstation.
tws.job.date	Date of the submitted job.
tws.job.fqname	Fully qualified name of the job (UNISON_JOB).
tws.job.ia	Input arrival time of the job.
tws.job.interactive	Job is interactive. Values can be `true` or `false`. Applies only to jobs compatible with earlier versions.
tws.job.logon	Credentials of the user who runs the job (LOGIN). Applies only to jobs compatible with earlier versions.
tws.job.name	Name of the submitted job.
this.name	The integration type of the job (for example, executable). Note that this variable returns theintegration type, not the name of the job.
job.<JOB_NAME>.name	The integration type of the specifically referenced job (for example, executable).
tws.job.num	UNISON_JOBNUM.
tws.job.priority	Priority of the submitted job.
tws.job.promoted	Job is promoted. Values can be `YES` or `No`. For more information about promotion for dynamic jobs, see Promoting jobs scheduled on dynamic pools on page 162 .
tws.job.recnum	Record number of the job.
tws.job.resourcesForPromoted	Quantity of the required logical resources assigned on a dynamic pool to a promoted job. Values can be `1` if the job is promoted or `10` if the job is not promoted. For more information about promotion for dynamic jobs, see Promoting jobs scheduled on dynamic pools on page 162.
tws.job.taskstring	Task string of the submitted job. Applies only to jobs compatible with earlier versions.
tws.job.workstation	Name of the workstation on which the job is defined.
tws.jobstream.id	ID of the job stream that includes the job (UNISON_SCHED_ID).
tws.jobstream.name	Name of the job stream that includes the job (UNISON_SCHED).
tws.jobstream.workstation	Name of the workstation on which the job stream that includes the job is defined.

Table 37. Supported HCL Workload Automation variables in JSDL definitions. (continued)

Variables that can be inserted in the dynamic workload broker job definition	Description
tws.master.workstation	Name of the master domain manager (UNISON_MASTER).
tws.plan.date	Start date of the production plan (UNISON_SCHED_DATE).
tws.plan.date.epoch	Start date of the production plan, in epoch format (UNISON_SCHED_EPOCH).
tws.plan.runnumber	Run number of the production plan (UNISON_RUN).

The following example illustrates a job definition where variables are used to return both the name of the job and the plug-in type. For a job named **TEST2** of type executable, the variables in the script are resolved as follows:

- `${tws.job.name}` resolves to TEST2
- `${this.name}` resolves to executable
- `${job.TEST2.name}` resolves to executable

```
**$JOBS
WA_AGT#TEST2
 TASK
    <?xml version="1.0" encoding="UTF-8"?>
<jsdl:jobDefinition xmlns:jsdl="http://www.ibm.com/xmlns/prod/scheduling/1.0/jsdl"
xmlns:xs="http://www.w3.org/2001/XMLSchema" xmlns:jsdle="http://www.ibm.com/xmlns/prod/scheduling/1.0/jsdle"
xs:text="resolveVariableTable" name="executable">
    <jsdl:application name="executable">
        <jsdle:executable interactive="false">
            <jsdle:script>echo ${tws.job.name}
echo ${this.name}
echo ${job.TEST2.name}</jsdle:script>
        </jsdle:executable>
    </jsdl:application>
</jsdl:jobDefinition>
 RECOVERY STOP**
```

Passing variables between jobs

In many scenarios, the job output or a job property of the first job in a job stream can be the input for the execution of the successive jobs in the same job stream or in a different job stream.

In the following scenario, you have *JobA* and *JobB* in the same job stream instance and *JobA* is a predecessor of *JobB*. *JobA* passes some variables values to *JobB* at run time.

You can pass the following variables from *JobA* to *JobB*:

- *JobA* exports some properties and *JobB* references these properties in its definition as variables in a predefined format. At run time, the *JobB* variables are automatically resolved. The job properties that you can export depend on the job type you are defining. See Passing job properties from one job to another in the same job stream instance on page 270.

- *JobA* exports its standard output value and *JobB* references this standard output as a variable. At run time the *JobB* variable is automatically resolved. See Passing job standard output from one job to another in the same job stream instance on page 278.
- *Only for executable jobs. JobA* exports its standard output value and the *JobB* references this standard output as its standard input value. See Passing job standard output from one job to another as standard input in the same job stream instance on page 279.
- *Only for native and executable jobs. JobA* sets some variable values by using the jobprop utility on UNIX operating systems and jobprop.exe utility on Windows operating systems that are installed on dynamic agents. *JobB* references these variable values in its definition. At run time, the *JobB* variables are automatically resolved. See Passing variables set by using jobprop in one job to another in the same job stream instance on page 280.

In a different scenario, *JobA* exports variables in a variable table. The variable table makes the exported variables available to *JobB*, where *JobB* is any successor job, in the same job stream or in a different job stream. See Passing variables from one job to another in the same job stream or in a different job stream by using variable tables on page 281.

Note: The USERJOBS job stream that is created by HCL Workload Automation processes, does not support the passing of variables among jobs that belong to it.

Passing job properties from one job to another in the same job stream instance

Export job properties, such as job status, integration type, and rerun counter, from one dynamic job to a successor job in the same job stream instance.

The job properties that you can export from one dynamic job to a successor job in the same job stream instance depend on the job type you are defining. To add a job property within a successor job definition so that it is resolved locally on the agent at run time, use the following syntax:

```
${job:<JOB_NAME>.<property_name>}
```

where *<JOB_NAME>* is the name value or alias name value of the job from which you are exporting the property values and *<property_name>* is the property that you are referring to. The *<property_name>* value is case insensitive.

Additionally, you can use this syntax to retrieve specific cross-job metadata by using the following property names:

Job status

Retrieve the status of a job belonging to the same job stream (in the format returned by the dynamic domain manager) by specifying `.status` as the property name: `${job:<JOB_NAME>.status}`.

Integration type

Retrieve the integration type of a job belonging to the same job stream by specifying `.name` as the property name: `${job:<JOB_NAME>.name}`. Note that this property returns the specific integration type of the referenced job (for example, `executable`), and not the actual name of the job.

Rerun counter

Pass the deterministic rerun counter of a job to a successor job by referencing the `${job:<JOB_NAME>.reruncount}` property. This allows your downstream jobs and diagnostics to instantly determine how many times the predecessor job was rerun and dynamically adapt their script logic accordingly.

Consider the following scenario:

1. You create a job stream containing jobs J1 and J2, where J2 is scheduled to run after J1.
2. J1 finishes and reaches its final state. From within J2, you want to know the status of J1 at runtime.
3. You use the syntax *${job:J1.status}* inside J2, and it returns the final state of J1 in the dynamic domain manager format (for example, if it finished successfully, it displays SUCCEEDED_EXECUTION).
4. Similarly, if J1 is an executable job and you want to retrieve its integration type at run time, you use the syntax *${job:J1.name}* inside J2, and it returns executable.

Note: You cannot use the following keywords as *<property_name>* variables:

- RC
- returncode
- name
- status
- stdlist
- stduri

If you use these keywords to define custom output properties, they are not resolved and valued as expected by subsequent jobs. For example, if you specify `${job:PREVIOUSJOB.name}`, the successor job fails to resolve the variable.

Only some job types can pass property values to other successor jobs. The following types of job can export variables:

Shadow jobs

Table 38: Properties for shadow jobs on page 273 shows the list of properties that you can pass from one shadow job to another and indicates the mapping between the `Extra information` properties of the job and the properties that you can use.

IBM Sterling Connect:Direct jobs

Table 39: Properties for IBM Sterling Connect:Direct jobs on page 274 shows the list of properties that you can pass from one IBM Sterling Connect:Direct job to another and indicates the mapping between the `Extra information` properties of the job and the properties that you can use.

File transfer jobs

Table 40: Properties for file transfer jobs on page 274 shows the list of properties that you can pass from one file transfer job to another and indicates the mapping between the `Extra information` properties of the job and the properties that you can use.

Since you can use wildcards to specify a partial name condition, you can transfer more than one file within the same job, and you have one full set of properties for each transferred file.

JSR 352 Java Batch jobs

Table 41: Properties for JSR 352 Java Batch jobs on page 276 shows the list of properties that you can pass from one JSR 352 Java Batch job to another and indicates the mapping between the `Extra information` properties of the job and the properties that you can use.

Job Management jobs

Table 42: Properties for Job Management jobs on page 277 shows the list of properties that you can pass from one Job Management job to another and indicates the mapping between the `Extra information` properties of the job and the properties that you can use.

Job Stream Submission jobs

Table 43: Properties for Job Stream Submission jobs on page 277 shows the list of properties that you can pass from one Job Stream Submission job to another and indicates the mapping between the `Extra information` properties of the job and the properties that you can use.

Database jobs

Table 44: Properties for database jobs on page 277 shows the list of properties that you can pass from one database job to another and indicates the mapping between the `Extra information` properties of the job and the properties that you can use.

Example

The following example demonstrates how specifying variables in different formats allows for variables to have different values because they are resolved at different times. It also demonstrates how variables can be passed from job to job in a job stream instance. The `WIN92MAS_REW#VP_JS_141800058` job stream contains `JOBA` and `JOBB` jobs. The `JOBB` executable job references the following properties of the `JOBA` shadow job:

- `ScheduledTime`
- `dJobNAme`
- `dJobStreamName`
- `dJobStreamWorkstation`

The database definitions:

```
SCHEDULE WIN92MAS_REW#VP_JS_141800058
:
WIN92MAS_REW#JOBA
 TASK
 <?xml version="1.0" encoding="UTF-8"?>
 <jsdl:jobDefinition xmlns:dshadow=
"http://www.ibm.com/xmlns/prod/scheduling/1.0/dshadow" xmlns:
 jsdl="http://www.ibm.com/xmlns/prod/scheduling/1.0/jsdl">
 <jsdl:application name="distributedShadowJob">
 <dshadow:DistributedShadowJob>
 <dshadow:JobStream>VPJS_141800058</dshadow:JobStream>
 <dshadow:Workstation>nc125133</dshadow:Workstation>
```

```
 <dshadow:Job>VP_JOBMON_141800058</dshadow:Job>
 <dshadow:matching>
 <dshadow:previous/>
 </dshadow:matching>
 </dshadow:DistributedShadowJob>
 </jsdl:application>
 </jsdl:jobDefinition>
 DESCRIPTION "Sample Job Definition for DISTRIBUTED environment"
 RECOVERY STOP
NC125133#JOBB
 TASK
 <?xml version="1.0" encoding="UTF-8"?>
 <jsdl:jobDefinition xmlns:XMLSchema=
"http://www.w3.org/2001/XMLSchema" xmlns:jsdl="http://www.ibm.com/xmlns/
    prod/scheduling/1.0/jsdl" xmlns:
jsdle="http://www.ibm.com/xmlns/prod/scheduling/1.0/jsdle"
 XMLSchema:text="resolveVariableTable" name="executable">
 <jsdl:application name="executable">
 <jsdle:executable>
 <jsdle:script>
echo ScheduledTime:${job:JOBA.ScheduledTime}
echo JobName:${job:JOBA.dJobName}
echo JobStreamName:${job:JOBA.dJobStreamName}
echo JobStreamWorkstation:${job:JOBA.dJobStreamWorkstation}
</jsdle:script>
</jsdle:executable>
</jsdl:application>
</jsdl:jobDefinition>
 DESCRIPTION "Added by composer."
 RECOVERY STOP
 FOLLOWS JOBA

END
```

Table 38. Properties for shadow jobs

Shadow job properties that can be passed to another job definition	Shadow job Extra Information properties
${job:<*JOB_NAME*>.ScheduledTime}	Remote Job Scheduled Time
${job:<*JOB_NAME*>.dJobName}	Remote Job
${job:<*JOB_NAME*>.dJobStreamName}	Remote Job Stream
${job:<*JOB_NAME*>.dJobStreamWorkstation}	Remote Job Stream Workstation

Table 39. Properties for IBM Sterling Connect:Direct jobs

IBM Sterling Connect:Direct properties that can be passed to another job definition	IBM Sterling Connect:Direct job Extra Information properties
${job:<*JOB_NAME*>.PrimaryNodeAddress}	Primary Node Address
${job:<*JOB_NAME*>.PrimaryNodeUserID}	Primary Node User
${job:<*JOB_NAME*>.SecondaryNodeName}	Secondary Node Name
${job:<*JOB_NAME*>.SecondaryNodeUserID}	Secondary Node User
${job:<*JOB_NAME*>.ProcessName}	Process Name
${job:<*JOB_NAME*>.DestinationDisposition}	Destination Disposition
${job:<*JOB_NAME*>.ProcessFileName}	Process File Name
${job:<*JOB_NAME*>.ProcessFileLocation}	Process File Location
${job:<*JOB_NAME*>.CompressType}	Compression Type
${job:<*JOB_NAME*>.CheckPointRestart}	Check Point Restart
${job:<*JOB_NAME*>.ActionSelected}	Action Selected
${job:<*JOB_NAME*>.SourceFilePath}	Source File Path
${job:<*JOB_NAME*>.DestinationFilePath}	Destination File Path
${job:<*JOB_NAME*>.ProcessNumber}	Process Number

Table 40. Properties for file transfer jobs

File Transfer properties that can be passed to another job definition	File Transfer job Extra Information properties
${job:<*JOB_NAME*>.File.LocalFile}	File.LocalFile
${job:<*JOB_NAME*>.File.NameCollisionPolicy	File.NameCollisionPolicy
${job:<*JOB_NAME*>.File.LocalUser}	File.LocalUser
${job:<*JOB_NAME*>.File.Protocol}	File.Protocol

Table 40. Properties for file transfer jobs (continued)

File Transfer properties that can be passed to another job definition	File Transfer job Extra Information properties
`${job:<JOB_NAME>.File.RemoteFile}`	`File.RemoteFile`
`${job:<JOB_NAME>.File.RemoteUser}`	`File.RemoteUser`
`${job:<JOB_NAME>.File.Size}`	`File.Size`
`${job:<JOB_NAME>.File.TotalTransferTime}`	`File.TotalTransferTime`
`${job:<JOB_NAME>.NumberOfTransferredFiles}`	`Number of transferred files` Only one value for each job.
`${job:<JOB_NAME>.Protocol}`	`File.Protocol`
`${job:<JOB_NAME>.Direction}`	`File.Direction`
`${job:<JOB_NAME>.RemoteUser}`	`File.Remote User`
`${job:<JOB_NAME>.LocalUser}`	`File.Local User`
`${job:<JOB_NAME>.DestinationPath}`	`File.Destination Path`
`${job:<JOB_NAME>.Number of transferred files}`	`File.Number of transferred files`
`${job:<JOB_NAME>.File.src|dst.RemoteFile}`	`File.src.RemoteFile` `File.dst.RemoteFile`
`${job:<JOB_NAME>.File.src|dst.RemoteUser}`	`File.src.RemoteUser` `File.dst.RemoteUser`
`${job:<JOB_NAME>.File.counter.LocalFile}`	`File.counter.LocalFile`
`${job:<JOB_NAME>.File.counter.LocalUser}`	`File.counter.LocalUser`
`${job:<JOB_NAME>.File.counter.Protocol}`	`File.counter.Protocol`
`${job:<JOB_NAME>.File.counter.RemoteFile}`	`File.counter.RemoteFile`
`${job:<JOB_NAME>.File.counter.RemoteUser}`	`File.counter.RemoteUser`

Table 40. Properties for file transfer jobs (continued)

File Transfer properties that can be passed to another job definition	File Transfer job Extra Information properties
`${job:<JOB_NAME>.File.counter.Size}`	`File.counter.Size`
`${job:<JOB_NAME>.File.counter.TotalTransferTime}`	`File.counter.TotalTransferTime`
`${job:<JOB_NAME>.NumberOfTransferredFiles}`	`Number of transferred files` Only one value for each job.
`${job:<JOB_NAME>.File.counter.src|dst.RemoteFile}`	`File.counter.src.RemoteFile` `File.counter.dst.RemoteFile`
`${job:<JOB_NAME>.File.counter.src|dst.RemoteUser}`	`File.counter.src.RemoteUser` `File.counter.dst.RemoteUser`

Table 41. Properties for JSR 352 Java Batch jobs

JSR 352 Java Batch job properties that can be passed to another job	Label
`${job:<JOB_NAME>.ExitStatus}`	`Job Exit Status`
`${job:<JOB_NAME>.JobInstanceId}`	`Job Instance Id`
`${job:<JOB_NAME>.executionId}`	`Job Execution Id`
`${job:<JOB_NAME>.JobName}`	`Job Name`
`${job:<JOB_NAME>.JobBatchStatus}`	`Job Batch Status`
`${job:<JOB_NAME>.self}`	`Self`
`${job:<JOB_NAME>.joblogs}`	`Job Logs`
`${job:<JOB_NAME>.Jobexecution}`	`Job Execution`

The following variables are exported for each step of the job:

`${job:<JOB_NAME>.<STEP_NAME>.Name}`	`Step Name`

Table 41. Properties for JSR 352 Java Batch jobs (continued)

JSR 352 Java Batch job properties that can be passed to another job	Label
`${job:<JOB_NAME>.<STEP_NAME>.BatchStatus}`	`Step Batch Status`

Table 42. Properties for Job Management jobs

Job Management job properties that can be passed to another job	Label
${job:<JOB_NAME>. jobName}	jobname
${job:<JOB_NAME>. jobstreamId}	jobstreamid
${job:<JOB_NAME>.workstation}	workstation
${job:<JOB_NAME>.user}	userName
${job:<JOB_NAME>.url}	Url

Table 43. Properties for Job Stream Submission jobs

Job Stream Submission job properties that can be passed to another job	Label
${job:<JOB_NAME>.Url}	Url
${job:<JOB_NAME>.workstation}	Workstation
${job:<JOB_NAME>. jobstreamname}	Job Stream Name
${job:<JOB_NAME>.datetime}	Earliest Start DateTime
${job:<JOB_NAME>.variabletablename}	Variable Table Name
${job:<JOB_NAME>.variablelistValues}	Variable Table Values {"key1":value1,"key2":value2,...}

Table 44. Properties for database jobs

Database job properties that can be passed to another job	Label
${job:<JOB_NAME>.NumberOfRows}	Number of rows
${job:<JOB_NAME>.PPS_PROPERTY_NAME}	The name of the procedure expressed
${job:<JOB_NAME>.PPS_PROPERTY_VALUE}	The value of the procedure expressed

Passing job standard output from one job to another in the same job stream instance

You can export the job standard output from one dynamic job to a successive job in the same job stream instance. The job standard output variable is used in the `script` field of the job definition

To add a job standard output within another job definition, to have it resolved locally on the agent at run time, use the following syntax:

```
${job:<JOB_NAME>.stdlist}
```

where <*JOB_NAME*> is the name or the alias name of the job from which you are exporting the job standard output.

Example

In this example, the `WIN92MAS_REW#VP_JS_141800058` job stream contains `JOBA_ALIAS` that is the `JOBA` alias and `JOBB` jobs. The `JOBB` executable job references the `JOBA_ALIAS` standard output.

The database definitions:

```
SCHEDULE WIN92MAS_REW#VP_JS_141800058
:
WIN92MAS_REW#JOBA as JOBA_ALIAS
 TASK
 <?xml version="1.0" encoding="UTF-8"?>
 <jsdl:jobDefinition xmlns:dshadow=
"http://www.ibm.com/xmlns/prod/scheduling/1.0/dshadow" xmlns:
 jsdl="http://www.ibm.com/xmlns/prod/scheduling/1.0/jsdl">
 <jsdl:application name="distributedShadowJob">
 <dshadow:DistributedShadowJob>
 <dshadow:JobStream>VPJS_141800058</dshadow:JobStream>
 <dshadow:Workstation>nc125133</dshadow:Workstation>
 <dshadow:Job>VP_JOBMON_141800058</dshadow:Job>
 <dshadow:matching>
 <dshadow:previous/>
 </dshadow:matching>
 </dshadow:DistributedShadowJob>
 </jsdl:application>
 </jsdl:jobDefinition>
 DESCRIPTION "Sample Job Definition for DISTRIBUTED environment"
 RECOVERY STOP

NC125133#JOBB
 TASK
 <?xml version="1.0" encoding="UTF-8"?>
 <jsdl:jobDefinition xmlns:XMLSchema="http://www.w3.org/2001/XMLSchema" xmlns:
jsdl="http://www.ibm.com/xmlns/prod/scheduling/1.0/jsdl" xmlns:
jsdle="http://www.ibm.com/xmlns/prod/scheduling/1.0/jsdle"
XMLSchema:text="resolveVariableTable" name="executable">
<jsdl:application name="executable">
<jsdle:executable>
  <jsdle:script>echo &quot;stdlist: ${job:JOBA_ALIAS.stdlist}&quot;
    </jsdle:script>
  </jsdle:executable>
  </jsdl:application>
  </jsdl:jobDefinition>
 DESCRIPTION "Added by composer."
 RECOVERY STOP
```

```
 FOLLOWS JOBA_ALIAS
END
```

Passing job standard output from one job to another as standard input in the same job stream instance

You can export the job standard output from a dynamic job to a successive executable job as standard input in the same job stream instance. The job standard output variable is used in the `input` field of the executable job definition

To add a job standard output within another executable job definition, to have it resolved locally on the agent at run time, use the following syntax:

```
${job:<JOB_NAME>.stduri}
```

where <*JOB_NAME*> is the name value or alias name value of the job from which you are exporting the job standard output.

Note: The *stduri* variable passing is not supported for shadow jobs. Because shadow jobs do not produce a job log, if you pass the job stdout variable as input to another job in the same job stream, the output of the job is empty. A shadow job can only print a status message.

Example

In this example, the `NC112019#JS_PROP` job stream contains the `JOBALIAS_A` that is the `NC112019#JOBA` alias and `NC112019#JOBB` jobs. The `NC112019#JOBB` executable job references the `JOBALIAS_A` standard output as standard input.

The database definitions:

```
SCHEDULE NC112019#JS_PROP
:
NC112019#JOBA AS JOBALIAS_A
 TASK
 <?xml version="1.0" encoding="UTF-8"?>
 <jsdl:jobDefinition xmlns:jsdl="http://www.ibm.com/xmlns/prod/scheduling/
  1.0/jsdl" xmlns:
jsdle="http://www.ibm.com/xmlns/prod/scheduling/1.0/jsdle" name="executable">
<jsdl:application name="executable">
<jsdle:executable interactive="false" path="ls"/>
</jsdl:application>
</jsdl:jobDefinition>
 DESCRIPTION "Added by composer for job stream: NC112019#JS_PROP."
 RECOVERY STOP

NC112019#JOBB
 TASK
 <?xml version="1.0" encoding="UTF-8"?>
 <jsdl:jobDefinition xmlns:XMLSchema="http://www.w3.org/2001/XMLSchema"
xmlns:jsdl="http://www.ibm.com/xmlns/prod/scheduling/1.0/jsdl" xmlns:
jsdle="http://www.ibm.com/xmlns/prod/scheduling/1.0/jsdle" XMLSchema:text=
   "resolveVariableTable"
         name="executable">
         <jsdl:application name="executable">
  <jsdle:executable input="${job:JOBALIAS_A.stduri}"
       interactive="false" path="cat"/>
  </jsdl:application>
```

```
  </jsdl:jobDefinition>
 DESCRIPTION "Added by composer for job stream: WIN92MAS#JS_PROP."
 RECOVERY STOP
 FOLLOWS JOBALIAS_A
END
```

Passing variables set by using jobprop in one job to another in the same job stream instance

You can use the jobprop utility installed on dynamic agents to set variable and its value in a job and pass the variable to the successive job in the same job stream instance.

To set variable and its value in the first job use the following syntax:

```
jobprop <VAR-NAME> <value>
```

where <*VAR-NAME*> is the variable that you can export into another job and `<value>` is the value assigned to the <*VAR-NAME*>. For more information about jobprop utility, see jobprop on page 987.

To define the variables in another job use the following syntax:

```
${job:<JOB_NAME>.<VAR-NAME>}
```

where <*JOB_NAME*> is the name value or alias name value of the job from which you are exporting the <*VAR-NAME*> variable value.

Example

In this example, the `WIN92MAS#JS_PROP` job stream contains `NC125133#JOBA` and `NC125133#JOBB` executable jobs. The `NC125133#JOBB` job references the following `NC125133#JOBA` variable values set by using the jobprop utility:

- *VAR1* variable set to `value1` value.
- *VAR2* variable set to `value2` value.
- *VAR3* variable set to `value3` value.
- *VAR4* variable set to `value4` value.

The database definitions:

```
SCHEDULE WIN92MAS#JS_PROP
                  :
                  NC125133#JOBA
                  TASK
                  <?xml version="1.0" encoding="UTF-8"?>
                  <jsdl:jobDefinition xmlns:jsdl="http://www.ibm.com/xmlns/prod/scheduling/1.0/jsdl"
 xmlns:jsdle="http://www.ibm.com/xmlns/prod/scheduling/1.0/jsdle">
                  <jsdl:application name="executable">
                  <jsdle:executable interactive="false">
                  <jsdle:script>#!/bin/sh
                  . /home/ITAuser/TWA/TWS/tws_env.sh
                  jobprop VAR1 value1
                  jobprop VAR2 value2
                  jobprop VAR3 value3
                  jobprop VAR4 value4
                  </jsdle:script>
```

```
                </jsdle:executable>
                </jsdl:application>
                </jsdl:jobDefinition>
                DESCRIPTION "Sample Job Definition"
                RCCONDSUCC "RC>=0"
                RECOVERY STOP

                NC125133#JOBB
                TASK
                <?xml version="1.0" encoding="UTF-8"?>
                <jsdl:jobDefinition xmlns:jsdl="http://www.ibm.com/xmlns/prod/scheduling/1.0/jsdl"
xmlns:jsdle="http://www.ibm.com/xmlns/prod/scheduling/1.0/jsdle">
                <jsdl:application name="executable">
                <jsdle:executable interactive="false">
                <jsdle:script>
                echo VAR1=${job:joba.VAR1}
                echo VAR2=${job:joba.VAR2}
                echo VAR3=${job:joba.VAR3}
                echo VAR4=${job:joba.VAR4}
                </jsdle:script>
                </jsdle:executable>
                </jsdl:application>
                </jsdl:jobDefinition>
                DESCRIPTION "Sample Job Definition"
                RCCONDSUCC "RC>=0"
                RECOVERY STOP
                FOLLOWS JOBA
                END
```

Passing variables from one job to another in the same job stream or in a different job stream by using variable tables

You can use variable tables to set variables exported from a job, and pass the variables to any successor job, in the same job stream or in a different job stream.

To export variables from a job into a variable table, an ad-hoc HCL Workload Automation job type is available: the VariableTable job type. The VariableTable job must be added to the job stream as a successor of the job that is exporting variables. The VariableTable job sets the exported variables in a variable table and makes them available to any other successor job, in the same job stream or in a different job stream.

You can easily define a VariableTable job by using the Dynamic Workload Console or composer command line.

User definition

The user names that are used as the **streamlogon** value for Windows® job definitions must have user definitions. This is not required for users who run jobs on other operating systems. If you are using job types with advanced options, you can use these values regardless of the operating system. For more information, see Using user definitions on job types with advanced options on page 285.

Note: If you have the `enAddUser` global option set to `"yes"`, the user definition is automatically added to the plan after you create or modify the user definition in the database.

Each user definition has the following format and arguments:

Syntax

```
username [workstation#] [domain\]username[@internet_domain]
password "password"
end
```

Arguments

username

[folder/] [workstation#]username

[folder/]workstation

Specifies the workstation on which the user is allowed to launch jobs and the folder where the workstation is defined, if any. The # symbol is required. The default is blank, meaning all workstations.

username

Specifies the name of the Windows user. The *username* field value can contain up to 47 characters.

[folder/] [workstation#]domain\username

[folder/]workstation

Specifies the workstation on which the user is allowed to launch jobs and the folder where the workstation is defined, if any. The # symbol is required. The default is blank, meaning all workstations.

domain\username

Specifies the name of the Windows® domain user. The *domain\username* field value can contain up to 47 characters.

[folder/] [workstation#]username@internet_domain

[folder/]workstation

Specifies the workstation on which the user is allowed to launch jobs and the folder where the workstation is defined, if any. The # symbol is required. The default is blank, meaning all workstations.

username@internet_domain

Specifies the name of the user in User Principal Name (UPN) format. UPN format is the name of a system user in an email address format. The user name is followed by the at symbol followed by the name of the Internet domain with which the user

is associated. The *username@internet_domain* field value can contain up to 47 characters.

Note:

If you define a user for **Windows® operating systems**:

- User names are case-sensitive. Also, the user must be authorized to log on to the workstation on which HCL Workload Automation launches jobs, and have the permission to **Log on as batch**.
- If the user name is not unique, it is taken to mean a local user, a domain user, or a trusted domain user, in that order.

password

Specifies the user password. The password can contain up to 31 characters, and must be enclosed in double quotation marks. To indicate a null password, use two consecutive double quotation marks with no blanks in between, ". When a user definition has been saved, you cannot read the password. Users with appropriate security privileges can modify or delete a user, but password information is never displayed.

Example

Examples

The following example defines four users:

```
username joe
     password "okidoki"
end
#
username server#jane
     password "okitay"
end
#
username dom1\jane
     password "righto"
end
#
username jack
     password ""
end
#
username administrator@twsbvt.com
     password "internetpwd"
end
#
username serverA#dom1\jack
     password "righto"
end
#
username serverB#user1@twsbvt.com
```

```
     password "internetpwd"
end
#
```

Comments

Passwords extracted with the `composer extract` command are of limited use. When you run the `composer extract` command on a user definition, the password is obfuscated with the "`**********`" reserved keyword. If you try running the composer `import`, `replace`, or `modify` commands on an extracted user password, the password replacement has no effect and the old password is maintained. Also, if you try running the composer `create`, `new`, or `add` commands on a user where the password equals the "`**********`" reserved keyword, the following error is returned:

```
AWSJCL521E The password specified for the Windows user "USER_NAME" does
not comply with password security policy requirements.
```

Note that the reserved keyword is a string of ten asterisks (*). You cannot enter a sequence of ten asterisks as a password, but you can have a password with any other number of asterisks.

To fix this problem, make sure you run the `composer extract` with the ;password on page 472 option.

See also

For more information about how to perform the same task from the Dynamic Workload Console, see:

the Dynamic Workload Console User's Guide, "Designing your Workload" section.

Using the HCL Workload Automation user and streamlogon definitions

On Windows® workstations, user definitions are specified using **composer** in the form ***[workstation#]username***. The instance ***[workstation#]username*** uniquely identifies the Windows® user in the HCL Workload Automation environment. The workstation name is optional; its absence indicates that the user named *username* is defined on all the Windows® workstations in the HCL Workload Automation network. If the user named *username* is only defined on some Windows® workstations in the HCL Workload Automation network, to avoid inconsistencies, you must create a user definition [*workstation#*]*username* for each workstation running on Windows® where the user *username* is defined.

If you schedule a job on an agent, on a pool or a dynamic pool, the job runs with the user defined on the pool or dynamic pool. However, the user must exist on all workstations in the pool or dynamic pool where you plan to run the job.

When you define a job using **composer**, you must specify both a workstation and a valid user logon for the workstation. The logon is just a valid user name for Windows®, without the workstation name. For example, in the following job definition:

```
$JOB
workstation#job01 docommand "dir"
streamlogon username
```

the value for `streamlogon` is *username* and not *workstation#username*.

However, when you use the **altpass** command, you must use the user definition in the format

```
workstation#username
```

For this command, you can omit the workstation name only when changing the password of the workstation from where you are running the command.

Trusted domain user

If HCL Workload Automation is to launch jobs for a trusted domain user, follow these guidelines when defining the user accounts. Assuming HCL Workload Automation is installed in `Domain1` for user account `maestro`, and user account `sue` in `Domain2` needs to launch a job, the following must be true:

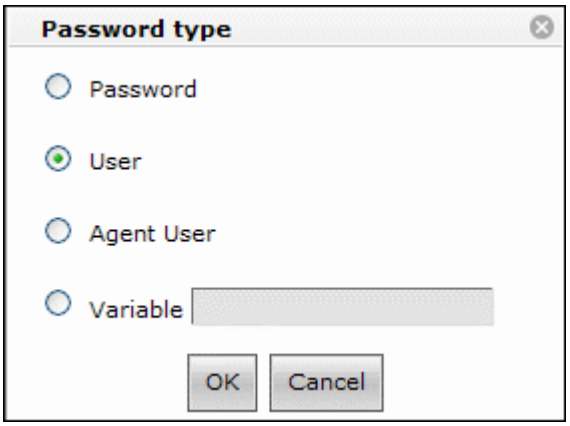
- There must be mutual trust between `Domain1` and `Domain2`.
- In `Domain1` on the computers where jobs are launched, `sue` must have the right to **Log on as batch**.
- In `Domain1`, `maestro` must be a domain user.
- On the domain controllers in `Domain2`, `maestro` must have the right to **Access this computer from network**.

Using user definitions on job types with advanced options

On job types with advanced options, regardless of the operating system of the dynamic agent that will run the job, you can provide the username of a user definition in the credentials section of the job and have the password field resolved at runtime with the password value stored in the database.

For example, when you define the job with the Dynamic Workload Console, you enter the username of a user definition and click the ellipsis (...) located next to the password field to display the following `Password type` widget:

Figure 24. User definition



where you select `User` as shown. You can likewise code this option in the task section (JSDL) of the job definition in composer. See the related sections for more information.

To be able to use this option when you define a job, you need to be authorized in the security file with the `use` access keyword for object type `userobj`, that is:

```
userobj    access=use
```

HCL Workload Automation follows this sequence when it is called to resolve the username and the password at runtime:

- If the workstation is not specified (for example, `${password:myuser}`):
    1. Searches `myuser` on the workstation running the job applying a case sensitive policy.
    2. Searches `myuser` on the workstation running the job applying a case insensitive policy.
    3. Searches `myuser` without an associated workstation applying a case sensitive policy.
    4. Searches `myuser` without an associated workstation applying a case insensitive policy.
- If the workstation is specified (for example, `${password:agent#myuser}`):
    1. Searches `myuser` on workstation `agent` applying a case sensitive policy.
    2. Searches `myuser` on workstation `agent` applying a case insensitive policy.
    3. Searches `myuser` without an associated workstation applying a case sensitive policy.
    4. Searches `myuser` without an associated workstation applying a case insensitive policy.
- If the workstation is specified but is empty (for example, `${password:#myuser}`):
    1. Searches `myuser` without an associated workstation applying a case sensitive policy.
    2. Searches `myuser` without an associated workstation applying a case insensitive policy.

Attention: User definitions lack referential integrity. This implies that, if a user definition referenced in the credentials section of a job type with advanced options is changed or deleted, no warning or error message is returned until the job is run.

Calendar definition

A calendar is a list of dates which define if and when a job stream runs.

Syntax

$calendar

[folder/]calendarname ["*description*"]

date [...]

[*calendarname* ...]

Arguments

[folder/]calendarname

Specifies the name of the calendar. For calendar definitions defined in the root (/) folder, the name can contain up to 8 alphanumeric characters. For calendar definitions defined in a folder different from the root, the name can contain up to 100 characters. The character limit includes dashes (-) and underscores (_), and the name must start with a letter.

"description"

Provides a description of the calendar. The description can contain up to 120 alphanumeric characters. It must be enclosed in double quotation marks. It can contain alphanumeric characters as long as it starts with a letter. It can contain the following characters: comma (,), period (.), dash (-), plus (+), single quote ('), and equal (=). It cannot contain double quotation marks (") other than the enclosing ones, colon (:), semi-colon (;), and ampersand (&).

***date* [...]**

Specifies one or more dates, separated by spaces. The format is *mm*/*dd*/*yy*.

Example

Examples

The following example defines three calendars named `monthend`, `paydays`, and `holidays`:

```
$calendar
monthend "Month end dates 1st half 2023"
     01/31/2023 02/28/2023 03/31/2023 04/30/2023 05/31/2023 06/30/2023
paydays
     01/15/2023 02/15/2023
     03/15/2023 04/15/2023
     05/14/2023 06/15/2023
holidays
     01/01/2023 02/15/2023 05/31/2023
```

See also

For more information about how to perform the same task from the Dynamic Workload Console, see:

the Dynamic Workload Console User's Guide, "Designing your Workload" section.

Folder definition

A smart workflow folder is a container of jobs, job streams, and other folders. Use smart workflow folders to organize your jobs and job streams according to your lines of business or other custom categories. A folder can contain one or more jobs or job streams, while each job stream can be associated to one folder. If no folder is defined for a job stream, the root folder (/) is used by default.

You can move jobs and job streams to specified folders using the composer rename command. This is especially useful if your jobs and job streams use a naming convention. You can rename and move these objects into a specified folder path in batch mode where, parts of the job and job stream names are mapped to the folder names.

When a job stream defined in a folder is scheduled and submitted to be run, the information about the folder is maintained in the plan. Jobs in the plan, however, make reference to the folder of the job stream and not the folder in which the job definition was originally defined. If a job stream is not specified when submitting, then the default JOBS job stream is considered. The folder of the JOBS job stream is the root folder (/).

Each folder definition has the following format and arguments:

Syntax

folder *foldername*
end
folder *foldername*/*foldername*
end

Arguments

folder *foldername*

A folder is a container of jobs, job streams, or sub-folders and has a tree-like structure similar to a file system. The folder name is an alphanumeric string that cannot start with a "-" (hyphen), but it can contain the following characters: "/" (forward slash), "-" (hyphen), and "_" (underscore). It cannot contain spaces. If an object is not defined in a folder, then the default folder "/" is used. If you specify an absolute path, include a "/" forward slash before the folder name. A forward slash is not necessary for relative paths. The maximum length for the full folder path (that is, the path name including the parent folder and all nested sub-folders) is 800 characters, while each folder name supports a maximum of 256 characters. Wildcard characters are supported. A single scheduling object belongs to one folder only, but each folder can contain numerous objects.

There are a number of convenient commands dedicated to managing folders:

Table 45. Folder commands

Command		Description
Composer	**Conman**	
CHFOLDER on page 457	CHFOLDER on page 600	Changes the working directory or the current folder.
LISTFOLDER on page 488	LISTFOLDER on page 623	Lists all folders in the root or in a folder.

Table 45. Folder commands (continued)

Command		Description
Composer	**Conman**	
MKFOLDER on page 493	-	Creates a new folder in the database.
RMFOLDER on page 505	-	Deletes a folder defined in the database.
RENAMEFOLDER on page 510	-	Renames a folder definition in the database.

Example

Examples

The following example defines a folder named `Europe`, and two folders nested within each other: `England` and `London`:

```
folder Europe
end
folder Europe/England
end
folder Europe/England/London
end
```

In this example, the folder "Europe" and the sub-folders "England" and "London" are created in the same directory path from where you launched composer.

See also

From the Dynamic Workload Console, you can perform the same task as described in:

the Dynamic Workload Console User's Guide, section about designing folders.

Variable and parameter definition

Variables and parameters are objects to which you assign different values.

Variables and parameters are useful when you have values that change depending on your job streams and jobs. Job stream, job, and prompt definitions that use them are updated automatically either at the start of the production cycle or at the time the job runs depending on the format used when the variable is specified.

Use variables and parameters as substitutes for repetitive values when defining prompts, jobs, and job streams. For example, using variables for user logon and script file names in job definitions and for file and prompt dependencies permits the use of values that can be maintained centrally in the database on the master.

While variables are scheduling objects that are defined in the HCL Workload Automation database and can be used by any authorized users in the domain, parameters are defined and used locally on individual agents.

You can define one or more variables whose value contains another variable to allow dynamic variable substitution at run time. This feature makes your definitions more flexible, maintainable, and reusable.

You can nest any number of variables: the variables are resolved at run time in a recursive pattern, until they are all resolved. If a variable is not set, it is ignored and HCL Workload Automation proceeds to the next variable, until the resolution process is completed.

Note: The following limitations apply:

- This functionality is supported only for integrations running on dynamic agents. It is not supported on native job types.
- The caret symbol (^) is not supported if you define the variable within a variable table. When defining nested variables within a variable table, you must use the `${variablename}` syntax. However, both the caret symbol (**^**) and the **${variablename}** syntax are supported when you reference the variable in the job definition of an integration running on a dynamic agent.

The following sections describe variables and parameters in detail.

Variables

Variables are defined as scheduling objects in the database. Variables can be defined individually with the following command:

$parm
[*tablename.*][*folder/]variablename "variablevalue"*
...

where:

tablename

Is the name of the variable table that is to contain the new variable. The variable table must be already defined. If you do not specify a variable table name, the variable is added to the default table.

[folder/]variablename

Is the name of the variable, optionally preceded by the folder name within which the variable table is defined. The variable table name can contain up to 64 alphanumeric characters, including dashes (-) and underscores (_), and must start with a letter. Do not use the following reserved keywords as variable names: `RC`, `returncode`, `name`, `status`, `stdlist`, and `stduri`.

value

Is the value assigned to the variable. The value can contain up to 1024 alphanumeric characters. Do not include the names of other variables. You can define one or more variables whose value contains another variable to allow dynamic variable substitution at run time. Variables defined within a variable table can be referenced by other variable table entries. These embedded variables are then resolved at run time and used in the jobs themselves.

However, the recommended way to define variables is to use a Variable table definition on page 295. In any case, all variables are placed in a variable table. If you define a variable and do not specify the name of a variable table, it is included in the default variable table.

Variables can be used in job and job stream definitions. They are resolved; that is, they are replaced with their assigned value when the production plan is generated or extended, or when you submit a job or a job stream for running. The format used to specify a variable also determines when the variable is resolved with a value. The following formats can be used when specifying a variable:

^*variablename*^

Specify the variable in this format if you want it resolved when the plan is generated or extended. The caret symbol (^) is not supported for nested variables. For this reason, nested variables are not supported on fault-tolerant agents.

${*variablename*}

Specify the variable in this format if you want it resolved or overwritten when the job or job stream is submitted to be run. An option in the job definition that indicates to resolve variables at the job run time must also be specified. If this variable is present only in the default variable table, the variable cannot be resolved. See an example of an application of this kind of variable in the section Examples on page 293.

Attention: When submitting a job stream from the Self-Service Catalog that contains variables or that has a variable table associated to it, variables specified in this format, ${*variablename*}, are not supported. They must be specified in the ^*variablename*^ format.

For details on variable resolution, see Variable resolution on page 166.

The variable names specified in these definitions are first resolved against variable table definitions and then on local parameters if the variables are not found.

When you specify a variable, enclose the entire string containing the variable in quotation marks `(" ")`.

If the variable contains a portion of a path, ensure that the caret characters are not immediately preceded by a backslash (\) because, in that case, the `\^` sequence could be wrongly interpreted as an escape sequence and resolved by the parser as caret character. If necessary, move the backslash into the definition of the variable between carets to avoid bad interpretation of the backslash character. For example, the following table shows the correct way for defining and using a variable named `MYDIR` in the default variable table:

Table 46. How to handle a backslash in variable substitution

Wrong way	Right way

1. Define the `MYDIR` variable as:
`$PARM`
`MYDIR "scripts"`
2. Use it in this way:
	1. Define the `MYDIR` variable as:
	`$PARM`
	`MYDIR "\scripts"`
	2. Use it in this way:

Table 46. How to handle a backslash in variable substitution (continued)

Wrong way	Right way
`job01 scriptname "c:\operid\^MYDIR^\test.cmd"`	`job01 scriptname "c:\operid^MYDIR^\test.cmd"`
3. Use it in this way: `job01 scriptname "c:\operid\${MYDIR}\test.cmd"`	3. Use it in this way: `job01 scriptname "c:\operid${MYDIR}\test.cmd"`

This is true for all command line commands, graphical user interfaces, and APIs through which you use variable substitution.

Parameters

Local parameters are defined in a local database on the workstation where the jobs using them will run. To define them, you do not use this **composer** command but the parms on page 958 utility command.

Local parameters can be used in:

- JCL
- Log on
- Prompts dependencies
- File dependencies
- Recovery prompts

A local parameter is defined within these keywords or from within the invoked job script using the following syntax:

```
'bin\parms PARAMETERNAME'
```

Local parameters are resolved using the definitions stored in the local PARMS database as follows:

- At run time on the workstation where job processing occurs.
- At submission time on the workstation where the job or job stream is submitted from the **conman** command line. Table 47: Keywords that can take local parameters in submit commands on page 292 summarizes in which submit command keyword you can use parameters.

Table 47. Keywords that can take local parameters in submit commands

Keyword	submit docommand (sbd command)	submit file (sbf command)	submit job (sbj command)	submit job stream (sbs command)
abendprompt	✓	✓	✓	
scriptname		✓		
docommand	✓			
logon	✓	✓		

Keyword	submit docommand (sbd command)	submit file (sbf command)	submit job (sbj command)	submit job stream (sbs command)
opens	✓	✓	✓	✓
prompt	✓	✓	✓	✓

For more information on how to submit jobs and job streams in production from the **conman** command line refer to Managing objects in the plan - conman on page 545.

On UNIX, when you define a job or job stream in the database, you must enclose the string

```
path/parms parametername
```

between ' ' characters to ensure the parameter is solved at run time on the workstation even if a parameter with the same name is defined as a global parameter in the HCL Workload Automation database. For example, if you add to the database the following job definition:

```
$jobs
myjob
  docommand "ls ^MYDIR^"
  streamlogon "^MYUSER^"
```

and two parameters named `MYDIR` and `MYUSER` are defined in the database, then, as the production plan is created or extended, the two parameters are resolved using the definitions contained in the database and their corresponding values are carried with the `Symphony` file. If you define in the database `myjob` as follows:

```
$jobs
myjob
  docommand "ls 'bin/parms MYDIR'"
  streamlogon "'bin MYUSER'"
```

then as the production plan is created or extended the only action that is performed against the two parameters in the definition of `myjob` is the removal of the ' ' characters, the parameters are carried in the `Symphony` file unresolved and then are resolved at run time locally on the target workstation using the value stored in the PARMS database.

Example

Examples

Two parameters, `glpah` and `gllogon`, are defined as follows:

```
$parm
glpath     "/glfiles/daily"
gllogon     "gluser"
```

The `glpath` and `gllogon` parameters are used in the `gljob2` job of the`glsched` job stream:

```
schedule glsched on weekdays
:
gljob2
     scriptname "/usr/gl^glpath^"
     streamlogon "^gllogon^"
     opens "^glpath^/datafile"
```

```
     prompt ":^glpath^ started by ^gllogon^"
end
```

An example of a variable used with the **docommand** keyword is:

```
docommand "ls ^MY_HOME^"
```

The following example demonstrates how specifying variables in different formats allow for variables to have different values because they are resolved at different times. It also demonstrates how variables can be passed from job to job in a job stream. The variable, *SWITCH_VAR* is defined in the variable table, STATETABLE, with an initial default value of `on`. The job, UPDATE1, is responsible for changing the value of the *SWITCH_VAR* variable in the STATETABLE variable table to `off`. The job stream PROCJS contains two identical jobs, PROC1 and PROC2, in which the *SWITCH_VAR* variable has been specified in two different formats. The first sets off the variable with the caret (^) symbol ^*var_name*^, and the second, uses the format ${*var_name*}:

```
<jsdle:script>echo ^SWITCH_VAR^:${SWITCH_VAR}</jsdle:script>
```

The order in which these jobs run is the following:

```
SCHEDULE NC117126#PROCJS
VARTABLE STATETABLE
:
NC117126_1# PROC1

NC117126_1# PROC2
 FOLLOWS UPDATE1

NC117126_1# UPDATE1
 FOLLOWS  PROC1
END
```

When the job stream is added to the plan, *SWITCH_VAR*, defined in both PROC1 and PROC2, immediately assumes the default value assigned in the variable table, `on`. When the job stream is submitted to be run, the first job to be submitted is PROC1 and the variable defined as *SWITCH_VAR* is resolved to `on` so that the variables in the PROC1 job are resolved as:

```
<jsdle:scriptecho>on:onjsdle:script>echo on:on</jsdle:script>
```

UPDATE1 then runs setting the value of *SWITCH_VAR* in the variable table to `off` so that when PROC2 runs, the variables are resolved as:

```
<jsdle:script>echo on:onjsdle:script>echo on:off </jsdle:script>
```

The variable specified as ^*SWITCH_VAR*^ in the job maintains the value of `on` because variables in this format are resolved when the job stream is added to the plan and are not refreshed when the job is submitted to run. Instead, the variable specified in the format, *${SWITCH_VAR}*, which was previously set to `on` is now updated with the new value in the variable table `off`.

Creating a variable definition using the Dynamic Workload Console

To create a variable definition in the Dynamic Workload Console, you must add it to a variable table definition:

1. Click **HCL Workload Automation**&→**; Workload**&→**; Design**&→**; Create Workload Definitions**
2. Select an engine name and click **Go**

3. Open in edit mode an existing variable table from the Quick Open pane, or create a new variable table as described in Variable table definition on page 295
4. In the Properties - Variable Table panel, click the Variables tab and add new variable definitions by clicking the "+" (Add) icon and specifying variable names and values

For more information, see Customizing your workload using variable tables on page 164.

Variable table definition

A variable table is an object that groups multiple variables. All the global parameters (now named *variables*) that you use in workload scheduling are contained in at least one variable table. Two ways of defining variables are available:

- Define them when you define a variable table in the way described here. This is the recommended way.
- Define them individually with the **composer** **$parm on page 289** command in the [*tablename.*]*variablename* "*variablevalue*" format. If you do not specify a table name, the new variable is placed in the default variable table.

You are not forced to create variable tables to be able to create and use variables. You might never create a table and never use one explicitly. In any case, the scheduler provides a default table and every time you create or manage a variable without naming the table, it stores it or looks for it there.

You can define more than one variable with the same name but different value and place them in different tables. Using variable tables you assign different values to the same variable and therefore reuse the same variable in job definitions and when defining prompts and file dependencies. Variable tables can be assigned at run cycle, job stream, and workstation level.

Variable tables can be particularly useful in job definitions when a job definition is used as a template for a job that belongs to more than one job stream. For example, you can assign different values to the same variable and reuse the same job definition in different job streams.

You can define one or more variables whose value contains another variable to allow dynamic variable substitution at run time. This feature makes your definitions more flexible, maintainable, and reusable.

You can nest any number of variables: the variables are resolved at run time in a recursive pattern, until they are all resolved. If a variable is not set, it is ignored and HCL Workload Automation proceeds to the next variable, until the resolution process is completed.

Note: The following limitations apply:

- This functionality is supported only for integrations running on dynamic agents. It is not supported on native job types.
- The caret symbol (^) is not supported if you define the variable within a variable table. When defining nested variables within a variable table, you must use the `${variablename}` syntax. However, both the caret symbol (**^**) and the **${variablename}** syntax are supported when you reference the variable in the job definition of an integration running on a dynamic agent.

When creating a scheduling object, you can define it in a folder. If no folder path is specified, then the object definition is created in the current folder. By default, the current folder is the root (\) folder, but you can customize it to a different folder path. You can also use the composer rename command to move and rename objects in batch mode that use a naming convention to a different folder using part of the object name to assign a name to the object.

Syntax

vartable *[folder/]tablename*
[**description** "*description*"]
[**isdefault**]
members
[*variablename* "*variablevalue*"]
...
[*variablename* "*variablevalue*"]
end

Arguments

vartable *[folder/]tablename*

The name of the variable table and, optionally, the name of the folder within which the variable table is defined. The name must start with a letter, and can contain alphanumeric characters, dashes, and underscores. It can contain up to 80 characters.

description "*tabledescription*"

The description of the variable table. The text must be enclosed within double quotation marks. The description can contain up to 120 alphanumeric characters. It cannot contain double quotation marks (") other than the enclosing ones, colon (:), semicolon (;), and ampersand (&).

isdefault

When specified, the table is the default table. You cannot mark more than one table as the default table. When you mark a variable table as the default variable table, the current variable table is no longer the default one. When migrating the database from a previous version, the product creates the default variable table with all the variables already defined.

members *variablename "variablevalue"*

The list of variables and their values separated by spaces. The name can contain up to 64 alphanumeric characters, including dashes (-) and underscores (_), and must start with a letter. The value can contain up to 1024 alphanumeric characters. Values must be enclosed within double quotation marks.

Example

The following example shows a variable table and its contents.

```
VARTABLE TEST1
  MEMBERS
   DEVBATCH "DOMD\IMSBATCH\SAME"
   PARAM_01 "date"
   PARAM_02 "root"
   PARM_01 "PARM_001"
   PRPT_02 "PARM_002"
   PRPT_03 "PARM_003"
   PRPT_04 "PARM_004"
   PRPT_05 "PARM_005"
   SAME17 "test/for/variable with samename > variable/table"
   SLAV10 "/nfsdir/billingprod/crmb/MAESTRO_JOB/AG82STGGDWHSCART"
   SLAV11 "/nfsdir/billingprod/crmb/MAESTRO_JOB/AG82CDMGALLBCV"
   SLAV12 "/nfsdir/billingprod/crmb/MAESTRO_JOB/AG82CDMGRISCTRAF"
   SLAV13 "/opt/crm/DWH_OK/Businness_Copy_ok"
   SLAV14 "/opt/crm/DWH_OK/DW_Canc_Cust_Gior_ok_"
   TRIGGER "/usr/local/samejobtriggers"
   VFILE2 "testforvarwithsamename2.sh"
   VUSER2 "same_user2"
   WRAPPER "/usr/local/sbin/same/phi_job.ksh"
END
```

Security file considerations

From the standpoint of security file authorizations, permission to act on the variable entries contained in a variable table is dependent on the overall permission granted on the variable table, as shown in following table.

Table 48. Required access keyword on variable table in Security file (vartable object) and allowed actions.

Required security file access keyword on enclosing variable table	Allowed action on listed variable entries
Modify	Add
	Delete
	Modify
	Rename
Display	Display
Unlock	Unlock

See also

For more information about how to perform the same task from the Dynamic Workload Console, see:

the Dynamic Workload Console User's Guide, "Designing your Workload" section.

Prompt definition

A prompt identifies a textual message that is displayed to the operator and halts processing of the job or job stream until an affirmative answer is replied (either manually by the operator or automatically by an event rule action). After the prompt is replied to, processing continues. You can use prompts as dependencies in jobs and job streams. You can use variables in prompts.

Note: Predefined or global prompt definitions are reset each time the **JnextPlan** job is run.

Syntax

$prompt

[folder/]promptname "[**:** | **!**]*text*?

[*promptname* ...]

Arguments

[folder/]promptname

Specifies the name of the prompt, optionally preceded by the folder within which the prompt is defined. The prompt name can contain up to 8 alphanumeric characters, including dashes (-) and underscores (_), and must start with a letter.

text

Provides the text of the prompt. The text of the prompt can contain up to two hundred alphanumeric characters. Based on the character preceding the text, the prompt can behave differently:

- If the text begins with a colon (:), the prompt is displayed, but no reply is required to continue processing.
- If the text begins with an exclamation mark (!), the prompt is displayed, but it is not recorded in the log file.

You can use one or more parameters as part or all of the text string for a prompt. If you use a parameter, the parameter string must be enclosed in carets (^). See Variable and parameter definition on page 289 for an example.

Note: Within local prompts, carets (^) not identifying a parameter, must be preceded by a backslash (\) to prevent them from causing errors in the prompt. Within global prompts, carets do not have to be preceded by a backslash.

You can include backslash n (**\n**) within the text to create a new line.

Example

Examples

The following example defines three prompts:

```
$prompt
     prmt1 "ready for job4? (y/n)"
     prmt2 ":job4 launched"
     prmt3 "!continue?"
```

See also

For more information about how to perform the same task from the Dynamic Workload Console, see:

the Dynamic Workload Console User's Guide, "Designing your Workload" section.

Resource definition

Resources represent physical or logical scheduling resources that can be used as dependencies for jobs and job streams. Resources can be used as dependencies only by jobs and job streams that run on the workstation where the resource is defined.

Syntax

$resource
[folder/]workstation#*[folder/]resourcename units* ["*description*?]

[*[folder/]workstation*#*[folder/]resourcename* ...]

Arguments

[folder/]workstation

Specifies the name of the workstation or workstation class on which the resource is used, optionally preceded by the folder name within which the workstation or workstation class is defined.

[folder/]resourcename

Specifies the name of the resource, optionally preceded by the folder name within which the resource is defined. The resource name can contain up to eight alphanumeric characters, including dashes (-) and underscores (_), and must start with a letter.

units

Specifies the number of available resource units. Values can be **0** through **1024**.

"description"?

Provides a description of the resource. The description can contain up to 120 alphanumeric characters. It must be enclosed in double quotation marks.

The resource units involved in needs dependencies for a job or for a job stream remain busy until the job or job stream is completed (successfully or not). The resource units are released as soon as the job or job stream is completed.

When multiple jobs and job streams depend on the same resource, if not enough resource units are currently available for all of them it is assigned according to the job or job stream priority. The status of a job or job stream becomes READY as soon as all its dependencies are resolved. If the limit CPU set on the workstation does not allow it to run at the moment, it waits in READY state. The only exception to this behavior is when the job or job stream is GO or HI, in which case it starts regardless of the value set for limit CPU.

Example

Examples

The following example defines four resources:

```
$resource
  ux1#tapes  3 "tape units"
  ux1#jobslots  24 "job slots"
  ux2#tapes  2 "tape units"
  ux2#jobslots  16 "job slots"
```

See also

For more information about how to perform the same task from the Dynamic Workload Console, see:

the Dynamic Workload Console User's Guide, "Designing your Workload" section.

Run cycle group definition

A run cycle group is a database object within which one or more run cycles are defined. The run cycles combined together produce a set of run dates for a job stream. A run cycle group can contain:

- *Inclusive* run cycles that specify when a job stream must run. The keywords defining the run cycle follow the `on` keyword.
- *Exclusive* run cycles that specify when a job stream is not to run. The keywords defining the run cycle follow the `except` keyword. Usually an *exclusive* run cycle is matched against an *inclusive* one to define specific dates when the job stream is exempted from running.

Each of the run cycles includes its own definition keywords. Some of these keywords can also be defined at run cycle group-level. When a run cycle and the run cycle group include the same keyword, the value in the run cycle definition is used for the run cycle. When the run cycle definition omits a certain keyword that is defined at run cycle group-level, it inherits the value.

When creating a scheduling object, you can define it in a folder. If no folder path is specified, then the object definition is created in the current folder. By default, the current folder is the root (\) folder, but you can customize it to a different folder path. You can also use the composer rename command to move and rename objects in batch mode that use a naming convention to a different folder using part of the object name to assign a name to the object.

For detailed information about run cycle groups, see Run cycle group on page 25.

Each run cycle group definition has the following format and arguments:

Syntax

runcyclegroup
[folder/]runcyclegroupname [**description**"*text*"]
vartable *[folder/]tablename*
[freedays *[folder/]calendarname* [**-sa**] [**-su**]]
[**on** [**runcycle** *[folder/]name*
[**validfrom** *date*] [**validto** *date*]
[**description** "*text*"]
[**vartable** *[folder/]table_name*]]
{*date*|*day*|*[folder/]calendar*|*request*|"*icalendar*"} [**,...**]
[**fdignore**|**fdnext**|**fdprev**][**subset** *subsetname* **AND**|**OR**]
[({**at** *time* [**+***n* **day**[**s**]] |
schedtime *time* [**+***n* **day**[**s**]]}
[**until** | **jsuntil** *time* [**timezone**|**tz** *tzname*][**+***n* **day**[**s**]]
[**onuntil***action*]]
[**every** *rate* {**everyendtime** *time*[**+***n* **day**[**s**]]}
[**deadline** *time* [**+***n* **day**[**s**]]])]]
[**,...**]]
[**except** [**runcycle** *[folder/]name*]
[**validfrom** *date*] [**validto** *date*]
[**description** "*text*"]
{*date*|*day*|*[folder/]calendar*|*request*|"*icalendar*"} [**,...**]
[**fdignore**|**fdnext**|**fdprev**][**subset** *subsetname* **AND**|**OR**]
[{(**at** *time* [**+***n* **day**[**s**]])] |
(**schedtime** *time* [**+***n* **day**[**s**]])}]
[**,...**]
[{**at** *time* [***timezone|tz*** *tzname*] [**+***n* **day**[**s**]] |
schedtime *time* [***timezone|tz*** *tzname*] [**+***n* **day**[**s**]]}]
[**until** | **jsuntil** *time* [**timezone**|**tz** *tzname*][**+***n* **day**[**s**]] [**onuntil***action*]]

[**deadline** *time* [**timezone**|**tz** *tzname*] [**+***n* **day**[**s**]]]
end

Arguments

[folder/]runcyclegroupname

Specifies the name of the run cycle group, optionally preceded by the folder name within which the run cycle group is defined. The run cycle group name can contain up to eight alphanumeric characters, including dashes (-) and underscores (_), and must start with a letter.

description "*text*"

Provides a description of the run cycle group. The description can contain up to 120 alphanumeric characters. It must be enclosed in double quotation marks. It can contain alphanumeric characters as long as it starts with a letter. It can contain the following characters: comma (,), period (.), dash (-), plus (+), single quote ('), and equal (=). It cannot contain double quotation marks (") other than the enclosing ones, colon (:), semi-colon (;), and ampersand (&).

vartable *[folder/]tablename*

The name of the variable table. The name must start with a letter, and can contain alphanumeric characters, dashes, and underscores. It can contain up to 80 characters.

[freedays *[folder/]Calendar_Name* [-sa] [-su]]

Specifies a freeday calendar for calculating workdays for the job stream. It can also set Saturdays and Sundays as workdays.

***[folder/]*Calendar_Name**

The name of the calendar that must be used as the non-working days calendar for the job stream. If *Calendar_Name* is not in the database, HCL Workload Automation issues a warning message when you save the job stream. If *Calendar_Name* is not in the database when the command **schedulr** runs, HCL Workload Automation issues an error message and uses the default calendar **holidays** in its place. Do not use the names of weekdays for the calendar names.

-sa

Saturdays are *workdays*.

-su

Sundays are *workdays*.

See freedays on page 348 for details and examples.

runcycle *name*

Specifies a label with a friendly name for the run cycle specified in the following lines.

valid from *date* ... valid to *date*

Delimits the time frame during which the job stream is active, that is the job stream is added to the production plan. Note that the date specified as **valid to** value is not included in the run cycle, therefore on this date the job stream is not active.

description *"text"*

Contains a description of the run cycle.

vartable

Specifies the name of the variable table to be used by the run cycle.

date

Specifies a run cycle that runs on specific dates. The syntax used for this type is:

yyyymmdd [,yyyymmdd][,...]

For example, for a job stream that is scheduled to run on the 25th of May 2023 and on the 12th of June 2023 the value is:

```
on
20230525,20230612
```

day

Specifies a run cycle that runs on specific days. The syntax used for this type is:

{mo|tu|we|th|fr|sa|su}

For example, for a job stream that is scheduled to run every Monday the value is:

```
on
mo
```

[folder/]calendar

The dates specified in a calendar with this name. The calendar name can be followed by an offset in the following format:

{**+** | **-**}*n* {**day[s]** | **weekday[s]** | **workday[s]**}

Where:

n

The number of days, weekdays, or workdays.

days

Every day of the week.

weekdays

Every day of the week, except Saturday and Sunday.

workdays

Every day of the week, except for Saturdays and Sundays (unless otherwise specified with the **freedays** keyword) and for the dates marked either in a designated non-working days calendar or in the **holidays** calendar.

request

Selects the job stream only when requested. This is used for job streams that are selected by name rather than date. To prevent a scheduled job stream from being selected for **JnextPlan**, change its definition to ON REQUEST.

Note: When attempting to run a job stream that contains "on request" times, consider that:

- "On request" always takes precedence over "at".
- "On request" never takes precedence over "on".

icalendar

Represents a standard used to specify a recurring rule that describes when a job stream runs.

The syntax used for run cycle with type *icalendar* is the following:

FREQ={DAILY|WEEKLY|MONTHLY|YEARLY}

[;INTERVAL=[-]*n*]

[;{BYFREEDAY|BYWORKDAY|BYDAY=*weekday_list*|

BYMONTHDAY=*monthday_list***}]**

where the default value for keyword **INTERVAL** is 1.

Using *icalendar* you can specify that a job stream runs:

every *n* days

by using the following format:

FREQ=DAILY[;INTERVAL=*n*]

where the value set for **valid from** is the first day of the resulting dates.

For example, for a job stream that is scheduled to run daily the value is:

```
FREQ=DAILY
```

For a job stream that is scheduled to run every second day the value is:

```
FREQ=DAILY;INTERVAL=2
```

every free or work days

by using the following format:

FREQ=DAILY[;INTERVAL=*n*]

;BYFREEDAY|BYWORKDAY

For example, for a job stream that is scheduled to run every non-working day the value is:

```
FREQ=DAILY;BYFREEDAY
```

For a job stream that is scheduled to run every second workday the value is:

```
FREQ=DAILY;INTERVAL=2;BYWORKDAY
```

every *n* weeks on specific *weekdays*

by using the following format:

FREQ=WEEKLY[;INTERVAL=*n*]

;BYDAY=*weekday_list*

where the value set for *weekday_list* is

```
[SU][,MO][,TU][,WE][,TH][,FR][,SA]
```

For example, for a job stream that is scheduled to run every week on Friday and Saturday the value is:

```
FREQ=WEEKLY;BYDAY=FR,SA
```

For a job stream that is scheduled to run every three weeks on Friday the value is:

```
FREQ=WEEKLY;INTERVAL=3;BYDAY=FR
```

every *n* months on specific dates of the month

by using the following format:

FREQ=MONTHLY[;INTERVAL=*n*]

;BYMONTHDAY=*monthday_list*

where the value set for *monthday_list* is represented by a list of

```
[+number_of_day_from_beginning_of_month]
[-number_of_day_from_end_of_month]
[number_of_day_of_the_month]
```

For example, for a job stream that is scheduled to run monthly on the 27th day the value is:

```
FREQ=MONTHLY;BYMONTHDAY=27
```

For a job stream that is scheduled to run every six months on the 15th and on the last day of the month the value is:

```
FREQ=MONTHLY;INTERVAL=6;BYMONTHDAY=15,-1
```

every *n* months on specific days of specific weeks

by using the following format:

FREQ=MONTHLY[;INTERVAL=*n*]

;BYDAY=*day_of_m_week_list*

where the value set for *day_of_m_week_list* is represented by a list of

```
[+number_of_week_from_beginning_of_month]
[-number_of_week_from_end_of_month]
[weekday]
```

For example, for a job stream that is scheduled to run monthly on the first Monday and on the last Friday the value is:

```
FREQ=MONTHLY;BYDAY=1MO,-1FR
```

For a job stream that is scheduled to run every six months on the 2nd Tuesday the value is:

```
FREQ=MONTHLY;INTERVAL=6;BYDAY=2TU
```

every *n* years

by using the following format:

FREQ=YEARLY[;INTERVAL=*n*]

where the value set for **valid from** is the first day of the resulting dates.

For example, for a job stream that is scheduled to run yearly the value is:

```
FREQ=YEARLY
```

For a job stream that is scheduled to run every two years the value is:

```
FREQ=YEARLY;INTERVAL=2
```

Note: The following limitations apply:

- the maximum supported interval for a daily run cycle is 31 days.
- the maximum supported interval for a weekly run cycle is 8 weeks.
- the maximum supported interval for a monthly run cycle is 12 months. For run cycles specifying the day of the week based on the month, for example the third Saturday or the second Friday, the maximum supported interval is 5 days.
- the maximum supported interval for a yearly run cycle is 10 years.

runcyclegroup

Specified one or more run cycles that combined together produce a set of run dates for a job stream. The run cycle group must be expressed using the following syntax: `$RCG runcyclegroupname`.

fdignore|fdnext|fdprev

Indicates the rule to be applied if the date selected for running the job or job stream falls on a non-working day. The available settings are:

fdignore

Do not add the date.

fdnext

Add the nearest workday after the non-working day.

fdprev

Add the nearest workday before the non-working day.

[subset *subsetname* AND|OR]

subset *subsetname*

Specifies the name of the subset. If you do not specify a name, `SUBSET_1`, is used by default.

AND|OR

By default, run cycles within a subset are in a logical **OR** relationship but you can change it to a logical **AND**, as long as the run cycle group result is a positive date or set of dates (Inclusive).

at *time* [timezone|tz *tzname*][+*n* day[s]]

Specifies a time dependency

time

Specifies a time of day. Possible values can range from **0000** to **2359**.

tzname

Specifies the time zone to be used when computing the start time. See Managing time zones on page 1064 for time zone names. The default is the time zone of the workstation on which the job or job stream is launched.

Note: If an **at** time and an **until** or **deadline** time are specified, the time zones must be the same.

n

Specifies an offset in days from the scheduled start date and time.

schedtime *time* [timezone|tz *tzname*][+*n* day[s]]

Represents the time when the job stream is positioned in the plan.

time

Specifies a time of day in the format: HHHHmm. Possible values are from **0000** to **240000**, or **0000 + <number of days>**.

Where <number of days> can be from 1 to 100 days.

tzname

Specifies the time zone to be used when calculating the start time. See Managing time zones on page 1064 for time zone names. The default is the time zone of the workstation on which the job or job stream is launched.

n

Specifies an offset in days from the scheduled start date and time.

until *time* [timezone|tz *tzname*][+*n* day[s]] [onuntil *action*]

Depending on the object definition the until keyword belongs to, specifies the latest time a job stream must be completed or the latest time a job can be launched. It is mutually exclusive with the **jsuntil** keyword.

time

Specifies the time of day. The possible values are **0000** through **2359**.

tzname

Specifies the time zone to be used when computing the time. See Managing time zones on page 1064 for time zone names. The default is the time zone of the workstation on which the job or job stream is launched.

Note: If an **until** time and an **at** or **deadline** time are specified, the time zones must be the same.

n

Specifies an offset, in days, from the scheduled date and time.

onuntil *action*

Depending on the object definition the until keyword belongs to, specifies:

- The action to be taken on a job whose until time has expired but the job has not yet started.
- The action to be taken on a job stream whose until time has expired but the job stream is not yet completed in SUCC state.

The following are the possible values of the *action* parameter:

suppr

The job or job stream and any dependent job or job stream do not run. This is the default behavior.

Once the until time expired on a job stream, the status for the job stream is calculated following the usual rules; suppressed jobs are not considered in the calculation. In case the job stream contains at least one **every** job its status is HOLD.

When the until time expires for a job, the job moves to HOLD status or keeps any previous status which is a final status.

If the until time is passed together with the **onuntil suppr** and the **carryforward** options, the job stream is *carry forwarded* by JnextPlan only if the until date is equal to the date when JnextPlan runs. If the until and the JnextPlan run dates are not the same, the job stream is not *carry forwarded*.

cont

The job or job stream runs when all necessary conditions are met and a notification message is written to the log when the until time elapses.

If the until time is passed together with the **onuntil cont** and the **carryforward** options, the job stream is always *carry forwarded* by JnextPlan.

canc

A job or job stream is cancelled when the **until** time specified expires. When using ***onuntil canc*** on jobs, the cancel operation on the job is issued by the FTA on which the job runs. Any job or job stream that was dependent on the completion of a job or job stream that was cancelled, runs because the dependency no longer exists.

If the until time is passed together with the **onuntil canc** and the **carryforward** options, the job stream is not *carry forwarded* by JnextPlan because it is already canceled.



Note: When using ***onuntil canc*** at job stream level, define as owner of the job stream the workstation highest in the hierarchy of the scheduling environment, among all workstations that own jobs contained in the job stream.

[jsuntil *time* [+*n* day[s]]] [onuntil*action*]

Defines the latest start time of a job stream. It also determines the behavior of the jobs in the job stream when the job stream is approaching its latest start time. This keyword is mutually exclusive with the **until** keyword. Use the **jsuntil** keyword to avoid your job stream being suppressed if it starts right before its latest start time and the duration of one or more jobs in it exceeds the latest start time. For example, if you have a job stream with **jsuntil** set to 1000, and one of the jobs starts running at 959 and its duration exceeds the latest start time, the job and its successors run as scheduled.

time

Specifies the time of day. The possible values are **0000** through **2359**.

tzname

Specifies the time zone to be used when computing the time. See Managing time zones on page 1064 for time zone names. The default is the time zone of the workstation on which the job or job stream is launched.

n

Specifies an offset, in days, from the scheduled date and time.

onuntil *action*

Depending on the object definition the until keyword belongs to, specifies:

- The action to be taken on a job whose until time has expired but the job has not yet started.
- The action to be taken on a job stream whose until time has expired but the job stream is not yet completed in SUCC state.

The following are the possible values of the *action* parameter:

suppr

The job or job stream and any dependent job or job stream do not run. This is the default behavior.

Once the until time expired on a job stream, the status for the job stream is calculated following the usual rules; suppressed jobs are not considered in the calculation. In case the job stream contains at least one **every** job its status is HOLD.

When the until time expires for a job, the job moves to HOLD status or keeps any previous status which is a final status.

If the until time is passed together with the **onuntil suppr** and the **carryforward** options, the job stream is *carry forwarded* by JnextPlan only if the until date is equal to the date when JnextPlan runs. If the until and the JnextPlan run dates are not the same, the job stream is not *carry forwarded*.

cont

The job or job stream runs when all necessary conditions are met and a notification message is written to the log when the until time elapses.

If the until time is passed together with the **onuntil cont** and the **carryforward** options, the job stream is always *carry forwarded* by JnextPlan.

canc

A job or job stream is cancelled when the **until** time specified expires. When using ***onuntil canc*** on jobs, the cancel operation on the job is issued by the FTA on which the job runs. Any job or job stream that was dependent on the completion of a job or job stream that was cancelled, runs because the dependency no longer exists.

If the until time is passed together with the **onuntil canc** and the **carryforward** options, the job stream is not *carry forwarded* by JnextPlan because it is already canceled.



Note: When using ***onuntil canc*** at job stream level, define as owner of the job stream the workstation highest in the hierarchy of the scheduling

environment, among all workstations that own jobs contained in the job stream.

every *rate* {everyendtime *time*[+*n* day[s]]}

Defines the repetition rate at which instances of the same job stream are run over a time interval. The job stream is launched repeatedly at the specified rate until the time specified in `everyendtime`.

rate

The repetition rate, expressed in hours and minutes (*hhmm*), at which the instances of the job stream are launched.

time

The end time, expressed in hours and minutes (*hhmm*) when the repetition interval stops. After this time no more instances of the job stream are run. Use of the `everyendtime` keyword is mandatory.

n

The number of days that the value of `everyendtime` can be extended. For example, if you specify:

```
everyendtime 1550 + 1
```

and the job stream is scheduled to start running today at 10:00, the end time that the job stream instances will stop being launched is tomorrow at 15:50.

Example

The following example defines a run cycle group named, RCG2, that contains one inclusive run cycle, RUN_CYCLE1, and two exclusive run cycles, RUN_CYCLE2, and RUN_CYCLE3. To determine the run schedule of the job stream associated to this run cycle group, the intersection of the two exclusive run cycles (the two exclusive run cycles have a logical AND relationship between them) is subtracted from the inclusive run cycle. The following are the characteristics of the run cycle group:

An inclusive run cycle RUN_CYCLE1

where,

- The calendar, CAL1, defines days that should be considered non-working days for the job stream. Saturday and Sunday are declared working days.
- The job stream runs not earlier than two days after March 31, 2024 (April 2), and not later than two days after April 12, 2024 (April 14). Every day, the job stream is delayed by two days.
- The job streams runs every day (after the two-day delay) beginning at 7 a.m. and it cannot start later than 9 a.m., otherwise, it is suppressed and does not run at all. The job stream should complete by 10 a.m.

An exclusive run cycle, RUN_CYCLE2

If the job stream falls on a non-working day, then the nearest workday before the non-working day is excluded.

If the job stream falls on April 1, 2024, and this day happens to be a non-working day, then the nearest workday after the non-working day is excluded.

An exclusive run cycle, RUN_CYCLE3

If the job stream falls on April 1, 2024, and this day happens to be a non-working day, then the nearest workday after the non-working day is excluded.

```
RUNCYCLEGROUP RCG2
DESCRIPTION "Sample RunCycle Group"
VARTABLE TABLE1
FREEDAYS CAL1 -SA -SU
  ON     RUNCYCLE RUN_CYCLE1 VALIDFROM 03/31/2024 VALIDTO 04/12/2024 DESCRIPTION
         "Inclusive Run Cycle" VARTABLE TABLE1 "FREQ=DAILY;" FDIGNORE
         (AT 0700 +2 DAYS UNTIL 0900 +2 DAYS ONUNTIL SUPPR DEADLINE 1000 +2 DAYS)

  EXCEPT RUNCYCLE RUN_CYCLE2 VALIDFROM 03/31/2024 VALIDTO 04/12/2024 DESCRIPTION
         "Exclusive Run Cycle" CAL1 FDPREV SUBSET SUBSET_A AND
         (AT 0700 +2 DAYS)

  EXCEPT RUNCYCLE RUN_CYCLE3 VALIDFROM 03/31/2024 VALIDTO 04/12/2024 DESCRIPTION
         "Exclusive Run Cycle" 04/01/2024 FDNEXT SUBSET SUBSET_A AND
         (SCHEDTIME 0700 +2 DAYS)
SCHEDTIME 0700 TZ Europe/Berlin +2 DAYS UNTIL 0900 TZ Europe/Berlin +2 DAYS ONUNTIL
          CONT DEADLINE 1000 TZ Europe/Berlin +2 DAYS
END
```

Job stream definition

A job stream consists of a sequence of jobs to be run, together with times, priorities, and other dependencies that determine the order of processing. You can define job streams related to the same line of business in a specified folder. You can also use the composer rename command to rename and move job streams in batch mode that use a naming convention to specified folders where the folder names are taken from the job stream name.

A job stream begins with a **schedule** keyword followed by attributes and dependencies. The colon delimiter introduces the jobs invoked by the job stream. Each job has its own attributes and dependencies.

Syntax

schedule [*[folder/]workstation#*][*folder/]jobstreamname*
  # comment
  [**validfrom** *date*]
  [**timezone** | **tz** *tzname*]
  [**description** *"text"*]
  [**draft**]
   [**isservice**
    servicename ***service_name***
    servicedescription ***service_description***
    servicetags " | ***service_tag*** |***service_tag***|"
    servicevariables "**{*var*: \"*value*\"}**"]

[**vartable** *[folder/]table_name*]
[**freedays** *[folder/]calendarname* [**-sa**] [**-su**]]
[**on** [**runcycle** *name*
[**validfrom** *date*] [**validto** *date*]
[**description** *"text"*]
[**vartable** *[folder/]table_name*]]
{*date* | *day*|*[folder/]calendar* | *request* | *"icalendar"* | **$RCG** *[folder/]runcyclegroup*} [**,...**]
[**fdignore** | **fdnext** | **fdprev**]
[({**at** *time* [**+***n* **day**[**s**]] |
schedtime *time* [**+***n* **day**[**s**]]}
[**until** | **jsuntil** *time* [**+***n* **day**[**s**]] [**onuntil** *action*]]
[**every** *rate* {**everyendtime** *time*[**+***n* **day**[**s**]]}
[**deadline** *time* [**+***n* **day**[**s**]]])]]
[**,...**]]

[**except** [**runcycle** *name*]
[**validfrom** *date*] [**validto** *date*]
[**description** *"text"*]
{*date* | *day*|*[folder/]calendar* | *request* | *"icalendar"* | **$RCG** *[folder/]runcyclegroup*} [**,...**]
[**fdignore** | **fdnext** | **fdprev**]
[{(**at** *time* [**+***n* **day**[**s**]])} |
(**schedtime** *time* [**+***n* **day**[**s**]])}]
[**,...**]
[**startcond filecreated | filemodified** *[folder/]workstation_name#file_name*
user *username*
interval *seconds*
[(**alias** *startcond_jobname*
rerun batch outfile *outputfilename*
params "*filemonitor additional parameters*")] |
startcond job *[folder/]workstation_name#*[*folder/*]*job_name*
outcond *joboutputcondition*
interval *seconds*
[(**alias** *startcond_jobname* **rerun)]]**
[{**at** *time* [***timezone | tz*** *tzname*] [**+***n* **day**[**s**]] |
schedtime *time* [***timezone | tz*** *tzname*] [**+***n* **day**[**s**]]}]
[**until** | **jsuntil** *time* [**timezone** | **tz** *tzname*] [**+***n* **day**[**s**]] [**onuntil** *action*]]
[**deadline** *time* [**timezone** | **tz** *tzname*] [**+***n* **day**[**s**]]]
[**carryforward**]
[**matching** {**previous** | **sameday** | **relative from [+** | **-]** *time* **to [+** | **-]** *time*|
from *time* [+ | *-n* day[s]] **to** *time* [+ *n* day[s]] [**,...**]}]
[**follows** {[*netagent::*][*workstation#*]*jobstreamname*[**.***jobname* |
@] [**previous**|

```
    sameday|relative from [+|-] time to [+|-] time|
    from time [+|-n day[s]] to time [+|-n day[s]]
  ][if <condition> [| <condition>...]]
    } ] [,...]] [...]
  [join condition_name [number | numconditions | all] of
    description "..."]
    .....
    endjoin

      [keysched]
  [limit joblimit]
  [needs { [n] [[folder/]workstation#][folder/]resourcename } [,...] ] [...]
  [opens { [[folder/]workstation#]"filename" [ (qualifier) ] [,...] }]  [...]
  [priority number | hi | go]
  [prompt {[folder/]promptname|"[:|!]text"} [,...] ] [...]
  [onoverlap {parallel|enqueue|donotstart}]
 :
job-statement
  # comment
job_name [job_alias]
[outcond joboutputcondition interval seconds]
  [{at time [timezone | tz tzname] [+n day[s]] |
  schedtime time [timezone | tz tzname] [+n day[s]]}][,...]
  [until  time [timezone | tz tzname] [+n day[s]] [onuntil action]
  [deadline time [timezone | tz tzname] [+n day[s]] [onlate action] ]
  [maxdur time | percentage % onmaxdur action]
  [mindur time | percentage % onmindur action]
  [every rate]
  [follows {[netagent::][workstation#]jobstreamname{.jobname @} [previous|
    sameday|relative from [+|-] time to [+|-] time |
    from time [+|-n day[s]] to time [+|-n day[s]]
    ]} ][if <condition> [| <condition>...]] [,...]] [...]
  [join condition_name [number | numconditions | all] of
    description "..."]
    .....
    endjoin

      [confirmed]
  [critical]
  [keyjob]
  [needs { [n] [[folder/]workstation#][folder/]resourcename } [,...] ] [...]
  [opens { [[folder/]workstation#]"filename" [ (qualifier) ] [,...] }]  [...]
```

[**priority** *number* | **hi** | **go**]
[**prompt** {*[folder/]promptname*|"[**:**|**!**]*text*"} [,...]] [...]
[**nop**]
[**statistictype custom**]

[*job-statement*...]
end

Arguments

Table 49: List of scheduling keywords on page 317 contains a brief description of the job stream definition keywords. A detailed description of each scheduling keyword is provided in the next subsections.

Table 49. List of scheduling keywords (continued)

Keyword	Description	Page
end	Marks the end of a job stream.	end on page 331
every	Launches a job stream or a job repeatedly at a specified rate.	every on page 332
except	Specifies dates that are exceptions to the **on** dates the job stream is selected to run. It can be followed by a run cycle definition.	except on page 337
fdignore | fdnext | fdprev	Specifies a rule that must be applied when the date selected for exclusion falls on a non-working day.	except on page 337
folder|fol	Specifies the folder where the scheduling object is stored. If you generally work from a precise folder, then you can use the chfolder command to navigate to folders and sub-folders. The chfolder command changes the working directory or current folder, which is set to root ("/") by default, so that you can use relative folder paths when submitting commands. If no folder path is specified, then the object definition is created in the current folder and not in the root. If a relative path is specified, the path is relative to the current folder. See CHFOLDER on page 457 for more information about changing folder paths.	folder on page 343
follows	Specifies jobs or job streams that must complete successfully or must satisfy one or more output conditions before the job or the job stream that is being defined is launched.	follows on page 344
freedays	Specifies a freeday calendar for calculating *workdays* for the job stream. It can also set Saturdays and Sundays as *workdays*.	freedays on page 348
interval	How often HCL Workload Automation checks whether the condition is met.	startcond on page 383
job statement	Defines a job and its dependencies.	job statement on page 349
isservice	Specifies that a job stream can be submitted as a service toSelf-Service Catalog. When creating and editing SSC-ready job streams, it is recommended you use the Dynamic Workload Console.	isservice on page 351

Table 49. List of scheduling keywords (continued)

Keyword	Description	Page
join	Defines a set of conditional dependencies on a job or job stream.	join on page 352
jsuntil	Specifies that the job stream continues running also if one of its jobs starts running right before the time specified in the **jsuntil** keyword. This keyword is enabled by default starting from version 9.4, Fix Pack 1. It is mutually exclusive with the **until** keyword. For more information about the **until** keyword, see until on page 390.	jsuntil on page 354
keyjob	Marks a job as key in both the database and in the plan .	keyjob on page 356
keysched	Marks a job stream as key in both the database and in the plan for monitoring by applications, such as IBM® Tivoli® Business Systems Manager or IBM® Tivoli Enterprise Console®.	keysched on page 357
limit	Sets a limit on the number of jobs that can be launched concurrently from the job stream.	limit on page 357
matching	Defines the matching criteria used when a matching criteria is not specified in the follows specifications in the job stream definition or in the job definition within the job stream.	matching on page 358
maxdur	Specifies the maximum length of time a job can run. You can express this time in either minutes, or as a percentage of the latest estimated duration for the job.	maxdur on page 360
mindur	Specifies the shortest amount of time within which a job normally runs and completes.	mindur on page 361
needs	Defines the number of units of a resource required by the job or job stream before it can be launched. The highest number of resources the job stream can be dependent from is 1024.	needs on page 362
nop	Specifies that a job is not to be run when the plan executes. The job is included in the plan but, as the plan runs, it is placed in `Cancel Pending` status and is not executed.	nop on page 364

Table 49. List of scheduling keywords (continued)

Keyword	Description	Page
on	Defines the dates on which the job stream is selected to run. It can be followed by a run cycle definition.	on on page 364
onlate	Defines the action to be taken on a job in the job stream when the job's deadline expires.	onlate on page 372
onoverlap	Specifies how to handle a job stream instance that is scheduled to start although the preceding instance has not yet completed.	onoverlap on page 373
opens	Defines files that must be accessible before the job or job stream is launched.	opens on page 374
onuntil	Specifies the action to take on a job or job stream whose until time has been reached.	until on page 390
outcond	The output condition which, when met, releases the remaining part of the job stream or the job where it is specified.	startcond on page 383
priority	Defines the priority for a job or job stream.	priority on page 377
prompt	Defines prompts that must be replied to before the job or job stream is launched.	prompt on page 378
runcycle	Specifies a label with a friendly name for the run cycle. It is used in conjunction with the following keywords: **except** For *exclusive* run cycles, that define when the job stream is not to run. **on** For *inclusive* run cycles, that define when the job stream will run. For more information about run cycles, see Run cycle on page 22.	• except on page 337 • on on page 364
schedule	Assigns a name to the job stream.	schedule on page 381
schedtime	Specifies the time used to set the job stream in the time line within the plan to determine successors and predecessors.	schedtime on page 379

Table 49. List of scheduling keywords (continued)

Keyword	Description	Page
startcond	Builds into the job stream a mechanism which checks for specific events and conditions and releases the job stream when the specified events or conditions take place.	startcond on page 383
servicename	Specifies the service name to be used for the job stream in Self-Service Catalog. Required if you enable the **isservice** keyword. When creating and editing SSC-ready job streams, it is recommended you use the Dynamic Workload Console.	servicename on page 387
servicedescription	Specifies the service description to be used for the job stream in Self-Service Catalog. When creating and editing SSC-ready job streams, it is recommended you use the Dynamic Workload Console.	servicedescription on page 388
servicetags	Specifies the service tags to be used to identify the job stream in Self-Service Catalog. When creating and editing SSC-ready job streams, it is recommended you use the Dynamic Workload Console.	servicetags on page 388
servicevariables	Specifies the service variables to be used for the job stream in Self-Service Catalog. When creating and editing SSC-ready job streams, it is recommended you use the Dynamic Workload Console.	servicevariables on page 389
timezone | tz	Specifies the time zone to be used when computing the start time.	timezone on page 389
until	Defines the latest time a job or a job stream can be launched. When defined in a run cycle specifies the latest time a job or a job stream can be launched for that specific run cycle. It is mutually exclusive with the **jsuntil** keyword. For more information about the **jsuntil** keyword, see jsuntil on page 354.	until on page 390
validfrom	Defines the date from which the job stream instance starts.	validfrom/validto on page 393
validto	Indicates the date on which the job stream instance ends.	validfrom/validto on page 393
vartable	Defines the variable table to be used by the job stream and the run cycle.	Variable table on page 394

Note:

1. Job streams scheduled to run on workstations marked as *ignored* are not added to the production plan when the plan is created or extended.
2. Wrongly typed keywords used in job definitions lead to truncated job definitions stored in the database. In fact the wrong keyword is considered extraneous to the job definition and so it is interpreted as the job name of an additional job definition. Usually this misinterpretation causes also a syntax error or an inexistent job definition error for the additional job definition.
3. Granting access to a workstation class or a domain means to give access just to the object itself, and grant no access to the workstations in the object.

Time zone specification rules

You can specify a time zone at several keyword levels within a job stream definition; that is:

- For the whole job stream (inclusive of all its keyword specifications)
- At time restriction level (with the `at`, `deadline`, `schedtime`, and `until` keywords)
- For each included job statement

The following rules apply when resolving the time zones specified within a job stream definition:

- When you specify the time zone at job stream level, this applies to the time definitions of the run cycle (defined with the `on` keyword) as well as to those in the time restrictions.
- If you specify a time zone both at job stream level and at time restriction level, they must be the same. If you specify no time zone, either at job stream and time restriction levels, the time zone specified on the workstation is used.
- The time zone specified at job level can differ from the one specified at job stream level and overrides it. If you specify no time zone, either at job stream and job levels, the time zone specified on the workstation running the job is used.

Time restriction specification rules

Within a job stream definition you can specify time restrictions (with the `at`, `deadline`, `schedtime`, and `until` keywords) at both job stream and run cycle levels. When both are specified, the time restrictions specified at run cycle level override the ones specified at job stream level.

Example

This is an example of job stream definition:

```
SCHEDULE M235062_99#TEST/SCHED_FIRST1 VALIDFROM 06/30/2024
ON RUNCYCLE SCHED1_PREDSIMPLE VALIDFROM 07/18/2024 "FREQ=DAILY;INTERVAL=1"
   ( AT 1010 )
ON RUNCYCLE SCHED1_PRED_SIMPLE VALIDFROM 07/18/2024 "FREQ=DAILY;INTERVAL=1"
CARRYFORWARD
PROMPT "Do you want the job to start?"
```

```
PRIORITY 55
:
M235062_99#MYFOLDER/JOBMDM
 PRIORITY 30
 NEEDS 16 M235062_99#JOBSLOTS
 PROMPT PRMT3

B236153_00#JOB_FTA
 FOLLOWS MYFOLDER/JOBMDM
END
```

Creating an embedded job in a job stream

You can create independent jobs within the job stream. This ensures greater customization, and control without being affected by external changes in the job definition. You can fully customize jobs inside a job stream tailored for your needs.

You can edit the job stream definition to add an embedded job. This type of job is correlated to the corresponding job stream. Traditionally, when creating a job definition for a job stream, any later changes to the job itself reflect on the job stream. This cannot happen with the embedded job that exists only inside the job stream definition.

You can add the entire job definition within the job stream as embedded that ensures the job is not impacted by external updates, as the updates must happen to the job stream itself. This approach creates greater flexibility and control over the specific job parameters used within the job stream. You need to run the display job stream | js command to view embedded jobs. The entire embedded definition is displayed in such cases and to modify it, you need to run the modify job stream | js command. You can add the embedded job definition as follows in a workflow:

Syntax

```
jobs:
  - workstation: worstation_name
    name: JOB_name
    jobDefinition:
        Specify the job_deinition
```

workstation

Mandatory parameter and specify the workstation for the job.

name

Mandatory parameter and specify a name for the job. You can provide any alphanumeric value and wildcard characters. You can use a maximum of 64 characters.

jobDefinition

Specify the job definition.

Important: If you are using schedlang format, use the **BUILTIN** parameter to create the definition. For more information, see Example on page 324.

Embedded jobs not supported by composer, dataextract, and dataimport commands.

Example

You can create a job strream definition as follows to add an embedded job that can run the ls command.

```
---
kind: JobStream
def:
  folder: /
  name: JS_EMBEDDED
  workstation: /WS_AGT_0
  description: Job stream containing an embedded job
  jobs:
  - workstation: /WS_AGT_0
    name: JOB_EMBEDDED1
    jobDefinition:
      workstation: /WS_AGT_0
      type: executable
      task:
        executable:
          interactive: false
          suffix: ""
          script: ls
          credential: {}
      recovery:
        action: STOP
        repeatAffinity: false
```

If you are using schedlang format, create the definition as follows:

```
$jobstream

JOBSTREAM /WS_AGT_0#/JS_EMBEDDED
  DESCRIPTION "Job stream containing an embedded job"
:

  /WS_AGT_0#JOB_EMBEDDED1 BUILTIN
    TASK
      {
        "executable" : {
          "interactive" : false,
          "suffix" : "",
          "script" : "ls",
          "credential" : { }
        }
      }
    RECOVERY STOP
    END
```

See also

From the Dynamic Workload Console you can perform the same task as described in:

the Dynamic Workload Console User's Guide, section about Creating job stream definitions.

Job stream definition keyword details

This section describes the job stream definition keywords listed in table Table 49: List of scheduling keywords on page 317.

at

Specifies a time dependency. If the **at** keyword is used, then the job or job stream cannot start before the time set with this keyword. When specifying an **at** time, consider the impact of Daylight Saving Time (DST) "spring forward" gaps. If the specified time falls within the skipped DST hour, the system automatically shifts the evaluated time forward. This forward shift can disrupt time-sensitive external dependencies (such as follows ... previous), potentially causing a predecessor to run later than its successor and preventing the dependency from being resolved.

Syntax

at *time* [**timezone**|**tz** *tzname*][**+***n* **day**[**s**]]

Arguments

time

Specifies a time of day. Possible values can range from **0000** to **2359**.

tzname

Specifies the time zone to be used when computing the start time. See Managing time zones on page 1064 for time zone names. The default is the time zone of the workstation on which the job or job stream is launched.

Note: If an **at** time and an **until** or **deadline** time are specified, the time zones must be the same.

n

Specifies an offset in days from the scheduled start date and time.

Comments

If an **at** time is not specified for a job or job stream, its launch time is determined by its dependencies and priority and its position in the preproduction plan is determined by the value assigned to the **schedtime** keyword. For more information about the **schedtime** keyword refer to schedtime on page 379.

If the run cycle and job stream start times are both defined, the run cycle start time takes precedence when the job stream is scheduled with JNextPlan. When the job stream is launched with the submit command, the run cycle start time is not used.

The time value in the **at** option is considered as follows:

- If the time value is less than the value set in the ***startOfDay*** global option, it is taken to be for the following day.
- If the time value is greater than the value set in the ***startOfDay*** global option, it is taken to be for the current day.

If neither the **at** nor the **schedtime** keywords are specified in the job stream definition then, by default, the job or job stream instance is positioned in the plan at the time specified in the *startOfDay* global option.

Example

Examples

The following examples assume that the HCL Workload Automation processing day starts at 6:00 a.m.

- The following job stream, selected on Tuesdays, is launched no sooner than 3:00 a.m. Wednesday morning. Its two jobs are launched as soon as possible after that time.

```
schedule sked7 on tu at 0300:
job1
job2
end
```

- The following example launches job stream mysked on Sundays at 8:00 a.m.. Jobs job1, job2, and job3 are all launched on Sundays.

```
schedule mysked on fr at 0800 + 2 days
:
job1
job2 at 0900
job3 follows job2 at 1200
end
```

- The time zone of workstation `sfran` is defined as America/Los_Angeles, and the time zone of workstation `nycity` is defined as America/New_York. The following job stream is selected to run on Friday. It is launched on workstation `sfran` at 10:00 a.m. America/Los_Angeles Saturday. `job1` is launched on `sfran` as soon as possible after that time. `job2` is launched on `sfran` at 2:00 p.m. America/New_York (11:00 a.m. America/Los_Angeles) Saturday. `job3` is launched on workstation `nycity` at 4:00 p.m. America/New_York (1:00 p.m. America/Los_Angeles) Saturday.

```
sfran#schedule sked8 on fr at 1000 + 1 day:
job1
job2 at 1400 tz America/New_York
nycity#job3 at 1600
end
```

carryforward

Makes a job stream eligible to be carried forward to the next production plan if it is not completed before the end of the current production plan.

Syntax

carryforward

Example

Examples

The following job stream is carried forward if its jobs have not completed before preproduction processing begins for a new production time frame.

```
schedule sked43 on th
carryforward
:
job12
job13
job13a
end
```

The following is an example of a job stream that does not have a job defined.

1. Define two job streams and submit them in the correct order:

   ```
   MDM#JS001
   PRIORITY 0
   :
   MDM#JOB001
   END

   MDM#JSNOJOB
   FOLLOWS MDM#JS001.@
   :
   END
   ```

2. Both job streams will be in **HOLD** status at the next JnextPlan time.
3. After JnextPlan, MDM#JSNOJOB will not carryforward.

   Note: Job streams that do not contain jobs are not carried forward.

comment

Includes comments in a job stream definition and the jobs contained in a job stream.

Syntax

\# *text*

Comments

Inserts a comment line. The first character in the line must be a pound sign #.

You can add comments in a job stream definition immediately after the line with the **schedule** keyword, or in a job contained in a job stream definition immediately after the *job statement* line.

Example

Examples

The following example includes both types of comments:

```
schedule wkend on fr at 1830
##########################
# The weekly cleanup jobs
##########################
#
carryforward
:
job1
 # final totals and reports
job2
 # update database
end
```

confirmed

Specifies that a job's completion must be confirmed.

Syntax

confirmed

Example

Examples

In the following job stream, confirmation of the completion of `job1` must be received before `job2` and `job3` are launched.

```
schedule test1 on fr:
job1 confirmed
job2 follows job1
job3 follows job1
end
```

critical

Specifies that the job is mission-critical and must be processed accordingly.

- While building the plan, or every time it runs the submit command, calculates the latest start time each of its predecessors can start so that the job successfully meets its deadline. This is called the *critical start time*. The critical job and every one of its predecessors are assigned a critical start time.

  The entire set of predecessors to the critical job is referred to as the *critical network* of the job.
- While running the plan, dynamically recalculates the critical start times within the critical network.

  When a predecessor risks compromising the timely completion of the critical job, it is *promoted*; that is, using different operating system mechanisms, such as implementing the nice command on UNIX or changing the priority level on Windows, it is assigned additional resources and its submission is prioritized with respect to other jobs that are out of the critical network. This action is recurrently run on any predecessor within the critical network and, if necessary, on the critical job as long as there is a risk that this job becomes late.

Important: Critical jobs, included in the plan because associated to a run cycle, must have a deadline specified at job, job stream or run cycle level. Whereas, critical jobs submitted in plan on request might not have a specified deadline, and, in this case, the global option `deadlineOffset` is used.

Syntax

critical

deadline

Specifies the time within which a job or job stream must complete. Jobs or job streams that have not yet started or that are still running when the deadline time is reached, are considered *late* in the plan. When a job (or job stream) is late, the following actions are performed:

information about the whole environment, define the option on the master domain manager. Deadlines for critical jobs are evaluated automatically, independently of the **bm check deadline** option.

Syntax

deadline *time* [**timezone**|**tz** *tzname*][**+***n* **day**[**s**]

Arguments

time

Specifies a time of day. Possible values range from **0000** to **2359**.

tzname

Specifies the time zone to be used when computing the deadline. See Managing time zones on page 1064 for time zone names. The default is the time zone of the workstation on which the job or job stream is launched.

n

Specifies an offset in days from the scheduled deadline time.

Note: If a **deadline** time and an **until** or **at** time are specified, the time zones must be the same.

Example

Examples

The following example launches job stream `sked7` every day and job `jobc` to start running at 14:30 and to be completed by 16:00.

```
schedule sked7 on everyday :
    jobc at 1430 deadline 1600
end
```

description

Includes a description for the job stream.

Syntax

description '*text*'

Comments

The maximum length of this field is 4094 bytes.

Example

Examples

```
schedule test1
description ?Revenue at the end of the month?
on monthend
:
job1
job2
job3
end
```

draft

Marks a job stream as draft. A draft job stream is not added to the preproduction plan.

Syntax

draft

Comments

A draft job stream is not considered when resolving dependencies and is not added to the production plan. After removing the draft keyword from a job stream you need to run **JnextPlan** command to add the job stream to the preproduction plan and so to the production plan.

Example

Examples

```
schedule test1 on monthend
draft
:
job1
job2
job3
end
```

end

Marks the end of a job stream definition.

Syntax

end

Example

Examples

```
schedule test1 on monthend
:
job1
job2
```

```
job3
end << end of job stream >>
```

every

Defines the repetition rate for a job stream or for a job. The syntax and the location of the keyword vary according to which object the keyword is being used for. When used for a job stream, it is specified within the definition of a run cycle for the job stream. When used for a job, it is specified within the definition of the job in the job stream.

For details, see:

- every (used for job streams) on page 332
- every (used for jobs) on page 333

every (used for job streams)

Defines the repetition rate for the job stream. The job stream is launched repeatedly at the specified rate. The keyword can be specified only within the definition of a run cycle for the job stream.

The use of this keyword results in the creation at plan-generation time of a number of instances of the job stream that depends on the repetition rate, on the start time (defined either with `at` or `schedtime`, or based on the start of day), and on the time defined for `everyendtime`.

Syntax

every *rate* {**everyendtime** *time*[**+***n* **day**[**s**]]}

Arguments

rate

The repetition rate, expressed in hours and minutes (*hhmm*), at which the instances of the job stream are launched.

time

The end time, expressed in hours and minutes (*hhmm*) when the repetition interval stops. After this time no more instances of the job stream are run. Use of the `everyendtime` keyword is mandatory.

n

The number of days that the value of `everyendtime` can be extended. For example, if you specify:

```
everyendtime 1550 + 1
```

and the job stream is scheduled to start running today at 10:00, the end time that the job stream instances will stop being launched is tomorrow at 15:50.

Comments

Based on the values used, the job stream instances are created at the time that the plan is generated and run following the defined start dates for the job stream. They are not created dynamically during the execution of the plan.

Example

Examples

1. In the following example, an instance of job stream JS_151439298 is run every 2 hours until 6 PM starting from 2 PM of June 15, 2024.

```
SCHEDULE MDM021#JS_151439298
ON RUNCYCLE RC1 06/15/2024
( AT 1400 EVERY 0200 EVERYENDTIME 1800 )
:
MDM021#J_151439298
END
```

2. In the following example, an instance of job stream JS_0415 is run every 10 minutes until 3:15 PM starting from 2:45 PM of June 16, 2024.

```
SCHEDULE MDM005#JS_0415
ON RUNCYCLE ACCTRC 06/16/2024
( SCHEDTIME 1445 DEADLINE 1530 EVERY 0010 EVERYENDTIME 1515 )
:
MDM005#JS_0415
END
```

every (used for jobs)

Defines the repetition rate for a job. The job is launched repeatedly at the specified rate. If the job has a dependency that is not satisfied, the iteration is started only after the dependency is satisfied. The keyword is specified within the definition of a job.

Syntax

every *rate*

Arguments

rate

The repetition rate expressed in hours and minutes, in the *hhmm* format. The rate can be longer than 24 hours. The maximum supported value is 99 hours and 59 minutes.

Comments

- The **every** iteration of a job does not stop even if one of the job repetitions abends.
- If the **every** option is used without the **at** dependency, the rerun jobs are scheduled respecting the **every** rate specified, starting from the time when the job actually started.
- In the specific case that the **every** option is used with the **at** dependency and one rerun is delayed (for a dependency or for any other reason), then, while HCL Workload Automation realigns to the **at** time, there might one or two iterations that do not respect the **every** rate. For all other cases the every rate is always respected.

Example 2 on page 334 explains how HCL Workload Automation realigns to the **at** time if the job starts later than the defined **at** time and some iterations are lost.

- If an **every** instance of a job does not start at its expected start time, use the **bm late every** option to set the maximum number of minutes that elapse before HCL Workload Automation skips the job. The value of the option must be defined in the `<TWSHOME>/localopts` file:

  bm late every = xx

  Where *xx* is the number of minutes.

  This option is local for each agent, therefore it must be defined on every fault-tolerant agent that has **every** jobs with **bm late every** option set.

  The **bm late every** option applies only to jobs with both the **every** option and the **at** time dependency defined, it has no impact on jobs that have only the **every** option defined. Only jobs whose **every** rate is greater than the **bm late every** value will be impacted.

  Example 4 on page 336 shows the behavior of HCL Workload Automation when the delay of an **every** instance does not exceed the **bm late every** option value.

  Example 5 on page 336 shows the behavior of HCL Workload Automation when the delay of an **every** instance exceeds the **bm late every** option value.

  Example 6 on page 337 shows the behavior of HCL Workload Automation when the first instance of a job does not run at its expected start time and exceeds the **bm late every** option value.

- If the **every** keyword is defined for a job when the Daylight Saving Time (DST) turns off, that is the clock is put one hour back, the command **every job** is DST aware and it runs also during the second, repeated time interval.

Example

Examples

1. The following example runs the `testjob` job every hour:

   ```
   testjob every 60
   ```

2. The following example shows the `testjob1` job that is defined to run every 15 minutes, between the hours of 6:00 p.m. and 8:00 p.m.:

   ```
   testjob1 at 1800 every 15 until 2000
   ```

   The job is supposed to run at 1800, 1815, 1830, and so on every 15 minutes.

   If the job is submitted `adhoc` at 1833, the reruns are at 1833, 1834, 1845, etc. The reason for this is explained next:

   At first notice that in a job there are two time values to consider:

- The *start_time*; this is the time when the job is expected to run. It is set to the **at** time specified for the job or to the time when the rerun should be launched. This value can be viewed using `conman showjobs` before the job iteration starts.
- The *time_started*; this is the time when the job actually starts, for example 1833. This value can be viewed by using `conman showjobs` after the job iteration started.

Because `testjob1` was submitted `adhoc` at 1833, this is the information you see immediately after submission:

with `conman showjobs`

```
TESTJOB1 HOLD 1800
```

in the Symphony file

`start_time=1800` (because the job is expected to run **at** 1800)

`time_started=NULL` (because the job has not yet started)

Since the `start_time` (1800) is smaller than the current time (1833), `testjob1` starts immediately and the updated information becomes:

with `conman showjobs`

```
TESTJOB1 SUCC 1833
```

in the Symphony file

`start_time=1800` (because the job was expected to run **at** 1800)

`time_started=1833` (because the job started at 1833)

When **batchman** calculates the time for the next iteration, it uses the following data:

```
start_time=1800
```

```
rate=0015
```

```
current_time=1833
```

Since the next iteration time (1800+0015=1815) would still be sooner than the *current_time* value (1833), **batchman** identifies the last planned iteration that was not run by adding to the *start_time* as many *every_rate* as possible without exceeding the *current_time*

1800 + 0015 + 0015 = **1830** < 1833

and then issues the command to run that iteration. Assuming that this iteration is run at 1834, the information, after the job starts, becomes the following:

with `conman showjobs`

```
TESTJOB1 SUCC 1834
```

in the Symphony file

`start_time=1830` (because that job iteration was expected to run **at** 1830)

`time_started=1834` (because that job iteration started at 1834)

After this job iteration completed, **batchman** calculates again the time the next iteration has to start using these updated values:

```
start_time=1830
```

```
rate=0015
```

```
current_time=1834
```

The fact that the next iteration time (1830+0015=1845) is later than the *current_time* value (1834), shows **batchman** that the iteration is recovered. The iteration time, starting from 1845 onwards, can now be realigned with the planned iteration times set in the job definition by the **at** and **every** keywords.

3. The following example does not start the `testjob2` job iteration until job `testjob1` has completed successfully:

   ```
   testjob2 every 15 follows testjob1
   ```

4. In the following example, the delay of an instance of an **every** job does not exceed the **bm late every** option value:

   ```
   bm late every = 10
   JOB  AT  1400 EVERY 0030
   ```

   This job is supposed to run at 1400, 1430, 1500, and so on every thirty minutes.

   If the server is down from 1435 to 1605, the instances at 1500, 1530, and 1600 do not run. At 1605, HCL Workload Automation restarts. When it analyses the Symphony file, it determines that the potential best time for the next **every** job instance is 1600. HCL Workload Automation checks if the potential best time (1600) exceeds the maximum allowed delay for an **every** job (10 minutes).

   In this case the delay has not exceeded the **bm late every** option, therefore HCL Workload Automation behaves as usual and creates the instance of the **every** job with start time set to 1600. The subsequent instances are at 1630, 1700 and so on, every thirty minutes.

5. In the following example, the delay of the instance of an **every** job exceeds the **bm late every** option value:

   ```
   bm late every = 10
   JOB  AT  1400 EVERY 00030
   ```

   This job is supposed to run at 1400, 1430, 1500, and so, on every thirty minutes.

   If the server is down from 1435 to 1620, the instances at 1500, 1530, and 1600 do not run. At 1620, HCL Workload Automation restarts. When it analyses the Symphony file, it determines that the potential best time for the next **every** job instance is 1600. HCL Workload Automation checks if the potential best time (1600) exceeds the maximum allowed delay for an **every** instance of a job (10 minutes).

   In this case the delay is greater that the **bm late every** option, therefore HCL Workload Automation applies the new behavior, it does not launch the instance of the **every** job at 1600 and it creates the instance of the **every** job with start time set to 1630.

6. The following example shows the behaviour of HCL Workload Automation when the first instance of a job does not run at its expected start time and exceeds the **bm late every** option value:

```
bm late every = 10
JOB  AT  1400 EVERY 00030
```

This job is supposed to run at 1400, 1430, 1500, and so on, every thirty minutes.

If the server is down from 1000 to 1415, the first instance of the job does not run. At 1415, HCL Workload Automation restarts. When it analyses the Symphony file, it determines that the first instance of this **every** job has not run. In this case HCL Workload Automation launches the job at 1415.

except

Defines the dates that are exceptions to the **on** dates of a job stream. See on on page 364 for more information.

Syntax

except [**runcycle** *name*]
[**validfrom** *date*] [**validto** *date*]
[**description** *"text"*]
{*date*|*day*|*[folder/]calendar*|*request*|*"icalendar"*|*runcyclegroup*}
[**,...**]
[**fdignore**|**fdnext**|**fdprev**][**subset** *subsetname* **AND**|**OR**]

Arguments

runcycle *name*

Specifies a label with a friendly name for the run cycle specified in the following lines.

valid from *date* ... valid to *date*

Delimits the time frame during which the job stream is active, that is the job stream is added to the production plan. Note that the date specified as **valid to** value is not included in the run cycle, therefore on this date the job stream is not active.

description *"text"*

Contains a description of the run cycle.

date

Specifies a run cycle that runs on specific dates. The syntax used for this type is:

yyyymmdd [,yyyymmdd][,...]

For example, for a job stream that is scheduled to run on the 25th of May 2023 and on the 12th of June 2023 the value is:

```
on
20230525,20230612
```

day

Specifies a run cycle that runs on specific days. The syntax used for this type is:

{mo|tu|we|th|fr|sa|su}

For example, for a job stream that is scheduled to run every Monday the value is:

```
on
mo
```

[folder/]calendar

The dates specified in a calendar with this name. The calendar name can be followed by an offset in the following format:

{**+** | **-**}*n* {**day**[**s**] | **weekday**[**s**] | **workday**[**s**]}

Where:

n

The number of days, weekdays, or workdays.

days

Every day of the week.

weekdays

Every day of the week, except Saturday and Sunday.

workdays

Every day of the week, except for Saturdays and Sundays (unless otherwise specified with the **freedays** keyword) and for the dates marked either in a designated non-working days calendar or in the **holidays** calendar.

request

Selects the job stream only when requested. This is used for job streams that are selected by name rather than date. To prevent a scheduled job stream from being selected for **JnextPlan**, change its definition to ON REQUEST.

Note: When attempting to run a job stream that contains "on request" times, consider that:

- "On request" always takes precedence over "at".
- "On request" never takes precedence over "on".

icalendar

Represents a standard used to specify a recurring rule that describes when a job stream runs.

The syntax used for run cycle with type *icalendar* is the following:

FREQ={DAILY|WEEKLY|MONTHLY|YEARLY}

[;INTERVAL=[-]*n*]

[;{BYFREEDAY|BYWORKDAY|BYDAY=*weekday_list*|

BYMONTHDAY=*monthday_list***}]**

where the default value for keyword **INTERVAL** is 1.

Using *icalendar* you can specify that a job stream runs:

every *n* days

by using the following format:

FREQ=DAILY[;INTERVAL=*n*]

where the value set for **valid from** is the first day of the resulting dates.

For example, for a job stream that is scheduled to run daily the value is:

```
FREQ=DAILY
```

For a job stream that is scheduled to run every second day the value is:

```
FREQ=DAILY;INTERVAL=2
```

every free or work days

by using the following format:

FREQ=DAILY[;INTERVAL=*n*]

;BYFREEDAY|BYWORKDAY

For example, for a job stream that is scheduled to run every non-working day the value is:

```
FREQ=DAILY;BYFREEDAY
```

For a job stream that is scheduled to run every second workday the value is:

```
FREQ=DAILY;INTERVAL=2;BYWORKDAY
```

every *n* weeks on specific *weekdays*

by using the following format:

FREQ=WEEKLY[;INTERVAL=*n*]

;BYDAY=*weekday_list*

where the value set for *weekday_list* is

```
[SU][,MO][,TU][,WE][,TH][,FR][,SA]
```

For example, for a job stream that is scheduled to run every week on Friday and Saturday the value is:

```
FREQ=WEEKLY;BYDAY=FR,SA
```

For a job stream that is scheduled to run every three weeks on Friday the value is:

```
FREQ=WEEKLY;INTERVAL=3;BYDAY=FR
```

every *n* months on specific dates of the month

by using the following format:

FREQ=MONTHLY[;INTERVAL=*n*]

;BYMONTHDAY=*monthday_list*

where the value set for *monthday_list* is represented by a list of

```
[+number_of_day_from_beginning_of_month]
[-number_of_day_from_end_of_month]
[number_of_day_of_the_month]
```

For example, for a job stream that is scheduled to run monthly on the 27th day the value is:

```
FREQ=MONTHLY;BYMONTHDAY=27
```

For a job stream that is scheduled to run every six months on the 15th and on the last day of the month the value is:

```
FREQ=MONTHLY;INTERVAL=6;BYMONTHDAY=15,-1
```

every *n* months on specific days of specific weeks

by using the following format:

FREQ=MONTHLY[;INTERVAL=*n*]

;BYDAY=*day_of_m_week_list*

where the value set for *day_of_m_week_list* is represented by a list of

```
[+number_of_week_from_beginning_of_month]
[-number_of_week_from_end_of_month]
[weekday]
```

For example, for a job stream that is scheduled to run monthly on the first Monday and on the last Friday the value is:

```
FREQ=MONTHLY;BYDAY=1MO,-1FR
```

For a job stream that is scheduled to run every six months on the 2nd Tuesday the value is:

```
FREQ=MONTHLY;INTERVAL=6;BYDAY=2TU
```

every *n* years

by using the following format:

FREQ=YEARLY[;INTERVAL=*n*]

where the value set for **valid from** is the first day of the resulting dates.

For example, for a job stream that is scheduled to run yearly the value is:

```
FREQ=YEARLY
```

For a job stream that is scheduled to run every two years the value is:

```
FREQ=YEARLY;INTERVAL=2
```

Note: The following limitations apply:

- the maximum supported interval for a daily run cycle is 31 days.
- the maximum supported interval for a weekly run cycle is 8 weeks.
- the maximum supported interval for a monthly run cycle is 12 months. For run cycles specifying the day of the week based on the month, for example the third Saturday or the second Friday, the maximum supported interval is 5 days.
- the maximum supported interval for a yearly run cycle is 10 years.

runcyclegroup

Specified one or more run cycles that combined together produce a set of run dates for a job stream. The run cycle group must be expressed using the following syntax: `$RCG runcyclegroupname`.

fdignore|fdnext|fdprev

Specifies a rule that must be applied when the date selected for exclusion falls on a non-working day. It can be one of the following:

fdignore

Do not exclude the date.

fdnext

Exclude the nearest workday after the non-working day.

fdprev

Exclude the nearest workday before the non-working day.

subset *subsetname*

Specifies the name of the subset. If you do not specify a name, `SUBSET_1`, is used by default.

AND|OR

By default, run cycles within a subset are in a logical **OR** relationship but you can change it to a logical **AND**, as long as the run cycle group result is a positive date or set of dates (Inclusive).

For an explanation about remaining keywords contained in the **except** syntax refer to on on page 364.

Comments

You can define multiple instances of the **except** keyword for the same job stream. Each instance is equivalent to a run cycle to which you can associate a freeday rule.

Multiple **except** instances must be consecutive within the job stream definition.

Each instance of the keyword can contain any of the values allowed by the **except** syntax.

Example

Examples

The following example selects job stream `testskd2` to run every weekday except those days whose dates appear on calendars named `monthend` and `holidays`:

```
schedule testskd2 on weekdays
except monthend,holidays
```

The following example selects job stream `testskd3` to run every weekday except May 15, 2024 and May 23, 2024:

```
schedule testskd3 on weekdays
except 05/15/2024,05/23/2024
```

The following example selects job stream `testskd4` to run every day except two weekdays prior to any date appearing on a calendar named `monthend`:

```
schedule testskd4 on everyday
except monthend-2 weekdays
```

Select job stream `sked4` to run on Mondays, Tuesdays, and 2 weekdays prior to each date listed in the `monthend` calendar. If the run date is a non-working day, run the job stream on the nearest following workday. Do not run the job stream on Wednesdays.

```
schedule sked4
on mo
on tu, MONTHEND -2 weekdays fdnext
except we
```

Select job stream `testskd2` to run every weekday except for the days listed in `monthend`. If a date in `monthend` falls on a non-working day, exclude the nearest workday before it. In this example, the non-working days are Saturdays, Sundays, and all the dates listed in the default `holidays` calendar .

```
schedule testskd2
on weekdays
except MONTHEND fdprev
```

folder

Specifies the folder where the scheduling object, such as job, job stream, workstation, and so on, is stored.

Syntax

folder | **fol** *foldername*

Arguments

foldername

Specifies the folder where the scheduling object is stored.

Comments

A folder is a container of jobs, job streams, or sub-folders and has a tree-like structure similar to a file system. The folder name is an alphanumeric string that cannot start with a "-" (hyphen), but it can contain the following characters: "/" (forward slash), "-" (hyphen), and "_" (underscore). It cannot contain spaces. If an object is not defined in a folder, then the default folder "/" is used. If you specify an absolute path, include a "/" forward slash before the folder name. A forward slash is not necessary for relative paths. The maximum length for the full folder path (that is, the path name including the parent folder

and all nested sub-folders) is 800 characters, while each folder name supports a maximum of 256 characters. Wildcard characters are supported. A single scheduling object belongs to one folder only, but each folder can contain numerous objects.

For information about commands you can use to manage folders, see Folder definition on page 287.

Example

Examples

```
SCHEDULE FOLDER3/FTA1#/FOLDER1/FOLDER2/JS4
STARTCOND JOB FTA1#JOBDEF1 OUTCOND "RC>0 AND RC<=20" INTERVAL 60
(ALIAS mystartcond RERUN)
:
FTA1#JOBDEF1
  DEADLINE 2000 +1 DAYS ONLATE KILL
END
```

follows

Defines the other jobs and job streams that must complete successfully before a job or job stream is launched.

Comments

Use the following syntax for job streams:

[**follows** {[*netagent::*][*workstation#*]*jobstreamname*

[**previous**|**sameday**|**relative from [+/-]** *time* **to [+/-]** *time*|**from** *time* [+/-*n* day[s]] **to** *time* [+/-*n* day[s]]

Use the following syntax for jobs:

[**follows** {[*netagent::*][*workstation#*]*jobstreamname*{.*jobname*}

[**previous**|**sameday**|**relative from [+/-]** *time* **to [+/-]** *time* | **from** *time* [+/-*n* day[s]] **to** *time* [+/-*n* day[s]] [**if** <*condition*> [| <condition>...]]

Arguments

netagent

The name of the network agent where the internetwork dependency is defined.

[folder/]workstation

The workstation on which the job or job stream that must have completed runs. The default is the same workstation as the dependent job or job stream.

If a *workstation* is not specified with *netagent*, the default is the workstation to which the network agent is connected.

Network agent workstations do not support folders, therefore neither the network agent nor the jobs or job streams running on them can be defined in folders. Folders are supported on all other workstation types, as follows:

```
 [follows {[folder/][workstation#]
[folder/]jobstreamname[.jobname]
```

[folder/]jobstreamname

The name of the job stream that must have completed. For a job, the default is the same job stream as the dependent job.

time

Specifies a time of day. Possible values range from **0000** to **2359**.

[folder/]jobname

The name of the job that must have completed. An `at` sign (@) can be used to indicate that all jobs in the job stream must complete successfully.

previous|sameday|relative from [+/-] *time* to [+/-] *time* | from *time* [+/-*n* day[s]] to *time* [+/-*n* day[s]]

Defines how the job stream or job referenced by an external follows dependency is matched to a specific job stream or job instance in the plan. See Comments on page 346 for a detailed explanation of these options.

[if <*condition*> [| <condition>...]]

Conditional dependencies are used when you need a successor job or job stream to start only after certain conditions are satisfied by the predecessor job or job stream. They can also be used to specify alternative flows in a job stream starting from a predecessor job or job stream. The successor job is determined by which conditions the predecessor job or job stream satisfies. The jobs in the flow that do not run, because the output conditions were not satisfied, are put in SUPPR state which is different from regular dependencies where jobs are put in Hold until the predecessor is in SUCC state.

The conditions expressed by [**if** <*condition*> [| <condition>...]] can be of different types: based on the job execution status of the predecessor job or job stream, the job status, and other conditions based on the output or outcome of the predecessor job or job stream. You can specify more than one condition, separated by the pipe (|) symbol but, you cannot specify a combination of types, one type only. For example, `IF ABEND | FAIL | SUPPR`.

When the predecessor is a job stream, the conditional dependency is only a status condition, as follows: `abend`, `succ`, and `suppr`. The successor job runs when the predecessor job stream status satisfies the job status specified using these arguments. You can specify one status, a combination of statuses, or all statuses. When specifying more than one status or condition name, separate the statuses or names by using the pipe (|) symbol.

if *Condition_Name*

Where, *Condition_Name* can represent both a status or an actual name that you assign to a condition that you define.

if exec

The successor job runs when the predecessor job has started.

if fail|abend|succ|suppr

The successor job runs when the predecessor job status satisfies the job status specified using these arguments. You can specify one status, a combination of statuses, or all statuses. When specifying more than one status, separate the statuses by using the pipe (|) symbol.

if *Condition_Name*]

The successor job runs when the predecessor job satisfies the output conditions defined for the *Condition_Name* specified. You can specify one condition name, or a combination of names. When specifying more than one condition name, separate the names by using the pipe (|) symbol. These output conditions are initially defined in the job definition.

Comments

Dependency resolution criteria define how the job stream or job referenced by an external follows dependency is matched to a specific job stream or job instance in the plan. Because the plan allows the inclusion of multiple instances of the same job or job stream, you can identify the instance that resolves the external follows dependency based on the following resolution criteria:

previous

Closest Preceding: The job or job stream instance that resolves the dependency is the closest preceding the instance that includes the dependency. Be aware that DST transitions can alter this chronological ordering. During a "spring forward" DST switch, a predecessor scheduled during the skipped hour will be shifted forward. If this newly evaluated time is later than the successor's time, the previous matching criteria is no longer met, the dependency will not resolve, and the successor will remain in a HOLD state.

sameday

Same Day: The job or job stream instance that resolves the dependency is the closest one in time scheduled to start on the day when the instance that includes the dependency is scheduled to run.

relative from [+/-] *time* to [+/-] *time*

Within a Relative Interval: The job or job stream instance that resolves the dependency is the closest one in a time interval of your choice, which is defined relatively to the scheduled start time of the dependent instance.

from *time* [+/-*n* day[s]] to *time* [+/-*n* day[s]

Within an Absolute Interval: The job or job stream instance that resolves the dependency is the closest one in a time interval of your choice. The time interval is not related to the scheduled start time of the dependent instance.

Regardless of which matching criteria are used, if multiple instances of potential predecessor job streams exist in the specified time interval, the rule used by the product to identify the correct predecessor instance is the following:

1. HCL Workload Automation searches for the closest instance that precedes the depending job or job stream start time. If such an instance exists, this is the predecessor instance.
2. If there is no preceding instance, HCL Workload Automation considers the correct predecessor instance as the closest instance that starts after the depending job or job stream start time.

The scheduler classifies follows dependencies as *internal* when they are specified only by their job name within the job stream. It classifies them as *external* when they are specified in the *workstation#/folder/jobstreamName.jobName* format.

When a job stream includes a job with a follows dependency that shares the same job stream name (for example, job stream `schedA` includes a job named `job6` that has a follows dependency on `schedA.job2`), the dependency is added to the plan as an *external* follows dependency. Because the scheduler uses the `sameday` matching criteria to resolve external dependencies, dependencies originated in this way are never added the first time the object is submitted.

For more information and examples on how external follows dependencies are resolved in the plan refer to Managing external follows dependencies for jobs and job streams on page 100.

Example

Examples

The following example specifies to not launch job stream `skedc` until the closest preceding job stream instance `sked4` on workstation `site1` has completed successfully:

```
schedule skedc  on fr  follows site1#sked4 previous
```

The following example specifies to not launch job stream `skedc` until the job stream instance of `sked4`, contained in the `payroll` folder, on workstation `site1` that run between 12:00 of 3 days before to 3:00 of the day after have completed successfully:

```
schedule skedc  on fr  follows site1#PAYROLL/sked4  from 1200 -3 days to 0300 -1 day
```

The following example specifies not to launch job stream `skedc` until job stream `sked4` on workstation `site1` and job `joba` in job stream `sked5` on workstation `site2` have completed successfully:

```
schedule skedc on fr
follows site1#sked4,site2#sked5.joba
```

Do not launch `sked6` until `jobx` in the job stream `skedx` on network agent `cluster4` has completed successfully:

```
sked6 follows cluster4::site4#skedx.jobx
```

The following example specifies not to launch `jobd` until `joba` in the same job stream, and `job3` in job stream `skeda` have completed successfully:

```
jobd follows joba,skeda.job3
```

The following example specifies to launch the job `LOADDATA_INFO` after the job `CHECKJOB` in the `CHECKDATA` job stream, only if job `CHECKJOB` completes in FAIL or ABEND state, and if job `CHECKJOB1` in the `CHECKDATA1` job stream satisfies the condition `STATUS_OK` defined in the `CHECKJOB1` job definition:

```
WK1#LOADDATA_INFO
 FOLLOWS WK2#CHECKDATA.CHECKJOB IF FAIL|ABEND
 FOLLOWS W32#CHECKDATA1.CHECKJOB1 IF  STATUS_OK
```

freedays

Use **freedays** to specify the name of a non-working days calendar that lists the non-working days for your enterprise. If and how a job stream runs on these particular days is defined in a *freedays* rule during the run cycle setup. HCL Workload Automation uses this calendar as the base calendar for calculating *workdays* for the job stream.

The keyword affects only the scheduling of the job streams for which it is specified.

Syntax

freedays *[folder/]Calendar_Name* [**-sa**] [**-su**]

Arguments

***[folder/]*Calendar_Name**

The name of the calendar that must be used as the non-working days calendar for the job stream. If *Calendar_Name* is not in the database, HCL Workload Automation issues a warning message when you save the job stream. If *Calendar_Name* is not in the database when the command **schedulr** runs, HCL Workload Automation issues an error message and uses the default calendar **holidays** in its place. Do not use the names of weekdays for the calendar names.

-sa

Saturdays are *workdays*.

-su

Sundays are *workdays*.

Comments

If you specify a non-working days calendar in the job stream definition, then the concept of *workdays* takes the following value: *workdays = everyday excluding saturday and sunday (unless you specified -sa or -su along with freedays) and excluding all the dates of Calendar_Name*

If you do not specify **freedays** in the job stream definition, then: *workdays = everyday excluding saturday and sunday and all the dates of the holidays calendar*

By default, *saturday* and *sunday* are considered as non-working days unless you specify the contrary by adding **-sa**, **-su** or both after *Calendar_Name*.

Example

Examples

Select job stream `sked2` to run on 01/01/2023 and on all workdays as long as they are not listed in the non-working days calendar named `GERMHOL`.

```
schedule sked2
freedays GERMHOL
on 01/01/2023, workdays
```

Select job stream `sked3` to run two workdays before each date in the `PAYCAL` calendar. Workdays are every day from Monday to Saturday as long as they are not listed in the non-working days calendar named `USAHOL`.

```
schedule sked3
freedays USAHOL -sa
on PAYCAL -2 workdays
```

Select job stream `sked3` on the dates listed in the `APDATES` calendar. If the selected date is a non-working day, do not run the job stream. In this example, Sundays and all the dates listed in the `GERMHOL` calendar are to be considered as non-working days. All days from Monday to Saturday, except for the dates listed in `GERMHOL`, are workdays.

```
schedule sked3
freedays GERMHOL -sa
on APDATES fdignore
```

Select job stream `testsked3` to run every weekday except 5/15/2023 and 5/23/2023. If 5/23/2023 is a non-working day, do not exclude it. In this example, Saturdays, Sundays, and all the dates listed in `GERMHOL` are to be considered as non-working days. All days from Monday to Friday, except for the dates listed in `GERMHOL`, are workdays.

```
schedule testskd3
freedays GERMHOL
on weekdays
except 5/15/2023 fdignore
except 5/23/2023
```

Select job stream `testsked4` to run every day except two weekdays prior to every date listed in the `MONTHEND` calendar. If the date to be excluded is a non-working day, do not exclude it, but exclude the nearest following workday. In this example, non-working days are all the dates listed in `USAHOL`, while workdays are all the days from Monday to Sunday that are not listed in `USAHOL`.

```
schedule testskd4
freedays USAHOL -sa -su
on everyday
except MONTHEND -2 weekdays fdnext
```

job statement

Jobs can be defined in the database independently (as described in Job definition on page 228), or as part of job streams. In either case, the changes are made in the database and do not affect the production plan until the start of a new production plan.

Syntax

To define a job as part of a job stream, use the following syntax inside the job stream definition:

[*workstation***#**]*jobname* [**as** *newname*]
{**scriptname** *filename* | **docommand** *"command"*}
streamlogon *username*
[**description** *"description"*]
[**tasktype** *tasktype*]
[**interactive**]
[**succoutputcond** *Condition_Name* "*Condition_Value*"]
[**outputcond** *Condition_Name* "*Condition_Value*"]
[**recovery**
{**stop**
[**after** [*[folder/]workstation#*][*folder/]jobname*]
[**abendprompt** "*text*"]]
| **continue**
[**after** [*[folder/]workstation#*][*folder/]jobname*]
[**abendprompt** "*text*"]]

|**rerun [same_workstation]**
[[**repeatevery** *hhmm*] [**for** *number* **attempts**]]
[**after** [*[folder/]workstation#*][*folder/]jobname*]
| [**after** [*[folder/]workstation#*][*folder/]jobname*]
[**abendprompt** "*text*"]}

To use a job already defined in the database in the job stream definition define *job statement* using the following syntax:

[*[folder/]workstation***#**][*folder/]jobname* [**as** *newname*]

Arguments

as

The name you want to use to refer to the job instance within that job stream.

For the other keywords refer to Job definition on page 228.

Comments

When defining a job as part of a job stream as the job stream definition is added to the database also the new job definition is added and can be referenced, from that moment on, also from other job streams.

Note: Wrongly typed keywords used in job definitions lead to truncated job definitions stored in the database. In fact the wrong keyword is considered extraneous to the job definition and so it is interpreted as the job name of an

additional job definition. Usually this misinterpretation causes also a syntax error or an inexistent job definition error for the additional job definition.

When a job stream is added or modified, the attributes or recovery options of its jobs are also added or modified. Remember that when you add or replace a job stream, any job modifications affect all other job streams that use the jobs. Note that the cross reference report, *xref*, can be used to determine the names of the job streams including a specific job.

For more information about cross reference report refer to xref on page 1020.

Note: Jobs scheduled to run on workstations marked as *ignored*, and belonging to job streams scheduled to run on active workstations, are added to the plan even though they are not processed.

Example

Examples

The following example defines a job stream with three previously defined jobs:

```
schedule bkup on fr at 20:00 :
  cpu1#jbk1
  cpu2#jbk2
    needs 1 tape
  cpu3#jbk3
    follows jbk1
end
```

The following job stream definition contains job statements that add or modify the definitions of two jobs in the database:

```
schedule sked4 on mo :
  job1  scriptname “d:\apps\maestro\scripts\jcljob1”
    streamlogon jack
    recovery stop abendprompt “continue production”
  site1#job2  scriptname “d:\apps\maestro\scripts\jcljob2”
    streamlogon jack
    follows job1
end
```

isservice

Marks a job stream as usable by Self-Service Catalog. When creating and editing SSC-ready job streams, it is recommended you use the Dynamic Workload Console.

Syntax

isservice

Comments

A job stream marked as **isservice** can be managed from the Self-Service Catalog as well as from the Dynamic Workload Console and command-line commands. If you define this keyword, the **servicename** keyword is required. For more information, see servicename on page 387.

Example

```
SCHEDULE BW-XFR-LNX53#JS2
DESCRIPTION "Added by composer."
ISSERVICE
SERVICENAME TEST2
SERVICEDESCRIPTION “This is my Service”
SERVICETAGS "|tag2|tag3|"
SERVICEVARIABLES "{var: \"value2\"}"
ON RUNCYCLE RC1 "FREQ=DAILY;"
:

BW-XFR-LNX53#JS2

END
```

join

Defines a set of joined dependencies on a job or job stream. The set of joined dependencies is satisfied when the specified number of dependencies are met.

Syntax

[**join** *join_name* [*number* | *numconditions* | ALL] OF

DESCRIPTION "..."]

...

ENDJOIN

Arguments

join_name

A meaningful name for the set of joined conditions. The maximum length is 16 characters.

number

The number of dependencies that must be met for the joined conditions to be satisfied. Supported values are:

0

All dependencies must be met.

1

The number of dependencies that must be met for the joined conditions to be satisfied.

numconditions

The number of dependencies that must be met for the joined conditions to be satisfied.

ALL

All dependencies must be met for the joined conditions to be satisfied

DESCRIPTION

A meaningful description for the set of joined conditions. This argument is optional.

Comments

For any single object, you are limited to a maximum of 4 join instances for each object with unlimited conditional statements within each join.

On a job, you can define standard and conditional dependencies, both internal and external.

On a job stream , you can define standard and conditional dependencies, only internal.

Internetwork dependencies are not supported.

Example

Examples

The following example shows a job stream containing the following dependencies:

1. A follows dependency on a job stream in ABEND or SUPPRESS state
2. A set of joined conditional dependencies, containing two dependencies, the first on job `nc007108#COND_DEP_JS1.COND_DEP_J1`, which must finish successfully for the dependency to be satisfied, the second on job `nc007108#COND_DEP_JS2.COND_DEP_J2`, which must meet the `EXT_STATUS_PREREQ_SUCC_2` condition. This condition is

defined by the user when creating the job definition. At least one of these conditions must be met for the set of joined conditional dependencies to be satisfied.

3. A follows dependency on an external job stream in EXEC state.
4. A second set of joined conditional dependencies, containing two dependencies, the first on job `CDJ_PRED_2_163955532` which must be in SUCC state and the second on job `CDJ_PRED_1_163955532`, which must be in INT_STATUS_PREDEC_SUCC_1 state. The second condition is defined by the user when creating the job definition. Both these conditions must be met for the set of joined conditional dependencies to be satisfied.

:

```
SCHEDULE nc007108#CDJS_163955532
FOLLOWS nc007108#COND_DEP_JS1.@ IF ABEND | SUPPRESS

JOIN JOINDEP_COV_01 1 OF
DESCRIPTION "Description for join JOINDEP_COV_01"
FOLLOWS nc007108#COND_DEP_JS1.COND_DEP_J1 IF SUCC
FOLLOWS nc007108#COND_DEP_JS2.COND_DEP_J2 IF EXT_STATUS_PREREQ_SUCC_2
ENDJOIN

:
nc007108#CDJ_PRED_1_163955532
nc007108#CDJ_PRED_2_163955532

FTA_nc007108#CDJ_FTA_163955532
FOLLOWS nc007108#COND_DEP_JS1.COND_DEP_J1 IF EXEC

JOIN JOINDEP_COV_02 ALL OF
FOLLOWS CDJ_PRED_2_163955532 IF SUCC
FOLLOWS CDJ_PRED_1_163955532 IF INT_STATUS_PREDEC_SUCC_1
ENDJOIN

END
```

jsuntil

The **jsuntil** keyword defines the latest start time of a job stream. It also determines the behavior of the jobs in the job stream when the job stream is approaching its latest start time. Use the **jsuntil** keyword to avoid that the job stream is either suppressed, canceled, or set to continue (depending on the action specified in the **onuntil** keyword) if it starts before its latest start time. For example, if you have a job stream with **jsuntil** set to 10:00 am, and one of the jobs starts running at 9:59 am, the job and its successors run as scheduled.

This keyword is mutually exclusive with the **until** keyword.

There is also a major difference with between the **until** and **jsuntil** keywords:

If you specify the until keyword in your job stream definition

This keyword is evaluated also after the job stream has started. As a result, if the latest start time expires before the job stream completes successfully, the action specified in the related **onuntil** keyword is performed on the job stream and on its jobs, which have not yet started.

If you specify the jsuntil keyword in your job stream definition

This keyword is evaluated only once, as soon as all dependencies of the job stream are satisfied and the job stream state changes to READY. If the latest start time defined using the **jsuntil** keyword has not expired at this time, it is no longer evaluated and the job stream runs independently of it. However, to prevent the job stream from remaining in READY state indefinitely, two days after the time specified in the **jsuntil** keyword has expired, the job stream is suppressed by default.

For more information about the **until** keyword, see until on page 390.

Syntax

[**jsuntil** *time* [**timezone**|**tz** *tzname*][**+***n* **day**[**s**]] [**onuntil***action*]]

Arguments

time

Specifies the time of day. The possible values are **0000** through **2359**.

tzname

Specifies the time zone to be used when computing the time. See Managing time zones on page 1064 for time zone names. The default is the time zone of the workstation on which the job or job stream is launched.

n

Specifies an offset, in days, from the scheduled date and time.

onuntil *action*

The action to be taken on a job stream whose until time has expired but the job stream is not yet completed in SUCC state. The following are the possible values of the *action* parameter:

suppr

The job or job stream and any dependent job or job stream do not run. This is the default behavior.

Once the until time expired on a job stream, the status for the job stream is calculated following the usual rules; suppressed jobs are not considered in the calculation. In case the job stream contains at least one **every** job its status is HOLD.

When the until time expires for a job, the job changes to HOLD status or keeps any previous status which is a final status.

If the until time is passed together with the **onuntil suppr** and the **carryforward** options, the job stream is *carry forwarded* by JnextPlan only if the until date is equal to the date when JnextPlan runs. If the until and the JnextPlan run dates are not the same, the job stream is not *carry forwarded*.

cont

The job or job stream runs when all necessary conditions are met and a notification message is written to the log when the until time elapses.

If the until time is passed together with the **onuntil cont** and the **carryforward** options, the job stream is always *carry forwarded* by JnextPlan.

canc

A job or job stream is cancelled when the **until** time specified expires. When using ***onuntil canc*** on jobs, the cancel operation on the job is issued by the FTA on which the job runs. Any job or job stream that was dependent on the completion of a job or job stream that was cancelled, runs because the dependency no longer exists.

If the until time is passed together with the **onuntil canc** and the **carryforward** options, the job stream is not *carry forwarded* by JnextPlan because it is already canceled.



Note: When using ***onuntil canc*** at job stream level, define as owner of the job stream the workstation highest in the hierarchy of the scheduling environment, among all workstations that own jobs contained in the job stream.

Comments

The **jsuntil** keyword is supported on components at version 9.4 Fix Pack 1, or later, with the exception of dynamic agents connected to a master at version 9.4 Fix Pack 1, or later. On all other agent types where a previous version of the product is installed, the **jsuntil** keyword is ignored.

Example

Examples

To schedule the BankReports job stream so that it starts at 08:00 and continues running if one of its jobs starts running within 9:59 and its duration exceeds 10:00, specify the following syntax:

```
schedule BankReports
on everyday
AT 0800 JSUntil 1000
end
```

keyjob

The **keyjob** keyword is used to mark a job as key in both the database and in the plan.

Syntax

keyjob

Example

Examples

In the following example, both the job and the job stream are marked as key.

```
SCHEDULE cpu1#sched1
ON everyday
KEYSCHED
AT 0100
cpu1#myjob1 KEYJOB
END
```

keysched

The **keysched** keyword is used to mark a job stream as key in both the database and in the plan and for monitoring by applications, such as Tivoli® Business Systems Manager. See the *HCL Workload Automation Integrating with Other Products* guide for information about enabling the key flag mechanism.

Syntax

keysched

Example

Examples

The following example :

```
SCHEDULE cpu1#sched1
ON everyday
KEYSCHED
AT 0100
cpu1#myjob1 KEYJOB
END
```

limit

The **limit** keyword limits the number of jobs that can run simultaneously in a job stream. This keyword works only if all the jobs in the job stream are defined on workstations that are managed by the same batchman process. There are three possible cases:

The workstation on which batchman runs is a fault-tolerant agent

In this case, the **limit** keyword limits the number of jobs, defined in the fault-tolerant agent and in the extended agents defined on this fault-tolerant agent, that can run simultaneously in the job stream. The job stream must contain only jobs defined in the fault-tolerant agent or in the extended agents running on the fault-tolerant agent.

The workstation on which batchman runs is a domain manager

In this case, in addition to workstations of the previous case, the job stream can contain jobs running on standard agents connected to the domain manager, or to any fault-tolerant agent managed by this domain manager, configured with the `FULLSTATUS` parameter set to **on**.

The workstation on which batchman runs is a dynamic domain manager

In addition to workstations of the previous cases, the job stream can contain jobs defined on the broker and its dynamic agents.

In other scenarios, the use of the **limit** keyword to limit the number of jobs that can run simultaneously might not work as expected.

Syntax

limit *joblimit*

Arguments

joblimit

Specifies the number of jobs that can be running at the same time in the job stream. Possible values are **0** through **1024**. If you specify **0**, you prevent all jobs from being launched, including the one with priority set to **GO** or **HI**.

Example

Examples

The following example limits to five the number of jobs that can run simultaneously in job stream `sked2`:

```
schedule sked2 on fr
  limit 5 :
```

matching

Sets a default for the matching criteria to be used in all follows dependencies where a matching criteria is not set in the job stream definition or in the jobs contained in the job stream.

Syntax

matching {**previous** |**sameday** | **relative from [+/-]** *time* **to [+/-]** *time*

Arguments

For information about the keyword used with **matching** see the follows on page 344 keyword.

Example

Examples

The following example shows the definition of job stream `SCHED2` that:

- Contains a `job1` that can be run today only if it was run yesterday.
- Needs the instance of job stream `SCHED1` running the same day to complete before running.

```
SCHEDULE PDIVITA1#SCHED2
ON RUNCYCLE RULE1 "FREQ=DAILY;"
ON RUNCYCLE CALENDAR2 CAL1
MATCHING PREVIOUS
FOLLOWS PDIVITA1#SCHED1.@ SAMEDAY
FOLLOWS PDIVITA1#SCHED2.JOB1
:
PDIVITA1#JOB1

PDIVITA1#JOB2
END
```

In this sample the external follows dependency from `PDIVITA1#SCHED2.JOB1` inherits the matching criteria specified in the **matching** keyword.

Comments

Note that if you delete a job stream and then add it again to the database, the job stream gets another identifier. For this reason, if the job stream contains `FOLLOWS` dependencies with `PREVIOUS` matching criteria, these dependencies are not matched when **JnextPlan** runs again, because they are cross-plan dependencies that refer to an old identifier.

In the following example, if you delete job stream `JS01`, to assure the referential integrity of the database, also `FOLLOWS TWS851MASTER#JS01.@` is deleted from the definition of `JS02` and from the Preproduction plan.

If you delete job stream `JS03`, to assure the referential integrity of the database, also `FOLLOWS TWS851MASTER#JS03.@ PREVIOUS` is deleted from the definition of `JS02` and from the Preproduction plan.

If you delete job stream `JS02` and then add it again to the plan, also its `FOLLOWS` dependencies are added again. When the plan is extended, the `FOLLOWS TWS851MASTER#JS03.@ PREVIOUS` dependency of job stream `JS02` does not match the instance of job stream `JS03` coming from the previous plan, and this dependency is not added.

At the next plan extension, the process works again.

```
SCHEDULE TWS851MASTER#JS01
ON RUNCYCLE RULE1 "FREQ=DAILY;"
:
TWS851MASTER#J02
END
SCHEDULE TWS851MASTER#JS03
ON RUNCYCLE RULE1 "FREQ=DAILY;"
SCHEDTIME 1000
CARRYFORWARD
:
TWS851MASTER#J03
END
SCHEDULE TWS851MASTER#JS02
ON RUNCYCLE RULE1 "FREQ=DAILY;"
:
TWS851MASTER#J01
FOLLOWS TWS851MASTER#JS01.@
FOLLOWS TWS851MASTER#JS03.@ PREVIOUS
END
```

To avoid this problem, use the composer command **replace** because, in this case, job stream identifiers do not change.

maxdur

Specifies the maximum length of time a job can run. You can express this time in either minutes, or as a percentage of the latest estimated duration for the job. If a job is running, and the maximum duration time has been exceeded, then the following actions occur:

- One of the following actions is triggered: Kill or Continue.
- The job is shown as exceeded in the following places:
  - When running showjob from the conman command line, `MaxDurationExceeded` is displayed.
  - From the Dynamic Workload Console in the job properties for the job.
  - An informational message is written to the *`TWS_home`*`/stdlist/logs/`*`yyyymmdd`*`_TWSMERGE.log` file.

If this job is still running when JnextPlan is run, then the job is inserted in the USERJOBS job stream. The maximum duration setting is not maintained for the job in the USERJOBS job stream and will not be monitored. To have the job stream carried forward and avoid having the job being moved to the USERJOBS job stream, flag the original job stream where the maximum duration setting was specified as a carryforward job stream (setting the carryforward keyword in the job stream) if the enCarryForward global option is set to `yes`, otherwise, set the enCarryForward global option to `all`.

Syntax

maxdur *time* | *percentage* % **onmaxdur** *action*

Arguments

time

Specifies a length of time expressed using the syntax *HHHMM* where,

HHH

Represents the number of hours and is a number ranging from 000-500.

MM

Represents the number of minutes and is a number ranging from 00-59.

percentage

Specifies the percentage of the latest estimated duration. It can be a number ranging from 0-1000000.

onmaxdur ***action***

Specifies the action to be triggered on a job that is still running when the maximum duration specified for this job has been exceeded. The following are the possible values of the *action* parameter:

Kill

Specify to stop the running job. Killed jobs end in the ABEND state. Any jobs or job streams that are dependent on a killed job are not released. Killed jobs can be rerun.

Continue

Specifies to let the running job continue to run even if it has exceeded the maximum time duration.

When submitting a conman command to set or change the **onmaxdur** action, you must also specify the **maxdur** keyword in connection with the **onmaxdur** argument.

Example

Examples

The following example specifies to continue a running job if it is still running after one hour and 20 minutes:

```
MAXDUR 80 ONMAXDUR CONT
```

The following example specifies to kill a running job when if the job runs longer than one hour and 20 minutes:

```
MAXDUR 80 ONMAXDUR KILL
```

The following example specifies to continue a running job if the job is still running after it has exceeded 120% of its maximum duration where the maximum duration is based on the latest estimated duration:

```
MAXDUR 120 % ONMAXDUR KILL
```

mindur

Specifies the shortest amount of time within which a job normally runs and completes. If a job completes before this minimum amount of time is reached, then the following actions are performed:

- One of the following actions is triggered: Abend, Confirm, or Continue.
- The job is shown as to have not reached its minimum duration in the following places only if the job completes with SUCCESS:
  - When running showjobs, showjobs ;props from the conman command line, `MinDurationNotReached` is displayed.
  - From the Dynamic Workload Console in the job properties for the job.
  - An informational message is written to the *`TWS_home`*`/stdlist/logs/`*`yyyymmdd`*`_TWSMERGE.log` file.

If this job is still running when JnextPlan is run, then the job is inserted in the USERJOBS job stream. The minimum duration setting is not maintained for the job in the USERJOBS job stream and will not be monitored. To have the job stream carried forward and avoid having the job being moved to the USERJOBS job stream, flag the original job stream where the minimum duration setting was specified as a carryforward job stream (setting the carryforward keyword in the job stream) if the enCarryForward global option is set to `yes`, otherwise, set the enCarryForward global option to `all`.

Syntax

mindur *time* | *percentage* % **onmindur** *action*

Arguments

time

Specifies a length of time expressed using the syntax *HHHMM* where,

HHH

Represents the number of hours and is a number ranging from 000-500.

MM

Represents the number of minutes and is a number ranging from 00-59.

percentage

Specifies the percentage of the latest estimated duration. It can be a number ranging from 0-1000000.

onmindur *action*

Specifies the action to be triggered on a job that completes before its minimum duration. The following are the possible values of the *action* parameter:

Abend

The job is set to **ABEND** status.

Confirm

The job is set to **CONFIRM** status The workload requires a user confirmation to proceed.

Continue

The workload running continues without taking any action.

Example

Examples

The following example specifies to continue a running workload even if the job does not reach a minimum duration of at least 80 minutes:

```
MINDUR 120 ONMINDUR CONT
```

The following example specifies to set the job status to Error if the job does not run for at least 80 minutes:

```
MINDUR 120 ONMINDUR ABEND
```

The following example requires a user to confirm the job when it has reached 50% or half of its latest estimated minimum duration:

```
MINDUR 50 % ONMINDUR CONFIRM
```

needs

The **needs** keyword defines resources that must be available before a job or job stream is launched. You can use the **needs** keyword either in a job stream definition or in the definition of the contained jobs, not in both.

Syntax

needs [*n*] *[folder/]*[*workstation***#**]*[folder/]resourcename* [,...]

Arguments

n

Specifies the number of resource units required. Possible values are **1** to **1024** for each **needs** statement. The default is 1.

[folder/]workstation

Specifies the name of the workstation on which the resource is locally defined, and optionally, the name of the folder in which the workstation is defined. If not specified, the default is the workstation where the dependent job or job stream runs. Resources can be used as dependencies only by jobs and job streams that run on the workstation where the resource is defined.

Due to the resources dependencies resolution mechanism, a resource dependency at job stream level can be considered 'local' (and then its use supported) rather than 'global', when both the job stream and all its jobs are defined on the same workstation as the resource.

However, a standard agent and its host can reference the same resources.

[folder/]resourcename

Specifies the name of the resource.

Comments

A job or job stream can request a maximum of 1024 units of a resource in a **needs** statement. At run time, each **needs** statement is converted in *holders*, each holding a maximum of 32 units of a specific resource. Independently from the amount of available units of the resource, for a single resource there can be a maximum of 32 holders. If 32 holders are already defined for a resource, the next job or job stream waiting for that resource waits until a current holder terminates AND the needed amount of resource becomes available.

Example

Examples

The following example prevents job stream `sked3` from being launched until three units of `cputime`, and two units of `tapes` become available:

```
schedule sked3 on fr
  needs 3 cputime,2 tapes :
```

The `jlimit` resource has been defined with two available units. The following example allows no more than two jobs to run concurrently in job stream `sked4`:

```
schedule sked4 on mo,we,fr :
  joba needs 1 jlimit
  jobb needs 1 jlimit
  jobc needs 2 jlimit    <<runs alone>>
  jobd needs 1 jlimit
end
```

nop

The `nop` keyword specifies that the job is not to run when the plan executes. The job is included in the plan, as part of the job stream in which it is featured, but as the plan runs, it is placed in `Cancel Pending` status and is not executed. If there are standard dependencies defined on the job, the dependencies are released and the successors are executed. In the case of conditional dependencies, the condition must be verified as true for the successors to be executed.

This option represents a quick and easy way to temporarily *disable* a job from executing without having to delete it from the job stream and change its dependencies. For example, if you need to temporarily disable a print job from running because the printer is out of service, you can NOP the job in the job stream definition until the printer is fixed or replaced.

Syntax

job-statement
[...]
[nop]
[...]

Example

Examples

In the following example job `jbk1` is defined in job stream `bkup` with the NOP option. It will be flagged as `Cancel Pending` in all the Symphony files to come as long as the option is defined. Successor job `jbk3` will run.

```
schedule bkup on fr at 20:00 :
cpu1#jbk1
nop
cpu2#jbk2
needs 1 tape
cpu3#jbk3
follows jbk1
end
```

on

This is a job stream keyword that defines when and how often a job stream is selected to run. If omitted the job stream is not added to the preproduction plan. The **on** keyword must follow the **schedule** keyword. See except on page 337 for more information.

Syntax

on [**runcycle** *name*

[**valid from** *date*] [**valid to** *date*]

[**description** *"text"*]

[**vartable** *table_name*]]

{*date*|*day*|*[folder/]calendar*|*request*|*"icalendar"*|*runcyclegroup*} [**,...**]

[**fdignore**|**fdnext**|**fdprev**][**subset** *subsetname* **AND**|**OR**]

Arguments

runcycle *name*

Specifies a label with a friendly name for the run cycle specified in the following lines.

valid from *date* ... valid to *date*

Delimits the time frame during which the job stream is active, that is the job stream is added to the production plan. Note that the date specified as **valid to** value is not included in the run cycle, therefore on this date the job stream is not active.

description *"text"*

Contains a description of the run cycle.

vartable

Specifies the name of the variable table to be used by the run cycle.

date

Specifies a run cycle that runs on specific dates. The syntax used for this type is:

yyyymmdd [,yyyymmdd][,...]

For example, for a job stream that is scheduled to run on the 25th of May 2023 and on the 12th of June 2023 the value is:

```
on
20230525,20230612
```

day

Specifies a run cycle that runs on specific days. The syntax used for this type is:

{mo|tu|we|th|fr|sa|su}

For example, for a job stream that is scheduled to run every Monday the value is:

```
on
mo
```

[folder/]calendar

The dates specified in a calendar with this name. The calendar name can be followed by an offset in the following format:

{**+** | **-**}*n* {**day**[**s**] | **weekday**[**s**] | **workday**[**s**]}

Where:

n

The number of days, weekdays, or workdays.

days

Every day of the week.

weekdays

Every day of the week, except Saturday and Sunday.

workdays

Every day of the week, except for Saturdays and Sundays (unless otherwise specified with the **freedays** keyword) and for the dates marked either in a designated non-working days calendar or in the **holidays** calendar.

request

Selects the job stream only when requested. This is used for job streams that are selected by name rather than date. To prevent a scheduled job stream from being selected for **JnextPlan**, change its definition to ON REQUEST.

Note: When attempting to run a job stream that contains "on request" times, consider that:

- "On request" always takes precedence over "at".
- "On request" never takes precedence over "on".

icalendar

Represents a standard used to specify a recurring rule that describes when a job stream runs.

The syntax used for run cycle with type *icalendar* is the following:

FREQ={DAILY|WEEKLY|MONTHLY|YEARLY}

[;INTERVAL=[-]*n*]

[;{BYFREEDAY|BYWORKDAY|BYDAY=*weekday_list*|

BYMONTHDAY=*monthday_list***}]**

where the default value for keyword **INTERVAL** is 1.

Using *icalendar* you can specify that a job stream runs:

every *n* days

by using the following format:

FREQ=DAILY[;INTERVAL=*n*]

where the value set for **valid from** is the first day of the resulting dates.

For example, for a job stream that is scheduled to run daily the value is:

```
FREQ=DAILY
```

For a job stream that is scheduled to run every second day the value is:

```
FREQ=DAILY;INTERVAL=2
```

every free or work days

by using the following format:

FREQ=DAILY[;INTERVAL=*n*]

;BYFREEDAY|BYWORKDAY

For example, for a job stream that is scheduled to run every non-working day the value is:

```
FREQ=DAILY;BYFREEDAY
```

For a job stream that is scheduled to run every second workday the value is:

```
FREQ=DAILY;INTERVAL=2;BYWORKDAY
```

every *n* weeks on specific *weekdays*

by using the following format:

FREQ=WEEKLY[;INTERVAL=*n*]

;BYDAY=*weekday_list*

where the value set for *weekday_list* is

```
[SU][,MO][,TU][,WE][,TH][,FR][,SA]
```

For example, for a job stream that is scheduled to run every week on Friday and Saturday the value is:

```
FREQ=WEEKLY;BYDAY=FR,SA
```

For a job stream that is scheduled to run every three weeks on Friday the value is:

```
FREQ=WEEKLY;INTERVAL=3;BYDAY=FR
```

every *n* months on specific dates of the month

by using the following format:

FREQ=MONTHLY[;INTERVAL=*n*]

;BYMONTHDAY=*monthday_list*

where the value set for *monthday_list* is represented by a list of

```
[+number_of_day_from_beginning_of_month]
[-number_of_day_from_end_of_month]
[number_of_day_of_the_month]
```

For example, for a job stream that is scheduled to run monthly on the 27th day the value is:

```
FREQ=MONTHLY;BYMONTHDAY=27
```

For a job stream that is scheduled to run every six months on the 15th and on the last day of the month the value is:

```
FREQ=MONTHLY;INTERVAL=6;BYMONTHDAY=15,-1
```

every *n* months on specific days of specific weeks

by using the following format:

FREQ=MONTHLY[;INTERVAL=*n*]

;BYDAY=*day_of_m_week_list*

where the value set for *day_of_m_week_list* is represented by a list of

```
[+number_of_week_from_beginning_of_month]
[-number_of_week_from_end_of_month]
[weekday]
```

For example, for a job stream that is scheduled to run monthly on the first Monday and on the last Friday the value is:

```
FREQ=MONTHLY;BYDAY=1MO,-1FR
```

For a job stream that is scheduled to run every six months on the 2nd Tuesday the value is:

```
FREQ=MONTHLY;INTERVAL=6;BYDAY=2TU
```

every *n* years

by using the following format:

FREQ=YEARLY[;INTERVAL=*n*]

where the value set for **valid from** is the first day of the resulting dates.

For example, for a job stream that is scheduled to run yearly the value is:

```
FREQ=YEARLY
```

For a job stream that is scheduled to run every two years the value is:

```
FREQ=YEARLY;INTERVAL=2
```

Note: The following limitations apply:

- the maximum supported interval for a daily run cycle is 31 days.
- the maximum supported interval for a weekly run cycle is 8 weeks.
- the maximum supported interval for a monthly run cycle is 12 months. For run cycles specifying the day of the week based on the month, for example the third Saturday or the second Friday, the maximum supported interval is 5 days.
- the maximum supported interval for a yearly run cycle is 10 years.

runcyclegroup

Specified one or more run cycles that combined together produce a set of run dates for a job stream. The run cycle group must be expressed using the following syntax: `$RCG runcyclegroupname`.

fdignore|fdnext|fdprev

Indicates the rule to be applied if the date selected for running the job or job stream falls on a non-working day. The available settings are:

fdignore

Do not add the date.

fdnext

Add the nearest workday after the non-working day.

fdprev

Add the nearest workday before the non-working day.

[subset *subsetname* AND|OR]

subset *subsetname*

Specifies the name of the subset. If you do not specify a name, `SUBSET_1`, is used by default.

AND|OR

By default, run cycles within a subset are in a logical **OR** relationship but you can change it to a logical **AND**, as long as the run cycle group result is a positive date or set of dates (Inclusive).

Comments

You can define multiple instances of the **on** keyword for the same job stream. Multiple **on** instances must be consecutive within the job stream definition. Each instance is equivalent to a run cycle to which you can associate a freeday rule.

Each instance of the keyword can contain any of the values allowed by the **on** syntax.

If the run cycle and job stream start times are both defined, the run cycle start time takes precedence when the job stream is scheduled with JNextPlan. When the job stream is launched with the submit command, the run cycle start time is not used.

Example

Examples

The following example selects job stream `sked1` on Mondays and Wednesdays:

```
schedule sked1 on mo,we
```

The following example selects job stream `sked3` on June 15, 2023, and on the dates listed in the `apdates` calendar:

```
schedule sked3 on 6/15/23,apdates
```

The following example selects job stream `sked4` two weekdays before each date appearing in the `monthend` calendar:

```
schedule sked4 on monthend -2 weekdays
```

The following example selects job stream `testskd1` every weekday except on Wednesdays:

```
schedule testskd1 on weekdays
  except we
```

The following example selects job stream `testskd3` every weekday except May 15, 2023 and May 24, 2023:

```
schedule testskd3 on weekdays
  except 05/16/2023,05/24/2023
```

The following example selects job stream `testskd4` every day except two weekdays prior to any date appearing in a calendar named `monthend`:

```
schedule testskd4 on everyday
  except monthend -2 weekdays
```

Select job stream `sked1` to run all Mondays, Fridays, and on 29/12/2023. If Mondays and 29/12/2023 are non-working days, run the job stream on the nearest following workday. If Fridays are non-working days, run the job stream on the nearest preceding day. In this example, the non-working days are Saturdays, Sundays, and all the dates listed in the default `HOLIDAYS` calendar. Workdays are all days from Monday to Friday if they are not listed in the `HOLIDAYS` calendar.

```
schedule sked1
on mo, 12/29/2023 fdnext
on fr fdprev
```

This example shows the output of the display command of job stream `testcli` defined to run on different run cycles on workstation `site2`:

```
display js=site2#testcli
```

obtained in 120-column format by setting *MAESTROCOLUMNS*=120 before accessing the **composer** command-line:

```
JobstreamName Workstation Draft Valid From  Valid To UpdatedBy UpdatedOn  LockedBy
------------- ----------- ----- ---------  -------- --------- ---------  --------
TESTCLI       SITE2       Y     08/25/2023 -       mdmDBE4   08/25/2023 mdmDBE4

SCHEDULE W5#TESTCLI VALID FROM 08/25/2023 TIMEZONE ACT
DESCRIPTION "Job stream with several run cycle settings."
DRAFT
ON RUNCYCLE M5 VALID FROM 08/25/2023
   DESCRIPTION "monthly"
   "FREQ=MONTHLY;INTERVAL=5;BYMONTHDAY=-3,1"
   ( AT 0000 )
ON RUNCYCLE W4 VALID FROM 08/25/2023
   DESCRIPTION "weekly"
   "FREQ=WEEKLY;INTERVAL=5;BYDAY=MO,WE"
   FDNEXT ( AT 0000 )
ON RUNCYCLE D3 VALID FROM 08/25/2023
   DESCRIPTION "daily"
   "FREQ=DAILY;INTERVAL=2"
   FDPREV ( AT 0000 )
ON RUNCYCLE C2 VALID FROM 08/25/2023
   DESCRIPTION "calendar"
   ITALY +2 DAYS
   ( AT 0000 )
ON RUNCYCLE M6 VALID FROM 08/25/2023
   DESCRIPTION "monthly"
   "FREQ=MONTHLY;INTERVAL=2;BYDAY=1MO,1TH,2WE"
   ( AT 0000 +2 DAYS )
ON RUNCYCLE Y7 VALID FROM 08/25/2023
   DESCRIPTION "yearly"
   "FREQ=YEARLY;INTERVAL=7"
   ( AT 0100 )
ON RUNCYCLE SS1 VALID FROM 08/25/2023
   08/10/2023,08/18/2023,08/20/2023,08/25/2023
   ( AT 0000 UNTIL 0000 +1 DAYS ONUNTIL SUPPR DEADLINE 0000 +2 DAYS )
EXCEPT RUNCYCLE S1 VALID FROM 08/25/2023
   DESCRIPTION "simple"
   08/26/2023,08/28/2023,08/30/2023,09/13/2023
   ( AT 0000 )
```

```
CARRYFORWARD
MATCHING SAMEDAY
FOLLOWS LAB235004#SROBY2.@
FOLLOWS X8#COPYOFJS2.RR
FOLLOWS XA15::TPA
KEYSCHED
LIMIT 22
PRIORITY 15
:
X8#PIPPO AS JOBTC
 CONFIRMED
 PRIORITY 13
 KEYJOB
 FOLLOWS W5#POPO.@
 FOLLOWS X8#JS2.F3
END

AWSBIA291I Total objects: 1
```

The calendar `ITALY` is a custom calendar defined in the database that sets the workdays and holidays of the calendar in use in Italy.

onlate

The **onlate** keyword defines the action to be taken on a job in job stream when the job's deadline expires.

Syntax

onlate *action*

Arguments

action

Specifies the action to be taken on the job when the job's deadline expires. The supported action is **kill**. If the job is running when the deadline expires, it is killed. Killed jobs end in the ABEND state. Any jobs or job streams that are dependent on a killed job are not released. If the dependency on the job is a conditional dependency on the job completing in ABEND state, that dependency is released.

Comments

This keyword applies only to jobs defined in job streams.

If you do not specify an **until** time for the job and specify a **deadline** time with an **onlate kill** action, the **until** keyword is automatically set to the same time as the **deadline** keyword. As a result, if the job has not yet started, it is suppressed anyway.

The **until** keyword is defined only at plan level, therefore it does not modify the job definition.

Example

Examples

The following example launches job stream `ABSENCES` every day and job `calc_job` to start running at 14:30 and to be completed by 16:00. If the `calc_job` job does not complete by the 1600 deadline, it is killed and stops running.

```
schedule ABSENCES on everyday :
    ABSENCES at 1430 deadline 1600 onlate kill
end
```

onoverlap

Defines the action that the scheduler must take on a job stream instance that is about to start but the previous instance has not yet completed. The options are to:

- Start the job stream instance anyway.
- Wait for the previous instance to complete.
- Cancel running the new instance altogether.

Syntax

onoverlap parallel|enqueue|donotstart

Arguments

parallel

The next instance is started regardless, and the two instances run concurrently. This is the default behavior for the job stream if you do not use the `onoverlap` keyword.

enqueue

The next instance is not started until the previous instance has completed its run.

Internally, this option is implemented as a **follows** dependency on the previous instance of the job stream. This internal dependency counts toward the maximum limit of 40 dependencies allowed for a job or job stream. Therefore, if you use **enqueue**, you can define a maximum of **39** additional manual dependencies (such as **follows**, **needs**, or **opens**) to avoid exceeding the plan limit.

donotstart

If, for any reason, the job stream cannot start within four minutes, the next instance does not start at all.

At planning time, a new dependency is added to the previous instance. The new instance includes a latest start condition, ensuring it can only begin if the dependency is released within four minutes of the previous instance start time. The added dependency prevents the new instance from getting stuck if the prior instance takes excessively long to complete. If the four-minute timeout is exceeded for any reason, the new instance does not start.

Note: Do not use `onoverlap donotstart` action on job stream instances that are defined on a domain different from the master domain manager.

Example

In the following example, an instance of job stream JS_0415 is run every 10 minutes. In case an instance has not completed when the next one is to start, the next instance waits for its completion.

Note: When using **onoverlap enqueue**, the system automatically generates a dependency in the plan to track the status of the previous instance. If the total number of manual dependencies plus the internal **onoverlap** dependency exceeds 40, the system returns error **AWSJOM138E** and the plan might experience delays or synchronization issues.

```
SCHEDULE MDM005#JS_0415
ON RUNCYCLE ACCTRC 06/16/2025
( SCHEDTIME 1445 DEADLINE 1530 EVERY 0010 EVERYENDTIME 1515 )
ONOVERLAP ENQUEUE
# Note: Because ENQUEUE is used, ensure the job stream
# below has no more than 39 additional FOLLOWS or NEEDS dependencies.
:
MDM005#JS_0415
END
```

opens

Specifies files that must be available before a job or job stream can be launched.

Syntax

opens *[folder/]*[*workstation#*]"*filename*" [(*qualifier*)] [,...]

Arguments

[folder/]workstation

Specifies the name of the workstation or workstation class on which the file exists, and optionally, the name of the folder in which the workstation is defined. The default is the workstation or workstation class of the dependent job or job stream. If you use a workstation class, it must be the same as that of the job stream that includes this statement.

filename

Specifies the name of the file, inclusive of its full path, enclosed in quotation marks. You can use HCL Workload Automation parameters as part or all of the file name string. You can also use variables defined in the variable table of the workstation on which the file exists. Refer to Variable and parameter definition on page 289 for additional information and examples.

qualifier

Specifies a valid test condition. In UNIX®, the qualifier is passed to a **test** command, which runs as **root** in bin/sh. However, on dynamic agents, if a no root agent installation was performed, then you must verify that the test command is available to the installation user and the test will be run as the HCL Workload Automation user (for example, the installation user).

For pools and dynamic pools, because it is not possible to know in advance on which member agent the test will run, and there is no affinity between the agent that satisfies the condition and the agent that runs the job, then a file dependency is recommended only in the case of a condition that is to be evaluated on a shared file system.

If you want to have at least one agent satisfy the condition and to run the job, see the event-driven workload automation file monitor with the TWSAction submit actions, using a variable as the workstation. See TWSAction actions in the Event-driven workload automation event and action definitions section of the *HCL Workload Automation: User's Guide and Reference*. See also the topic about using file dependencies in dynamic scheduling for more information about managing file dependencies with dynamic agents, pools, and dynamic pools in the *HCL Workload Automation: User's Guide and Reference*.

Attention:

- On UNIX operating systems, the list of the supported qualifiers depends on the operating system type. You can verify the supported qualifiers by running the **bin/sh/test** command.
- On Windows®, the test function is performed as theHCL Workload Automation user (for example, the installation user).

The valid qualifiers are:

-d %p

True if the file exists and is a directory.

-e %p

True if the file exists.

-f %p

True if the file exists and is a regular file.

-r %p

True if the file exists and is readable.

-s %p

True if the file exists and its size is greater than zero.

-w %p

True if the file exists and is writable.

-a

Boolean operator AND.

-o

Boolean operator OR.

In both UNIX® and Windows®, the expression **%p**, is used to pass the value assigned to *filename* to the test function.

Entering **(notempty)** is the same as entering **(-s %p)**. If no qualifier is specified, the default is **(-f %p)**.

Comments

The combination of the *path of the file* and the *qualifiers* cannot exceed 145 characters, and the *name of the file* cannot exceed 28 characters.

Example

Examples

The following example checks to see that file `c:\users\fred\datafiles\file88` on workstation `nt5` is available for read access before launching `ux2#sked6`:

```
schedule ux2#sked6 on tu opens nt5#"c:\users\fred\datafiles\file88"
```

The following example checks to see if three directories, `/john`, `/mary`, and `/roger`, exist under `/users` before launching job `jobr2`:

```
jobr2  opens "/users"(-d %p/john -a -d %p/mary -a -d %p/roger)
```

The following example checks to see if `cron` has created its FIFO file before launching job `job6`:

```
job6  opens "/usr/lib/cron/FIFO"(-p %p)
```

The following example checks to see that file `d:\work\john\execit1` on workstation `dev3` exists and is not empty, before running job `jobt2`:

```
jobt2  opens dev3#"d:\work\john\execit1"(notempty)
```

The following example checks to see that file `c:\tech\checker\startf` on workstation `nyc` exists, is not empty, and is writable, before running job `job77`:

```
job77  opens nyc#"C:\tech\checker\startf"(-s %p -a -w %p)
```

Security for test(1) Commands

In UNIX®, a special security feature prevents unauthorized use of other commands in the qualifier. For example, the file below contains a command in the qualifier:

```
/users/xpr/hp3000/send2(-n "`ls /users/xpr/hp3000/m*`" -o -r %p)
```

If the qualifier contains another command, the following checks are made:

- The local option ***jm no root*** must be set to no.
- In the security file, the user documenting the schedule or adding the Open Files dependency with a **conman adddep** command, must have submit access to a job with the following attributes:

  name=`cmdstest.fileeq`

  logon=root

  jcl=the path of the opens files

  cpu=the CPU on which the opens files reside

  Note that `cmdstest` and `fileeq` do not exist.

priority

Sets the priority of a job or job stream. By assigning a different priority to jobs or job streams you determine which one starts first, if the dependencies are solved.

Assuming the jobs and job streams are ready to be launched , if you set a priority for the job streams and for the jobs in the job streams:

- The job stream that starts first is the one with the highest priority.
- Among the jobs in the job stream with the highest priority, the job that starts first is the one with the highest priority.

Syntax

priority *number* | **hi** | **go**

Arguments

number

Specifies the priority. Possible values are **0** through **99**. A priority of 0 prevents the job or job stream from launching. The default value is 10 and is not displayed when viewing the job stream definition.

hi

Represents a value higher than any value that can be specified with a number. When set, the job or job stream is immediately launched as soon as it is free from all dependencies.

go

Represents the highest priority that can be set. When set, the job or job stream is immediately launched as soon as it is free from all dependencies.

Comments

Jobs and job streams with **hi** or **go** priority levels are launched as soon as all their dependencies are resolved. In this case:

- Job streams override the cpu job limit.
- Jobs override the cpu job limit, but they override neither the schedule job limit nor the cpu job fence.

Example

Examples

The following example shows the relationship between job stream and job priorities. The two job streams, `sked1` and `sked2` have the following definitions in the database:

```
schedule sked1 on tu
priority 50
:
job1 priority 15
job2 priority 10
end
```

```
schedule sked2 on tu
priority 10
:
joba priority 60
jobb priority 50
end
```

Since the job stream `sked1` has the highest priority then the jobs are launched in the following order: `job1`, `job2`, `joba`, `jobb`.

If, instead, the job stream priorities are the same, the jobs are launched in the following order: `joba`, `jobb`, `job1`, `job2`.

If `job2` has a dependency **A** and `job1` has a dependency **B** and the dependency **A** becomes solved (while **B** remains not solved) then `job2` starts before `job1` even though `job2` has a priority lower than the one set for `job1`.

prompt

Specifies prompts that must be answered affirmatively before a job or job stream is launched.

Syntax

prompt *[folder/]promptname* [**,**...]

prompt "[**:** | **!**]*text*" [**,**...]

Arguments

[folder/]promptname

Specifies the name of a prompt in the database. You can specify more than one *promptname* separated by commas but you cannot mix under the same **prompt** keyword prompts defined in the database with literal prompts.

text

Specifies a literal prompt as a text string enclosed in quotes ("). Multiple strings separated by backlash **n** (\n) can be used for long messages. If the string begins with a colon (:), the message is displayed but no reply is

necessary. If the string begins with an exclamation mark (!), the message is displayed, but it is not recorded in the log file. You can include backslash **n** (\n) within the text for new lines.

You can use one or more parameters as part or all of the text string. To use a parameter, place its name between carets (^). Refer to Variable and parameter definition on page 289 for additional information and examples.



Note: Within a local prompt, when not specifying a parameter, carets (^) must be preceded by a backslash (\) or they cause errors in the prompt. Within global prompts, carets do not have to be preceded by a backslash.

Example

Examples

The following example shows both literal and named prompts. The first prompt is a literal prompt that uses a parameter named `sys1`. When a single affirmative reply is received for the prompt named `apmsg`, the dependencies for both `job1` and `job2` are satisfied.

```
schedule sked3 on tu,th
  prompt "All ap users logged out of ^sys1^? (y/n)"
:
  job1 prompt apmsg
  job2 prompt apmsg
end
```

The following example defines a literal prompt that appears on more than one line. It is defined with backlash **n** (\n) at the end of each line:

```
schedule sked5 on fr
  prompt  "The jobs in this job stream consume \n
an enormous amount of cpu time.\n
Do you want to launch it now? (y/n)"
:
  j1
  j2 follows j1
end
```

schedtime

Represents the time when the job stream is positioned in the plan. The value assigned to **schedtime** does not represent a dependency for the job stream. While the production plan is in process, the job or job stream instance might start processing before the time set in the **schedtime** keyword if all its dependencies are resolved and if its priority allows it to start.

Syntax

schedtime *time* [**timezone**|**tz** *tzname*][**+***n* **day**[**s**]] [**,...**]

Arguments

time

Specifies a time of day in the format: HHHHmm. Possible values are from **0000** to **240000**, or **0000 + <number of days>**.

Where <number of days> can be from 1 to 100 days.

tzname

Specifies the time zone to be used when calculating the start time. See Managing time zones on page 1064 for time zone names. The default is the time zone of the workstation on which the job or job stream is launched.

n

Specifies an offset in days from the scheduled start date and time.

Comments

Differently from the **at** key, the **schedtime** key does not represent a time dependency, that is it does not state a time before which a job or job stream cannot start. Instead, the value specified in the **schedtime** keyword is used only to position the specific job or job stream instance in the preproduction plan. While the production plan is in process, the job or job stream instance might start processing before the time set in the **schedtime** keyword if all its dependencies are resolved and if its priority allows it to start.

For an explanation on how the **schedtime** keyword is used to identify predecessors in the preproduction plan, refer to Managing external follows dependencies for jobs and job streams on page 100.

The **at** and **schedtime** keywords are mutually exclusive. If **schedtime** is not specified and the **at** keyword is specified in the job or job stream, then its value is used to position the instance in the preproduction plan.

If neither the **at** nor the **schedtime** keywords are specified in the job or job stream definition, it is assumed by default to be the value assigned to the *startOfDay* global option set on the master domain manager.

For job streams with a **schedtime** definition, the value of the `Start time` field displayed on the Dynamic Workload Console depends on the setting of the `enPreventStart` global option (which determines if job streams without an `at` dependency can start immediately, without waiting for the run cycle specified in the job stream):

- If `enPreventStart` is set to `yes`, the start time is 12:00 AM converted to the time zone specified on the graphical user interface.
- If `enPreventStart` is set to `no`, the start time field is blank.

Example

Examples

The following examples assume that the HCL Workload Automation processing day starts at 6:00 a.m.

- The following job stream, selected on Tuesdays, is scheduled to start at 3:00 a.m. on Wednesday morning. Its two jobs are launched as soon as possible after the job stream starts processing.

```
schedule sked7 on tu schedtime 0300:
job1
job2
end
```

- The time zone of workstation `sfran` is defined as America/Los_Angeles, and the time zone of workstation `nycity` is defined as America/New_York. Job stream `sked8` is selected to run on Friday. It is scheduled to start on workstation `sfran` at 10:00 a.m. America/Los_Angeles Saturday (as specified by the `+ 1 day` offset). Job `job1` is launched on `sfran` as soon as possible after the job stream starts processing. Job `job2` is launched on `sfran` at 2:00 p.m. America/New_York (11:00 a.m. America/Los_Angeles) Saturday. `job3` is launched on workstation `nycity` at 4:00 p.m. America/New_York (1:00 p.m. America/Los_Angeles) Saturday.

```
sfran#schedule sked8 on fr schedtime 1000 + 1 day:
job1
job2 at 1400 tz America/New_York
nycity#job3 at 1600
end
```

schedule

Specifies the job stream name. With the exception of comments, this must be the first keyword in a job stream, and must be followed by the **on** keyword.

Syntax

schedule [*[folder/]workstation***#**][*folder/*]*jstreamname*

[**timezone**|**tz** *tzname*]

Arguments

[folder/]workstation

Specifies the name of the workstation on which the job stream is launched. The default is the workstation on which **composer** runs to add the job stream.

[*folder/*]*jstreamname*

Specifies the name of the job stream and optionally, the folder that contains the job stream. The name must start with a letter, and can contain alphanumeric characters, dashes, and underscores. It can contain up to 16 characters.

timezone|**tz** ***tzname***

Specifies the time zone to be used when managing for the job stream. This setting is ignored if the global option *enTimeZone* is set to **no** on the master domain manager. For information on time zone settings, refer to Managing time zones on page 1064.

Comments

In a job stream definition you can set a time zone for the entire job stream by using the **timezone** keyword in the validity interval or when specifying time restrictions using **at**, **until**, or **deadline**.

You can set also a time zone for a job contained in a job stream by setting keywords **at**, **until**, or **deadline** for that job.

Regardless of whether you are defining a job or a job stream, if you use a time zone in a time restriction, for example **at** then you must use the same time zone when specifying the other time restrictions, such as **deadline** and **until**.

In a job stream definition you can set a time zone for the entire job stream and for the jobs it contains. These time zones can differ from each other, in which case the time zone set for the job is converted into the time zone set for the job stream.

To manage all possible time zone settings, the time zone conversion that is performed when processing jobs and job streams across the HCL Workload Automation network is made respecting the following criteria:

1. If a time zone is not set for a job within a job stream, then that job inherits the time zone set on the workstation where the job is planned to run.
2. If a time zone is not set for a job stream, then the time zone set is the one set on the workstation where the job stream is planned to run.
3. If none of the mentioned time zones is set, then the time zone used is the one set on the master domain manager.

Example

Examples

This is the definition of a time zone of the job `sked8` contained in the folder `TEST`, on workstation `sfran` on which is set the time zone `America/New_York`. The time zone set for `job2` to run on workstation `LA` is defined as `America/Los_Angeles`.

```
schedule sfran#TEST/sked8
tz America/New_York
on fr at 1000 + 1 day:
job1
LA#job2 at 1400 tz America/Los_Angeles
end
```

statisticstype custom

Flags a job so that its estimated duration is calculated by an AI-based set of algorithms instead of the default logman process (automatic). For more information, see The logman command on page 146.

- Job properties
- Orchestration Monitor
- Critical path
- What-if Analysis

This tool can be particularly useful to see beyond the accepted impacts of already known periodicities, and understand what apparently hidden events affect the duration of a job. It can be effective to measure and forecast the durations of jobs along a critical path that occasionally does not meet its deadline.

See *AI Data Advisor (AIDA) User's Guide* for more information about predicting job duration using AI-based algorithms in addition to using this keyword.

Syntax

statisticstype custom

If you omit this keyword, the estimated duration - and its associated confidence interval - for the job is by default calculated by the process run by the logman command and made available on the same media. For more information, see The logman command on page 146.

startcond

Use this keyword to build into the job stream a mechanism which checks for specific events and conditions and releases the job stream when the specified events or conditions take place.

You can check for one of the following event types:

filecreated

You can check whether one or more files have been created on the specified workstation.

filemodified

You can check for modifications in one or more files present on the specified workstation.

job

You can define a job definition whose output condition is to be monitored.

For conditions based on files being created or modified, when you save the job stream, a monitoring job is automatically created to monitor the specified condition. This job becomes the first job in the job stream and remains in EXEC status until the condition it is monitoring is met or the job's deadline expires.

Note: If you do not specify a deadline, by default the value is defined using the **startConditionDeadlineOffset** optman option.

This job type is identified by the **+AUTOGEN+** label in the command line and by an icon with the **A** character in the Dynamic Workload Console.

For conditions based on the result of a specified job, when you save the job stream, the job becomes the first job in the job stream and restarts until the condition is satisfied, or the job's deadline expires.

Note: If you do not specify a deadline, by default the value is defined using the **startConditionDeadlineOffset** optman option.

This applies also if the job completes in Success status. This is the monitoring job. When you specify this condition type, HCL Workload Automation automatically defines a success output condition on the monitoring job. As a result, the monitoring job completes successfully when any of its output conditions is satisfied, including the condition on the monitoring job itself. You can apply this logic to the job stream or to specific jobs in the job stream. For more information about output conditions, see the section about Applying conditional branching logic in *User's Guide and Reference*.

If you do not specify a name for the monitoring job, by default the **file_StartCond** name is used. This value is defined using the **fileStartConditionJobName** optman option. For more information about the **fileStartConditionJobName** option, see the description of global options in Administration Guide.

By default, HCL Workload Automation keeps monitoring the condition also after it is first met. This is accomplished by automatically creating a new Job Stream Submission job and adding it to the job stream as a successor of the monitoring job. This job type is identified by the **+AUTOGEN+** label in the command line and by an icon with the **A** character in the Dynamic Workload Console. To have HCL Workload Automation stop when the condition is first met, select the **Start once** check box in the Workload Designer, or omit the **rerun** keyword in the **composer** command line.

The Job Stream Submission job is defined on a specific pool workstation named MASTERAGENTS. This pool workstation contains the dynamic agent installed on the master domain manager and on the backup domain manager, if present. The dynamic agent installed on the master domain manager and backup domain manager (if present) are automatically added at installation time to this pool workstation. If you delete the MASTERAGENTS pool workstation and then recreate it, you must stop and restart the dynamic agent to add it back to the MASTERAGENTS pool workstation. See the topic about automatically registering agents to pools in the *Planning and Installation Guide*.

Note: The default name for the pool workstation, MASTERAGENTS, can be modified using the optman global option resubmitJobName. See the detailed description of the global options in the *Administration Guide* for details about this option.

The Job Stream Submission job creates a new instance of the job stream in which the start condition is defined. By default, the new job stream instance starts also if the previous instance is still running and the two instances run concurrently. To change this behavior, in the Workload Designer, switch to the **Scheduling options** tab and select **Queue the new instance** in the **Actions** section. From the **composer** command line, use the **onoverlap** keyword. For more information, see . The newly-generated instance is identical to the previous one it and is set to repeat the condition check, therefore a series of new instances is created until the job stream's deadline expires.

The name of the Job Stream Submission job is defined using the value set for the **resubmitJobname** optman option. By default, the value assigned to this option is **restart_StartCond**. For more information about the **resubmitJobname** option, see the description of global options in Administration Guide.

Syntax

startcond filecreated | filemodified | job

startcond filecreated *[folder/]workstation_name#file_name* **user** *username* **interval** *seconds* [**alias** *startcond_jobname* **rerun batch outfile** *outputfilename* **params** "*filemonitor additional parameters*"]

startcond filemodified *[folder/]workstation_name#file_name* **user** *username* **interval** *seconds* [**alias** *startcond_jobname* **rerun batch outfile** *outputfilename* **params** "*filemonitor additional parameters*"]

startcond job *[folder/]workstation_name#*[*folder/*]*job_name* **outcond** *joboutputcondition* **interval** *seconds* [**alias** *startcond_jobname* **rerun]**

Arguments

filecreated *[folder/]workstation_name#file_name*

Check whether the specified file or files are created, where:

[folder/]workstation_name#file_name

Specifies the workstation name and the fully qualified path to file or files to be monitored.

filemodified *[folder/]workstation_name#file_name*

Check whether the specified file or files are modified, where:

[folder/]workstation_name#file_name

Specifies the workstation name and the fully qualified path to file or files to be monitored.

job *[folder/]workstation_name#*[*folder/*]*job_name*

Check whether the specified job definition has completed successfully meeting the condition specified with the **outcond** keyword.

user *username*

The login information for accessing the workstation where the file or files to be monitored are located. Applicable only to **filecreated** and **filemodified** keywords.

interval *seconds*

How often HCL Workload Automation checks whether the condition is met, expressed in seconds. For the **job** start condition only, the value will be approximated to 60 seconds, if lower than 60 seconds. If the value is higher than 60 seconds and not divisible by 60, it will be approximated to the nearest value which is also divisible by 60.

alias *startcond_jobname*

The name of the job which is automatically added to the plan to run the monitoring task.

For conditions of type filecreated or filemodified

If you do not specify any names, the **file_StartCond** name is used by default. The default name is retrieved from the **fileStartConditionJobName** global option. For more information, see the section about global options in the Administration Guide

For conditions of type job,

If you do not specify any value, the name of the job definition is used.

rerun

Have HCL Workload Automation automatically create a Job Stream Submission job, which is added as a successor of the monitoring job. The Job Stream Submission job submits a new instance of the job stream in which the start condition is defined.

batch

When the process returns multiple files at the same time, a single job stream instance is used to process them in batch mode. If do not specify this parameter, a job stream instance is launched for each file retrieved. Applicable only to **filecreated** and **filemodified** keywords.

outfile *outputfilename*

The names of the retrieved file or files are returned as a variable. You can optionally specify an output file where to store the file names. Ensure that the directory where the output file is to be created is already existing. Applicable only to **filecreated** and **filemodified** keywords.

params "*filemonitor additional parameters*"

Optionally specify **filemonitor** additional parameters. Applicable only to **filecreated** and **filemodified** keywords. For more information, see Filemonitor on page 941.

outcond

The output condition which, when met, releases the remaining part of the job stream. Applicable only to the **job** keyword. You can specify this keyword both at the job stream and job level. When you save the job stream, the job restarts until the condition is met or the job's deadline expires.

Comments

Ensure that all the components in the HCL Workload Automation environment are at version 9.4, Fix Pack 1, or later.

The following workstation types are not supported if you specify a start condition based on files being created or modified:

- Extended agent
- Workload broker
- Remote engine

Example

Examples

The following example illustrates a job stream which starts only when the `Reports.txt` file is created on workstation `S_MDM` in the `/Reports` path. The `FileMngr` user is used to connect to the specified workstation and the check on the condition is performed every 100 seconds. The monitoring job is named `ReportCheck` and the retrieved information is stored in the `/logs/ReportsOutput.txt` file. Using the **params** keyword, two **filemonitor** parameters (**-recursive** **-maxEventsThreshold**) have been inserted to specify that the check is performed also on sub folders and all events must be returned.

```
SCHEDULE S_MDM#JS1
VARTABLE MAIN_TABLE
ON RUNCYCLE RC2 08/04/2024
( AT 0800 +1 DAYS )
STARTCOND FILECREATED S_MDM#/Reports/Report.txt USER FileMngr INTERVAL 100
( ALIAS ReportCheck PARAMS "-recursive -maxEventsThreshold all"
OUTFILE /logs/ReportsOutput.txt )
LIMIT 5
OPENS S_MDM#"/my file . txt" (-f %p)
:
S_MDM#NATIVE
NOP
FOLLOWS DYNAMIC

S_AGT#DYNAMIC

S_AGT#SLEEP3
FOLLOWS NATIVE
FOLLOWS S_MDM#JS1_EXT.SLEEP3

S_AGT#JOB_MGMT
FOLLOWS SLEEP3
END
```

servicename

Specifies the service name for a job stream which is usable by Self-Service Catalog. When creating and editing SSC-ready job streams, it is recommended you use the Dynamic Workload Console.

Syntax

servicename

Comments

Use this keyword to specify the name of the job stream in Self-Service Catalog. This keyword is required if you define the **isservice** keyword. For more information, see isservice on page 351. The maximum supported length is 16 characters.

Example

```
SCHEDULE BW-XFR-LNX53#JS2
DESCRIPTION "Added by composer."
ISSERVICE
SERVICENAME TEST2
SERVICEDESCRIPTION “This is my Service”
SERVICETAGS "|tag2|tag3|"
SERVICEVARIABLES "{var: \"value2\"}"
ON RUNCYCLE RC1 "FREQ=DAILY;"
:

BW-XFR-LNX53#JS2

END
```

servicedescription

Specifies the service description for a job stream which is usable by Self-Service Catalog. When creating and editing SSC-ready job streams, it is recommended you use the Dynamic Workload Console.

Syntax

servicedescription

Comments

Use this keyword to specify the description of the job stream in Self-Service Catalog. The maximum supported length is 16 characters.

Example

```
SCHEDULE BW-XFR-LNX53#JS2
DESCRIPTION "Added by composer."
ISSERVICE
SERVICENAME TEST2
SERVICEDESCRIPTION “This is my Service”
SERVICETAGS "|tag2|tag3|"
SERVICEVARIABLES "{var: \"value2\"}"
ON RUNCYCLE RC1 "FREQ=DAILY;"
:

BW-XFR-LNX53#JS2

END
```

servicetags

Specifies the service tags for a job stream which is usable by Self-Service Catalog. You can use service tags to identify and categorize the job stream. When creating and editing SSC-ready job streams, it is recommended you use the Dynamic Workload Console.

Syntax

servicetags

Comments

You can define a maximum of 10 tags. The maximum supported length is 16 characters. Separate the tags using one of the following:

- tab
- space
- comma
- pipe
- semicolon

When you save the job stream, separators are converted into pipes.

Example

```
SCHEDULE BW-XFR-LNX53#JS2
DESCRIPTION "Added by composer."
ISSERVICE
SERVICENAME TEST2
SERVICEDESCRIPTION “This is my Service”
SERVICETAGS "|tag2|tag3|"
SERVICEVARIABLES "{var: \"value2\"}"
ON RUNCYCLE RC1 "FREQ=DAILY;"
:

BW-XFR-LNX53#JS2

END
```

servicevariables

Specifies the service variables for a job stream which is usable by Self-Service Catalog. You can use service variables to define variables in your job stream. When creating and editing SSC-ready job streams, it is recommended you use the Dynamic Workload Console.

Syntax

servicevariables

Comments

Variables are defined as CLOB in Json format. The maximum supported length is 16 characters.

Example

```
SCHEDULE BW-XFR-LNX53#JS2
DESCRIPTION "Added by composer."
ISSERVICE
SERVICENAME TEST2
SERVICEDESCRIPTION “This is my Service”
SERVICETAGS "|tag2|tag3|"
SERVICEVARIABLES "{var: \"value2\"}"
ON RUNCYCLE RC1 "FREQ=DAILY;"
:

BW-XFR-LNX53#JS2

END
```

timezone

Specifies-at job stream level-the time zone used to calculate the time when the job stream must start processing.

Syntax

timezone|**tz** *tzname*

Arguments

tzname

Specifies the name of the time zone. See Managing time zones on page 1064 for time zone names.

Comments

The time zone specified at job stream level applies to the time definitions for the run cycles and the time restrictions (defined by the `at`, `deadline`, `schedtime`, and `until` keywords).

If you specify a time zone for the job stream and one for a time restriction keyword, they must be the same.

If you specify no time zone, either at job stream and time restriction levels, the time zone specified for the workstation is used.

until

Depending on the object definition the until keyword belongs to, specifies the latest time a job stream must be completed or the latest time a job can be launched. This keyword is mutually exclusive with the **jsuntil** keyword. For more information about the **jsuntil** keyword, see jsuntil on page 354.

Syntax

[**until** *time* [**timezone**|**tz** *tzname*][**+***n* **day**[**s**]][**onuntil** *action*]]

Arguments

time

Specifies the time of day. The possible values are **0000** through **2359**.

timezone *tzname*

Specifies the time zone to be used when computing the time. See Managing time zones on page 1064 for time zone names. The default is the time zone of the workstation on which the job or job stream is launched.

Note: If an **until** time and an **at** or **deadline** time are specified, the time zones must be the same.

day *n*

Specifies an offset, in days, from the scheduled date and time.

onuntil *action*

Depending on the object definition the until keyword belongs to, specifies:

- The action to be taken on a job whose until time has expired but the job has not yet started.
- The action to be taken on a job stream whose until time has expired but the job stream is not yet completed.

The following are the possible values of the *action* parameter:

suppr

The job or job stream and any dependent job or job stream do not run. This is the default behavior.

Once the until time expired on a job stream, the status for the job stream is calculated following the usual rules; suppressed jobs are not considered in the calculation. In case the job stream contains at least one **every** job its status is HOLD.

When the until time expires for a job, the job moves to HOLD status or keeps any previous status which is a final status.

If the until time is passed together with the **onuntil suppr** and the **carryforward** options, the job stream is *carry forwarded* by JnextPlan only if the until date is equal to the date when JnextPlan runs. If the until and the JnextPlan run dates are not the same, the job stream is not *carry forwarded*.

cont

The job or job stream runs when all necessary conditions are met and a notification message is written to the log when the until time elapses.

If the until time is passed together with the **onuntil cont** and the **carryforward** options, the job stream is always *carry forwarded* by JnextPlan.

canc

A job or job stream is cancelled when the **until** time specified expires. When using ***onuntil canc*** on jobs, the cancel operation on the job is issued by the FTA on which the job runs. Any job or job stream that was dependent on the completion of a job or job stream that was cancelled, runs because the dependency no longer exists.

If the until time is passed together with the **onuntil canc** and the **carryforward** options, the job stream is not *carry forwarded* by JnextPlan because it is already canceled.

Note: When using ***onuntil canc*** at job stream level, define as owner of the job stream the workstation highest in the hierarchy of the scheduling environment, among all workstations that own jobs contained in the job stream.

Note:

- Both the keyword **until** and **deadline** can be used in the same definition but they must be expressed using the same time zone setting.
- The **untilDays** global option can only apply to a jobstream when the jobstream is added to the plan. If the **untilDays** was set to 0 when a jobstream was added to the plan changing the **untilDays** parameter after the jobstream is in the plan can have no impact on that jobstream.
- If an **until** time (latest start time) is not specified for a job stream, then a default **until** time can be set using the **untilDays** global option. The **until** time is calculated adding the value of the **untilDays** global option, expressed in number of days, to the scheduled time for the job stream. If the *enCarryForward* option is set to **all**, and the number of days specified for *untilDays* is reached, then any job stream instances in the plan that ended in error are automatically removed from the plan and are not added to the new production plan. If the default value for **untilDays** is maintained (0), then no default time is set for the **until** time.

Example

Examples

The following example prevents `sked1` from launching after 5:00 p.m. on Tuesdays:

```
schedule sked1  on tu  until 1700 :
```

The following example launches `sked1` at 5:00 p.m., when its **until** time is reached:

```
schedule sked1 until 1700 onuntil cont
```

The following example launches `job1` between 1:00 p.m. and 5:00 p.m. on weekdays:

```
schedule sked2  on weekdays :
  job1  at 1300  until 1700
end
```

The following example launches `joba` every 15 minutes between 10:30 p.m. and 11:30 p.m. on Mondays:

```
schedule sked3 on mo :
  joba  at 2230  every 0015  until 2330
end
```

The following example launches job stream `sked4` on Sundays between 8:00 a.m. and 1:00 p.m. The jobs are to be launched within this interval:

```
schedule sked4  on fr  at 0800 + 2 days
  until 1300 + 2 days
:
  job1
  job2  at 0900  <<launched on sunday>>
  job3  follows job2  at 1200  <<launched on sunday>>
end
```

The following example launches job stream `sked8` on weekdays at 4:00 p.m. and should complete running by 5 p.m. If the job stream is not completed by 5 p.m., it is considered a late job stream. The jobs are to be launched as follows: `job1` runs at 4

p.m., or at the latest, 4:20 p.m., at which time, if `job1` has not yet started, a notification message is written to the log and it starts running. `job2` runs at 4:30 p.m. or at the latest 4:50 p.m., at which time, if `job2` has not yet started, it is cancelled.

```
schedule sked8 on weekdays at 1600 deadline 1700 :
   job1  at 1600 until 1620 onuntil cont
   job2 at 1630 until 1650 onuntil canc
end
```

The following example launches job stream `sked01`. When the **until** event occurs, the job stream `sked02` is run because the job stream `sked01` is placed in SUCC state. The job stream `sked03`, instead, is not run because it has a punctual time dependency on job `job01` and this dependency has not been released.

```
SCHEDULE sked01 on everyday:
job01 until   2035 onuntil suppr
end
```

```
SCHEDULE sked02 on everyday follows sked01.@
:
job02
end
```

```
SCHEDULE sked03 on everyday follows  sked01.job01
:
job03
END
```

validfrom/validto

You can set a validity time for a job stream, which is a time frame within which the job stream is included in the preproduction plan. The validity time is set using the **validfrom** key in the job stream definition.

Syntax

validfrom *date*

Arguments

validfrom *date*

Defines from which date the job stream is active, that is it must be included in a production plan if the production plan duration includes that date. It also defines the first date the instances in the preproduction plan are included in the current plan.

Comments

Different *versions* of the same job stream can be defined by creating different job streams with the same name and workstation, but having different validity intervals. The concept of versions of the same job stream sharing the same *jobstreamname* and the same *workstationname* are key when managing dependency on that job stream. In fact when you define an external follows dependencies on a job stream you identify the predecessor job stream using its *jobstreamname* and *workstationname*. The job stream identified as the dependency is the one whose validity interval is during the period the dependency is active.

If you change the *jobstreamname* or the *workstationname* in one version of the job stream, the change is propagated in all its versions.

If you lock a version of the job stream, all versions of that job stream are locked.

If you change the name of a job defined in a job stream version then the new job name is propagated in all versions of the job stream. This means that, if you modify something, other than the *jobstreamname* or the *workstationname*, the internal and external job stream associations remain consistent.

When defining a job stream version, you are only asked to enter the **validfrom** date, and the **validto** date is automatically set to the value of the **validfrom** date of the following version. The **validto** date is shown when issuing **list** and **display** command when *MAESTROCOLUMNS* is set to 120. Different versions of the same job stream continue to share the name and workstation fields after their creation. If you modify the name of a version or change the workstation on which it was defined, the change is applied to all versions of that job stream.



Note: If the keywords used in the job stream definition are **validfrom** and **validto**, the corresponding filtering keywords used when issuing commands against object definitions stored in the database are **validfrom** and **validto**. For more information refer to Managing objects in the database - composer on page 434.

The date specified as **validto** value is not included in the run cycle, therefore the job stream is not active on this date.

Variable table

Using variable tables you assign different values to the same variable and therefore reuse the same variable in job definitions and when defining prompts and file dependencies.

When creating a scheduling object, you can define it in a folder. If no folder path is specified, then the object definition is created in the current folder. By default, the current folder is the root (\) folder, but you can customize it to a different folder path. You can also use the composer rename command to move and rename objects in batch mode that use a naming convention to a different folder using part of the object name to assign a name to the object.

Syntax

vartable *[folder/]table_name*

Arguments

vartable*[folder/]table_name*

The name of the variable table. The name can contain up to 80 alphanumeric characters including dashes (-) and underscores (_), and must start with a letter.

For more information, see Customizing your workload using variable tables on page 164.

Event rule definition

A scheduling event rule defines a set of actions that are to run upon the occurrence of specific event conditions. The definition of an event rule correlates events and triggers actions.

An event rule definition is identified by a rule name and by a set of attributes that specify if the rule is in draft state or not, the time interval it is active, the time frame of its validity, and other information required to decide when actions are triggered. It includes information related to the specific events `(eventCondition)` that the rule must detect and the specific actions it is to trigger upon their detection or timeout `(action)`. Complex rules may include multiple events and multiple actions.

When creating a scheduling object, you can define it in a folder. If no folder path is specified, then the object definition is created in the current folder. By default, the current folder is the root (\) folder, but you can customize it to a different folder path. You can also use the composer rename command to move and rename objects in batch mode that use a naming convention to a different folder using part of the object name to assign a name to the object.

Folders are also supported for the scheduling objects contained in event rules, such as workstations, jobs, job streams.

For an overview of scheduling event rules, see Running event-driven workload automation on page 174.

Syntax

You define event rules directly in XML language with the use of any XML editor. You can configure an environment variable on your computer to automatically open an XML editor of your choice to work with event rule definitions. See The composer editor on page 436 for details. The XML describing the event rule must match the rule language schemas defined in `EventRules.xsd` and in `FilteringPredicate.xsd`.

The rule language schemas defined in `eventRules.xsd` and in `FilteringPredicate.xsd` are used to validate your rule definitions and, depending upon the XML editor you have, to provide syntactic help. The files are located in the `schemas` subdirectory of the HCL Workload Automation installation directory.

Table 50. Explanation of the notation defining the number of occurrences for a language element. (continued)

Notation	Meaning
(0, n)	0 indicates that the language element is optional. n indicates that if coded, multiple occurrences are allowed.
(1, 1)	The first 1 indicates that the language element is required. The second 1 indicates that only 1 occurrence is allowed.
(1, 2)	1 indicates that the language element is required. 2 indicates that 2 occurrences are required.
(1, n)	1 indicates that the language element is required. n indicates that multiple occurrences are allowed.

- eventRule name=" " ruleType=" " isDraft=" " (1, 1)
  - description (0, 1)
  - timeZone (0, 1)
  - validity from=" " to=" " (0, 1)
  - activeTime start=" " end=" " (0, 1)
  - timeInterval amount=" " unit=" " (0, 1)
  - eventCondition eventProvider=" " eventType=" " (1, n)
    - scope (0, 1)
    - filteringPredicate (0, 1)
      - attributeFilter name=" " operator="eq" (0, n)
        - value (1, n)
      - attributeFilter name=" " operator="ne" (0, n)
        - value (1, n)
      - attributeFilter name=" " operator="le" (0, n)
        - value (1, 1)
      - attributeFilter name=" " operator="ge" (0, n)
        - value (1, 1)
      - attributeFilter name=" " operator="range" (0, 1)
        - value (1, 2)
  - correlationAttributes (0, 1)
    - attribute name=" " (1, n)
  - action actionProvider=" " actionType=" " responseType=" " (0, n)
    - description (0, 1)
    - scope (0, 1)
    - parameter name=" "(1, n)
    - value (1, 1)

Event rule definitions are grouped into event rule sets.

- eventRuleSet (1, 1)
  - eventRule (1, n)

Use the `eventRuleSet` language element also if you have to enclose a single rule definition.

Note: None of the comments that you write in the XML, in the form , are saved in the database. The next time that you open a rule definition, the comments that you wrote the first time are not there. Instead, use the `description` attribute to write any additional information.

Arguments

The keywords that describe an event rule are the following XML tags:

eventRule

The scheduling object that encloses the definition of multiple event conditions and multiple rule actions in addition to a set of attributes that define when the rule is activated. An `eventRule` element typically includes:

- A number of required and optional rule attributes
- One or more event conditions
- One or more rule actions, although rules with no actions are also allowed

The value of the name attribute is expressed as *[folder/]name*. The folder can be up to 841 characters in length. For more information about the folder syntax, see Folder definition on page 287. The name of the event rule can be up to 40 alphanumeric characters. The name of the event rule must start with a letter, and cannot contain blanks. Underscore `(_)` and dash `(-)` characters are allowed.

The rule type is based on the number of events - and on their correlation - that the rule is defined to detect. It can be one of the following:

filter

The rule is activated upon the detection of a single specific event.

sequence

The rule is activated when an ordered sequence of events arrives within a specific time interval.

set

The rule is activated when an unordered sequence of events arrives within a specific time interval.

Rules of type `set` and `sequence` can also be activated on **timeout**, when one or more events arrive but the complete sequence does not arrive within the specified time window.

isDraft

Specifies if the rule definition is currently enabled. Values can be **yes** or **no**. The default is **no**.

- Optional attributes:

description

A description of the rule. It can be of up to 120 characters.

Remember to write any XML special characters you might use in the XML special notation, such as:

- `&` for &
- `>` for >
- `<` for <
- `"` for "

and so on.

timeZone

Specifies a different time zone for the execution of the rule. The default time zone is the time zone defined on the workstation where the event processing server resides.

The application of daylight saving time (DST) has an impact on the `activeTime` interval (described next) of event rules:

- On the day that DST is turned on (the clock is adjusted forward one hour) the rule operation times that fall within the hour absorbed by the application of DST are moved forward by one hour. For example, 2:10 becomes 3:10.
- On the day that DST is turned off (the clock is adjusted backward one hour) the rule operation times that fall within the hour duplicated by the application of DST are regarded without DST.

validity

Specifies the rule validity period in terms of:

from *yyyy-mm-dd*

The validity period starts at midnight (of the rule time zone) of the specified day.

to *yyyy-mm-dd*

The validity period ends at midnight (of the rule time zone) of the specified day.

activeTime

Specifies the rule activity time window within each day of validity in terms of:

start *hh:mm:ss*

The beginning of the time window when the rule is active in hours, minutes, and seconds.

end *hh:mm:ss*

The end of the time window when the rule is active in hours, minutes, and seconds. If the time is earlier than in **start** *hh:mm:ss*, it refers to the next day.

timeInterval

Applies to rules that include timeout actions. Specifies the time interval within which all the events specified in the rule must have been received before a corrective timeout action is started. The time interval starts from the moment the first event specified in the rule is detected. Specify the time interval in terms of:

amount

The length of the time interval in time units.

unit

The time unit in one of the following:

- hours
- seconds
- milliseconds

This attribute is mandatory when there are timeout actions in the event rule definition.

eventCondition

The event condition is made up by the following attributes:

- Required attributes:

  eventProvider

Identifies the event monitoring provider that can capture a type of event. The event providers supplied at installation time are:

TWSObjectsMonitor

Monitors the status of HCL Workload Automation plan objects. This event provider runs on every HCL Workload Automation agent and sends the events to the event processing server.

TWSApplicationMonitor

Monitors HCL Workload Automation processes, file system, and message box. `TWSApplicationMonitor` events are supported on fault-tolerant agents only.

For a detailed example about how to set up an event rule that monitors the used disk space, see the section about Maintaining the file system in the *Administration Guide*.

FileMonitor

Monitors events affecting files.

DatasetMonitor

Monitors events affecting Data sets.

eventType

Specifies the type of event that is to be monitored. Every event can be referred to an event provider. The following tables list the event types by event provider.

To see the properties of each event type, go to Event-driven workload automation event and action definitions on page 1381.

Table 51: TWSObjectsMonitor events. on page 401 lists the `TWSObjectsMonitor` events.

Table 51. `TWSObjectsMonitor` events.

Event type	When the event is sent
JobStatusChanged	The status of a job changes
JobUntil	The latest start time of a job has elapsed
JobSubmit	A job is submitted
JobCancel	A job is canceled
JobRestart	A job is restarted
JobLate	A job becomes late
JobPromoted	A job in a critical network approaches the critical start time and has not yet started
JobRiskLevelChanged	◦ A new critical job is added to the plan with the risk level set to either high or potential ◦ The risk level for a job is set to high risk or potential risk and the risk level changes ◦ A critical job with the risk level set to either high or potential is removed from the plan An event is not sent if the critical job is in the job stream named `JOBS`.
JobExceededMaximumDuration	A job exceeds the maximum duration time established for the job

Event type	When the event is sent
JobDidnotReachMinimumDuration	A job does not run long enough to reach the minimum duration time established for the job
JobStreamStatusChanged	The status of a job stream changes
JobStreamCompleted	A job stream has completed
JobStreamUntil	The latest start time of a job stream has elapsed
JobStreamSubmit	A job stream is submitted
JobStreamCancel	A job stream is canceled
JobStreamLate	A job stream becomes late
WorkstationStatusChanged	A workstation is started or stopped
ApplicationServerStatusChanged	WebSphere Application Server Liberty has stopped or is restarting
ChildWorkstationLinkChanged	The workstation defined in the event rule links or unlinks from its parent workstation (the parent workstation sends the event)
ParentWorkstationLinkChanged	The parent workstation links or unlinks from the workstation defined in the event rule (the child workstation sends the event)
PromptStatusChanged	A prompt is asked or replied
ProductAlert	The Symphony file in the workstation specified in the event rule contains a corrupt record

Notice:

Any change performed on a workstation referenced in a rule is not reported in the rule. For example if you modify, update, or delete a workstation that is referenced in a rule, the rule ignores the change and continues to consider the workstation as it was when it was included in the rule.

A rule with event type ParentWorkstationLinkChanged is not applicable when the Filters Workstation is set to agent, pool, dynamic pool, or remote engine and the ParentWorkstation attribute is set to broker. To monitor a link status change between the workload broker server and a workstation managed by the workload broker server, define a rule with event type equal to ChildWorkstationLinkChanged.

A rule with event type equal to ChildWorkstationLinkChanged works only when the broker workstation is linked, unlinked, stopped, or started. If the change in the

link status is due to a stop or start operation on the agent workstation with the StartupLwa and ShutDownLwa commands, no action is started. To monitor stop or start operations on agent workstations, define a rule with event type equal to WorkstationStatusChanged.

Table 52: TWSApplicationMonitor events. on page 403 lists the `TWSApplicationMonitor` events.

Table 52. `TWSApplicationMonitor` events.

Event type	When the event is sent
MessageQueuesFilling	A specified mailbox exceeds the percentage full value.
TivoliWorkloadSchedulerFileSystemFilling	The file system where the HCL Workload Automation instance is installed exceeds the disk usage percentage full value.
TivoliWorkloadSchedulerProcessNotRunning	A specified process is not running.

Table 53: FileMonitor events. on page 403 lists the `FileMonitor` events.

Table 53. `FileMonitor` events.

Event type	When the event is sent
FileCreated	A file is created
FileDeleted	A file is deleted
ModificationCompleted	A file is modified (the event is sent only if two consecutive monitoring cycles have passed since the file was created or modified with no additional changes being detected)
LoggedMessageWritten	A specific string is found in a file (this event can be used to monitor application or system logs)

Table 54: DatasetMonitor events. on page 403 lists the `DatasetMonitor` events.

Table 54. `DatasetMonitor` events.

Event type	When the event is sent
ModificationCompleted	A data set is modified (the event is sent only if two consecutive monitoring cycles have passed since the file was created or modified with no additional changes being detected)

Event type	When the event is sent
ReadCompleted	A data set is read.

- Optional attributes:

  scope

  One or more qualifying attributes that describe the event. It can be up to 120 characters. The scope is automatically generated from what is defined in the XML. It cannot be specified by users.

  filteringPredicate

  The filtering predicate sets the event conditions that are to be monitored for each event type. It is made up by:

  attributeFilter

  The attribute filter is a particular attribute of the event type that is to be monitored:

  - Is defined by the following elements:

    name

    The attribute, or property name, of the event type that is to be monitored. Refer to Event providers and definitions on page 1381 for a list of property names for every event type.

    operator

    Can be:

    - `eq` (equal to)
    - `ne` (not equal to)
    - `ge` (equal or greater than)
    - `le` (equal or less than)
    - `range` (range)

- Includes one or more:

  value

  The value on which the operator must be matched.

Note: Every event type has a number of mandatory attributes, or property names. Not all the mandatory property names have default values. All the mandatory property names without a default value must have a filtering predicate defined.

correlationAttributes

The correlation attributes provide a way to direct the rule to create a separate rule copy for each group of events that share common characteristics. Typically, each active rule has one rule copy that runs in the event processing server. However, sometimes the same rule is needed for different groups of events, which are often related to different groups of resources. Using one or more correlation attributes is a method for directing a rule to create a separate rule copy for each group of events with common characteristics. Use with `set` and `sequence` rule types.

You can specify one or more attributes. Each is defined by:

attribute name=" "

The object attribute that you are correlating.

action

The action that is to be triggered if the event is detected. Event rule definitions with events but no actions are syntactically accepted, although they may have no practical significance. You may save such rules as draft and add actions later before they are deployed.

- Is defined by the following required attributes:

  actionProvider

  The name of the action provider that can implement one or more configurable actions. The action providers available at installation time are:

  GenericAction

  Runs non-HCL Workload Automation commands. The commands are run on the same computer where the event processor runs.

MailSender

Connects to an SMTP server to send an email.

MessageLogger

Logs the occurrence of a situation in an internal auditing database.

TECEventForwarder

Forwards the event to an external Tivoli Enterprise Console (TEC) server, or any other application capable of listening to events in TEC format.

TWSAction

Runs one of the following `conman` commands:

- submit job (`sbj`)
- submit job stream (`sbs`)
- submit command (`sbd`)
- reply to prompt (`reply`)

TWSForZosAction

Adds an application occurrence (job stream) to the current plan on HCL Workload Automation for Z. This provider is for use in HCL Workload Automation end-to-end scheduling configurations.

The application description of the occurrence to be added must exist in the AD database of HCL Workload Automation for Z.

actionType

Specifies the type of action that is to be triggered when a specified event is detected. Every action can be referred to an action provider. The following table lists the action types by action provider.

To see the properties of each action type, go to Event-driven workload automation event and action definitions on page 1381.

Table 55. Action types by action provider.

Action provider	Action types
GenericAction	RunCommand
MailSender	SendMail
MessageLogger	PostOperatorMessage
TECEventForwarder	TECFWD

Action provider	Action types
TWSAction	reply (ReplyPrompt) sbd (SubmitAdHocJob) sbj (SubmitJob) sbs (SubmitJobStream)
TWSForZosAction	AddJobStream

responseType

Specifies when the action is to run. Values can be:

onDetection

The action starts as soon as all the events defined in the rule have been detected. Applies to all rule types. See also Rule operation notes on page 192.

onTimeOut

The action starts after the time specified in `timeInterval on page 399` has expired but not all the events defined in the rule have been received. Applies to `set` and `sequence` rules only.

Note that timeout actions are not run if you do not specify a time interval. The scheduler will however let you save event rules where timeout actions have been defined without specifying a time interval, because you could set the time interval at a later time. Until then, only actions with the `onDetection` response type are processed.

Timeout actions for which a time interval was not defined are run only when the rules are deactivated. An event rule is deactivated in either of two cases:

- The `planman deploy -scratch` command is issued
- The rule is modified (it is then deactivated as soon as the `planman deploy` command is run)

In either case the rule is first deactivated and then reactivated. At this time all pending actions are executed.

- Includes the following optional attributes:

  description

  A description of the action. It can be of up to 120 characters.

  Remember to write any XML special characters you might use in the XML special notation, such as:

  - `&` for &
  - `>` for >
  - `<` for <
  - `"` for "

  and so on.

  scope

  One or more qualifying attributes that describe the action. It can be of up to 120 characters. The scope is automatically generated from what is defined in the XML. It cannot be specified by users.

- Includes a list of one or more parameters, or property names. All action types have at least one mandatory parameter. Every parameter is defined by:

  parameter name=" "

  See Action providers and definitions on page 1397 for a list of parameters, or property names, available for every action type.

  value

  See Action providers and definitions on page 1397 for a list of possible values or value types.

You can use variable substitution. This means that when you define action parameters, you can use the property names of the events that trigger the event rule to replace the value of the action property name. To do this, write the value for the action parameter you intend to substitute in either of these two forms:

- `${event.property}`

  Replaces the value as is. This is useful to pass the information to an action that works programmatically with that information, for example the `schedTime` of a job stream.

- `%{event.property}`

  Replaces the value formatted in human readable format. This is useful to pass the information to an action that forwards that information to a user, for example to format the `schedTime` of a job stream in the body of an email.

Where:

event

Is the name you set for the triggering `eventCondition`.

property

Is the `attributeFilter` name in the filtering predicate of the triggering event condition. The value taken by the attribute filter when the rule is triggered is replaced as a parameter value in the action definition before it is performed.

Note that you can use variable substitution also if no `attributeFilter` was specified for an attribute and also if the attribute is read-only.

For example, the task of an event rule is to detect when any of the jobs that have a name starting with `job15` end in error and, when that happens, submit that job again. The `eventCondition` section of the rule is coded as follows:

```
<eventCondition
      name=“event1”
      eventProvider=“TWSObjectsMonitor”
      eventType=“JobStatusChanged”>
  <filteringPredicate>
     <attributeFilter
           name=“JobName”
           operator=“eq”>
        <value>job15*<?value>
     </attributeFilter>
     <attributeFilter
           name=“Workstation”
           operator=“eq”>
        <value>*<?value>
     </attributeFilter>
     <attributeFilter
           name=“Status”
           operator=“eq”>
        <value>Error<?value>
     </attributeFilter>
  </filteringPredicate>
</eventCondition>
```

Wildcards (* for multiple characters or ? for single characters) are used to generalize the event condition that you want to apply to all the job instances whose name starts with `job15` and to their associated workstation. Variable substitution is used in the `action` section to submit again the specific job that ended in error on the same workstation. The `action` section is:

```
<action
      actionProvider=“TWSAction”
```

```
      actionType="sbj"
      responseType="onDetection">
  <description>Submit again the job that ended in error</description>
      <parameter name="JobDefinitionName">
        <value>${event1.JobName}<?value>
      </parameter>
      <parameter name="JobDefinitionWorkstationName">
        <value>${event1.Workstation}</value>
      </parameter>
</action>
```

Example

Examples

`JOB7` has a file dependency on `DAILYOPS.XLS`. As soon as the file is received, `JOB7` must start to process the file. The following rule controls that `JOB7` starts within one minute after the transfer of `DAILYOPS.XLS` is finished. If this does not happen, an email is sent to the evening operator. This is accomplished by defining two sequential event conditions that have to monitored:

1. The first event that triggers the rule is the creation of file `DAILYOPS.XLS` on the workstation to which it is to be transferred. As soon as this event is detected, a rule instance is created and a one minute interval count is begun to detect the next event condition.
2. The second event is the submission of `JOB7`. If this event fails to be detected within the specified time interval, the rule times out and the `SendMail` action is started.

```
<?xml version="1.0"?>
<eventRuleSet
      xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
      xmlns="http://www.ibm.com/xmlns/prod?tws/1.0/event-management/rules"
      xsi:schemaLocation="http://www.ibm.com/xmlns/prod?tws/1.0/event-management/rules
      EventRules.xsd">
   <eventRule
         name="myfolder/sample_rule"
         ruleType="sequence"
         isDraft="no">
      <description>An email is sent if job JOB7 does not start within
                  a minute after file DAILYOPS.XLS is created</description>
      <timeZone>America/Indianapolis</timeZone>
      <validity
            from="2023-01-01"
            to="2023-12-31" ?>
      <activeTime
            start="20:00:00"
            end="22:00:00" ?>
      <timeInterval
            amount="60"
            unit="seconds" ?>
      <eventCondition
            name="DAILYOPS_FTPed_event"
            eventProvider="FileMonitor"
            eventType="FileCreated">
         <filteringPredicate>
            <attributeFilter
                  name="FileName"
```

```
                operator="eq">
             <value>c:?dailybus?DAILYOPS.XLS<?value>
          </attributeFilter>
          <attributeFilter
                name="Workstation"
                operator="eq">
             <value>ACCREC03<?value>
          </attributeFilter>
          <attributeFilter
                name="SampleInterval"
                operator="eq">
             <value>300</value>
          </attributeFilter>
       </filteringPredicate>
    </eventCondition>
    <eventCondition
          name="job_JOB7_problem_event"
          eventProvider="TWSObjectsMonitor"
          eventType="JobSubmit">
      <filteringPredicate>
         <attributeFilter
               name="JobStreamWorkstation"
               operator="eq">
            <value>ACCREC03<?value>
         </attributeFilter>
         <attributeFilter
               name="Workstation"
               operator="eq">
            <value>ACCREC03<?value>
         </attributeFilter>
         <attributeFilter
               name="JobStreamName"
               operator="eq">
            <value>JSDAILY<?value>
         </attributeFilter>
         <attributeFilter
               name="JobName"
               operator="eq">
            <value>JOB7</value>
         </attributeFilter>
      </filteringPredicate>
    </eventCondition>
    <action
          actionProvider="MailSender"
          actionType="SendMail"
          responseType="onTimeOut">
       <description>Send email to evening operator stating job did not
                    start</description>
       <parameter name="To">
          <value>eveoper@bigcorp.com</value>
       </parameter>
       <parameter name="Subject">
          <value>Job JOB7 failed to start within scheduled time on
                 workstation ACCREC03.</value>
       </parameter>
    </action>
  </eventRule>
</eventRuleSet>
```

Note that the scope does not show the first time the rule is defined.

For more event rule examples, see Event rule examples on page 185 .

See also

From the Dynamic Workload Console you can perform the same task as described in:

the Dynamic Workload Console User's Guide, section about Creating an event rule.

Workload application definition

You can use workload applications to standardize a workload automation solution so that the solution can be reused in one or more HCL Workload Automation environments thereby automating business processes.

Workload applications are defined in an HCL Workload Automation environment, referred to as the source environment, using **composer** command line or the Workload Designer graphical user interface accessible from the Dynamic Workload Console. You create a new workload application template and then add one or more job streams to it. To reproduce the workload automation solution in another HCL Workload Automation environment, the workload application template is exported and then after some manual customizations, it can be imported in the target environment.

When creating a scheduling object, you can define it in a folder. If no folder path is specified, then the object definition is created in the current folder. By default, the current folder is the root (\) folder, but you can customize it to a different folder path. You can also use the composer rename command to move and rename objects in batch mode that use a naming convention to a different folder using part of the object name to assign a name to the object.

The export process produces a compressed file containing all the files and information required to make the workload run in a different HCL Workload Automation environment. The compressed file contains:

A definitions file

An XML file, *workload applicationtemplatename*_Definitions.UTF8.xml, that contains the definitions of all the exported objects. These definitions will be deployed in the target environment so as to populate the target database with the same objects existing in the source environment.

A mapping file

A mapping file, *workload applicationtemplatename*_Mapping.UTF8.properties, required to address those objects that are dependent on the topology of the environment and that cannot be reproduced without some manual customization. The target user will modify the file replacing the names of the objects in the source environment with the names that these objects have in the target environment.

A reference information file

A reference information file, *workload applicationtemplatename*_SourceEnv_reference.txt, containing the definitions of the workstations used in the workload application template and other information that can be useful to correctly map the source environment into the target environment so as to make the workload application run.

You can import the compressed package from the Dynamic Workload Console with an intuitive guided procedure, as described in the section about importing a workload application template in the *Dynamic Workload Console User's Guide*.

You can also manually import the compressed package into the target environment where the workload application will be deployed, thus creating all the required objects in the target environment. In the target environment, the `workload application name_Mapping.UTF8.properties` file must be edited manually, using a generic text editor, by specifying the names of the objects as they are defined in the target environment (for example, the names of the workstations on which the job streams run). The import operation must be performed in the target environment by using the command line. For more details, see *User's Guide and Reference* sections about workload applications and the wappman command.

When using **composer** command line, each workload application definition has the following format and arguments:

Syntax

wat *[folder/]wat_name*
[**description** "description"]
[**vendor** "vendor"]
jstreams
[*[folder/]workstation#[folder/]jobstream* [*[folder/]workstation#[folder/]jobstream*]...]
end

Arguments

***[folder/]*wat**

Mandatory field that contains the name of the workload application template. The maximum length is 80 characters. In product versions prior to HCL Workload Automation version 9.4, Fix Pack 1, this keyword was named **bapplication**.

description

Optional descriptive text to help workload application users understand the purpose and characteristics of the workload application. The maximum length is 120 characters.

vendor

Optional field that specifies the creator of the workload application template. It can be useful to let workload application users know who created and provided it. The maximum length is 120 characters.

[folder/]jstreams

The job streams that you want to add to the workload application template.

Example

Examples

To create two workload application templates, WAT_NAME1 and WAT_NAME2, run:

```
new wat
WAT WAT_NAME1
```

```
  DESCRIPTION "Description"
  VENDOR "Provider"
  JSTREAMS
      FTAfolder/FTA1#JS_1_1
      Agentfolder/AGENT1#JS_1_2
END

WAT WAT_NAME2
  DESCRIPTION "Description"
  VENDOR "Provider"
  JSTREAMS
      JS_2_1
      JS_2_2
END
```

Security object definition

You can use **composer** command line program to define security objects in the database.

Access control lists

Each access control list assigns security roles to users or groups, in a certain security domain.

Security domains

Each domain represents the set of scheduling objects that users or groups can manage.

Security roles

Each role represents a certain level of authorization that defines the set of actions that users or groups can perform on a set of object types.

Security access control list definition

In the role-based security model, an access control list assigns security roles to users or groups, in a certain security domain or on a specific folder or folder hierarchy. You can include multiple security access control list definitions in the same text file, along with security domain definitions and security role definitions.

Each security access control list definition has the following format and arguments:

Syntax

accesscontrollist for *security_domain_name*

user_or_group_name [*security_role*[, *security_role*]...]

[*user_or_group_name* [*security_role*[, *security_role*]...]]...
end

[**securitydomain** ...]

[**securityrole** ...]

accesscontrollist folder *folder_name*
user_or_group_name [*security_role*[, *security_role*]...]
[*user_or_group_name* [*security_role*[, *security_role*]...]]...
end

Arguments

security_domain_name

Specifies the name of the security domain on which you are defining the access control list.

***user_or_group_name* [*security_role*[, *security_role*]**

Assigns one or more security roles to a certain user or group, on the specified security domain.

folder_name

Specifies the name of the folder to which you can associate an access control list. If the access control list is associated to a folder, then the security roles are valid for all of the objects contained in the folder. When specifying folder names, ensure you include a forward slash (/) before the folder name. Include a forward slash after the folder name to indicate that the access control list is defined only on the folder specified, excluding any sub-folders. A folder name without a final forward slash indicates that the access control list is defined on the folder, as well as on any sub-folders.

Associating an access control list to a folder is a quick and easy method to grant access to all of the objects defined in a folder. If, instead, you need to restrict access to a subset of objects in the folder (for example, objects with a certain name, or specific userlogon, cpu or jcl), then using an access control list associated to a security domain is more effective. With security domains you can filter objects by specifying one or more attributes for each security object type.

See the following composer commands documented in the *User's Guide and Reference* when working with folders: Chfolder, Listfolder, Mkfolder, Rmfolder, and Renamefolder.

Example

Examples

The following example defines:

- An access control list on the `SECDOM1` domain
- An access control list on `SECDOM2` domain
- An access control list on the folder `/FOL1/FOL2/`
- An access control list on the folder `/APPS/APP1` and any sub-folders, if present, for example, `/APPS/APP1/APP1A`.

```
ACCESSCONTROLLIST FOR SECDOM1
  USER1 SECROLE1, SECROLE2, SECROLE3
  USER2 SECROLE4
  USER3 SECROLE2, SECROLE4
END

ACCESSCONTROLLIST FOR SECDOM2
  USER1 SECROLE1, SECROLE2
  USER2 SECROLE3
END

ACCESSCONTROLLIST FOLDER /FOL1/FOL2/
             USER1 SECROLE1
END
ACCESSCONTROLLIST FOLDER /APPS/APP1
             USER1 SECROLE1
END
```

Security domain definition

In the role-based security model, a security domain represents the set of objects that users or groups can manage. For example, you can define a domain that contains all objects named with a prefix 'AA'. If you want to specify different security attributes for some or all of your users, you can create additional security domains based on specific matching criteria. You can filter objects by specifying one or more attributes for each security object type. You can include or exclude each attribute from the selection. For example, you can restrict access to a set of objects having the same name or being defined on the same workstation, or both.

You can include multiple security domain definitions in the same text file, along with security role definitions and access control list definitions.

By default, a security domain named ALLOBJECTS is available. It contains all scheduling objects and cannot be renamed nor modified. If you try to rename it, a copy of the domain is created with the new name.

Each security domain definition has the following format and arguments:

Syntax

Each attribute can be included or excluded from the selection using the plus (+) and tilde (~) symbols.

securitydomain *security_domain_name*
  [**description** "*description*"]
    [**common** [[+|~*object_attribute* [= *value* | @[, *value* | @]...]]]
    object_type [[+|~]*object_attribute* [= *value* | @[, *value* | @]...]]
    [*object_type* [[+|~]*object_attribute* [= *value* | @[, *value* | @]...]]]...
  end

[**securityrole** ...]

[**accesscontrollist** ...]

Arguments

securitydomain *security_domain_name*

Specifies the name of the security domain. The name must start with a letter, and can contain alphanumeric characters, dashes, and underscores. It can contain up to 16 characters.

description *"description"*

Provides a description of the security domain. The description can contain up to 120 alphanumeric characters. The text must be enclosed within double quotes.

common [[+|~]*object_attribute* [= *value* | @[, *value* | @]...]]

Provides object attributes that are common to all the security object types.

***object_type* [[+|~]*object_attribute* [= *value* | @[, *value* | @]...]]**

For each object type, specifies the attributes that apply to that object type and the related values. Each attribute can be included or excluded from the selection using the plus (+) and tilde (~) symbols. Wildcard (@) is supported for the attribute value: *object_attribute* =@ means that all the objects matching the object attribute must be included in the domain. For the use of wildcard (@), see the examples below.

For the attributes that you can specify for each security object type, see the section about managing security with the Dynamic Workload Console, in the *Dynamic Workload Console User's Guide*.

For the values that you can specify for each object attribute, see the section about managing security with the Dynamic Workload Console, in the *Dynamic Workload Console User's Guide*.

Example

Examples

The following example defines a security domain named `SECDOM1` and a security domain named `SECDOM2`:

```
securitydomain SECDOM1
description "Sample Security Domain1"
job       cpu =   $THISCPU, # The workstation where the user logs on
                  $MASTER,  # The master workstation
                  $AGENTS,  # Any fault tolerant agent
                  $REMOTES  # Any standard agent
                  cogs@     # Any workstation whose name starts with "cogs"
         + folder = /  # Jobs defined in any folder
         + cpufolder = /  # Workstations defined in any folder
        + name =  A@        # Any job whose name starts with "A"
        ~  name =  A2@       # but doesn't start with A2
        + jcltype = SCRIPTNAME # Allow only SCRIPTNAME type of job definition
        + jcltype = DOCOMMAND  # Allow only DOCOMMAND type of job definition
        + logon =   $USER,  # Streamlogon is the conman/composer user
                    $OWNER, # Streamlogon is the job creator
                    $JCLOWNER, # Streamlogon is the OS owner of the file
                    $JCLGROUP  # Streamlogon is the OS group of the file
        ~  logon =   root, twsuser  # The job cannot logon as "root" or "twsuser"
        + jcl   =   "/usr/local/bin/@"  # The jobs whose executable file that is
   present in /usr/local/bin
        ~  jcl   =  "@rm@" # but whose JSDL definition does not contain the
   string "rm"
end

securitydomain SECDOM2
description "Sample Security Domain2"
    common      cpu=@+name=@
    userobj     cpu=@  + cpufolder = /
    job         cpu=@+ folder = / + cpufolder = /
    schedule    cpu=@+name=AP@+ folder = / + cpufolder = /
    resource    cpu=@ + folder = / +  + cpufolder = /
    prompt      folder = /
    file        name=@
    cpu         cpu=@  + folder = /
    parameter   cpu=@  + folder = /  + cpufolder = /
    calendar    folder = /
    report      name=@
    eventrule   name=@ + folder = /
    action      provider=@
    event       provider=@
    vartable    name=@  +  folder = /
    wkldapp     name=@  +  folder = /
    runcygrp    name=@  +  folder = /
    lob         name=@
    folder      name=/
end
```

Security role definition

In the role-based security model, a security role represents a certain level of authorization and includes the set of actions that users or groups can perform. You can include multiple security role definitions in the same text file, along with security domain definitions and access control list definitions.

Each security role definition has the following format and arguments:

Syntax

securityrole *security_role_name*
  [**description** "*description*"]
    object_type **access**[=*action*[,*action*]...]
    [*object_type* **access**[=*action*[,*action*]...]]...
  end

[**securitydomain** ...]

[**accesscontrollist** ...]

Arguments

securityrole*securityrolename*

Specifies the name of the security role. The name must start with a letter, and can contain alphanumeric characters, dashes, and underscores. It can contain up to 16 characters.

description *"description"*

Provides a description of the security role. The description can contain up to 120 alphanumeric characters. The text must be enclosed within double quotes.

***object_type* access[=*action*[,*action*]...]**

For each object type, specifies a list of actions that users or groups can perform on that specific object type.

Table 56: Security object types on page 420 shows the different object types and how they are referenced with **composer** and with the Dynamic Workload Console:

Table 56. Security object types

Object type - composer	Object type - Dynamic Workload Console	Description
action	Actions	Actions defined in scheduling event rules
calendar	Calendars	User calendars
cpu	Workstations	Workstations, domains, and workstation classes
event	Events	Event conditions in scheduling event rules
eventrule	Event Rules	Scheduling event rule definitions
file	Files	HCL Workload Automation database files
folder	Folders	The folder within which jobs and job streams are defined.
job	Jobs	Scheduled jobs and job definitions
parameter	Parameters	Local parameters
prompt	Prompts	Global prompts
report	Reports	The following reports in Dynamic Workload Console: **RUNHIST** Job Run History **RUNSTATS** Job Run Statistics **WWS** Workstation Workload Summary **WWR** Workstation Workload Runtimes **SQL** Custom SQL **ACTPROD** Actual production details (for current and archived plans) **PLAPROD** Planned production details (for trial and forecast plans)
resource	Resources	Scheduling resources
runcygrp	Run Cycle Groups	Run cycle groups

Table 56. Security object types (continued)

Object type - composer	Object type - Dynamic Workload Console	Description
schedule	Job Streams	Job streams
userobj	User Objects	User objects
vartable	Variable Tables	Variable tables
wkldappl	Workload Application	Workload application

Table 57: Actions that users or groups can perform on the different objects on page 421 shows the actions that users or groups can perform on the different objects.

Table 57. Actions that users or groups can perform on the different objects

Actions that users or groups can perform on the different objects			
acl	deldep	modify	stop
add	delete	release	submit
adddep	display	reply	submitdb
altpass	fence	rerun	unlink
altpri	kill	resetfta	unlock
build	limit	resource	use
cancel	link	run	
confirm	list	shutdown	
console	manage	start	

For the actions that users or groups can perform on a specific object type, for each of the HCL Workload Automation task, see the section about managing security roles with the Dynamic Workload Console, in the *Dynamic Workload Console User's Guide*.

Example

Examples

The following example defines security role `SECROLE1` and security role `SECROLE2`:

```
SECURITYROLE SECROLE1
```

```
 DESCRIPTION "Sample Security Role"
 SCHEDULE    ACCESS=ADD,ADDDEP,ALTPRI,CANCEL,DELDEP,DELETE,
                                  DISPLAY,LIMIT,MODIFY,
  RELEASE
       RESOURCE          ACCESS=ADD,DELETE,DISPLAY,MODIFY,RESOURCE,USE,LIST,UNLOCK
       PROMPT            ACCESS=ADD,DELETE,DISPLAY,MODIFY,REPLY,USE,LIST,UNLOCK
       FILE              ACCESS=BUILD,DELETE,DISPLAY,MODIFY,UNLOCK
       FOLDER            ACCESS=ADD,DELETE,DISPLAY,MODIFY,USE,LIST,UNLOCK,ACL
       CPU               ACCESS=LIMIT,LINK,MODIFY,SHUTDOWN,START,STOP,UNLINK,LIST,UNLOCK,RUN
       PARAMETER         ACCESS=ADD,DELETE,DISPLAY,MODIFY,LIST,UNLOCK
       CALENDAR          ACCESS=ADD,DELETE,DISPLAY,MODIFY,USE,LIST,UNLOCK
       REPORT            ACCESS=DISPLAY
       EVENTRULE         ACCESS=ADD,DELETE,DISPLAY,MODIFY,LIST,UNLOCK
       ACTION            ACCESS=DISPLAY,SUBMIT,USE,LIST
       EVENT             ACCESS=USE
       VARTABLE          ACCESS=ADD,DELETE,DISPLAY,MODIFY,USE,LIST,UNLOCK
       WKLDAPPL          ACCESS=ADD,DELETE,DISPLAY,MODIFY,LIST,UNLOCK
       RUNCYGRP          ACCESS=ADD,DELETE,DISPLAY,MODIFY,USE,LIST,UNLOCK
       LOB               ACCESS=USE
END

SECURITYROLE SECROLE2
 DESCRIPTION "Sample Security Role"
 SCHEDULE          ACCESS=ADD,ADDDEP,ALTPRI,CANCEL,DELDEP,DELETE,
                            DISPLAY,LIMIT,MODIFY,
  RELEASE
 RESOURCE          ACCESS=ADD,DELETE,DISPLAY,MODIFY,RESOURCE,USE,LIST,UNLOCK
 PROMPT            ACCESS=ADD,DELETE,DISPLAY,MODIFY,REPLY,USE,LIST,UNLOCK
END
```

The following example defines a new security role `APP_ADMIN`, for the user `APP1_ADMIN` and assigns administrator permissions on the folder hierarchy `/PRD/APP1/`, so that the `APP1_ADMIN` user can create access control lists to give other users access to the objects in this folder or its sub-folders:

Security role definition

```
SECURITYROLE APP_ADMIN
  DESCRIPTION "Security Role"
  JOB  ADD,MODIFY,SUBMITDB,USE,ADDDEP,RUN,RELEASE,REPLY,DELETE,DISPLAY,
       CANCEL,SUBMIT,CONFIRM,RERUN,LIST,DELDEP,KILL,UNLOCK,ALTPRI
  SCHEDULE ADD,ADDDEP,ALTPRI,CANCEL,DELDEP,DELETE,DISPLAY,LIMIT,MODIFY,RELEASE
  FOLDER   ADD,DELETE,DISPLAY,MODIFY,USE,LIST,UNLOCK,ACL
```

Security file

```
USER APP_ADMINofAPP1
  CPU=@+LOGON="APP_ADMIN"
BEGIN
  JOB    FOLDER="/PRD/APP1/","/PRD/APP1" + CPUFOLDER = / ACCESS=ADD,ADDDEP,
         ALTRPRI,CANCEL,SUBMIT,
         CONFIRM,RERUN,LIST,DELDEP,KILL,UNLOCK,ALTPRI
  SCHEDULE  FOLDER="/PRD/APP1/","/PRD/APP1" + CPUFOLDER = / ACCESS=ADD,ADDDEP,
         ALTPRI,CANCEL,DELDEP,
         DELETE,DISPLAY,LIMIT,MODIFY,RELEASE
  FOLDER NAME="/PRD/APP1/","PRD/APP1"   ACCESS=ADD,DELETE,DISPLAY,MODIFY,USE,
         LIST,UNLOCK,ACL
```

Actions on security objects

The following tables show the actions that users or groups can perform on the different object types, for each HCL Workload Automation task. See in parenthesis the corresponding *actions* and *objects* values that you must use when defining role-based security with **composer** command line interface.

Table 58. Actions that users or groups can perform when designing and monitoring the workload

Design and Monitor Workload	
Actions that users or groups can perform	**Security object types**
List (list)	Jobs (job)
Display (display)	Job Streams (schedule)
Create (add)	User Objects (userobj)
Delete (delete)	Prompts (prompt)
Modify (modify)	Resources (resource)
Use (use)	Calendars (calendar)
Unlock (unlock)	Run Cycle Groups (runcygrp)
Actions on remote workstations while modeling jobs (cpu-run) **Note:** See in parenthesis the corresponding *actions* and *objects* values that you must use when defining role-based security with the **composer** command-line interface.	Variable Tables (vartable) Workload Application (wkldappl) Workflow Folders (folder) Parameters (parameter)

Table 59. Actions that users or groups can perform when modifying current plan

Modify current plan
Actions that users or groups can perform on the current plan
Add job stream dependency (schedule - adddep)
Add job dependency (job - adddep)
Remove job dependency (job - deldep)
Remove job stream dependency (schedule - deldep)
Change job priority (job - altpri)

Table 59. Actions that users or groups can perform when modifying current plan

(continued)

Modify current plan
Actions that users or groups can perform on the current plan
Change job stream priority (schedule - altpri)
Cancel job (job - cancel)
Cancel job stream (schedule - cancel)
Rerun job (job - rerun)
Confirm job (job - confirm)
Release job (job - release)
Release job stream (schedule - release)
Kill jobs (job - kill)
Reply to prompts (prompt - reply)
Reply to job prompts (job - reply)
Reply to job stream prompts (schedule - reply)
Alter user password (userobj - altpass)
Change jobs limit (schedule - limit)
Actions on job remote system (job - run)
Change resource quantity (resource - resource)

Note: See in parenthesis the corresponding *actions* and *objects* values that you must use when defining role-based security with the **composer** command line interface.

Table 60. Actions that users or groups can perform when submitting workload

Submit Workload
Workload definitions that can be added to the current plan
Only existing job definitions (job - submitdb)
Existing jobs definitions and ad hoc jobs (job - submit)
Existing job stream definitions (schedule - submit)

Table 60. Actions that users or groups can perform when submitting workload

(continued)

Submit Workload
Workload definitions that can be added to the current plan

Note: See in parenthesis the corresponding *actions* and *objects* values that you must use when defining role-based security with the **composer** command line interface.

Table 61. Actions that users or groups can perform when managing the workload environment

Manage Workload Environment
Actions that users or groups can perform on workstations, domains, and workstation classes
List workstations (cpu - list)
Display workstation details (cpu - display)
Create workstations (cpu - add)
Delete workstations (cpu - delete)
Modify workstations (cpu - modify)
Use workstations (cpu - use)
Unlock workstations (cpu - unlock)
Start a workstation (cpu - start)
Stop a workstation (cpu - stop)
Change limit (cpu - limit)
Change fence (cpu - fence)
Shutdown (cpu - shutdown)
Reset FTA (cpu - resetfta)
Link (cpu - link)
Unlink (cpu - unlink)
Use 'console' command from conman (cpu - console)
Upgrade workstation (cpu - manage)

Table 61. Actions that users or groups can perform when managing the workload environment

(continued)

Manage Workload Environment
Actions that users or groups can perform on workstations, domains, and workstation classes

Note: See in parenthesis the corresponding *actions* and *objects* values that you must use when defining role-based security with the **composer** command line interface.

Table 62. Actions that users or groups can perform when managing event rules

Manage Event Rules
Actions that users or groups can perform on event rules
List event rules (eventrule - list)
Display event rules details (eventrule - display)
Create event rules (eventrule - add)
Delete event rules (eventrule - delete)
Modify event rules (eventrule - modify)
Use event rules (eventrule - use)
Unlock event rules (eventrule - unlock)
Display actions in the event rules (action - display)
Monitor triggered actions (action - list)
Use action types in the event rules (action - use)
Submit action (action - submit)
Use events in the event rules (event - use)
Use a File Monitor event on the workstation where the file resides. (event - display)

Table 62. Actions that users or groups can perform when managing event rules

(continued)

Manage Event Rules
Actions that users or groups can perform on event rules

Note: See in parenthesis the corresponding *actions* and *objects* values that you must use when defining role-based security with the **composer** command line interface.

Table 63. Administrative tasks that users or groups can perform

Administrative Tasks
Administrative tasks that users or groups can perform
View configuration (dump security and global options) (file - display)
Change configuration (makesec, optman add) (file - modify)
Delete objects definitions (file - delete)
Unlock objects definitions (file - unlock)
Allow planman deploy, prodsked and stageman (file - build)
Delegate security on folders (folder - acl)

Note: See in parenthesis the corresponding *action* and *object* values that you must use when defining role-based security with the **composer** command-line interface.

Table 64. Actions that users or groups can perform on workload reports

Workload Reports	
Actions that users or groups can perform on workload reports	
Generate workload reports (display report)	**Reports in Dynamic Workload Console** **RUNHIST** Job Run History **RUNSTATS** Job Run Statistics **WWS** Workstation Workload Summary **WWR** Workstation Workload Runtimes

Table 64. Actions that users or groups can perform on workload reports

(continued)

Workload Reports
Actions that users or groups can perform on workload reports
SQL Custom SQL **ACTPROD** Actual production details (for current and archived plans) **PLAPROD** Planned production details (for trial and forecast plans)

Note: See in parenthesis the corresponding *actions* and *objects* values that you must use when defining role-based security with the **composer** command line interface.

Table 65. Actions that users or groups can perform on folders.

Folders
Actions that users or groups can perform on folders
Access folders chfolder (display) listfolder (list or list and display) mkfolder (modify) rmfolder (delete) renamefolder (add)

Note: See in parenthesis the corresponding *actions* and *objects* values that you must use when defining role-based security with the **composer** command line interface.

Attributes for security object types

Table 66: Attributes for security object types on page 429 shows the attributes that you can specify for each security object type (see in parenthesis the corresponding object type and object attribute that you must use when defining security objects with the **composer** command line interface).

Table 66. Attributes for security object types

Attribute Security object type	Name (name)	Workstation (cpu)	Custom (custom)	JCL (jcl)	JCLtype (jcltype)	Logon (logon)	Provider (provider)	Type (type)	Host (host)	Port (port)	Folder (folder)	CPU Folder (cpufolder)
Actions (action)							✓	✓	✓	✓		
Calendars (calendar)	✓										✓	
Workstations (cpu)	✓							✓			✓	
Events (event)			✓				✓	✓				
Event rules (eventrule)	✓										✓	
Files (file)	✓											
Jobs (job)	✓	✓		✓	✓	✓					✓	✓
Parameters (parameter)	✓	✓									✓	✓
Prompts (prompt)	✓										✓	
Reports (report)	✓											
Resource (resource)	✓	✓									✓	
RunCycle groups (runcygrp)	✓										✓	
Job streams (schedule)	✓	✓									✓	✓
User objects (userobj)		✓				✓						✓
Variable tables (vartable)	✓										✓	
Workload applications (wkldappl)	✓										✓	
Folders (folder)	✓											

For the values that are allowed for each object attribute, see Specifying object attribute values on page 429.

Specifying object attribute values

The following values are allowed for each object attribute (see in parenthesis the corresponding object type and object attribute for the **composer** command line interface):

Name (name)

Specifies one or more names for the object type.

- For the Files (file) object type, the following values apply:

  globalopts

  Allows the user to set global options with the `optman` command. The following access types are allowed:

  - Display access for `optman ls` and `optman show`
  - Modify access for `optman chg`

  prodsked

  Allows the user to create, extend, or reset the production plan.

  security

  Allows the user to manage the security file.

  Symphony

  Allows the user to run **stageman** and **JnextPlan**.

  trialsked

  Allows the user to create trial and forecast plans or to extend trial plans.

  Note: Users who have restricted access to files should be given at least the following privilege to be able to display other object types that is, Calendars (calendar) and Workstations (cpu):

  ```
  file  name=globalopts  action=display
  ```

- For the **Variable Tables (vartable)** object type, you can use the $DEFAULT value for the **Name (name)** attribute to indicate the default variable table. This selects the table that is defined with the `isdefault` attribute.

Workstation (cpu)

Specifies one or more workstation, domain, or workstation class name. Workstations and workstation classes can optionally be defined in a folder. If this attribute is not specified, all defined workstations and domains can be accessed. Workstation variables can be used:

$MASTER

The HCL Workload Automation master domain manager.

$AGENTS

Any fault-tolerant agent.

$REMOTES

Any standard agent.

$THISCPU

The workstation on which the user is running the HCL Workload Automation command or program.

If you use the **composer** command line to define security domains, the following syntax applies:

```
cpu=[folder/]workstation]...
```

folder=*foldername*

Scheduling objects such as, jobs, job streams, and workstations, to name a few, can be defined in a folder. A folder can contain one or more scheduling objects, while each object can be associated to only one folder. The default folder is the root folder (/).

cpufolder=*foldername*

The folder within which the workstation or workstation class is defined.

Custom (custom)

Use this attribute to assign access rights to events defined in event plug-ins. The precise syntax of the value depends on the plug-in. For example:

- Specify different rights for different users based on SAP R/3 event names when defining event rules for SAP R/3 events.
- Define your own security attribute for your custom-made event providers.
- Specify the type of event that is to be monitored. Every event can refer to an event provider.

If you use **composer** command line to define security domains, the following syntax applies:

```
custom=value[,value]...
```

JCL (jcl)

Specifies the command or the path name of a job object's executable file. If omitted, all defined job files and commands qualify.

You can also specify a string that is contained in the task string of a JSDL definition to be used for pattern matching.

If you use **composer** command line to define security domains, the following syntax applies:

```
jcl="path" | "command" | "jsdl"
```

JCL Type (jcltype)

Specifies that the user is allowed to act on the definitions of jobs that run only scripts (if set to scriptname) or commands (if set to docommand). Use this optional attribute to restrict user authorization to actions on the definitions of jobs of one type only. Actions are granted for both scripts and commands when JCL Type (jcltype) is missing.

A user who is not granted authorization to work on job definitions that run either a command or a script is returned a security error message when attempting to run an action on them.

If you use **composer** command line to define security domains, the following syntax applies:

```
jcltype=[scriptname | docommand]
```

Logon (logon)

Specifies the user IDs. If omitted, all user IDs qualify.

You can use the following values for the **Logon (logon)** attribute to indicate default logon:

$USER

Streamlogon is the conman/composer user.

$OWNER

Streamlogon is the job creator.

$JCLOWNER

Streamlogon is the OS owner of the file.

$JCLGROUP

Streamlogon is the OS group of the file.

If you use **composer** command line to define security domains, the following syntax applies:

```
logon=username[,username]...
```

Provider (provider)

For **Actions (action)** object types, specifies the name of the action provider.

For **Events (event)** object types, specifies the name of the event provider.

If **Provider (provider)** is not specified, no defined objects can be accessed.

If you use **composer** command line to define security domains, the following syntax applies:

```
provider=provider_name[,provider_name]...
```

Type (type)

For **Actions (action)** object types, is the `actionType`.

For **Events (event)** object types, is the `eventType`.

For **Workstations (cpu)** object types, the permitted values are those used in composer or the Dynamic Workload Console when defining workstations, such as `manager`, `broker`, `fta`, `agent`, `s-agent`, `x-agent`, `rem-eng`, `pool`, `d-pool`, `cpuclass`, and `domain`.

Note: The value `master`, used in conman is mapped against the `manager` security attributes.

If **Type (type)** is not specified, all defined objects are accessed for the specified providers (this is always the case after installation or upgrade, as the type attribute is not supplied by default).

If you use **composer** command line to define security domains, the following syntax applies:

```
type=type[,type]...
```

Host (host)

For **Actions (action)** object types, specifies the TEC or SNMP host name (used for some types of actions, such as sending TEC events, or sending SNMP). If it does not apply, this field must be empty.

If you use **composer** command line to define security domains, the following syntax applies:

```
host=host_name
```

Port (port)

For **Actions (action)** object types, specifies the TEC or SNMP port number (used for some types of actions, such as sending TEC events, or sending SNMP). If it does not apply, this field must be empty.

If you use **composer** command line to define security domains, the following syntax applies:

```
port=port_number
```

Chapter 10. Managing objects in the database - composer

This section describes how you use the **composer** command-line program to manage scheduling objects in the HCL Workload Automation database. It is divided into the following sections:

- SETUPCOMPOSER Setting up the composer command-line program on page 434
- RUNCOMPOSER Running commands from composer on page 441
- COMPOSER COMMANDS on page 447

SETUPCOMPOSER Setting up the composer command-line program

About this task

The **composer** command line program manages scheduling objects in database.

You must install the HCL Workload Automation Command Line Client feature on fault-tolerant agents and systems outside the HCL Workload Automation network to use the **composer** command-line program.

Users can decide to maintain an audit trail recording any changes they perform and the related justifications. To enable the justification option, set up in a system shell the HCL Workload Automation environment variables listed below before running any **composer** commands:

IWS_DESCRIPTION

Specify the description to be recorded for each change performed by commands in the shell. The maximum length for this value is 512 characters. A warning message displays if you exceed the maximum and excess characters are truncated.

IWS_CATEGORY

Specify the category to be recorded for each change performed by commands in the shell. The maximum length for this value is 128 characters. A warning message displays if you exceed the maximum and excess characters are truncated.

IWS_TICKET

Specify the ticket to be recorded for each change performed by commands in the shell. The maximum length for this value is 128 characters. A warning message displays if you exceed the maximum and excess characters are truncated.

For more information about the justification option, see the section about keeping track of changes in Dynamic Workload Console User's Guide.

To authenticate using a JSON Web Token (JWT), retrieve the token from the Dynamic Workload Console, start a shell session, and run the command with the **-jwt** parameter, for example:

```
composer -jwt aòsdfjasldfkjasdflòjahsdfjlkasd
```

Note: On AIX systems, if you plan to define a large number of objects, the data segment should be increased for ensuring product reliability.

Setting up the composer environment

About this task

This section describes how you set up your composer environment.

Terminal output

The shell variables named ***MAESTROLINES*** and ***MAESTROCOLUMNS*** determine the output to your computer. The variables are defined in the `tws_env` script, which you run before running command line commands. If either variable is not set, the standard shell variables, ***LINES*** and ***COLUMNS***, are used. At the end of each screen page, composer does not pause at the end of a page. If ***MAESTROLINES*** (or ***LINES***) is set to a positive number, composer prompts to continue.

Depending on the value set in the ***MAESTROCOLUMNS*** local variable, two different sets of information are displayed about the selected object. There are two possibilities:

- MAESTROCOLUMNS < 120 characters
- MAESTROCOLUMNS >= 120 characters

The value set in the ***MAESTROCOLUMNS*** local variable cannot be higher than 1024.

Refer to the tables in DISPLAY and LIST to learn about the different output formats.

Offline output

The **;offline** option in **composer** commands is used to print the output of a command on a specified file. When you include it, the following shell variables control the output:

MAESTROLP

Specifies the destination file of a command's output. Set it to one of the following:

> *outputfilename*

Redirects output to a file and overwrites the contents of that file. If the file does not exist, it is created. For example:

```
MAESTROLP='> /tmp/outputfilename.txt'
```

>> *outputfilename*

Redirects output to a file and appends the output to the end of that file. If the file does not exist, it is created.

| *command*

Pipes output to a system command or process. The system command is run whether or not output is generated.

|| ***command***

Pipes output to a system command or process. The system command is not run if there is no output.

The default value for ***MAESTROLP*** is | **lp -tCONLIST** which pipes the command output to the printer and places the title "CONLIST" in the printout's banner page.

MAESTROLPLINES

Specifies the number of lines per page. The default is **60**.

MAESTROLPCOLUMNS

Specifies the number of characters per line. The default is **132**.

The variables must be exported before running **composer**.

The composer editor

Several composer commands automatically open a text editor. You can select which editor you want composer to use.

In addition, in both Windows® and UNIX®, you can set the `XMLEDIT` environment variable to point to an XML editor of your choice to edit event rule definitions. The XML editor opens automatically each time you run the `composer add`, `new`, or `modify` commands on an event rule.

Windows®

In Windows®, **Notepad** is used as the default editor. To change the editor, set the *EDITOR* environment variable to the path and name of the new editor before you run **composer**.

UNIX®

Several commands you can issue from **composer** automatically open a text editor. The type of editor is determined by the value of two shell variables. If the variable ***VISUAL*** is set, it defines the editor, otherwise the variable ***EDITOR*** defines the editor. If neither of the variables is set, a **vi** editor is opened.

Selecting the composer prompt on UNIX®

About this task

The **composer** command prompt is defined in the *TWS_home*`/localopts` file. The default command prompt is a dash (-). To select a different prompt, edit the ***composer prompt*** option in the `localopts` file and change the dash. The prompt can be up to ten characters long, not including the required trailing pound sign (#):

```
#------------------------------------------------------------------------
# Custom format attributes
#
date format =               1         # The possible values are 0-ymd, 1-mdy,
```

```
2-dmy, 3-NLS.
composer prompt =           -
conman prompt =             %
switch sym prompt =        <n>%
#-------------------------------------------------------------------------
```

For additional information about `localopts` configuration file, refer to *Administration Guide*.

Running the composer program

About this task

To configure your environment to use composer, set the *PATH* and *TWS_TISDIR* variables by running one of the following scripts:

On UNIX® operating systems:

- `. ./TWA_home/TWS/tws_env.sh` for Bourne and Korn shells
- `. ./TWA_home/TWS/tws_env.csh` for C shells

On Windows® operating systems:

- `TWA_home\TWS\tws_env.cmd`

Then use the following syntax to run commands from the composer user interface:

composer [*custom_parameters*] [**-file** *customPropertiesFile*][***connection_parameters***] ["*command*[**&**[*command*]][...]"]

where:

custom_parameters

Sets the working directory or current folder using the following syntax:

```
-cf /foldername
```

The current directory from where commands are submitted. The default current directory is the root (/).

-file *customPropertiesFile*

The custom properties file where you can specify connection parameters or custom parameters that override the values specified in the `useropts`, `localopts` and `jobmanager.ini` files. Connection parameters specified in the custom properties file must have the following syntax:

```
HOST=hostname
  PORT=port
  PROTOCOL=http/https
  PROXY=proxyName
  PROXYPORT=proxyPortNumber
  PASSWORD=password
  TIMEOUT=seconds
```

```
USERNAME=username
CURRENT FOLDER=/foldername
```

connection_parameters

If you are using **composer** from the master domain manager, the connection parameters were configured at installation time and do not need to be supplied, unless you do not want to use the default values.

If you are using **composer** from the command line client on another workstation, the connection parameters might be supplied by one or more of these methods:

- Stored in the `localopts` file
- Stored in the `useropts` file
- Supplied to the command in a parameter file
- Supplied to the command as part of the command string

For an overview of these options see the section about setting up options for using the user interfaces in the *User's Guide and Reference*. For full details of the configuration parameters see the topic on configuring the command-line client access in *Administration Guide*.

Note: If you are using composer from the command-line client on another workstation, for the following subset of scheduling objects, the composer command line connects to the server by using an HTTPS connection:

- jobs
- job streams
- folders
- run cycle groups
- workload applications
- access control lists
- security domains
- security roles

In this case, the command-line client assembles the full set of connection parameters in the following order:

1. Parameters specified in the command string itself
2. Parameters specified in the custom properties file
3. Parameters specified in the useropts file
4. Parameters specified in the localopts file
5. Parameters specified in the jobmanager.ini file

Valid values include:

[-host *hostname*]

The name of the host that you want to access by using wappman command line.

[-port *port_number*]

The TCP/IP port number used to connect to the specified host.

[-protocol {http | https}]

The protocol used to connect to the specified host.

[-proxy *proxyName*]

The name of the proxy server used when accessing a command-line client.

[-proxyport *proxyPortNumber*]

The TCP/IP port number of the proxy server used when accessing using a command-line client.

[-jwt *JSON Web Token*]

Specify the JWT to be used for authentication between the master domain manager and agents. You can retrieve the token from the Dynamic Workload Console. This parameter is mutually exclusive with the **username** and **password** parameters. The JWT authentication applies to the commands you launch in the shell where you specify the JWT.

[-username *user_name*]

An HCL Workload Automation user with sufficient privileges to perform the operation. This parameter is mutually exclusive with the **jwt** parameter.

[-password *password*]

The password of the HCL Workload Automation user. This parameter is mutually exclusive with the **jwt** parameter.

[-timeout *seconds*]

The timeout in seconds when accessing using a command-line client. The default is 3600 seconds.

[-defaultws *twscpu*]

The default workstation. This workstation is used when you run composer commands without specifying a workstation.

If host, port, and protocol parameters are specified in a file, all of them must be specified in the same file.

The composer command-line program is installed automatically when installing the master domain manager. It must be installed separately on top of an HCL Workload Automation agent workstation or stand-alone on a node outside the HCL Workload Automation network. The feature that installs the composer command-line program is named *Command Line Client* that is installed together with the fault-tolerant agent and dynamic agent.

You can use the composer command line both in *batch* and in *interactive* mode.

When running composer in *interactive* mode, you first launch the composer command-line program and then, from the composer command line prompt, you run commands one at a time, for example:

```
composer -username  admin2  -password  admin2pwd
   add myjobs.txt
   create myjobs.txt from jobs=@
```

When running composer in *batch* mode, you launch the composer command-line program specifying as input parameter the command to be issued. When the command is processed, the composer command-line program exits, for example,

```
composer -f "c:\TWS\network\mylocalopts" add myjobs.txt
```

Note: If you use the batch mode to issue more than one command from within the composer, make sure you manage the semi-colon (;) character in one of the following ways:

- Using double quotation marks, for example:

  ```
  composer "delete dom=old_domain; noask"
  ```

- Using a space character, for example:

  ```
  composer delete dom=old_domain noask
  ```

- Escaping the ; character, for example:

  ```
  composer delete dom=old_domain \; noask
  ```

Other examples on how to use the command, assuming connection parameters are set in the local configuration scripts, are the following:

- Runs print and version commands, and quits:

  ```
  composer "p parms&v"
  ```

- Runs print and version commands, and then prompts for a command:

  ```
  composer "p parms&v&"
  ```

- Reads commands from `cmdfile`:

  ```
  composer cmdfile
  ```

- Pipes commands from `cmdfile` to composer:

  ```
  cat cmdfile | composer
  ```

- Authenticates using JWT:

  ```
  composer -jwt igJraVRiOiTls2l.....JUNnrWmdC-kR
  ```

Note:

- On Windows workstations, if the User Account Control (UAC) is turned on and the UAC exception list does not contain the `cmd.exe` file, open the DOS command prompt shell with the `"Run As Admnistrator"` option to run composer on your workstation as a generic user different from Administrator or HCL Workload Automation user.
-

If you install an HCL Workload Automation component, either agent or server, in the `C:\` or `C:\Program Files\` directories, the **twsuser** or any user other than the Administrator might encounter issues when running commands from the command-line interface due to security settings. For example, the resource command-line interface might fail to run commands, and the conman command might display error or warning messages. To prevent these issues, open the command prompt with the **Run as administrator** option to run commands.

Control characters

You can enter the following control characters in conversational mode to interrupt **composer** if your ***stty*** settings are configured to do so.

Ctrl+c

composer stops running the current command at the next step that can be interrupted and returns a command prompt.

Ctrl+d

composer quits after running the current command.

RUNCOMPOSER Running commands from composer

About this task

Composer commands consist of the following elements:

commandname selection arguments

where:

commandname

Specifies the command name.

selection

Specifies the object or set of objects to be acted upon. Most scheduling objects support the use of folders. The objects can be organized into a tree-like structure of folders similar to a file system.

arguments

Specifies the command arguments.

When submitting commands involving folders, you can specify either a relative or absolute path to the folder. If no folder is specified, then the command is performed using the current folder which is set to root ("/") by default. If you generally work from a specific folder, then you can use the chfolder command to navigate to folders and sub-folders. The chfolder command changes the working directory or current folder, so that you can use relative folder paths when submitting commands.

To use the composer command to manage folders on components at versions earlier than 9.5.*x*, ensure that both the master domain manager and the backup master domain manager are at version 9.5 Fix Pack 2 or later.

See CHFOLDER on page 457 for detailed information about the ways you can modify the current folder.

Filters and wildcards

In HCL Workload Automation **composer** you can use wildcards and filters when issuing some specific commands to filter scheduling objects defined in the database. The wildcards you can use from **composer** are:

@

Replaces one or more alphanumeric characters.

?

Replaces one alphanumeric character.

To search for occurrences with names that contain either @ or ?, make sure you use the backslash character \ before @ or ? to escape them so that they are not interpreted as wildcards. Similarly, the backslash character must be prefixed by another backslash character to be interpreted as an occurrence to be found. The following examples clarify these rules, which also apply when specifying search strings using the **filter** keyword.

S@E

Search for all strings starting with S and ending with E, whatever is their length.

S?E

Search for all strings starting with S and ending with E, and whose length is three characters.

S\@E

Search for an exact match with string S@E.

S\?E

Search for an exact match with string S?E.

S\\E

Search for an exact match with string S\E.

When searching for scheduling objects in folders, the position of wildcards in the string indicates the object being searched. The following examples clarify these rules:

/@/@#/@/@

Search for all objects in all folders for all workstations defined in all folders. The first statement ("/@/" in the example) indicates the folder, the second statement ("@#", in the example) indicates the workstation name, the third statement ("/@/", in the example) indicates the folder, the fourth statement ("@", in the example) indicates the scheduling object.

[folder/]workstationname#/[folder/]jobstreamname

Search for all job streams with the specified name on the specified workstation. For both job streams and workstations, you can specify the folder where the object is stored, if any.

The commands you can issue from composer and that support filtering are:

- display
- create
- delete
- list
- lock
- modify
- print
- unlock
- update

The syntax used to filter objects when issuing one of these commands is the following:

command_name type_of_object=selection; [*option*;] [**filter** *filter_keyword*=selection [...]]

Table 67: Scheduling objects filtering criteria on page 443 shows the scheduling objects you can filter when issuing the commands listed above, and for each object, which fields can be filtered (in *italic*) or which key (in **bold)** is used to filter its fields.

For all objects which can be defined in a folder, such as jobs, job streams, workstations, and so on, you can optionally specify the folder where the object is defined. If no folder is specified, the root folder is used by default.

Table 67. Scheduling objects filtering criteria

Scheduling object	Filter keywords or fields that can be filtered	Description	Example
workstation	*[folder/]workstationname*	Applies the command to the workstations whose name satisfies the criteria.	list ws=p@
	domain	Applies the command to the workstations which belong to a domain.	mod ws=@; filter **domain**=dom1

Table 67. Scheduling objects filtering criteria (continued)

Scheduling object	Filter keywords or fields that can be filtered	Description	Example
	vartable	Applies the command to the workstations which refer the specified variable table.	mod ws=@; filter **vartable**=table2
domain	*domainname*	Applies the command to the domains whose name satisfies the criteria.	display dom=dom?
	parent	Applies the command to the domains whose parent domain satisfies the criteria.	list dom=@; filter **parent**=rome
prompt	*promptname*	Applies the command to the global prompts whose name satisfies the criteria.	lock prompt=p@
user	*[folder/]workstationname# username*	Applies the command to the users whose identifier satisfies the criteria.	list users=cpu1#operator?
resource	*[folder/]workstationname# resourcename*	Applies the command to the resources whose identifier satisfies the criteria.	print res=cpu?#operator?
variable	*variablename*	Applies the command to the parameters whose name satisfies the criteria.	delete vb=mytable.myparm@
folder	*folder*	Applies the command to the folders whose name satisfies the criteria.	list folder myfolder
job definition	*[folder/]workstationname#folder/jobname*	Applies the command to the job definitions whose name satisfies the criteria.	mod jd=mycpu#/myfolder/myjob@
	RecoveryJob	Applies the command to the jobs whose definition contains the specified recovery job definition.	list jobdefinition=@; filter **RecoveryJob**=CPUA#/job01
job stream	*[folder/]workstationname#folder/jobstreamname*	Applies the command to the job stream definitions whose name satisfies the criteria.	mod js=mycpu#testfolder/myjs@ mod js=mycpu#/myfolder/myjs@
	Calendar	Applies the command to the job streams that contain the calendar specified in the filter.	list js=@#@; filter **Calendar**=cal1

Table 67. Scheduling objects filtering criteria (continued)

Scheduling object	Filter keywords or fields that can be filtered	Description	Example
	Jobdefinition	Applies the command to the job streams that contain the job definition specified in the filter.	list js=@#@; filter **jobdefinition**=CPUA#job01
	Resource	Applies the command to the job streams that refer to the resource specified in the filter.	list js=@#@; filter **Resource**=cpu1#disk1
	Prompt	Applies the command to the job streams that refer to the prompt specified in the filter.	list js=@#@; filter **Prompt**=myprompt
	Vartable	Applies the command to the job streams that refer to the variable table specified in the filter. The variable table can be specified either in the run cycle or in the job stream section.	list js=@#@; filter **Vartable**=table1
	Rcvartable	Applies the command to the run cycles in the job streams that refer to the variable table specified in the filter.	list js=@#@; filter **Rcvartable**=table1
	Jsvartable	Applies the command to the job streams that refer to the variable table specified in the filter regardless of what is specified in the run cycle.	list js=@#@; filter **Jsvartable**=table1
	draft	Displays only job streams in draft status	list js=@#@; filter **draft**=table1
	active	Displays only job streams in active status	list js=@#@; filter **active**=table1
event rule	*eventrulename*	Applies the command to the event rules that include an action on a specific job or job stream.	list er=@; filter js=accrecjs5
vartable	*vartablename*	Applies the command to the variable tables whose name satisfies the criteria.	list vartable=A@
	isdefault	Applies the command to the default variable table.	list vartable=A@; filter isdefault

You can combine more than one filter for the same object type as shown in the following example:

```
list js=@#@; filter Calendar=cal1 jobdefinition=CPUA#myfolder/mysubfolder/job01
```

The output of the command is a list of job streams using calendar `cal1` and containing a job with job definition `CPUA#job01`stored in the path `myfolder/mysubfolder`.

Delimeters and special characters

Table 68: Delimeters and special characters for composer on page 446 lists characters have special meanings in composer commands.

Table 68. Delimeters and special characters for composer

Character	Description
&	Command delimiter. See Running the composer program on page 437.
;	Argument delimiter. For example: `;info;offline`
=	Value delimiter. For example: `sched=sked5`
:!	Command prefixes that pass the command on to the system. These prefixes are optional; if composer does not recognize the command, it is passed automatically to the system. For example: `!ls or :ls`
<< >>	Comment brackets. Comments can be placed on a single line anywhere in a job stream. For example: `schedule foo <<comment>> on everyday`

*	Comment prefix. When this prefix is the first character on a line, the entire line is a comment. When the prefix follows a command, the remainder of the line is a comment. For example:
	`*comment`
	`or`
	`print& *comment`

>	Redirects command output to a file, overwriting the contents of that file. If the file does not exist, it is created. For example: `display parms > parmlist`
>>	Redirects command output to a file and appends the output to the end of file. If the file does not exist, it is created. For example: `display parms >> parmlist`
|	Pipes command output to a system command or process. The system command is run whether or not output is generated. For example:

Table 68. Delimeters and special characters for composer (continued)

Character	Description
	`display parms | grep alparm`
||	Pipes command output to a system command or process. The system command is not run if there is no output. For example: `display parms || grep alparm`

Composer return codes

Composer return codes

Composer return codes management

When you run a composer command, the command line can show an outcome return code. To find the return code, perform the following action:

On Windows Operating systems:

```
echo %ERRORLEVEL%
```

On UNIX Operating systems:

```
echo $?
```

The composer command line has the following return codes:

0

Command completed successfully.

4

Command completed with a warning.

8

Command completed with an error.

16

Command fails.

32

Command has a syntax error.

COMPOSER COMMANDS

Note: Command names and keywords can be entered in either uppercase or lowercase characters, and can be abbreviated to as few leading characters as are needed to uniquely distinguish them from each other. Some of the command names also have short forms.

However there are some abbreviations, such as **v**, that point to a specific command, **version** in this case, even though they do not uniquely identify that command in the list. This happens when the abbreviation is hard coded in the product and so mismatches in invoking the wrong command are automatically bypassed.

Table 69. List of composer commands

Command	Short Name	Description
add	**a**	Adds a scheduling objects definition to the database from a text file.
authenticate	**au**	Changes the credentials of the user running **composer**.
chfolder	cf	Changes the working directory.
continue	**co**	Ignores the next error.
create	**cr**	Extracts an object definition from the database and writes it in a text file. Synonym for the **extract** command.
delete	**de**	Deletes scheduling objects.
display	**di**	Displays the details of the specified scheduling object.
edit	**ed**	Edits a file.
exit	**e**	Exits **composer**.
extract	**ext**	Extracts an object definition from the database and writes it in a text file.
help	**h**	Invoke the help on line for a command.
list	**l**	Lists scheduling objects.
listfolder	**lf**	Lists folders.
lock	**lo**	Locks the access to database objects.
mkfolder	**mf**	Creates a new folder.
modify	**m**	Modifies scheduling objects.
new		Creates a scheduling object using a text file where the object definition is inserted online. If you specify the type of scheduling object you want to define after *new* command, a predefined object definition is written in the text file.
print	**p**	Prints scheduling objects.
redo	**red**	Edits and reruns the previous command.
rmfolder	**rf**	Deletes a folder.
rename	**rn**	Changes the object name.
renamefolder	**rnf**	Renames a folder.
replace	**rep**	Replaces scheduling objects.
system command		Invokes an operating system command.

Table 69. List of composer commands (continued)

Command	Short Name	Description
unlock	**u**	Releases lock on the scheduling object defined in the database.
update	**up**	Updates the attributes settings of the scheduling object in the database.
validate	**val**	Validates the syntax, semantic, and data integrity of an object definition.
version	**v**	Displays the **composer** command-line program banner.

Other HELP topics:

Command	Description
COMMANDS	List of composer commands
SETUPCOMPOSER	Describes how to setup to use composer.
RUNCOMPOSER	How to run composer.
SPECIALCHAR	Describes the wildcards, delimiters and other special characters you can use.

Referential integrity check

HCL Workload Automation automatically performs referential checks to avoid lack of integrity in the object definitions in the database whenever you run commands that create, modify, or delete the definition of a referenced object. These are the checks performed by the product:

- Every time you use a command that creates a new object in the database, HCL Workload Automation checks that:
  - An object of the same type and with the same identifier does not already exist.
  - The objects referenced by this object already exist in the database.
- Every time you run a command that modifies an object definition in the database, HCL Workload Automation checks that:
  - The object definition to be modified exists in the database.
  - The objects referenced by this object exist in the database.
  - To avoid integrity inconsistencies, the object definition does not appear in the definition of an object belonging to the chain of his ancestors.
- Every time you run a command that deletes an object definition in the database, HCL Workload Automation checks that:
  - The object definition to be deleted exists in the database.
  - The object definition to be deleted is not referenced by other objects defined in the database.

In event rules, no referential integrity check is applied to the event section of the rule. In the action section, referential integrity is checked.

Table 70: Object identifiers for each type of object defined in the database on page 450 shows, for each object type, the identifiers that are used to uniquely identify the object in the database when creating or modifying object definitions:

Table 70. Object identifiers for each type of object defined in the database

Object type	Object identifiers
domain	*domainname*
workstation	*workstationname* (checked across workstations and workstation classes)
workstation class	*workstationclassname* (checked across workstations and workstation classes)
calendar	*calendarname*
job definition	*workstationname* and *jobname*
user	*workstationname* and *username*
job stream	*workstationname* and *jobstreamname* and, if defined, *validfrom*
job within a job stream	*workstationname* and *jobstreamname*, *jobname*, and, if defined, *validfrom*
resource	*workstationname* and *resourcename*
prompt	*promptname*
variable table	*variabletablename*
variable	*variabletablename.variablename*
event rule	*eventrulename*
access control list	*securitydomainname*
security domain	*securitydomainname*
security role	*securityrolename*

In general, referential integrity prevents the deletion of objects when they are referenced by other objects in the database. However, in some cases where the deletion of an object (for example a workstation) implies only the update of a referencing object (for example a workstation class that includes it), the deletion might be allowed. Table 71: Object definition update upon deletion of referenced object on page 450 shows all cases when a referenced object can be deleted even if other objects reference it:

Table 71. Object definition update upon deletion of referenced object

Object	References	Upon deletion of the referenced object ...
Internetwork Dependency	Workstation	... remove the dependency from the job or job stream
External Follows Depend	Job Stream	... remove the dependency from the job or job stream
	Job	... remove the dependency from the job or job stream
Internal Dependency	Job	... remove the dependency from the job or job stream
Workstation Class	Workstation	... remove the workstation from the workstation class

Table 72: Referential integrity check when deleting an object from the database on page 451 describes how the product behaves when it is requested to delete an object referenced by another object with using a specific relationship:

Table 72. Referential integrity check when deleting an object from the database

Object to be deleted	Referenced by object	Relationship	Delete rule
domain A	domain B	domain A is parent of domain B	An error specifying the existing relationship is displayed.
	workstation B	workstation B belongs to domain A	An error specifying the existing relationship is displayed.
workstation A	workstation B	workstation A is host for workstation B	An error specifying the existing relationship is displayed.
	job B	job B is defined on workstation A	An error specifying the existing relationship is displayed.
	job stream B	job stream B is defined on workstation A	An error specifying the existing relationship is displayed.
	user B	user B is defined on workstation A	An error specifying the existing relationship is displayed.
	job stream B	workstation A works as network agent for internetwork dependencies set in job stream B	Both workstation A and the internetwork dependency are deleted
	job stream B	job stream B has a file dependency from a file defined on workstation A	Both workstation A and the file dependency are deleted
	job B within job stream B	workstation A works as network agent for internetwork dependencies set in job B	Both workstation A and the internetwork dependency are deleted
	job B within job stream B	job B has a file dependency from a file defined on workstation A	Both workstation A and the file dependency are deleted
	resource B	resource B is defined on workstation A	An error specifying the existing relationship is displayed.
	file B	file B is defined on workstation A	An error specifying the existing relationship is displayed.
	workstation class B	workstation A belongs to workstation class B	Both workstation A and its entry in workstation class B are deleted.
	job B within job stream B	job B contained in job stream B is defined on workstation A	An error specifying the existing relationship is displayed.

Table 72. Referential integrity check when deleting an object from the database (continued)

Object to be deleted	Referenced by object	Relationship	Delete rule
job A	job B	job A is recovery job for job B	An error specifying the existing relationship is displayed.
	job stream B	job A is contained in job stream B	An error specifying the existing relationship is displayed.
	job stream B	job stream B follows job A	job A and the follows dependency in job stream B are deleted.
	job B within job stream B	job B follows job A	job A and the follows dependency in job B are deleted.
	event rule B	job A is in the action definition of event rule B (and does not use variable substitution)	An error specifying the existing relationship is displayed.
calendar A	job stream B	job stream B uses calendar A	An error specifying the existing relationship is displayed.
workstation class A	job B	job B is defined on workstation class A	An error specifying the existing relationship is displayed.
	job stream B	job stream B is defined on workstation class A	An error specifying the existing relationship is displayed.
	resource B	resource B is defined on workstation class A	An error specifying the existing relationship is displayed.
	file B	file B is defined on workstation class A	An error specifying the existing relationship is displayed.
resource A	job stream B	needs dependency defined in job stream B	An error specifying the existing relationship is displayed.
	job B within job stream B	needs dependency defined in job B	An error specifying the existing relationship is displayed.
prompt A	job stream B	prompt dependency defined in job stream B	An error specifying the existing relationship is displayed.
	job B within job stream B	prompt dependency defined in job B	An error specifying the existing relationship is displayed.
variable A	job stream B	variable A is used in job stream B in:	variable A is deleted without checking

Table 72. Referential integrity check when deleting an object from the database (continued)

Object to be deleted	Referenced by object	Relationship	Delete rule
		• in the text of an ad hoc prompt • or in the file name specified in a file dependency	
	job B	variable A is used in job stream B in: • in the text of an ad hoc prompt • or in the file name specified in a file dependency • or in the value specified for streamlogon • or in the value specified for scriptname	variable A is deleted without checking
	prompt B	variable A is used in the text of prompt B	variable A is deleted without checking
variable table A	job stream B	variable table A is referenced in job stream B	variable table A is not deleted
	job B	variable table A is referenced in job B	variable table A is not deleted
	prompt B	variable table A is referenced in the text of prompt B	variable table A is not deleted
job stream A	job stream B	job stream B follows job stream A	job stream A and the follows dependency in job stream B are deleted.
	job B within a job stream B	job B follows job stream A	job stream A and the follows dependency in job B are deleted.
	event rule B	job stream A is in the action definition of event rule B (and does not use variable substitution)	An error specifying the existing relationship is displayed.
security domain A	access control list B	access control list B is defined on security domain A	access control list B is deleted.
security role A	access control list B	security role A is referenced in access control list B	security role A is not deleted.
folder A	workstation B	workstation B nested in folder A	An error specifying the existing relationship is displayed.

Table 72. Referential integrity check when deleting an object from the database (continued)

Object to be deleted	Referenced by object	Relationship	Delete rule
	workstation class B	workstation class B nested in folder A	An error specifying the existing relationship is displayed.
	job B	job B nested in folder A	An error specifying the existing relationship is displayed.
	calendar B	calendar B nested in folder A	An error specifying the existing relationship is displayed.
	folder B	folder B nested in folder A	An error specifying the existing relationship is displayed.
	variable table B	variable table B nested in folder A	An error specifying the existing relationship is displayed.
	prompt B	prompt B nested in folder A	An error specifying the existing relationship is displayed.
	resource B	resource B nested in folder A	An error specifying the existing relationship is displayed.
	run cycle group B	run cycle group B nested in folder A	An error specifying the existing relationship is displayed.
	job stream B	job stream B nested in folder A	An error specifying the existing relationship is displayed.
	event rule B	event rule B nested in folder A	An error specifying the existing relationship is displayed.
	workload application B	workload application B nested in folder A	An error specifying the existing relationship is displayed.
	access control list B (might contain an object definition defined on a specific folder)	access control list B nested in folder A	An error specifying the existing relationship is displayed.

ADD

Adds or updates scheduling objects to the database.

Authorization

You must have ***add*** access to add a new scheduling object. If the object already exists in the database you must have:

- ***modify*** access to the object if the object is not locked.
- ***modify*** and ***unlock*** access to the object if the object is locked by another user.

To **add** security objects, you must have permission for the **modify** action on the object type **file** with attribute **name=security**.

Syntax

{**add** | **a**} *filename* [**;unlock**]

Arguments

filename

Specifies the name of the text file that contains the object definitions. For event rules, *filename* specifies the name of the XML file containing the definitions of the event rules that you want to add (see the topic on event rule definition in the chapter on "Defining objects in the database" in *User's Guide and Reference* for XML reference and see the topic on the composer editor in SETUPCOMPOSER for details about setting up an XML editor).

;unlock

Indicates that the object definitions must be unlocked if locked by the same user in the same session. If you did not lock the object and you use the ;unlock option, when you issue the command you receive an error message and the object is not replaced.

Comments

The text file is validated at the client and, if correct, objects are inserted into the database on the master domain manager. **Composer** transforms object definitions into an XML definition used at the server; otherwise the command is interrupted and an error message is displayed. This does not apply to event rule definitions because they are provided directly in XML format.

With the add command, if an object already exists, you are asked whether or not to replace it. This behavior does not affect existing job definitions inside job streams, and the job definitions are automatically updated without prompting any message. You can use the option **unlock** to update existing objects you previously locked by using only one command. For all new objects inserted, the option is ignored. If you change the name of an object, it is interpreted by **composer** as a new object and will be inserted. A **rename** command is recommended in this case.

The add command checks for loop dependencies inside job streams. For example, if `job1` follows `job2`, and `job2` follows `job1` there is a loop dependency. When a loop dependency inside a job stream is found, an error is displayed.

The add command does not check for loop dependencies between job streams because, depending on the complexity of the scheduling activities, this check might take too long.

Example

Examples

To add the jobs from the file `myjobs`, run the following command:

```
add myjobs
```

To add the job streams from the file `mysked`, run the following command:

```
a mysked
```

To add the workstations, workstation classes, and domains from the file `cpus.src`, run the following command:

```
a cpus.src
```

To add the user definitions from the file `users_nt`, run the following command:

```
add users_nt
```

To add the event rule definitions you edited in a file named `newrules.xml`, run:

```
a newrules.xml
```

See also

From the Dynamic Workload Console you can perform the same tasks as described in:

the *Dynamic Workload Console User's Guide*.

the *Dynamic Workload Console User's Guide*.

- To add workstations, see

  the *Dynamic Workload Console User's Guide*, section about Creating distributed workstations.

  the *Dynamic Workload Console User's Guide*, section about Creating distributed workstations.
- To add event rules, see

  the *Dynamic Workload Console User's Guide*, section about Creating an event rule.

  the *Dynamic Workload Console User's Guide*, section about Creating an event rule.
- To add access control lists, security domains, and security roles, see

  the *Dynamic Workload Console User's Guide*, section about Managing Workload Security.

  the *Dynamic Workload Console User's Guide*, section about Managing Workload Security.
- To add all other objects, see

  the *Dynamic Workload Console User's Guide*, section about Designing your Workload.

  the *Dynamic Workload Console User's Guide*, section about Designing your Workload.
- To add workload application templates, see

the *Dynamic Workload Console User's Guide*, section about Creating a workload application template.

the *Dynamic Workload Console User's Guide*, section about Creating a workload application template.

AUTHENTICATE

Switches to another user credentials while running **composer**.

Authorization

Any user authorized to run **composer** is authorized to issue this command.

Syntax

{**authenticate** | **au**} [**username=***username* **password=***password*]

Arguments

username=*username*

The username of the user you want to switch to.

password=*password*

The password of the user you want to switch to.

Comments

A message is displayed communicating the authentication failure or success. This command is used only in interactive mode.

Example

Examples

To switch to user **tws_user1** with password **mypasswd1** from within the **composer** command-line program, run the following command:

```
au username=tws_user1 password=mypasswd1
```

CHFOLDER

Use to navigate folders in the database. This command changes the working directory or current directory, which is set to root ("/") by default, so that you can use relative folder paths when submitting commands.

Authorization

To **change** folder, you must have ***display*** access to the folder.

Syntax

{**chfolder** | **cf**} *foldername*

Arguments

foldername

Is the name and path of the folder that becomes the current folder.

Comments

When submitting commands involving folders, you can specify either a relative or absolute path to the folder. If no folder is specified, then the command is performed using the current folder which is set to root ("/") by default. If you generally work from a specific folder, then you can use the chfolder command to navigate to folders and sub-folders. The chfolder command changes the working directory or current folder, so that you can use relative folder paths when submitting commands.

There are additional ways you can modify the current folder. The following methods for setting the current folder are also ordered by precedence, for example, the value specified in the customer properties file overrides the current folder setting in the `localopts` file.

CustomParameters

You can pass **chfolder | cf** as a parameter together with a composer command from the composer command line.

For example, to list job streams in the folder named `/TEMPLATE/`, then submit the following from the composer command line:

```
composer -cf /TEMPLATE/ "li js @#@"
```

You can obtain the same result by changing the current folder in composer first, then submitting the command:

```
-cf /TEMPLATE/
```

```
-li js @#@
```

UNISON_CURRENT_FOLDER

Set the value of the current folder using the UNISON_CURRENT_FOLDER environment variable.

Custom properties file

You can use the **CURRENT FOLDER** parameter to set the value of the current folder in the custom properties file that can be passed to composer. The parameters in the custom properties file override the values specified in the `useropts` and `localopts` files. For example, set the parameter as follows:

```
CURRENT FOLDER = /<foldername>
```

See the section about running the composer program in *User's Guide and Reference* for information about the custom properties file.

`useropts`

You can set the current folder in the `useropts` file. The `useropts` file contains values for `localopts` parameters that you can personalize for individual users. Set the current folder as follows:

```
current folder = /<<varname>foldername</varname>>
```

localopts

You can set the current folder in the localopts file by adding `current folder = /<<varname>foldername</varname>>`. For example:

```
-----------------------------------
# Current Folder
#
current folder = /bluefolder
```

The maximum length for the full folder path (that is, the path name including the parent folder and all nested sub-folders) is 800 characters, while each folder name supports a maximum of 256 characters. When you create a nested folder, the parent folder must be existing. See the section about folder definition in *User's Guide and Reference* for details about folder specifications.

Example

Examples

To change the working directory or the current folder from the root folder to the folder path "TEST/APPS/APP1", run:

```
chfolder /TEST/APPS/APP1/
```

TEST/APPS/APP1 is now the current folder path and commands can be submitted on objects in this folder using relative paths.

CONTINUE

Specifies that the next command error is to be ignored.

Authorization

Any user authorized to run composer is authorized to issue this command.

Syntax

{**continue** | **co**}

Comments

This command is useful when multiple commands are entered on the command line or redirected from a file. It instructs **composer** to continue running commands even if the next command, following **continue**, results in an error. This command is not needed when you enter commands interactively because **composer** does not quit on an error.

Example

Examples

If you want the composer to continue with the **print** command if the **delete** command results in an error, run the following command:

```
composer "co&delete cpu=site4&print cpu=@"
```

DELETE

Deletes object definitions in the database.

If the scheduling object is stored in a folder, the command is performed on the folder where the scheduling object definition is stored. If folder is omitted, the default folder ("/") is used.

Authorization

To **delete** scheduling objects, you must have ***delete*** access to the objects being deleted.

To **delete** security objects, you must have permission for the **modify** action on the object type **file** with attribute **name=security**.

Syntax

{**delete** | **de**}
{[**calendars** | **calendar** | **cal**=*[folder/]calname*] |
[**domain** | **dom**]=*domainame*] |
[**eventrule** | **erule** | **er**=*[folder/]eventrulename*] |
[**folder** | **fol**=*foldername*] |
[**parms** | **parm** | **vb**=[*[folder/]tablename*.]*variablename*] |
[**prompts** | **prom**=*[folder/]promptname*] |
[**resources** | **resource** | **res**=[*[folder/]workstationname#*]*[folder/]resourcename*] |
[**runcyclegroup** | **rcg**=*[folder/]runcyclegroupname*] |
[**vartable** | **vt**=*[folder/]tablename*] |
[**wat**=*[folder/]workloadapplicationtemplatename*]
[**cpu**={*[folder/]workstationame* [**;force**] | *[folder/]workstationclassname* [**;force**]| *domainame*}]
[**workstation** | **ws**=*[folder/]workstationame*] [**;force**] |
[**workstationclass** | **wscl**]=*[folder/]workstationclassname* [**;force**] |
[**jobs** | **jobdefinition** | **jd**]=[*[folder/]workstationame#*][*folder/*]*jobname* |
[**sched** | **jobstream** | **js**]= [*[folder/]workstationame#*][*folder/*]*jstreamname*
*[**valid from** date*|***valid to** date* |***valid in** date date]*] |
[**users** | **user**=[*[folder/]workstationame#*]*username*] |
[**accesscontrollist** | **acl for** *securitydomainname*] |
[**securitydomain** | **sdom**=*securitydomainname*] |
[**securityrole** | **srol**=*securityrolename*]}
[**;noask**]

Arguments

calendars | **calendar** | **cal**

If no argument follows, deletes all calendar definitions.

If argument *[folder/]calname* follows, deletes the specified calendar. Wildcard characters are permitted.

domain | dom

If argument *domainname* follows, deletes the specified domain. Wildcard characters are permitted.

eventrule | erule | er

If no argument follows, deletes all event rule definitions.

If argument *[folder/]eventrulename* follows, deletes the specified event rule. Wildcard characters are permitted.

folder

If no argument follows, an error is returned.

If argument *foldername* follows, deletes the specified folder and its sub-folders. Wildcard characters are permitted. The command is performed provided the folder does not contain any scheduling objects, neither in the sub-folders, otherwise an error message is returned.

You can also delete a folder using the dedicated composer command RMFOLDER on page 505.

parms|parm|vb

If no argument follows, deletes all global variable definitions found in the default variable table.

If argument *[folder/]tablename.variablename* follows, deletes the *variablename* variable of the *tablename* table. If *[folder/]tablename* is omitted, **composer** looks for the variable definition in the default variable table. Wildcard characters are permitted on both *[folder/]tablename* and *variablename*. For example:

```
delete parms=@.@
```

Deletes all variables from all tables.

```
delete parms=@
```

Deletes all variables from the default table.

```
delete parms=@.acct@
```

Deletes all the variables whose name starts with `acct` from all the existing tables.



Remember: While you delete a variable, the variable table that contains it is locked. This implies that, while the table is locked, no other user can run any other locking commands on it or on the variables it contains.

prompts | prom

If no argument follows, deletes all prompt definitions.

If argument *[folder/]promptname* follows, deletes the specified prompt. Wildcard characters are permitted.

resources | resource | res

If no argument follows, deletes all resource definitions.

If argument *[folder/]workstationname#[folder/]resourcename* follows, deletes the *[folder/]resourcename* resource of the *[folder/]workstationname* workstation on which the resource is defined. If *[folder/]workstationname* is omitted, the default is the workstation on which **composer** is running. Wildcard characters are permitted.

runcyclegroup | rcg

If no argument follows, deletes all run cycle group definitions.

If argument *[folder/]runcyclegroupname* follows, deletes the specified run cycle group. Wildcard characters are permitted.

vartable | vt

If no argument follows, deletes all variable table definitions.

If argument *[folder/]tablename* variable table follows, deletes the specified variable table. Wildcard characters are permitted.

wat

If no argument follows, deletes all workload application template definitions.

If argument *[folder/]workloadapplicationtemplate* follows, deletes the specified workload application template. Wildcard characters are permitted.

cpu

Deletes workstations, workstation classes, or domains.

workstation

The name of the workstation. Wildcard characters are permitted. If you specify the **force** argument, the workstation definition is removed from the HCL Workload Automation database.

workstationclass

The name of the workstation class. Wildcard characters are permitted. If you specify the **force** argument, the workstationclass definition is removed from the HCL Workload Automation database.

domain

The name of the domain. Wildcard characters are permitted.

workstation | ws

If argument *workstationname* follows, deletes the specified workstation. Wildcard characters are permitted. If you specify the **force** argument, the workstation definition is removed from the HCL Workload Automation database.

workstationclass | wscl

If argument *workstationclassname* follows, deletes the specified workstation class. Wildcard characters are permitted. If you specify the **force** argument, the workstation class definition is removed from the HCL Workload Automation database.

jobs | jobdefinition | jd

If argument *[folder/]workstationame#[folder/]jobname* follows, deletes the *[folder/]jobname* job stored on the *[folder/]workstationame* workstation on which the job runs. If *[folder/]workstationame* is omitted, the default is the workstation on which **composer** is running. Wildcard characters are permitted.

sched | jobstream | js

If argument *[folder/]workstationame#[folder/]jstreamname* follows, deletes the *[folder/]jstreamname* job stream on the *[folder/]workstationame* workstation on which the job stream is defined. If *[folder/]workstationame* is omitted, the default is the workstation on which **composer** is running. Wildcard characters are permitted.

valid from

date Restricts the selection to job streams that have a *valid from* date equal to the indicated value. The format is *mm/dd/yyyy*.

valid to

date Restricts the selection to job streams that have a *valid to* date equal to the indicated value. The format is *mm/dd/yyyy*.

valid in

date date The time frame during which the job stream can run. The format is *mm/dd/yyyy - mm/dd/yyyy*. One of the two dates can be represented by @.

users | user

If argument *[folder/]workstationame#username* follows, deletes the *username* user of the *[folder/]workstationame* workstation on which the user is defined. If *[folder/]workstationame* is omitted, the default is the workstation on which **composer** is running. Wildcard characters are permitted. The password field is not copied for security reasons.

accesscontrollist | acl

If no *securitydomainname* argument follows, delete access control list definitions for all the security domains.

If argument *securitydomainname* follows, delete the access control list definitions for the *securitydomainname* security domain. Wildcard characters are permitted for *securitydomainname*.

securitydomain | sdom

If no *securitydomainname* argument follows, delete all the security domains definitions.

If argument *securitydomainname* follows, delete the *securitydomainname* security domain definition. Wildcard characters are permitted.

securityrole | srol

If no *securityrolename* argument follows, delete all the security roles definitions.

If argument *securityrolename* follows, delete the *securityrolename* security role definition. Wildcard characters are permitted.

;noask

Specifies not to prompt for confirmation before taking action on each qualifying object.

Comments

If you use wildcard characters to specify a set of definitions, **composer** requires confirmation before deleting each matching definition. A confirmation is required before deleting each matching definition if you do not specify the **noask** option.

To delete an object, it must not have been locked by another user. If some matching objects are locked during the command processing, an error message with the list of these objects is shown to the user.

Note: Users can delete objects they have locked without unlocking them first.

Deleting a dynamic agent stored in a folder from the database might lead to inconsistent behavior. To work around this problem, run JnextPlan after deleting an agent.

Example

Examples

To delete `job3` stored in the `myfolder` folder that is launched on workstation `site3` stored in the test folder, run the following command:

```
delete jobs=test/site3#myfolder/job3
```

To delete all workstations with names starting with `ux`, run the following command:

```
de cpu=ux@
```

To delete all job streams stored in the folder path `test/redfolder` on all workstations, run the following command:

```
de sched=@#test/redfolder/@
```

To delete all the event rules named from `rulejs320` to `rulejs329`, run the following command:

```
de erule=rulejs32?
```

See also

From the Dynamic Workload Console you can perform the same tasks as described in:

the *Dynamic Workload Console User's Guide*.

the *Dynamic Workload Console User's Guide*.

- To delete event rules, see

  the *Dynamic Workload Console User's Guide*, section about Editing an event rule.

  the *Dynamic Workload Console User's Guide*, section about Editing an event rule.

- To delete access control lists, security domains, and security roles, see

  the *Dynamic Workload Console User's Guide*, section about Managing Workload Security.

  the *Dynamic Workload Console User's Guide*, section about Managing Workload Security.

DISPLAY

Displays the details of one or more object definitions of the same type stored in the database. The entire definition of the object is displayed.

Authorization

To **display** scheduling objects, you must have ***display*** access to the object being displayed. If you want to use the **full** keyword you must have also the ***display*** access to the jobs contained in the job stream definition. If you do not have the required access, composer is unable to find the objects.

To **display** security objects, you must have permission for the **display** action on the object type **file** with attribute **name=security**.

Syntax

{**display** | **di**}
{[**calendars** | **calendar** | **cal**=*[folder/]calname*] |
[**eventrule** | **erule** | **er**=*[folder/]eventrulename*] |
[**folder**|**fol**=*foldername* |]
[**parms** | **parm** | **vb**=*variablename*.]*variablename*] |
[**vartable** | **vt**=*[folder/]tablename*] |
[**prompts** | **prom**=*[folder/]promptname*] |
[**resources** | **resource** | **res**=[*[folder/]workstationname#*]*[folder/]resourcename*] |
[**runcyclegroup** | **rcg**=*[folder/]runcyclegroupname*] |
[**cpu**={*[folder/]workstationname* | *[folder/]workstationclassname* | *domainame*}]
[**wat**=*[folder/]workloadapplicationtemplatename*]
[**workstation** | **ws**=*[folder/]workstationname*] |
[**workstationclass** | **wscl**=*[folder/]workstationclassname*] |
[**domain** | **dom**=*domainame*] |
[**jobs** | **jobdefinition** | **jd**=[*[folder/]workstationname#*][*folder/]jobname*] |
[**sched** | **jobstream** | **js**= [*[folder/]workstationname#*][*folder/]jstreamname*
  [**valid from** *date*|**valid to** *date* |**valid in** *date date*]
  [**;full**]] |
[**users** | **user**=[*[folder/]workstationname#*]*username*] |
[**accesscontrollist** | **acl for** *securitydomainname*] |
[**securitydomain** | **sdom**=*securitydomainname*] |
[**securityrole** | **srol**=*securityrolename*]}
[**;offline**]

Arguments

calendars | calendar | cal

If no argument follows, displays all calendar definitions.

If argument *[folder/]calname* follows, displays the *[folder/]calname* calendar. Wildcard characters are permitted.

eventrule | erule | er

If argument *[folder/]eventrulename* follows, displays the *[folder/]eventrulename* event rule. Wildcard characters are permitted.

folder | fol

If no argument follows, displays all folder definitions.

If argument *foldername* follows, displays the *foldername* folders. Wildcard characters are permitted.

parms | parm | vb

If no argument follows, displays all global variable definitions found in the default variable table.

If argument *[folder/]tablename.variablename* follows, displays the *variablename* variable of the specified table. If *[folder/]tablename* variable table is omitted, **composer** looks for the variable definition in the default variable table. Wildcard characters can be used on both *[folder/]tablename* variable table and *variablename* variable. For example:

```
display parms=@.@
```

Displays all variables on all tables.

```
display parms=@
```

Displays all variables on the default table.

```
display parms=@.acct@
```

Displays all the variables whose name starts with `acct` on all the existing tables.

vartable | vt

If no argument follows, displays all variable table definitions.

If argument *[folder/]tablename* variable table follows, displays the *[folder/]tablename* variable table. Wildcard characters are permitted.

prompts | prom

If no argument follows, displays all prompt definitions.

If argument *[folder/]promptname* follows, displays the *[folder/]promptname* prompt. Wildcard characters are permitted.

resources | resource | res

If no argument follows, displays all resource definitions.

If argument *[folder/]workstationname#[folder/]resourcename* follows, displays the *[folder/]resourcename* resource of the *[folder/]workstationname* workstation on which the resource is defined. If *[folder/]workstationname* is omitted, the default is the workstation on which **composer** is running. Wildcard characters are permitted.

runcyclegroup | rcg

If no argument follows, displays all run cycle group definitions.

If argument *[folder/]runcyclegroupname* follows, displays the *[folder/]runcyclegroupname* run cycle group. Wildcard characters are permitted.

cpu

Displays workstations, workstation classes, or domains.

[folder/]workstation

The name of the workstation. Wildcard characters are permitted.

[folder/]workstationclass

The name of the workstation class. Wildcard characters are permitted.

domain

The name of the domain. Wildcard characters are permitted.

wat

If no argument follows, displays all workload application template definitions.

If argument *[folder/]workloadapplicationtemplate* follows, displays the specified workload application template. Wildcard characters are permitted.

workstation | ws

If no argument follows, displays all workstation definitions.

If argument *[folder/]workstationname* follows, displays the specified workstation. Wildcard characters are permitted.

domain | dom

If no argument follows, displays all domain definitions.

If argument *domainname* follows, displays the *domainname* domain. Wildcard characters are permitted.

workstationclass | wscl

If no argument follows, displays all workstation class definitions.

If argument *[folder/]workstationclassname* follows, displays the specified workstation class. Wildcard characters are permitted.

jobs | jobdefinition | jd

If no argument follows, displays all job definitions.

If argument *workstationname#/folder/jobname* follows, displays the folder where the *jobname* is stored, and the *workstationname* workstation *workstationname* on which the job runs. If *folder* is omitted, the default folder ("/") is used.

You can use the **com.hcl.ws.showRootFolder** property in the `TWSConfig.properties` file to manage how jobs located in the root folder are displayed. The job name can either be displayed indicating the root folder (`/MYJOB1`), or without indicating it (`MYJOB1`). By default, this property is set to `NO` which specifies to display the job without indicating the root folder. Set the property to `YES` to display job names preceded by root ("/") . The `TWSConfig.properties` file is located in the path:

On Windows operating systems

TWA_home\usr\servers\engineServer\resources\properties

On UNIX operating systems

TWA_DATA_DIR/usr/servers/engineServer/resources/properties

If *[folder/]workstationname* is omitted, the default is the workstation on which **composer** is running. Wildcard characters are permitted.

sched | jobstream | js

If no argument follows, displays all job stream definitions in the current folder.

If argument *workstationname#/folder/jstreamname* follows, displays the folder where the job stream definition *jstreamname* is stored and the workstation *workstationname* on which the job stream is defined. If *folder* is omitted, the default folder ("/") is used.

You can use the **com.hcl.ws.showRootFolder** property in the `TWSConfig.properties` file to manage how job streams located in the root folder are displayed. The job stream name can either be displayed indicating the root folder (`/MYJOBSTREAM1`), or without indicating it (`MYJOBSTREAM1`). By default, this property is set to `NO` which specifies to display the job stream name without indicating the root folder. Set the property to `YES` to display job stream names preceded by root ("/"). The `TWSConfig.properties` file is located in the path:

On Windows operating systems

TWA_home\usr\servers\engineServer\resources\properties

On UNIX operating systems

TWA_DATA_DIR/usr/servers/engineServer/resources/properties

If *[folder/]workstationname* is omitted, the default is the workstation on which **composer** is running. Wildcard characters are permitted.

valid from

date Restricts the selection to job streams that have a *valid from* date equal to the indicated value. The format is *mm/dd/yyyy*.

valid to

date Restricts the selection to job streams that have a *valid to* date equal to the indicated value. The format is *mm/dd/yyyy*.

valid in

date date The time frame during which the job stream can run. The format is *mm/dd/yyyy* - *mm/dd/yyyy*. One of the two dates can be represented by @.

full

Displays also all job definitions contained in the job stream.

users | user

If no argument follows, displays all user definitions.

If argument *[folder/]workstationame#username* follows, displays the *username* user of the *[folder/]workstationame* workstation on which the user is defined. If *[folder/]workstationame* is omitted, the default is the workstation on which **composer** is running. Wildcard characters are permitted.

accesscontrollist | acl

If no *securitydomainname* argument follows, displays access control list definitions for all the security domains.

If argument *securitydomainname* follows, displays the access control list definitions for the *securitydomainname* security domain. Wildcard characters are permitted for *securitydomainname* .

securitydomain | sdom

If no *securitydomainname* argument follows, displays all the security domains definitions.

If argument *securitydomainname* follows, displays the *securitydomainname* security domain definition. Wildcard characters are permitted for *securitydomainname* .

securityrole | srol

If no *securityrolename* argument follows, displays all the security roles definitions.

If argument *securityrolename* follows, displays the *securityrolename* security role definition. Wildcard characters are permitted for *securityrolename* .

;offline

Sends the output of the command to the **composer** output device. For information about this device, see the topic about offline output in "SETUPCOMPOSER".

Results

The **display** command returns you the following information about the object to be displayed:

- a summary row containing information about the selected object
- the selected object definition

Depending on the value set in the ***MAESTROCOLUMNS*** local variable the summary row shows different sets of information about the selected object.

A table showing the output formats for displaying scheduling objects can be found in the topic about the "display" command in *User's Guide and Reference*.

See the topic on the offline output in "SETUPCOMPOSER" for more information on how to set ***MAESTROCOLUMNS*.**

Example

Examples

To display all calendars, run the following command:

```
display calendars=@
```

this is a sample output:

```
Calendar Name     Updated On  Locked By
----------------  ----------  --------------------
HOLIDAYS          12/31/2022  twsuser
HOLIDAYS
  01/01/2023 02/15/2023 05/31/2023


Calendar Name     Updated On  Locked By
----------------  ----------  --------------------
MONTHEND          01/01/2023  -

MONTHEND
  "Month end dates 1st half 2023"
  01/31/2023 02/28/2023 03/31/2023 04/30/2023 05/31/2023 06/30/2023

Calendar Name     Updated On  Locked By
----------------  ----------  --------------------
PAYDAYS           01/02/2023  -

PAYDAYS
  01/15/2023 02/15/2023 03/15/2023 04/15/2023 05/14/2023 06/15/2023
```

To print the output of the display command on all job streams that are launched on workstation `site2`, stored in folder myfolder run the following command:

```
di sched=myfolder/site2#@;offline
```

See also

From the Dynamic Workload Console you can perform the same tasks as described in:

the *Dynamic Workload Console User's Guide*.

the *Dynamic Workload Console User's Guide*.

- To display event rules, see

  the *Dynamic Workload Console User's Guide*, section about Editing an event rule.

  the *Dynamic Workload Console User's Guide*, section about Editing an event rule.
- To display access control lists, security domains, and security roles, see

  the *Dynamic Workload Console User's Guide*, section about Managing Workload Security.

  the *Dynamic Workload Console User's Guide*, section about Managing Workload Security.

EDIT

Edits a file.

Authorization

Any user authorized to run composer is authorized to issue this command.

Syntax

{**edit** | **ed**} *filename*

Arguments

filename

The name of the file to be edited.

Comments

An editor is started and the specified file is opened for editing. See the topic on the composer editor in SETUPCOMPOSER for more information.

Example

Examples

To open the file `mytemp` for editing, run the following command:

```
edit mytemp
```

To open the file `resfile` for editing, run the following command:

```
ed resfile
```

EXIT

Exits the **composer** command line program.

Authorization

Any user authorized to run composer is authorized to issue this command.

Syntax

{**exit** | **e**}

Comments

When you are running the **composer** command line program in help mode, this command returns **composer** to command input mode.

Example

Examples

To exit the **composer** command line program, run the following command:

```
exit
```

or:

```
e
```

EXTRACT

Creates a text file containing object definitions extracted from the database.

Authorization

To **extract** scheduling object, you must have ***display*** access to the objects being copied and, if you want to use the **;lock** keyword, also the ***modify*** access. If you do not have the required access composer is unable to find the objects.

To **extract** security objects, you must have permission for the **modify** action on the object type **file** with attribute **name=security**.

To extract users, specify the passphrase to be used for encrypting the user password in one of the following ways:

- set the **WA_EXPORT_PWD** environment variable with your password in the shell where you plan to run the command. When you run the `export WA_EXPORT_PWD` command, this makes the password accessible to any command run from that shell session, allowing subsequent commands to run without prompting for it again.
- specify the password at the command prompt

Syntax

{**create** | **cr** | **extract** | **ext**} *filename* **from**
{[**calendars** | **calendar** | **cal**=*[folder/]calname*] |
[**eventrule** | **erule** | **er**=*[folder/]eventrulename*] |
[**parms** | **parm** | **vb**=[*[folder/]tablename*.]*variablename*] |

[**vartable** | **vt**=*[folder/]tablename*] |
[**prompts** | **prom**=*[folder/]promptname*] |
[**resources** | **resource** | **res**=[*[folder/]workstationname#*]*[folder/]resourcename*] |
[**runcyclegroup** | **rcg**=*[folder/]runcyclegroupname*] |
[**cpu**={*[folder/]workstationname* | *[folder/]workstationclassname* | *domainame*}] |
[**workstation** | **ws**=*[folder/]workstationname*] |
[**workstationclass** | **wscl**=*[folder/]workstationclassname*] |
[**domain** | **dom**=*domainame*] |
[**jobs** | **jobdefinition** | **jd**=[*[folder/]workstationname#*][*folder/*]*jobname*] |
[**sched** | **jobstream** | **js**= [*[folder/]workstationname#*][*folder/*]*jstreamname*
*[**valid from** date*|***valid to** date* |***valid in** date date]*
*[**;full**]*] |
[**users** | **user**=[*[folder/]workstationname#*]*username* [;password]] |
[**accesscontrollist** | **acl for** *securitydomainname*] |
[**securitydomain** | **sdom**=*securitydomainname*] |
[**securityrole** | **srol**=*securityrolename*]}
[**;lock**]

Arguments

filename

Specifies the name of the file to contain the object definitions.

calendars | **calendar** | **cal**

If no argument follows, copies all calendar definitions into the file.

If argument *[folder/]calname* follows, copies the *calname* calendar into the file. Wildcard characters are permitted.

eventrule | **erule** | **er**

If no argument follows, copies all event rule definitions into the XML file.

If argument *[folder/]eventrulename* follows, copies the *eventrulename* event rule into the file. Wildcard characters are permitted.

parms | **parm** | **vb**

If no argument follows, copies all global variable definitions found in the default variable table into the file.

If argument *[folder/]tablename.variablename* follows, copies the *variablename* variable of the specified *tablename* variable table into the file. If the *tablename* variable table is omitted, **composer** looks for the variable definition in the default variable table. Wildcard characters are permitted on both *tablename* variable table and *variablename* variable.

For example:

```
create parmfile from parms=@.@
```

Copies all variables from all tables.

```
create parmfile from parms=@
```

Copies all variables from the default table.

```
create parmfile from parms=@.acct@
```

Copies all the variables whose name starts with `acct` from all the existing tables.

Remember: Using the **;lock** option on a variable locks the variable table that contains it. This implies that, while the table is locked, no other user can run any other locking commands on it or on the variables it contains.

vartable | vt

If no argument follows, copies all variable table definitions into the file.

If argument *[folder/]tablename* variable table follows, copies the *tablename* variable table into the file. Wildcard characters are permitted.

prompts | prom

If no argument follows, copies all prompt definitions into the file.

If argument *[folder/]promptname* follows, copies the *promptname* prompt into the file. Wildcard characters are permitted.

resources | resource | res

If no argument follows, copies all resource definitions into the file.

If argument *[folder/]workstationname#resourcename* follows, copies the *resourcename* resource of the *workstationname* workstation in the specified folder on which the resource is defined. If *workstationname* is omitted, the default is the workstation on which **composer** is running. Wildcard characters are permitted.

runcyclegroup | rcg

If no argument follows, copies all run cycle group definitions into the file.

If argument *[folder/]runcyclegroupname* follows, copies the *runcyclegroupname* run cycle group into the file. Wildcard characters are permitted.

cpu

Copies workstations, workstation classes, or domains into the file.

[folder/]workstation

The name of the workstation. Wildcard characters are permitted.

[folder/]workstationclass

The name of the workstation class. Wildcard characters are permitted.

domain

The name of the domain. Wildcard characters are permitted.

workstation | ws

If no argument follows, copies all workstation definitions into the file.

If argument *[folder/]workstationname* follows, copies the *workstationname* workstation. Wildcard characters are permitted.

domain | dom

If no argument follows, copies all domain definitions into the file.

If argument *domainname* follows, copies the *domainname* domain into the file. Wildcard characters are permitted.

workstationclass | wscl

If no argument follows, copies all workstation class definitions into the file.

If argument *[folder/]workstationclassname* follows, copies the *workstationclassname* workstation class. Wildcard characters are permitted.

jobs | jobdefinition | jd

If no argument follows, copies all job definitions into the file.

If argument *workstationame#[folder/]jobname* follows, copies the *[folder/]jobname* job, optionally defined in the folder named [*folder/*] of the *workstationame* workstation on which the job runs into the file. If *folder* is omitted, the default folder ("/") is used.

You can use the **com.hcl.ws.showRootFolder** property in the `TWSConfig.properties` file to manage how jobs located in the root folder are displayed. The job name can either be displayed indicating the root folder (`/MYJOB1`), or without indicating it (`MYJOB1`). By default, this property is set to `NO` which specifies to display the job without indicating the root folder. Set the property to `YES` to display job names preceded by root ("/") . The `TWSConfig.properties` file is located in the path:

On Windows operating systems

`TWA_home\usr\servers\engineServer\resources\properties`

On UNIX operating systems

TWA_DATA_DIR`/usr/servers/engineServer/resources/properties`

If *workstationame* is omitted, the default is the workstation on which **composer** is running. Wildcard characters are permitted for *workstationame*, [*folder/*], and *jobname*.

sched | jobstream | js

If no argument follows, copies all job stream definitions into the file.

If argument *workstationame#[folder/]jstreamname* follows, copies the *jstreamname* job stream, optionally defined in the folder named [*folder/*], of the *workstationame* workstation on which the job stream is defined into the file. If *folder* is omitted, the default folder ("/") is used.

You can use the **com.hcl.ws.showRootFolder** property in the `TWSConfig.properties` file to manage how job streams located in the root folder are displayed. The job stream name can either be displayed indicating the root folder (`/MYJOBSTREAM1`), or without indicating it (`MYJOBSTREAM1`). By default, this property is set to `NO` which specifies to display the job stream name without indicating the root folder. Set the property to `YES` to display job stream names preceded by root ("/"). The `TWSConfig.properties` file is located in the path:

On Windows operating systems

TWA_home\usr\servers\engineServer\resources\properties

On UNIX operating systems

TWA_DATA_DIR/usr/servers/engineServer/resources/properties

If *workstationame* is omitted, the default is the workstation on which **composer** is running. Wildcard characters are permitted for *workstationame*, [*folder/*], and *jstreamname*.

valid from

date Restricts the selection to job streams that have a *valid from* date equal to the indicated value. The format is *mm/dd/yyyy*.

valid to

date Restricts the selection to job streams that have a *valid to* date equal to the indicated value. The format is *mm/dd/yyyy*.

valid in

date date The time frame during which the job stream can run. The format is *mm/dd/yyyy - mm/dd/yyyy*. One of the two dates can be represented by @.

full

Copies also all job definitions contained in the job stream.

users | **user**

If no argument follows, copies all user definitions into the file.

If argument *[folder/]workstationame#username* follows, copies the *username* user of the *workstationame* workstation in the specified folder on which the user is defined into the file. If *workstationame* is omitted, the default is the workstation on which **composer** is running. Wildcard characters are permitted.

If you do not add the **;password** option, the password defined for the user is saved in the output file as a sequence of 10 asterisks (*) and cannot be reused.

If you do add the **;password** option, the password defined for the user is encrypted and saved in the output file. It can thus be re-imported and used again.

accesscontrollist | acl

If no *securitydomainname* argument follows, copies the access control list definitions for all the security domains into the file.

If argument *securitydomainname* follows, copies the access control list definitions for the *securitydomainname* security domain into the file. Wildcard characters are permitted for *securitydomainname* .

securitydomain | sdom

If no *securitydomainname* argument follows, copies all the security domain definitions into the file.

If argument *securitydomainname* follows, copies the *securitydomainname* security domain definition into the file. Wildcard characters are permitted for *securitydomainname* .

securityrole | srol

If no *securityrolename* argument follows, copies all the security role definitions into the file.

If argument *securityrolename* follows, copies the *securityrolename* security role definition into the file. Wildcard characters are permitted for *securityrolename* .

;lock

Specifies to keep locked the selected object.

Comments

You can use this command to create a file containing parameter definitions to be imported into the parameter database defined locally on a workstation. For more information on how to import parameter definitions locally, refer to the "parms" utility documented in the chapter "Utility commands" in *User's Guide and Reference*.

You can invoke the command with the old name “create” or the new name “extract”. Without the **lock** option, database locking is not checked and all matching objects are extracted to the file. After you create a file, you can use the **edit** command to make changes to the file and the **add** or **replace** command to add or update the database.

You can specify with the **lock** option, if the objects that respond to the selected criteria, must remain locked by the user in the database. If **composer**, during the extraction, find some of these objects already locked by someone else, these objects are not inserted into the file and a message to *stdout* is presented for each locked object.

Example

Examples

To create a file named `caltemp` containing all calendars, run the following command:

```
create caltemp from calendars=@
```

To create a file named `stemp` containing all job streams defined on the workstation where **composer** runs, run the following command:

```
cr stemp from jobstream=@
```

To create a file named `alljobs.txt` containing all job definitions, run the following command:

```
extract alljobs.txt from jd=@#@
```

To create a file named `allrules.xml` containing all event rule definitions, run the following command:

```
ext allrules.xml from erule=@
```

To create a file named `dbmainadm.txt` with the definition of user `princeps` of workstation `dbserv349`, including the encrypted password, run:

```
composer extract c:\dbmainadm.txt from user=dbserv349#princeps;password
```

The contents of file `dbmainadm.txt` will be:

```
USERNAME princeps
  PASSWORD "ENCRYPT:EIu7PP+gvS8="
END
```

To create workstation `Austin` in folder `Texas`, run the command:

```
create ws Texas/Austin
```

In the following two examples, you can see how exporting the `WA_EXPORT_PWD` environment variable, ensures you do not have to specify the password again when running subsequent commands:

```
export WA_EXPORT_PWD=<password>
composer "extract /tmp/users.def users=@#@;password"
```

Result: The extract command does not ask for the password, and the specified user is extracted with encrypted password, as requested.

```
export WA_EXPORT_PWD=<password>
composer "extract /tmp/users.def users=@#@"
```

Result: The extract command does not ask for the password, and the specified user is extracted with the password in clrear format, as requested.

HELP

Displays the on-line help for a command or displays the list of commands that can be issued from **composer**. Not available in Windows®.

Authorization

Any user authorized to run composer is authorized to issue this command.

Syntax

{**help** | **h**} {*command*|*keyword*}

Arguments

command

Specifies the name of a **composer** or system command. For **composer** commands, enter the full command name; abbreviations and short forms are not supported.

keyword

You can also enter the following keywords:

COMMANDS

Lists all composer commands.

RUNCONPOSER

How to run composer.

SETUPCOMPOSER

Describes how to setup to use composer.

SPECIALCHAR

Describes the wildcards, delimiters and other special characters you can use.

Example

Examples

To display a list of all **composer** commands, run the following command:

```
help commands
```

To display information about the **add** command, run the following command:

```
help add
```

To display information about the special characters you can use, run the following command:

```
h specialchar
```

LIST

Lists, or prints summary information about objects defined in the HCL Workload Automation database. List provides you with the list of objects names with their attributes. Print sends the list of objects names with their attributes to the device or file specified in the ***MAESTROLP*** local variable. The print command can be used to send the output to a local printer, if the ***MAESTROLP*** variable is set accordingly.

Authorization

If the ***enListSecChk*** global option is set to **yes** on the master domain manager, then to list or print an object you must have either ***list*** access, or ***list*** and ***display*** access.

To **list** security objects, you must have permission for the **display** action on the object type **file** with attribute **name=security**.

Syntax

{**list** | **l**}
{[**calendars** | **calendar** | **cal**=*[folder/]calname*] |
[**eventrule** | **erule** | **er**=*[folder/]eventrulename*] |
folder|**fol**=*foldername* |
[**parms** | **parm** | **vb**=[*[folder/]tablename*.]*variablename*] |
[**vartable** | **vt**=*[folder/]tablename*] |
[**prompts** | **prom**=*[folder/]promptname*] |
[**resources** | **resource** | **res**=[*[folder/]workstationname#*]*resourcename*] |
[**runcyclegroup** | **rcg**=*[folder/]runcyclegroupname*] |
[**cpu**={*[folder/]workstationname* | *[folder/]workstationclassname* | *domainame*}]
[**wat**=*workloadapplicationtemplatename*]
[**workstation** | **ws**=*[folder/]workstationame*] |
[**workstationclass** | **wscl**=*[folder/]workstationclassname*] |
[**domain** | **dom**=*domainame*] |
[**jobs** | **jobdefinition** | **jd**=[*[folder/]workstationame#*][*folder/*]*jobname*] |
[**sched** |**jobstream** | **js**= [*[folder/]workstationame#*][*folder/*]*jstreamname*
*[**valid from** date*|
valid to** date* |valid in** date date]* |
[**users** | **user**=[*[folder/]workstationame#*]*username*] |
[**accesscontrollist** | **acl for** *securitydomainname*] |
[**securitydomain** | **sdom**=*securitydomainname*] |
[**securityrole** | **srol**=*securityrolename*]}
[**;offline**]
[**;showid**]

Arguments

calendars | **calendar** | **cal**

If no argument follows, lists or prints all calendar definitions.

If argument *[folder/]calname* follows, lists or prints the *calname* calendar. Wildcard characters are permitted.

eventrule | **erule** | **er**

If no argument follows, lists or prints all event rule definitions.

If argument *[folder/]eventrulename* follows, lists or prints the *eventrulename* event rule. Wildcard characters are permitted.

folder | **fol**

If no argument follows, lists or prints the current folder.

If argument *foldername* follows, lists or prints the *foldername* folders. Wildcard characters are permitted.

You can also use the dedicated LISTFOLDER on page 488 composer command to list folders.

parms | parm | vb

If no argument follows, lists or prints all global variable definitions found in the default variable table.

If argument *[folder/]tablename.variablename* follows, lists or prints the *variablename* variable of the *tablename* table. If *tablename* is omitted, **composer** looks for the variable definition in the default variable table. Wildcard characters can be used on both *tablename* and *variablename*. For example:

```
list parms=@.@
```

Lists all variables on all tables.

```
list parms=@
```

Lists all variables on the default table.

```
list parms=@.acct@
```

Lists all the variables whose name starts with `acct` on all the existing tables.

vartable | vt

If no argument follows, lists or prints all variable table definitions.

If argument *[folder/]tablename* variable table follows, lists or prints the *tablename* variable table. Wildcard characters are permitted.

prompts | prom

If no argument follows, lists or prints all prompt definitions.

If argument *[folder/]promptname* follows, lists or prints the *promptname* prompt. Wildcard characters are permitted.

resources | resource | res

If no argument follows, lists or prints all resource definitions.

If argument *[folder/]workstationname#[folder/]resourcename* follows, lists or prints the *resourcename* resource of the *workstationname* workstation in the specified folder on which the resource is defined. If *workstationname* is omitted, the default is the workstation on which **composer** is running. Wildcard characters are permitted.

runcyclegroup | rcg

If no argument follows, lists or prints all run cycle groups.

If argument *[folder/]runcyclegroupname* follows, lists or prints the *runcyclegroupname* run cycle group. Wildcard characters are permitted.

[folder/]workstation

The name of the workstation. Wildcard characters are permitted.

[folder/]workstationclass

The name of the workstation class. Wildcard characters are permitted.

domain

The name of the domain. Wildcard characters are permitted.

wat

If no argument follows, lists or prints all workload application template definitions.

If argument *[folder/]workloadapplicationtemplate* follows, lists or prints the specified workload application template. Wildcard characters are permitted.

workstation | ws

If no argument follows, lists or prints all workstation definitions.

If argument *[folder/]workstationname* follows, lists or prints the *workstationname* workstation. Wildcard characters are permitted.

domain | dom

If no argument follows, lists or prints all domain definitions.

If argument *domainname* follows, lists or prints the *domainname* domain. Wildcard characters are permitted.

workstationclass | wscl

If no argument follows, lists or prints all workstation class definitions.

If argument *[folder/]workstationclassname* follows, lists or prints the *workstationclassname* workstation class. Wildcard characters are permitted.

[*folder/*]

The folder where the job or job stream definition is stored. You can specify folder names using absolute or relative paths. If you want to submit the command from a precise folder, then you can use the chfolder command to navigate folders and sub-folders. The chfolder command changes the working directory, which is set to root ("/") by default, so that you can use relative folder paths when submitting commands. See CHFOLDER on page 457 for more information about changing folder paths.

jobs | jobdefinition | jd

If no argument follows, lists or prints all job definitions.

If argument *workstationame#[folder/]jobname* follows, lists or prints the *jobname* job stored in the folder named [*folder/*] of the *workstationame* workstation on which the job runs. If *workstationame* is omitted, the default is the workstation on which **composer** is running. Wildcard characters are permitted for *workstationame*, [*folder/*], and *jobname*.

sched | jobstream | js

If no argument follows, lists or prints all job stream definitions.

If argument *workstationame#[folder/]jstreamname* follows, lists or prints the *jstreamname* job stream stored in the folder named [*folder/*] of the *workstationame* workstation on which the job stream is defined. If

workstationame is omitted, the default is the workstation on which **composer** is running. Wildcard characters are permitted for *workstationame*, [*folder/*], and *jstreamname*.

valid from

date Restricts the selection to job streams that have a *valid from* date equal to the indicated value. The format is *mm/dd/yyyy*.

valid to

date Restricts the selection to job streams that have a *valid to* date equal to the indicated value. The format is *mm/dd/yyyy*.

valid in

date date The time frame during which the job stream can run. The format is *mm/dd/yyyy - mm/dd/yyyy*. One of the two dates can be represented by @.

users | user

If no argument follows, lists or prints all user definitions.

If argument *[folder/]workstationame#username* follows, lists or prints the *username* user of the *workstationame* workstation in the specified folder on which the user is defined. If *workstationame* is omitted, the default is the workstation on which **composer** is running. Wildcard characters are permitted.

Note: If you are listing windows users in the UPN format `username@internet_domain` , insert the escape character '\' before the '@' character in the `username@internet_domain` value. For example if you are listing the `administrator@bvt.com` user, run the following command:

```
list users=administrator\@bvt.com
```

accesscontrollist | acl

If no *securitydomainname* argument follows, lists or prints the access control list definitions for all the security domains.

If argument *securitydomainname* follows, lists or prints the access control list definitions for the *securitydomainname* security domain. Wildcard characters are permitted for *securitydomainname* .

securitydomain | sdom

If no *securitydomainname* argument follows, lists or prints all the security domain definitions.

If argument *securitydomainname* follows, lists or prints the *securitydomainname* security domain definition. Wildcard characters are permitted for *securitydomainname* .

securityrole | srol

If no *securityrolename* argument follows, lists or prints all the security role definitions.

If argument *securityrolename* follows, lists or prints the *securityrolename* security role definition. Wildcard characters are permitted for *securityrolename* .

;offline

Sends the output of the command to the **composer** output device. For information about this device, see the topic on UNIX variables in "SETUPCOMPOSER". The **list ..... ;offline** command is equivalent to the **print** command.

showid

Displays a unique identifier that identifies a workstation, resource or prompt. These objects are no longer identified in the plan solely by their names, but also by the folder in which they are defined. The name and folder association is mapped to a unique identifier. For example, for workstations, the **this_cpu** option is the unique identifier of the workstation in the `localopts` file. You can verify the unique identifier for workstations, resources, and prompts by submitting the composer list command, together with the `;showid` filter, or by submitting the conman command, `showcpus`, `showresources`, or `showprompts`, in combination with the `;showid` filter. See the related example in the **Examples** section.

Identifying workstations by their unique identifier avoids the problem that occurred in previous versions when objects were renamed in the plan. For example, if an object was renamed and then carried forward to a new production plan, references to the old object name were lost. With the implementation of the unique identifier, this will no longer occur and dependencies will be correctly resolved.

When deleting a workstation, if the workstation is still in the plan, then another workstation cannot be renamed with the name of the deleted workstation for the number of days specified by the global option folderDays. However, a brand new workstation can be created with the name of the deleted workstation. This behavior applies only to dynamic agents, pools, and dynamic pools. The default value is 10 days.

When deleting a folder, a prompt, or resource, if there are still objects in the plan that reference these objects, then another folder, prompt, or resource cannot be renamed with the name of the deleted folder, prompt or resource for the number of days specified by folderDays. However, a brand new folder, prompt, or resource can be created with the name of the deleted object. See the folderDays global option in the *Administration Guide*

Results

List provides you with the list of objects names with their attributes. Print sends the list of objects names with their attributes to the device or file set in the *MAESTROLP* local variable. The print command can be used to send the output to a local printer, if you set the *MAESTROLP* variable accordingly. Make sure the *MAESTROLP* is set in your environment before running the print command.

Depending on the value set in the ***MAESTROCOLUMNS*** local variable the different sets of information about the selected object can be shown.

A table showing the output formats for printing scheduling objects can be found in the topic about the print command in *User's Guide and Reference*.

See the topic on the offline output in "SETUPCOMPOSER" for more information on how to set ***MAESTROLP*.**

Example

Examples

- To list all calendars, run the following command:

```
list calendars=@
```

this is a sample output:

```
Calendar Name     Updated On  Locked By
----------------  ----------  --------------------
HOLIDAYS          03/02/2023
PAYDAYS           03/02/2023
HOLIDAYS          03/02/2023
  01/01/2023 02/15/2023 05/31/2023


Calendar Name     Updated On  Locked By
----------------  ----------  --------------------
MONTHEND          01/01/2023  -

MONTHEND
  "Month end dates 1st half 2023"
  01/31/2023 02/28/2023 03/31/2023 04/30/2023 05/31/2023 06/30/2023

Calendar Name     Updated On  Locked By
----------------  ----------  --------------------
PAYDAYS           01/02/2023  -

PAYDAYS
  01/15/2023 02/15/2023 03/15/2023 04/15/2023 05/14/2023 06/15/2023
```

- To list all your defined event rules, run the following command:

```
list er=@
```

If `MAESTROCOLUMNS=80`, the output looks something like this:

```
Event Rule Name   Type       Draft  Status     Updated On  Locked By
----------------  ---------  -----  ---------  ----------  -------------
EVENT-MULTIPLE1   filter            active     06/06/2023  -
EVENT-MULTIPLE2   filter            active     06/06/2023  -
EVENT-MULTIPLE3   filter            active     06/06/2023  -
M_SUCC_12_S       sequence   Y      inactive   06/07/2023  -
M_SUCC_12_S_A     filter            active     06/07/2023  -
M_SUCC_12_S_B     filter     Y      inactive   06/07/2023  -
NEWEVENTRULE      filter            active     06/01/2023  administrator
```

If MAESTROCOLUMNS≥120, the output looks something like this:

```
Event Rule Name         Type      Draft Status   Updated On  Locked By
----------------------  --------- ----- -------- ----------- ---------
EVENT-MULTIPLE1         filter          active   06/06/2023 -
EVENT-MULTIPLE2         filter          active   06/06/2023 -
EVENT-MULTIPLE3         filter          active   06/06/2023 -
M_SUCC_12_S             sequence   Y    inactive 06/07/2023 -
M_SUCC_12_S_A           filter          active   06/07/2023 -
```

```
M_SUCC_12_S_B           filter     Y    inactive 06/07/2023 -
NEWEVENTRULE            filter          active   06/01/2023 administrator
```

- To view the properties of the NC1150691 agent workstation, run the following command:

  ```
  list ws=NC1150691
  ```

  An output similar to the following is displayed:

  ```
  Workstation Name  Type     Domain            Ignored  Updated On  Locked By
  ----------------  -------  ----------------  -------  ----------  ----------
  NC1150691         agent    -                           03/31/2023  -

  CPUNAME NC1150691
    DESCRIPTION "This workstation was automatically created at agent
                 installation time."
    OS WNT
    NODE nc115069.LondonLab.uk.vbtredfk.com SECUREADDR 22114
    TIMEZONE GMT+1
    FOR MAESTRO HOST NC115069_DWB
      TYPE AGENT
      PROTOCOL HTTPS
  END
  ```

- To view the properties of the POOL_A pool workstation, including all its members, run the following command:

  ```
  list ws=POOL_A
  ```

  An output similar to the following is displayed:

  ```
  Workstation Name  Type     Domain            Ignored  Updated On  Locked By
  ----------------  -------  ----------------  -------  ----------  ----------------
  POOL_A            pool     -                           03/31/2023  -

  CPUNAME POOL_A
    DESCRIPTION "This is a manually created pool"
    VARTABLE TABLE1
    OS OTHER
    TIMEZONE America/Argentina/Buenos_Aires
    FOR MAESTRO HOST NC115069_DWB
      TYPE POOL
    MEMBERS
      NC1150691
      NC1150692
  END
  ```

- To list job streams defined in the folder named `FOLDJS_API` on all workstations using an absolute path, submit the following command:

  ```
  composer li js @#/FOLDJS_API/@
  ```

  The following is the output of this command:

  ```
  -li js @#/FOLDJS_API_132/

  Workstation       Job Stream Name   Valid From  Updated On  Locked By
  ----------------  ----------------  ----------  ----------  ----------------
  >>                /FOLDJS_API_132/
  EU-HWS-LNX127     JS_FOLD_API_132   08/22/2023  07/31/2023  -
  ```

Optionally, you can obtain this same result, by submitting the command using a relative path. To submit the command using a relative path, use the -cf option to change the current folder from root (/) to `/FOLDJS_API/` as follows:

```
composer -cf /FOLDJS_API_132/ "li js @#@"
```

An additional way to obtain the same result is to change the current folder in composer first, then submit the command:

```
-cf /FOLDJS_API/
```

```
-li js @#@
```

- To list workstation `Austin`, stored in folder `Texas`, run the command:

  ```
  list ws Texas/Austin
  ```

- To list all workstations defined in all folders, including the unique identifier assigned to each workstation:

  ```
  -li ws=/@/@;showid
  ```

  The following is the output for this command:

  ```
  Workstation Name  Type     Domain      Ignored  Updated On  Locked By
  ----------------  -------  ----------  -------  ----------  ----------------
  AGENT_DYN       agent    -                    09/30/2023  -
  EU-HWS-LNX36      manager  MASTERDM               10/03/2023  -
  EU-HWS-LNX36_1    agent    -           Y        09/30/2023  -
  EU-HWS-LNX36_1_1  agent    -                    09/30/2023  -                {07775EB26F77524X}
  EU-HWS-LNX36_1_2  agent    -                    10/03/2023  -                {0AAEYDX55XC22E6C}
  EU-HWS-LNX36_DWB  broker   MASTERDM              09/30/2023  -
  MASTERAGENTS      pool     -                    09/30/2023  -

  /WSFTA/
  FTAMAR         fta      MASTERDM             09/30/2023  -                {0AAA5D7ZC7BY24A6}
  ```

See also

From the Dynamic Workload Console you can perform the same tasks as described in:

the *Dynamic Workload Console User's Guide*.

the *Dynamic Workload Console User's Guide*.

- To list or print event rules, see

  the *Dynamic Workload Console User's Guide*, section about Editing an event rule.

  the *Dynamic Workload Console User's Guide*, section about Editing an event rule.

- To list or print access control lists, security domains, and security roles, see

  the *Dynamic Workload Console User's Guide*, section about Managing Workload Security.

  the *Dynamic Workload Console User's Guide*, section about Managing Workload Security.

LISTFOLDER

Lists folders defined in the HCL Workload Automation database, either in the root or in a folder.

Authorization

If the ***enListSecChk*** global option is set to **yes** on the master domain manager, then to list a folder, you must have either ***list*** access, or ***list*** and ***display*** access.

Syntax

{**listfolder** | **lf**} *foldername*

Arguments

foldername

If only a forward slash ("/") is specified, then all folders in the root are listed.

If a folder name follows, then all folders contained in the specified folder are listed, excluding the specified folder. Wildcard characters are permitted.

To list all folders in an entire tree structure, specify the ampersand character ("@")

Comments

You can perform this same operation using the LIST on page 479 composer command.

Example

Examples

To list the entire tree structure of folders folders in the root, run:

```
listfolder /@
```

To list all folders contained in the folder named "Test", run:

```
listfolder /Test
```

or

```
listfolder /Test/
```

To list all folders and sub-folders contained in the folder named "Test", run:

```
listfolder /Test/@
```

LOCK

Locks the access to scheduling objects definitions in the database.

Authorization

To **lock** scheduling objects, you must have ***modify*** access to the object.

To **lock** security objects, you must have permission for the **modify** action on the object type **file** with attribute **name=security**.

Syntax

{**lock** | **lo**}
{[**calendars** | **calendar** | **cal**=*[folder/]calname*] |
[**eventrule** | **erule** | **er**=*[folder/]eventrulename*] |
[**folder** | **fol**=*foldername*] |
[**parms** | **parm** | **vb**=[*[folder/]tablename*.]*variablename*] |
[**vartable** | **vt**=*[folder/]tablename*] |
[**prompts** | **prom**=*[folder/]promptname*] |
[**resources** | **resource** | **res**=[*[folder/]workstationname#*]*[folder/]resourcename*] |
[**runcyclegroup** | **rcg**=*[folder/]runcyclegroupname*] |
[**cpu**={*[folder/]workstationname* | *[folder/]workstationclassname* | *domainame*}]
[**workstation** | **ws**=*[folder/]workstationname*] |
[**workstationclass** | **wscl**=*[folder/]workstationclassname*] |
[**domain** | **dom**=*domainame*] |
[**jobs** | **jobdefinition** | **jd**=[*[folder/]workstationname#*][*folder/*]*jobname*] |
[**sched**|**jobstream**|**js**= [*[folder/]workstationname#*][*folder/*]*jstreamname*
*[**valid from** date*|***valid to** date* |***valid in** date date]*] |
[**users** | **user**=[*[folder/]workstationname#*]*username*] |
[**accesscontrollist** | **acl for** *securitydomainname*] |
[**securitydomain** | **sdom**=*securitydomainname*] |
[**securityrole** | **srol**=*securityrolename*]}

Arguments

calendars

Locks all calendar definitions.

calendars | **calendar** | **cal**

If no argument follows, locks all calendar definitions.

If argument *[folder/]calname* follows, locks the *calname* calendar. Wildcard characters are permitted.

eventrule | **erule** | **er**

If no argument follows, locks all event rule definitions.

If argument *[folder/]eventrulename* follows, locks the *eventrulename* event rule. Wildcard characters are permitted.

folder

If no argument follows, locks the current folder.

If argument *foldername* follows, locks the *foldername* folder. If a parent folder is locked, a child folder can be modified or deleted. Wildcard characters are permitted.

parms | parm | vb

If no argument follows, locks the entire default variable table.

If argument *[folder/]tablename.variablename* follows, locks the entire table containing the *variablename* variable. If *tablename* is omitted, **composer** locks the entire default variable table.

Note: When you lock a variable, this locks the entire variable table that contains it. This implies that, while the table is locked, no other user can run any other locking commands on it.

Wildcard characters can be used on both *tablename* and *variablename*. For example:

```
lock parms=@.@
```

Locks all variables on all tables. As a result, all variable tables are locked.

```
lock parms=@
```

Locks all variables on the default table. As a result, the variable table is locked.

```
lock parms=@.acct@
```

Locks all the variables whose name starts with `acct` on all the existing tables. As a result, all the variable tables that contain at least one variable named in this way are locked.

vartable | vt

If no argument follows, locks all variable table definitions.

If argument *[folder/]tablename* variable table follows, locks the *tablename* variable table. Wildcard characters are permitted.

prompts | prom

If no argument follows, locks all prompt definitions.

If argument *[folder/]promptname* follows, locks the *promptname* prompt. Wildcard characters are permitted.

resources | resource | res

If no argument follows, locks all resource definitions.

If argument *[folder/]workstationame#[folder/]resourcename* follows, locks the *resourcename* resource of the *workstationame* workstation in the specified folder on which the resource is defined. If *workstationame* is omitted, the default is the workstation on which **composer** is running. Wildcard characters are permitted.

runcyclegroup | rcg

If no argument follows, locks all run cycle group definitions.

If argument *[folder/]runcyclegroupname* follows, locks the *runcyclegroupname* run cycle group. Wildcard characters are permitted.

cpu

Locks workstations, workstation classes, or domains.

[folder/]workstation

The name of the workstation. Wildcard characters are permitted.

[folder/]workstationclass

The name of the workstation class. Wildcard characters are permitted.

domain

The name of the domain. Wildcard characters are permitted.

workstation | ws

If no argument follows, locks all workstation definitions.

If argument *[folder/]workstationname* follows, locks the *workstationname* workstation. Wildcard characters are permitted.

domain | dom

If no argument follows, locks all domain definitions.

If argument *domainname* follows, locks the *domainname* domain. Wildcard characters are permitted.

workstationclass | wscl

If no argument follows, locks all workstation class definitions.

If argument *[folder/]workstationclassname* follows, locks the *workstationclassname* workstation class. Wildcard characters are permitted.

jobs | jobdefinition | jd

If no argument follows, locks all job definitions.

If argument *[folder/]workstationame#[folder/]jobname* follows, locks the *jobname* job of the *workstationame* workstation on which the job runs. If *workstationame* is omitted, the default is the workstation on which **composer** is running. Wildcard characters are permitted.

sched | jobstream | js

If no argument follows, locks all job stream definitions.

If argument *[folder/]workstationame#[folder/]jstreamname* follows, locks the *jstreamname* job stream of the *workstationame* workstation on which the job stream is defined. If *workstationame* is omitted, the default is the workstation on which **composer** is running. Wildcard characters are permitted.

valid from

date Restricts the selection to job streams that have a *valid from* date equal to the indicated value. The format is *mm/dd/yyyy*.

valid to

date Restricts the selection to job streams that have a *valid to* date equal to the indicated value. The format is *mm/dd/yyyy*.

valid in

date date The time frame during which the job stream can run. The format is *mm/dd/yyyy - mm/dd/yyyy*. One of the two dates can be represented by @.

users | user

If no argument follows, locks all user definitions.

If argument *[folder/]workstationame#username* follows, locks the *username* user of the *workstationame* workstation in the specified folder on which the user is defined. If *workstationame* is omitted, the default is the workstation on which **composer** is running. Wildcard characters are permitted.

accesscontrollist | acl

If no *securitydomainname* argument follows, locks the access control list definitions for all the security domains.

If argument *securitydomainname* follows, locks the access control list definitions for the *securitydomainname* security domain. Wildcard characters are permitted for *securitydomainname* .

securitydomain | sdom

If no *securitydomainname* argument follows, locks all the security domain definitions.

If argument *securitydomainname* follows, locks the definition of the *securitydomainname* security domain. Wildcard characters are permitted for *securitydomainname* .

securityrole | srol

If no *securityrolename* argument follows, locks all the security roles definitions.

If argument *securityrolename* follows, locks the definition of the *securityrolename* security role. Wildcard characters are permitted for *securityrolename* .

Comments

Objects are locked to make sure that definitions in the database are not overwritten by different users accessing concurrently to the same objects.

With this command the user explicitly acquires locks of database objects. When one user has an object locked, any other user has read only access until the object is released or explicitly unlocked by the administrator. If one user tries to lock an object that is already locked by someone else (other user), an error message is returned.

Locks on database objects are acquired by the user using `username` and `session`, where `session` is a string that can be set in the environment variable **TWS_SESSION** identifying that specific user work session.

This means that, on a machine, the **TWS_SESSION** identifier is different for:

- a user connected in two different shells to the **composer** command line program.
- a user connected, disconnected and then connected again to the **composer** command line from the same shell.

If no value is assigned to **TWS_SESSION**, then the default value identifying the session is set as follows:

- If using **composer** in batch mode, the default value is the *username* used by the user when connecting to the master domain manager.
- If using **composer** in interactive mode, the default value corresponds to an alphanumeric string automatically created by the product.

Note: In the database the *username* of the user locking an object definition is saved in uppercase.

Example

Examples

To lock the calendar named `Holidays`, run the command:

```
lock calendar=HOLIDAYS
```

To lock a folder named `Chicago`, run the command:

```
lock folder /CHICAGO
```

See also

In the Dynamic Workload Console, objects are automatically locked as long as you or another user have them open using the **Edit** button. Objects are not locked if you or another user opened them with **View**.

MKFOLDER

Creates a new folder definition in the database.

Authorization

You must have ***add*** access if you are creating a new folder. If the folder already exists in the database, you must have ***modify*** access to the folder.

Syntax

{**mkfolder** | **mf**} *foldername*

Arguments

foldername

Specify the folder name offset by forward slashes (/folder/) if you specify an absolute path, and without slashes if you specify a relative path.

Comments

You can also create a folder using the new composer command. See NEW on page 501. To create folders in a specific folder, then you can use the chfolder command to navigate to folders and sub-folders. The chfolder command changes the working directory, which is set to root ("/") by default, so that you can use relative folder paths when submitting commands. See CHFOLDER on page 457 for more information about changing folder paths.

The command opens a predefined template that helps you edit the folder definition and adds it in the database when you save it.

The object templates are located in the `templates` subfolder in the HCL Workload Automation installation directory. They can be customized to fit your preferences.

The maximum length for the full folder path (that is, the path name including the parent folder and all nested sub-folders) is 800 characters, while each folder name supports a maximum of 256 characters. When you create a nested folder, the parent folder must exist.

Example

Examples

To create a new folder named "Tokyo", run:

```
mkfolder /Tokyo/
```

To create a new sub-folder named "Tokyo", under an existing folder named "Japan":

```
mkfolder /Japan/Tokyo
```

To create a new sub-folder named "Tokyo", under an existing folder named "Japan" using a relative path:

```
mkfolder Japan/Tokyo
```

See also

From the Dynamic Workload Console you can perform the same tasks as described in:

the *Dynamic Workload Console User's Guide*.

the *Dynamic Workload Console User's Guide*.

- To create database objects, see

  the *Dynamic Workload Console User's Guide*, section about Designing your Workload.

  the *Dynamic Workload Console User's Guide*, section about Designing your Workload.

MODIFY

Modifies or adds scheduling objects. When modifying objects, the **modify** command extracts only the objects that can be locked by the current user.

Authorization

You must have **add** access if you add a new scheduling object. If the object already exists in the database, you must have **modify** access to the object, otherwise, composer is unable to find the objects.

To **modify** security objects, you must have permission for the **modify** action on the object type **file** with attribute **name=security**.

Syntax

{**modify** | **m**}
{[**calendars** | **calendar** | **cal**=*[folder/]calname*] |
[**eventrule** | **erule** | **er**=*[folder/]eventrulename*] |
folder|**fol** |
[**parms** | **parm** | **vb**=[*[folder/]tablename*.]*variablename*] |
[**vartable** | **vt**=*[folder/]tablename*] |
[**prompts** | **prom**=*[folder/]promptname*] |
[**resources** | **resource** | **res**=[*[folder/]workstationname#*]*[folder/]resourcename*] |
[**runcyclegroup** | **rcg**=*[folder/]runcyclegroupname*] |
[**cpu**={*[folder/]workstationname* | *[folder/]workstationclassname* | *domainame*}]
[**wat**=*workloadapplicationtemplatename*]
[**workstation** | **ws**=*[folder/]workstationname*] |
[**workstationclass** | **wscl**=*[folder/]workstationclassname*] |
[**domain** | **dom**=*domainame*] |
[**jobs** | **jobdefinition** | **jd**=[*[folder/]workstationname#*][*folder/*]*jobname*] |
[**sched**|**jobstream**|**js**= [*[folder/]workstationname#*][*folder/*]*jstreamname*
*[**valid from** date*|***valid to** date* |***valid in** date date]*
*[**;full**]*] |
[**users** | **user**=[*[folder/]workstationname#*]*username*] |
[**accesscontrollist** | **acl for** *securitydomainname*] |
[**securitydomain** | **sdom**=*securitydomainname*] |
[**securityrole** | **srol**=*securityrolename*]}

Arguments

calendars | calendar | cal

If no argument follows, modifies all calendar definitions.

If argument *[folder/]calname* follows, modifies the *calname* calendar. Wildcard characters are permitted.

eventrule | erule | er

If no argument follows, modifies all event rule definitions.

If argument *[folder/]eventrulename* follows, modifies the *eventrulename* event rule. Wildcard characters are permitted.

folder | fol

If no argument follows, modifies all folder definitions.

If argument *foldername* follows, modifies the *foldername* folders. Wildcard characters are permitted.

parms|parm|vb

If no argument follows, modifies all global variable definitions found in the default variable table.

If argument *[folder/]tablename.variablename* follows, modifies the specified variable of the *tablename* table. If *tablename* is omitted, **composer** looks for the *variablename* variable definition in the default variable table. Wildcard characters can be used on both *tablename* and *variablename*. For example:

```
modify parms=@.@
```

Modifies all variables on all tables.

```
modify parms=@
```

Modifies all variables on the default table.

```
modify parms=@.acct@
```

Modifies all the variables whose name starts with `acct` on all the existing tables.

Remember: The action of modifying or adding a variable locks the variable table that contains it. This implies that, while the table is locked, no other user can run any other locking commands on it or on the variables it contains.

vartable | vt

If no argument follows, modifies all variable table definitions.

If argument *[folder/]tablename* variable table follows, modifies the *tablename* variable table. Wildcard characters are permitted.

prompts | prom

If no argument follows, modifies all prompt definitions.

If argument *[folder/]promptname* follows, modifies the *promptname* prompt. Wildcard characters are permitted.

resources | resource | res

If no argument follows, modifies all resource definitions.

If argument *[folder/]workstationame#[folder/]resourcename* follows, modifies the *resourcename* resource of the *workstationame* workstation in the specified folder on which the resource is defined. If *workstationame* is omitted, the default is the workstation on which **composer** is running. Wildcard characters are permitted.

runcyclegroup | rcg

If no argument follows, modifies all run cycle group definitions.

If argument *[folder/]runcyclegroupname* follows, modifies the *runcyclegroupname* run cycle group. Wildcard characters are permitted.

cpu

Modifies workstations, workstation classes, or domains.

[folder/]workstation

The name of the workstation. Wildcard characters are permitted.

[folder/]workstationclass

The name of the workstation class. Wildcard characters are permitted.

domain

The name of the domain. Wildcard characters are permitted.

wat

If no argument follows, modifies all workload application template definitions.

If argument *[folder/]workloadapplicationtemplatename* follows, modifies the specified workload application template. Wildcard characters are permitted.

workstation | ws

If no argument follows, modifies all workstation definitions.

If argument *[folder/]workstationname* follows, modifies the *workstationname* workstation. Wildcard characters are permitted.

domain | dom

If no argument follows, modifies all domain definitions.

If argument *domainname* follows, modifies the *domainname* domain. Wildcard characters are permitted.

workstationclass | wscl

If no argument follows, modifies all workstation class definitions.

If argument *[folder/]workstationclassname* follows, modifies the *workstationclassname* workstation class. Wildcard characters are permitted.

jobs | jobdefinition | jd

If no argument follows, modifies all job definitions.

If argument *[folder/]workstationame#[folder/]jobname* follows, modifies the *jobname* job of the *workstationame* workstation on which the job runs. If *workstationame* is omitted, the default is the workstation on which **composer** is running. If [*folder/*] is omitted, then the root folder is assumed. If a relative path is used to specify the folder then the current folder is assumed. Wildcard characters are permitted.

sched | jobstream | js

If no argument follows, modifies all job stream definitions.

If argument *[folder/]workstationame#[folder/]jstreamname* follows, modifies the *jstreamname* job stream of the *workstationame* workstation on which the job stream is defined. If *workstationame* is omitted, the default is the workstation on which **composer** is running. If [*folder/*] is omitted, then the root folder is assumed. If a relative path is used to specify the folder then the current folder is assumed. Wildcard characters are permitted.

valid from

date Restricts the selection to job streams that have a *valid from* date equal to the indicated value. The format is *mm/dd/yyyy*.

valid to

date Restricts the selection to job streams that have a *valid to* date equal to the indicated value. The format is *mm/dd/yyyy*.

valid in

date date The time frame during which the job stream can run. The format is *mm/dd/yyyy - mm/dd/yyyy*. One of the two dates can be represented by @.

full

Modifies all job definitions contained in the job stream.

users | user

If no argument follows, modifies all user definitions.

If argument *[folder/]workstationame#username* follows, modifies the *username* user of the *workstationame* workstation in the specified folder on which the user is defined. If *workstationame* is omitted, the default is the workstation on which **composer** is running. Wildcard characters are permitted.

accesscontrollist | acl

If no *securitydomainname* argument follows, modifies the access control list definitions for all the security domains.

If argument *securitydomainname* follows, modifies the access control list definitions for the *securitydomainname* security domain. Wildcard characters are permitted for *securitydomainname* .

securitydomain | sdom

If no *securitydomainname* argument follows, modifies all the security domains definitions.

If argument *securitydomainname* follows, modifies the *securitydomainname* security domain definition. Wildcard characters are permitted for *securitydomainname* .

securityrole | srol

If no *securityrolename* argument follows, modifies all the security roles definitions.

If argument *securityrolename* follows, modifies the *securityrolename* security role definition. Wildcard characters are permitted for *securityrolename* .

Comments

The **modify** command performs the following sequence of actions:

1. Locks the objects in the database.
2. Copies the objects definition into a temporary file.
3. Edits the file.
4. Replaces the definition contained in the temporary file to the database.
5. If the **modify** command fails on a subset of the selected objects, **composer** asks *"do you want to re-edit?"* and the file saved before is reopened for editing and the next steps of the sequence are repeated.
6. Unlocks the objects in the database.

Event rule definitions are opened with an XML editor (see the topic on event rule definition in the chapter about defining objects in the database in *User's Guide and Reference* for XML reference and see the topic on the composer editor in SETUPCOMPOSER for details on setting up an XML editor).

If you modify with the same **modify** command two or more objects linked together by any relationship, for example a successor job and its predecessor job, then it might be relevant for the successful result of the **modify** command the order in which the objects are listed in the temporary file. This happens because the **modify** command reads in sequence the objects contained in the temporary file; so, if the referencing object is displayed before the object being referenced, the modify command might fail on the referencing object.

For example, if the command:

```
modify FTA1#TESTDIR/@PROVA
```

produces the following temporary file:

```
SCHEDULE FTA1#TESTDIR/PROVA VALIDFROM 08/31/2023
MATCHING SAMEDAY
:
FTA2#TESTDIR/MY-JOB
 FOLLOWS FTA1#DEVDIR/COPYOFPROVA.MY-JOB06
END
```

```
SCHEDULE FTA1#DEVDIR/COPYOFPROVA VALIDFROM 08/31/2023
MATCHING SAMEDAY
:
FTA1#TESTDIR/MY-JOBO6
END
```

and you change the name of the predecessor job from `FTA1#DEVDIR/MY-JOB06` to `FTA1#DEVDIR/MY-JOB05` in both job streams `FTA1#TESTDIR/PROVA` and `FTA1#TESTDIR/COPYOFPROVA`, then the **modify** command:

1. At first tries to change the definition of job stream `FTA1#TESTDIR/PROVA` and it fails because it finds a follows dependency from a job `FTA1#TESTDIR/MY-JOB05` which is still unknown.
2. Then it tries to change the definition of `FTA1#DEVDIR/COPYOFPROVA` and it succeeds.

The second time you run **modify** to change the predecessor job from `FTA1#DEVDIR/MY-JOB06` to `FTA1#DEVDIR/MY-JOB05` in job stream `FTA1#TESTDIR/PROVA`, the command is successfully performed since the predecessor job `FTA1#TESTDIR/MY-JOB05` now exists in the database.

If job stream `FTA1#DEVDIR/COPYOFPROVA` had been listed in the temporary file before `FTA1#PROVA`, then the **modify** command would have run successfully the first time because the name of the predecessor job would have been modified before changing the dependency definition in the successor job.

For user definitions, if the password field keeps the `"*******"` value when you exit the editor, the old password is retained. To specify a null password use two consecutive double quotation marks (").

The modify command checks for loop dependencies inside job streams. For example, if `job1` follows `job2`, and `job2` follows `job1` there is a loop dependency. When a loop dependency inside a job stream is found an error is displayed. The modify command does not check for loop dependencies between job streams because, depending on the complexity of the scheduling activities, this check might be too time and CPU consuming.

Example

Examples

To modify all calendars, run the following command:

```
modify calendars=@
```

To modify job stream `sked9` stored in the folder named "Tools" that is launched on workstation `site1`, run the following command:

```
m sched=site1#Tools/sked9
```

To modify all the event rules that include an action with job `DPJOB10`, run:

```
mod er=@;filter job=DPJOB10
```

To modify workstation `Austin`, stored in folder `Texas`, run the command:

```
m ws Texas/Austin
```

See also

From the Dynamic Workload Console you can perform the same tasks as described in:

the *Dynamic Workload Console User's Guide*.

the *Dynamic Workload Console User's Guide*.

- To modify event rules, see

  the *Dynamic Workload Console User's Guide*, section about Editing an event rule.

  the *Dynamic Workload Console User's Guide*, section about Editing an event rule.

NEW

Adds a new scheduling object definition in the database.

Authorization

You must have ***add*** access if you add a new scheduling object. If the object already exists in the database you must have ***modify*** access to the object.

To **add** new security objects, you must have permission for the **modify** action on the object type **file** with attribute **name=security**.

Syntax

new
[**calendar** |
domain |
eventrule |
folder|**fol** |
job |
jobstream |
parameter |
prompt |
resource |
runcyclegroup |
user |
vartable |
wat |
workstation |
workstationclass |
accesscontrollist |

securitydomain |
securityrole]

Arguments

The object you want to define: a calendar, a domain, an event rule, a folder, a job, a job stream, a variable, a prompt, a resource, a user, a variable table, a workload application template, a workstation, or a workstation class.

Comments

The command opens a predefined template that helps you edit the object definition and adds it in the database when you save it.

The object templates are located in the `templates` subfolder in the HCL Workload Automation installation directory. They can be customized to fit your preferences.

Event rule definitions are opened with an XML editor (see the topic on event rule definition in the chapter on "Defining objects in the database" in *User's Guide and Reference* for XML reference and see the topic on the composer editor in SETUPCOMPOSER for details on setting up an XML editor).

While you create a variable, the destination variable table is locked. This implies that, while the table is locked, no other user can run any other locking commands on it.

The maximum length for the full folder path (that is, the path name including the parent folder and all nested sub-folders) is 800 characters, while each folder name supports a maximum of 256 characters. When you create a nested folder, the parent folder must be existing.

The maximum length for passwords is **30 bytes**. However, 30 bytes does not always equal 30 characters, as some characters may require multiple bytes, depending on the chosen encoding.

Example

Examples

To create a new user definition, run:

```
new user
```

To create a new folder, run:

```
new folder
```

You can also create a new folder using the dedicated composer command MKFOLDER on page 493.

To create a new prompt definition, run:

```
new prompt
```

To create a new event rule definition, run:

```
new erule
```

To create a new variable table definition, run:

```
new vartable
```

To create a new variable definition, run:

```
new parameter
```

To create two workload application templates, WAT_NAME1 and WAT_NAME2, run:

```
new wat
BAPPLICATION WAT_NAME1
  DESCRIPTION "Description"
  VENDOR "Provider"
  JSTREAMS
      FTA1#JS_1_1
      AGENT1#JS_1_2
END

BAPPLICATION WAT_NAME2
  DESCRIPTION "Description"
  VENDOR "Provider"
  JSTREAMS
      JS_2_1
      JS_2_2
END
```

See also

From the Dynamic Workload Console you can perform the same tasks as described in:

the *Dynamic Workload Console User's Guide*.

the *Dynamic Workload Console User's Guide*.

- To create event rules, see

  the *Dynamic Workload Console User's Guide*, section about Creating an event rule.

  the *Dynamic Workload Console User's Guide*, section about Creating an event rule.
- To create access control lists, security domains, and security roles, see

  the *Dynamic Workload Console User's Guide*, section about Managing Workload Security.

  the *Dynamic Workload Console User's Guide*, section about Managing Workload Security.
- To create all other objects, see

  the *Dynamic Workload Console User's Guide*, section about Designing your Workload.

  the *Dynamic Workload Console User's Guide*, section about Designing your Workload.

PRINT

This is a synonym for the display command used with the **;offline** option. See DISPLAY for details.

REDO

Edits and runs the previous command again.

Note: If the previous command was **authenticate**, **redo** does not display the password specified.

Authorization

Any user authorized to run composer is authorized to issue this command.

Syntax

{**redo** | **red**}

Context

When you run the **redo** command, composer displays the previous command, so that it can be edited and run again. Use the spacebar to move the cursor under the character to be modified, and enter the following directives.

Directives

d[*dir*]

Deletes the character above the **d**. This can be followed by other directives.

i*text*

Inserts text before the character above the **i**.

r*text*

Replaces one or more characters with text, beginning with the character above the **r**. Replace is implied if no other directive is entered.

>*text*

Appends text to the end of the line.

>d[*dir* | *text*]

Deletes characters at the end of the line. This can be followed by another directive or text.

>r*text*

Replaces characters at the end of the line with text.

Directive Examples

ddd

Deletes the three characters above the **d**s.

iabc

Inserts **abc** before the character above the **i**.

rabc

Replaces the three characters, starting with the one above the **r**, with **abc**.

abc

Replaces the three characters above **abc** with **abc**.

d diabc

Deletes the character above the first **d**, skips one character, deletes the character above the second **d**, and inserts **abc** in its place.

>abc

Appends **abc** to the end of the line.

>ddabc

Deletes the last two characters in the line, and inserts **abc** in their place.

>rabc

Replaces the last three characters in the line with **abc**.

Example

Examples

To insert a character, run the following command:

```
redo
dislay site1#sa@
   ip
display site1#sa@
```

To replace three characters, for example change `site` into `serv` by replacing `ite` with `erv`, run the following command:

```
redo
display site1#sa@
         rerv
display serv1#sa@
```

RMFOLDER

Deletes folders defined in the HCL Workload Automation database. The folder and any sub-folders are deleted provided they do not contain any scheduling objects.

Authorization

To **delete** a folder, you must have ***delete*** access to the folder.

Syntax

{**rmfolder** | **rf**} *foldername*

Arguments

foldername

If a *foldername* is not specified, then by default the command deletes all folders and sub-folders in the root. If a *foldername* follows, then deletes the specified folder and its sub-folders. A folder is deleted provided that the folder and its sub-folders do not contain scheduling objects, otherwise, an error message is returned.

To delete all folders in an entire tree structure, specify the ampersand character ("@").

Comments

To be removed, the folder must be unlocked or locked by the user who issues the rmfolder command.

You can perform this same operation using the DELETE on page 460 composer command.

Example

Examples

To delete the folder named "Test" located in the root, run:

```
rmfolder /Test
```

or,

```
rmfolder Test
```

In this case, by default, the command checks for the folder in the root.

RENAME

Renames a scheduling object already existing in the database. The new name must not identify an object already defined in the database. When renaming a scheduling object, in addition to modifying the name, you can also move the object to a specified folder path, where the folder names reflect strings contained in the scheduling object name. The folder must already exist.

Authorization

To **rename** scheduling objects, you must have ***delete*** access to the object with the old name and ***add*** access to the object with the new name.

To **rename** security objects, you must have permission for the **modify** action on the object type **file** with attribute **name=security**.

Syntax

{**rename** | **rn**}
old_object_identifier new_object_identifier [;preview]
{[**calendars** | **calendar** | **cal=*[folder/]calname*]** |

[**parms** | **parm** | **vb**=[*[folder/]tablename*.]*variablename*] |
[**vartable** | **vt**=*[folder/]tablename*] |

[**prompts** | **prom**=*[folder/]promptname*] |
[**resources** | **resource** | **res**=[*[folder/]workstationname#*]*[folder/]resourcename*] |
[**runcyclegroup** | **rcg**=*[folder/]runcyclegroupname*] |

[**workstation** | **ws**=*[folder/]workstationname*]

[**workstationclass** | **wscl**]=*[folder/]workstationclassname* |
[**domain** | **dom**]=*domainame*] |

[**jobs** | **jobdefinition** | **jd**]=[*[folder/]workstationname#*][*folder/*]*jobname* |
[**sched** | **jobstream** | **js**]= [*[folder/]workstationname#*]
[**eventrule** | **erule** | **er**=*[folder/]eventrulename*] |
[**wat**=*[folder/]workloadapplicationtemplatename*]
[**folder**|**fol**] =*foldername*] |
[**sched** | **jobstream** | **js**]= [*[folder/]workstationname#*][*folder/*]*jstreamname*
*[**valid from** date*|***valid to** date* |***valid in** date date]*] |
[**accesscontrollist** | **acl for** *securitydomainname*] |
[**securitydomain** | **sdom**=*securitydomainname*] |
[**securityrole** | **srol**=*securityrolename*] |
[**users**|**user**=*usesrname*] }

Arguments

old_object_identifier

Specifies the old external key identifying the scheduling object, for example calendar name `cal1` as identifier for a defined calendar object to be renamed.

new_object_identifier

Specifies the new external key identifying the scheduling object, for example calendar name `cal2` as new identifier to be assigned to calendar object previously named `cal1`.

;preview

Optional. Use this option to review the result without actually performing the rename operation. No validation is performed for a preview, for example, a check on the existence of the target folders specified, or the use of reserved words in job or job stream names. This option can be used together with the following options: jobdefinition, jobstream, runcycle, runcyclegroup, folder, accesscontrollist, securityrole, securitydomain.

For what concerns jobs, job streams, resources and users both the ***old_object_identifier*** and ***new_object_identifier*** have the following formats:

[folder/][workstationname#][folder/]jobname

The command applies to this job definition. This format is used with the **jobs|jobdefinition|jd** key.

[folder/][workstationname#][folder/]jstreamname

The command applies to all versions of this job stream optionally defined in the specified folder. This format is used with the **sched|jobstream|js** key.

[workstationname#][folder/]jstreamname.jobname

The command applies to this job instance defined in this job stream where the job stream is optionally defined in this folder. For additional details, see the "js" keyword in the Job stream definition topic in the chapter about defining objects in the database in *User's Guide and Reference*. This format is used with the **jobsched|jb** key.

[folder/][workstationname#]resourcename

The command applies to this resource definition. You can optionally specify the folder where the workstation hosting the resource is defined. This format is used with the **resources|resource|res** key.

[folder/][workstationname#][domain\]username

The command applies to this user definition. You can optionally specify the folder where the workstation hosting the domain manager is defined. This format is used with the **users|user** key.

For what concerns variables (global parameters):

old_object_identifier

Must be specified in the *tablename.variablename* format. If *tablename* is omitted, composer looks for the variable in the default variable table.

new_object_identifier

Must be specified in the *variablename* format. Adding the table name here generates an error.

Comments

To be renamed, the object must be unlocked or locked by the user who issues the rename command.

When renaming a folder, if the folder already exists in the plan, the new folder name is also reflected in the plan. After the change, you must use the new folder name when using LISTFOLDER on page 488 and CHFOLDER on page 457 commands or when displaying scheduling objects from the Dynamic Workload Console. You can also rename a folder using the dedicated composer command RENAMEFOLDER on page 510.

The variable table containing the variable is locked, while the variable is renamed. This implies that, while the table is locked, no other user can run any other locking commands on it. Folders, however, are not locked in the database while being renamed, therefore two concurrent **rename** commands on the same folder result in the folder being renamed as specified in the most recent command submitted.

If an object named as specified in the *old_object_identifier* field does not exist in the database an error message is displayed.

One or more wildcards can be used to express part of the job or job stream name. Wildcards are not supported for other objects. When you rename a job or job stream, you can tokenize part of the job or job stream name and then use the tokens to specify the names of the folders and sub-folders into which you want to move the jobs or job streams. When multiple wildcards are used, the positional order is respected after the rename operation. See Examples on page 509 for details about how to move jobs and job streams into folders that are named after tokens in the job and job stream name. See the section about organizing jobs and job streams into folders in *User's Guide and Reference* for more information about how to move jobs and job streams into folders in batch mode and for more complex examples.

When ***workstationame*** is not specified for objects that have the workstation name as part of their object identifier (for example, job or job stream definitions), the program uses one of the following for ***workstationame***:

- The default workstation specified in the *localopts* file
- The master domain manager if the **composer** command line program is running on a node outside the HCL Workload Automation network. In this case, in fact, the default workstation set in the *localopts* file is the master domain manager.

The **rename** command is used to assign new names to objects already existing in the database. The new name assigned to an object becomes immediately effective in the database, while in the plan it maintains the previous name until the **JnextPlan** script is run again. If you submit ad-hoc jobs before generating again the production plan, use the previous name.

Note: Renaming any kind of domain manager, such as master domain manager, dynamic domain manager, or their backups is not a supported operation and might lead to unpredictable results.

Example

Examples

To rename domain object `DOMAIN1` to `DOMAIN2`, run the following command:

```
rename dom=DOMAIN1 DOMAIN2
```

To rename variable `ACCTOLD` (defined in table `ACCTAB`) to `ACCTNEW`, run the following command:

```
rename parm=ACCTAB.ACCTOLD ACCTNEW
```

To rename job stream `LABJST1` to `LABJST2` on workstation `CPU1`, defined in folder `myfolder`, run the following command:

```
rename js=myfolder/CPU1#LABJST1 CPU1#LABJST2
```

To rename and move all job stream definitions currently defined in the root (/) folder, corresponding to the naming convention `ITALY_WS#ROME_MILAN_`*`JOB_STREAM_NAME`*, to a folder structure with folder names based on strings contained in the job stream name, run the following command:

```
rename js @#ROME_MILAN_@ @#/ROME/MILAN/@
```

The result is structured as follows:

```
ITALY_WS#/ROME/MILAN/_JOB_STREAM_NAME
```

See also

From the Dynamic Workload Console you can perform the same tasks as described in:

the *Dynamic Workload Console User's Guide*.

the *Dynamic Workload Console User's Guide*.

- To rename event rules, see

  the *Dynamic Workload Console User's Guide*, section about Editing an event rule.

  the *Dynamic Workload Console User's Guide*, section about Editing an event rule.
- To rename security domains and security roles, see

  the *Dynamic Workload Console User's Guide*, section about Managing Workload Security.

  the *Dynamic Workload Console User's Guide*, section about Managing Workload Security.

RENAMEFOLDER

Use to rename a folder definition in the database. If the folder already exists in the plan, the new folder name is also reflected in the plan. After the change, you must use the new folder name when using LISTFOLDER on page 488 and CHFOLDER on page 457 commands or when displaying jobs and job streams from the Dynamic Workload Console.

Authorization

To **rename** a folder, you must have ***delete*** access to the folder with the old name and ***add*** access to the folder with the new name.

Syntax

{**renamefolder** | **rn**} *previousname newname*

Arguments

previousname

Is the previous name of the folder.

newname

Is the new name of the folder.

Comments

You can perform this same operation using the RENAME on page 506 composer command.

The maximum length for the full folder path (that is, the path name including the parent folder and all nested sub-folders) is 800 characters, while each folder name supports a maximum of 256 characters. When you create a nested folder, the parent folder must be existing.

Example

Examples

To rename a folder in the root named "TEST1" to "TEST2", run:

```
renamefolder /TEST1 /TEST2
```

REPLACE

Replaces scheduling object definitions in the database.

Authorization

You must have ***add*** access if you add a new scheduling object. If the object already exists in the database you must have:

- ***modify*** access to the object if the object is not locked.
- ***modify*** and ***unlock*** accesses to the object if you want to use the **;unlock** option against objects locked by other users.

To **replace** security objects, you must have permission for the **modify** action on the object type **file** with attribute **name=security**.

Syntax

{**replace** | **rep**} *filename* [**;unlock**]

Arguments

filename

Specifies the name of a file containing the object definitions to replace. The file can contain all types of scheduling objects definition.

unlock

Updates existing objects previously locked and unlocks them. An error is displayed if the objects are not previously locked. For all new objects inserted, this option, if specified, is ignored.

Comments

The **replace** command is similar to the **add** command, except that there is no confirmation prompt to replace existing objects. For more information, refer to ADD.

The replace command checks for loop dependencies inside job streams. For example, if `job1` follows `job2`, and `job2` follows `job1` there is a loop dependency. When a loop dependency inside a job stream is found, an error is displayed. To resolve the error, run the replace command again.

Example

Examples

To replace the jobs from the file `myjobs`, run the following command:

```
replace myjobs
```

To replace all resources with those contained in the file `myres`, run the following command:

```
rep myres
```

You want to change some existing event rule definitions in the database. You also want to add some new ones as well. You use this command in the following way:

1. You write the entire definitions in an XML file you name `2Q07rules.xml`.
2. You run:

   ```
   rep 2Q07rules.xml
   ```

SYSTEM COMMAND

Runs a system command.

Syntax

[: | **!]** *system-command*

Arguments

system-command

Specifies any valid system command. The prefix of colon (:) or exclamation mark (!) is required only when the command is spelled the same as a composer command.

Example

Examples

To run a **ps** command on UNIX®, run the following command:

```
ps -ef
```

To run a **dir** command on Windows®, run the following command:

```
dir \bin
```

UNLOCK

Releases access locks on scheduling objects defined in the database. By default, to unlock an object, the object must have been locked using the same user and session.

If the scheduling object is stored in a folder, the command is performed on the folder where the scheduling object definition is stored. If folder is omitted, the default folder ("/") is used.

Authorization

You must have the ***unlock*** access to unlock scheduling objects locked by other users.

To **unlock** security objects, you must have permission for the **unlock** action on the object type **file** with attribute **name=security**.

Syntax

{**unlock** | **un**}
{[**calendars** | **calendar** | **cal**=*[folder/]calname*] |
[**eventrule** | **erule** | **er**=*[folder/]eventrulename*] |
[**folder** | **fol**=*foldername*] |
[**parms** | **parm** | **vb**=[*[folder/]tablename*.]*variablename*] |
[**vartable** | **vt**=*[folder/]tablename*] |
[**prompts** | **prom**=*[folder/]promptname*] |
[**resources** | **resource** | **res**=[*[folder/]workstationname#*][*[folder/]resourcename*] |
[**runcyclegroup** | **rcg**=*[folder/]runcyclegroupname*] |
[**cpu**={*[folder/]workstationname* | *[folder/]workstationclassname* | *domainame*}]
[**workstation** | **ws**=*[folder/]workstationname*] |
[**workstationclass** | **wscl**=*[folder/]workstationclassname*] |
[**domain** | **dom**=*domainame*] |
[**jobs** | **jobdefinition** | **jd**=[*[folder/]workstationname#*][*folder/*]*jobname*] |
[**sched**|**jobstream**|**js**= [*[folder/]workstationname#*][*folder/*]*jstreamname*
*[**valid from** date*|***valid to** date* |***valid in** date date]*] |
[**users** | **user**=[*[folder/]workstationname#*]*username*] |
[**accesscontrollist** | **acl for** *securitydomainname*] |
[**securitydomain** | **sdom**=*securitydomainname*] |
[**securityrole** | **srol**=*securityrolename*]}
[**;forced**]

Arguments

calendars | **calendar** | **cal**

If no argument follows, unlocks all calendar definitions.

If argument *[folder/]calname* follows, unlocks the *calname* calendar. Wildcard characters are permitted.

eventrule | erule | er

If no argument follows, unlocks all event rule definitions.

If argument *[folder/]eventrulename* follows, unlocks the *eventrulename* event rule. Wildcard characters are permitted.

folder

If no argument follows, unlocks all folder definitions.

If argument *foldername* follows, unlocks the *foldername* folder. Wildcard characters are permitted.

parms|parm|vb

If no argument follows, unlocks the default variable table.

If argument *[folder/]tablename.variablename* follows, unlocks the entire table containing the *variablename* variable. If *tablename* is omitted, unlocks the default variable table. Wildcard characters can be used on both *tablename* and *variablename*. For example:

```
unlock parms=@.@
```

Unlocks all tables.

```
unlock parms=@
```

Unlocks the default table.

```
unlock parms=@.acct@
```

Unlocks all the tables containing the variables whose name starts with `acct`.

```
unlock parms=acct@
```

Unlocks the default table.



Remember: The action on a single variable unlocks the variable table that contains it.

vartable | vt

If no argument follows, unlocks all variable table definitions.

If argument *[folder/]tablename* variable table follows, unlocks the *tablename* variable table. Wildcard characters are permitted.

prompts | prom

If no argument follows, unlocks all prompt definitions.

If argument *[folder/]promptname* follows, unlocks the *promptname* prompt. Wildcard characters are permitted.

resources | resource | res

If no argument follows, unlocks all resource definitions.

If argument *[folder/]workstationame#[folder/]resourcename* follows, unlocks the *resourcename* resource of the *workstationame* workstation in the specified folder on which the resource is defined. If *workstationame* is omitted, the default is the workstation on which **composer** is running. Wildcard characters are permitted.

runcyclegroup | rcg

If no argument follows, unlocks all run cycle group definitions.

If argument *[folder/]runcyclegroupname* follows, unlocks the *runcyclegroupname* run cycle group. Wildcard characters are permitted.

cpu

Unlocks workstations, workstation classes, or domains.

[folder/]workstation

The name of the workstation. Wildcard characters are permitted.

[folder/]workstationclass

The name of the workstation class. Wildcard characters are permitted.

domain

The name of the domain. Wildcard characters are permitted.

workstation | ws

If no argument follows, unlocks all workstation definitions.

If argument *[folder/]workstationname* follows, unlocks the *workstationname* workstation. Wildcard characters are permitted.

domain | dom

If no argument follows, unlocks all domain definitions.

If argument *domainname* follows, unlocks the *domainname* domain. Wildcard characters are permitted.

workstationclass | wscl

If no argument follows, unlocks all workstation class definitions.

If argument *[folder/]workstationclassname* follows, unlocks the *workstationclassname* workstation class. Wildcard characters are permitted.

jobs | jobdefinition | jd

If no argument follows, unlocks all job definitions.

If argument *[folder/]workstationame#[folder/]jobname* follows, unlocks the *jobname* job of the *workstationame* workstation on which the job runs. If *workstationame* is omitted, the default is the workstation on which **composer** is running. Wildcard characters are permitted.

sched | jobstream | js

If no argument follows, unlocks all job stream definitions.

If argument *[folder/]workstationame#[folder/]jstreamname* follows, unlocks the *jstreamname* job stream of the *workstationame* workstation on which the job stream is defined. If *workstationame* is omitted, the default is the workstation on which **composer** is running. Wildcard characters are permitted.

valid from

date Restricts the selection to job streams that have a *valid from* date equal to the indicated value. The format is *mm/dd/yyyy*.

valid to

date Restricts the selection to job streams that have a *valid to* date equal to the indicated value. The format is *mm/dd/yyyy*.

valid in

date date The time frame during which the job stream can run. The format is *mm/dd/yyyy - mm/dd/yyyy*. One of the two dates can be represented by @.

users | user

If no argument follows, unlocks all user definitions.

If argument *[folder/]workstationame#username* follows, unlocks the *username* user of the *workstationame* workstation in the specified folder on which the user is defined. If *workstationame* is omitted, the default is the workstation on which **composer** is running. Wildcard characters are permitted.

accesscontrollist | acl

If no *securitydomainname* argument follows, unlocks the access control list definitions for all the security domains.

If argument *securitydomainname* follows, unlocks the access control list definitions for the *securitydomainname* security domain. Wildcard characters are permitted for *securitydomainname* .

securitydomain | sdom

If no *securitydomainname* argument follows, unlocks all the security domain definitions.

If argument *securitydomainname* follows, unlocks the definition of the *securitydomainname* security domain. Wildcard characters are permitted for *securitydomainname* .

securityrole | srol

If no *securityrolename* argument follows, unlocks all the security roles definitions.

If argument *securityrolename* follows, unlocks the definition of the *securityrolename* security role. Wildcard characters are permitted for *securityrolename* .

forced

If specified, allows the user who locked the object to unlock it regardless of the session.

If this option is used by the *superuser*, then the **unlock** command can operate regardless to the user and the session used to lock the object.

Comments

If a user, other than the *superuser*, tries to unlock an object that is locked by another user, an error message is returned.

Example

Examples

To unlock job definition `JOBDEF1`, run the following command:

```
unlock jd=@#JOBDEF1
```

To unlock event rule definition `ERJS21`, run the following command:

```
unlock erule=ERJS21
```

To unlock workstation `Austin`, stored in folder `Texas`, run the command:

```
unlock Texas/Austin
```

To unlock a folder named `Chicago`, run the command:

```
unlock folder /CHICAGO
```

See also

From the Dynamic Workload Console you can perform the same tasks as described in:

the *Dynamic Workload Console User's Guide*.

the *Dynamic Workload Console User's Guide*.

- To unlock event rules, see

  the *Dynamic Workload Console User's Guide*, section about Editing an event rule.

  the *Dynamic Workload Console User's Guide*, section about Editing an event rule.
- To unlock access control lists, security domains, and security roles, see

  the *Dynamic Workload Console User's Guide*, section about Managing Workload Security.

  the *Dynamic Workload Console User's Guide*, section about Managing Workload Security.

UPDATE

Modifies the attribute settings of some specific scheduling object type that is defined in the database without using the modify or replace commands. You might update some attribute settings for the specific object type in the database without opening and modifying the object definition in a text editor or replace the object definition by using the new content that is saved in a text file.

Authorization

You must have the ***modify*** and ***display*** access to update object properties, otherwise composer is unable to update the object attributes.

Syntax

{**update** | **up**}
{[**cpu**={*[folder/]workstationname* | *[folder/]workstationclassname*}] |
[**workstation** | **ws**=*[folder/]workstationname*] |
[**workstationclass** | **wscl**=*[folder/]workstationclassname*];
[**filter** *workstation_filter_criteria*= selection [...]];
set [**ignore**= *on* | *off*]}
[;noask]

Arguments

cpu

Updates workstations or workstation classes.

workstationame

The name of the workstation. Wildcard characters are permitted.

workstationclassname

The name of the workstation class. Wildcard characters are permitted.

Note: The command does not update domains.

***[folder/]*workstation** | **ws**

If no argument follows, update all workstation definitions.

If argument *[folder/]workstationname* follows, updates the *workstationname* workstation. Wildcard characters are permitted.

***[folder/]*workstationclass** | **wscl**

If no argument follows, modify all workstation class definitions.

If argument *[folder/]workstationclassname* follows, updates the *workstationclassname* workstation class. Wildcard characters are permitted.

filter

The workstation scheduling object filter criteria to use.

For more information about workstation scheduling object filtering criteria, see table Scheduling objects filtering criteria in Filters and wildcards.

set

The set criteria is mandatory and introduces the list of the object attribute settings to modify.

ignore=on | off

The `ignore` workstation, cpu, and workstation class attribute. If you specify `on`, the workstation or workstation class is set to ignore in the database and the workstation or all workstations which belong to a workstation class are not included in the next production plan. If you specify `off`, the workstation or all workstations belong to a workstation class are included in the next production plan.

Comments

Note: The scheduling object definition changes are applied only in the database. To have the changes in the plan, you must wait for next production plan (`Final` job stream running) or you might run JnextPlan command.

Example

Examples

If you want that all workstations with names starting with `AB` are not included in the next production plan and you need to update the `ignore` attribute to `ON` on their definitions, run the following command:

```
update ws=AB@;set ignore=on
```

If you want that all your workstations in the network are included in the next production plan and you need to update the `ignore` attribute to `OFF` on all workstations definition in your database, without replying to the composer prompting confirmation, run the following command:

```
update ws;set ignore=off; noask
```

If you want that all workstations which belong to *New York* workstation class are not included in the next production plan and you need to update the `ignore` attribute to `ON` on their definitions, run the following command:

```
update wscl=NewYork;set ignore=on
```

If you want that all your workstations in the domain `DOMWIN1` are included in the next production plan and you need to update the `ignore` attribute to `OFF` on all workstations definition in your database, without replying to the composer prompting confirmation, run the following command:

```
update ws;filter domain=DOMWIN1; set ignore=off; noask
```

If you want that all your workstations in the `MAIN_TABLE` vartable are included in the next production plan, and you need to update the `ignore` attribute to `ON` on all workstations definition in your database, run the following command:

```
update ws=@; filter vartable=MAIN_TABLE domain=MASTER@; set ignore=on
```

To update workstation `Austin`, stored in folder `Texas`, run the command:

```
update ws Texas/Austin
```

See also

From the Dynamic Workload Console you can perform the same tasks as described in:

the *Dynamic Workload Console User's Guide*.

the *Dynamic Workload Console User's Guide*.

- To delete event rules, see

  the *Dynamic Workload Console User's Guide*, section about Editing an event rule.

  the *Dynamic Workload Console User's Guide*, section about Editing an event rule.

VALIDATE

Performs the validation of the object definitions contained in a user file.

Authorization

You do not need any specific authorization to objects to run this command.

To validate users you have previously extracted by setting the **WA_EXPORT_PWD** environment variable, as described in EXTRACT on page 472, specify the passphrase to be used for decrypting the user password in one of the following ways:

- set the **WA_EXPORT_PWD** environment variable with your password in the shell where you plan to run the command. When you run the `export WA_EXPORT_PWD` command, this makes the password accessible to any command run from that shell session, allowing subsequent commands to run without prompting for it again.
- specify the password at the command prompt

Syntax

{**validate** | **val**} *filename* [**;syntax**]

Arguments

filename

Specifies the name of a file that contains calendars, workstations, workstation classes, domains, jobs, parameters, prompts, resources, job streams, event rules, or variable tables. For event rule definitions the file must be in the XML language. See the topic on event rule definition in the chapter on "Defining objects in the database" in *User's Guide and Reference* for details on writing event rule definitions.

syntax

Checks the file for syntax errors.

Comments

The output of the **validate** command can be redirected to a file as follows:

```
composer "validate filename" > outfile
```

To include error messages in the output file, use the following:

```
composer "validate filename" > outfile 2>&1
```

Example

Examples

To check the syntax of a file containing workstation definitions, run the following command:

```
validate mycpus;syntax
```

VERSION

Displays the **composer** command line program banner.

Authorization

Any user authorized to run composer is authorized to issue this command.

Syntax

{**version** | **v**}

Example

Examples

To display the **composer** command line program banner, run the following command:

```
version
```

or:

```
v
```

Organizing scheduling objects into folders

Move a set of scheduling objects in batch mode to specified folders using the scheduling object naming convention to name the folders.

Before you begin

The following scheduling objects can be defined in or moved into folders:

- jobs
- job streams
- workstations
- workstation classes
- calendars
- event rules
- prompts
- resources
- run cycle groups
- variable tables
- workload applications

You can organize your existing scheduling objects that use a specific naming convention into folders. Folders might represent a line of business or any other meaningful category to help you identify and monitor your workload. When moving scheduling objects into folders, you can choose to use part of the object's name as the folder name. The folder name might be a prefix or suffix used in the object's name. Removing this string from the object name frees up some characters in the object name allowing you to create more readable or descriptive names.

Wildcards are used to represent tokens in the object names which are then used to indicate the folder names. Folders with these designated names must already exist. Rename the objects and move them to specified folders in batch mode using the composer rename command. The ;preview option is provided so you can first test the command, without actually submitting it, to get a preview of the output and verify the expected result.

Before you go ahead and move scheduling objects into folders, it is recommended that you first verify if there are any existing event rule definitions that reference these objects. If references are found, duplicate the event rule and update the references to the objects using the new name and folder path. Then, proceed to rename and move the object to the desired folder as described in the following procedure. You can remove the original event rule definition that references the old object name only after verifying that no other instances of the object with the old name exist in the plan.

About this task

The high-level steps for organizing your scheduling objects into folders are as follows:

1. **Create the folders first.** Create the folder hierarchy with the required names in the HCL Workload Automation database. See MKFOLDER on page 493 for creating folders from the composer CLI. See the topic about designing folders in the *Dynamic Workload Console User's Guide* for information about creating folders from the Dynamic Workload Console.
2. **Rename and move the scheduling object into the desired folder**. Run the composer rename command mapping the object name to folder names using wildcards. Use the ;preview option the first time to ensure you obtain the expected result, and then run the command a second time without the ;preview. There are several approaches to organizing your jobs and job streams into designated folders with meaningful names. The following are some examples of how to use wildcards to map tokens contained in the object definitions to folder names. The examples are divided in two categories: renaming objects without the workstation name in the object identifier and renaming objects with the workstation in the object identifier.

   Renaming objects that do not have the workstation name as part of their object identifier (calendars, event rules, prompts, workstations, workstation classes, variable tables, workload applications, run cycle groups)

   Move a workstation from the root folder to a child folder

/AA/BB/EF_GHI
/AB/CD/GH_IJK
/BB/CC/EF_GHI

Renaming objects that have the workstation name as part of their object identifier (jobs, job streams, resources)



```
composer rename js @#ABC_DEF_@ @#/ABC/DEF/@
```

After: WS_NAME#/**ABC**/**DEF**/JSNAME

Before: WS_NAME#**ABCDEF**_JSNAME

```
composer rename js @#ABCDEF_@ @#/AB/CD/EF/@
```

After: WS_NAME#/**AB**/**CD**/**EF**/JSNAME

Move them from an existing folder to sub-folders

Before: WS_NAME#ABC/**DEFG**_JSNAME

```
composer rename js @#ABC/DEFG_@ @#/ABC/DE/FG/@
```

After: WS_NAME#/ABC/**DE**/**FG**/JSNAME

Move them from an existing folder to a different folder using the "?" wildcard character

Before: WS_NAME#MK_1_T/JOB_MY_DEF1_NOW_A

```
composer rename jd /MK_?_T/JOB_MY_DEF?_NOW_A /MY_TEST?/TEST_JOB1
```

After: WS_NAME#/MY_TEST1/TEST_JOB1

Move them from the root to a hierarchy of folders using both the "@" and "?" wildcard characters

Before: WS_NAME#/F1_JSMT01

```
composer rename WS@#F?_JS@ WS@#/F?/JS/@
```

After: WS_NAME#/F1/JS/MT01

Move them from an existing folder to a different folder and rename both the workstation and job name

Before: WS_NAME#/ROME/JOBNAME

```
composer rename jd @#/ROME/@ WKS_NAME#/PARIS/???_DA
```

After: WKS_NAME#/PARIS/**A_B_DA**

WKS_NAME#/PARIS/**A_C_DA**

Move them from an existing folder to a different folder and rename both the workstation and job name

Before: WS_NAME#/ROME/DPT

```
composer rename jd @#/ROME/@ S_MDM#/ROME/???_DA@
```

After: S_MDM#/ROME/**RE_**DA**DPT**, where the wildcard group "???" is matched to "RE" because there were no other matches with more than two characters.

For more information about the composer rename command and syntax, see RENAME on page 506.

3. **Update the security file to assign the appropriate security access.** If access was originally granted on a scheduling object based on matching criteria in the object name, then this access must be modified to include the folder. Use a security domain and add the folder name to the scheduling object, and the folder security object to the security file. You must also add the CPUFOLDER attribute to all objects that include the CPU object attribute, for example, jobs, job streams, userobj, resources, and parameters. See Security domain definition on page 416 if you configure security using the composer command line or the topic about managing security domains in the *Dynamic Workload Console User's Guide* if you configure security from the Dynamic Workload Console.

Results

The objects are now defined in the specified folder paths.

Chapter 11. Managing workload applications

Workload applications can be created and then exported so that they can be shared with other HCL Workload Automation environments. In the new environment, the workload application can be subsequently updated, replaced, or deleted.

Efficient workload processes can be reused in multiple environments. If you have a standardized solution that has been tested and fine-tuned then you can include it in a workload application template and deploy it to other environments for reuse.

The lifecycle of a workload application begins with the definition of the workload application template. The template is then exported to be deployed into the target environment.

You can define the workload application template by using either **composer** command line or the Dynamic Workload Console.

You can export the workload application template by using the procedure available in the Dynamic Workload Console, as described in Reusing a workload in another environment on page 528.

You can import the workload application template by using the procedure available in the Dynamic Workload Console, as described in Reusing a workload in another environment on page 528. When importing, you can choose whether to import the workload application bound to the objects it contains, or to import only the contents of the workload application eliminating any ties to the workload application so that objects can be managed in the new environment with more flexibility.

To export and import the workload application, you can also use the **wappman** command line as described in the following topics.

Storing scheduling objects in folders is supported starting from version 9.5 and later. When importing a workload application from a product version earlier than 9.5, all imported objects are stored in the root folder. Ensure you have write access to the root folder before starting the import.

The export process produces a compressed file containing five files:

***workload applicationname*_Definitions.UTF8.xml**

A file in XML format containing a definition of all the objects referenced in the workload application. The definitions are deployed in the target environment to populate the target database with the same objects existing in the source environment. Do not edit this file.

***workload applicationname*__Mapping.UTF8.properties**

A mapping file that the target user modifies replacing the names of the objects in the source environment with the names that the objects will have in the target environment.

When the import process is performed from the **wappman** command line, you can optionally request that the mapping file is automatically modified according to rules defined using regular expressions and specified in one of the following ad-hoc files:

- *workload applicationname*_BasicRegExpMapping.UTF8.rules
- *workload applicationname*_AdvancedRegExpMapping.UTF8.rules

These files are produced by the export process and, if used, must be properly customized.

***workload applicationname*_SourceEnv_reference.txt**

A file containing reference information about the workstations used in the workload application and other information that can be useful to correctly map the source environment to the target environment.

***workload applicationname*_BasicRegExpMapping.UTF8.rules**

A file containing rules, defined using basic regular expressions, to modify the mapping file. Optionally customize this file according to the names that the objects will have in the target environment. The import process performed from the **wappman** command line then applies the defined rules to modify the mapping file.

***workload applicationname*_AdvancedRegExpMapping.UTF8.rules**

A file containing rules, defined using advanced regular expressions, to modify the mapping file. Optionally customize this file according to the names that the objects will have in the target environment. The import process performed from the **wappman** command line then applies the defined rules to modify the mapping file.

You can skip objects when importing a workload application template by adding the SKP prefix before the object you want to skip. If you skip an object, an object with the same name must be present in the target environment, otherwise an error message is returned.

Each line in the mapping file specifies the objects to be imported in the new environment. To avoid importing a specific object, modify the *mapping_properties_file* by adding the **SKP** prefix before each object you want to skip.

Consider a mapping file containing two objects, as follows:

```
JOBSTREAM_TESTJS=TESTJS
JOB_TESTJOB1=TESTJOB1
```

If the `JOB_TESTJOB1` job already exists in the target environment, modify the *mapping_properties_file* by adding the **SKP** prefix before the `JOB_TESTJOB1` job, as follows:

```
JOBSTREAM_TESTJS=TESTJS
SKPJOB_TESTJOB1=TESTJOB1
```

Reusing a workload in another environment

You can use workload application template to utilize workloads that have been used in other environments. For example, when necessary, it is possible to import a workload application template in a different environment.

About this task

The purpose of this task is to export a workload from an environment and import into another environment.

The scope of the scenario is to export the workload for managing the payroll in a company from the test environment and import it in the production environment with a Workstation in production called **MAIN_WKS**, replacing existing job streams. The job stream named **PAYROLL_PROC_21**, that monthly creates and uploads payrolls for the year 2021 and the event rule **PAYROLL_21**, that monthly send a notification to your employees will be exported in a template called **PAYROLL.WAT** and imported in the production environment.

1. Export the existing job stream by creating a workload application template
    a. From the **Design** menu, click **Workload Designer** page and then select the engine.
    b. In **Explore area**, click **Create new +** and select **workload application template**.
    c. In **General info**, type the name `PAYROLL`
    d. From the **Details view**, add the job stream `PAYROLL_PROC_21` by clicking the plus button near **Job stream**.
    e. Save the template, and then click on the export button on the workload application template row to export it.
2. Import the workload application template in the production environment.
    a. From the **Design** menu, click **Workload Designer** page, click **Import**, and select **Import workload application**.
    b. Define the engine and upload the exported template.
    c. In **Jobs**, map the jobs of the workload application template and in **Job Streams** map `PAYROLL_PROC_21` as **target job stream**.
    d. In **Workstation**, select the `MAIN_WKS` workstation in the **Target** field .
    e. In **Event Rules** map the event rule `PAYROLL_21` as a **target event rule**.
    f. Click **Import**.

Results
You have exported the job stream named `PAYROLL_PROC_21` containing the jobs and the event rule `PAYROLL_21` as a workload application template and imported in the production environment.

Resolving the mapping file

The mapping file produced by the export process of a workload application contains a list of elements, some of which are dependent on the topology of the environment in which it is used. These elements need to be customized to reflect the target environment.

To use the workload application in a new environment, modify the mapping file to reflect the destination environment using the information provided in the reference file and then perform an import operation. The import operation is performed passing the mapping file and the definition file as input to the wappman command.

The wappman command can be used to import, export, replace, list, display, and delete a workload application. See WAPPMAN on page 538 for the complete command line usage and syntax to perform these actions on a workload application and any special considerations to be aware of.

During the export process, the objects contained in the workload application are extracted to the definitions file with the same definition they have in the source environment. The definitions file can contain a complete object definition or in some cases, only a name or reference to the object that is extracted. Simple references and not a full object definition is extracted for those objects that require to be mapped to an object already present in the target environment. For some objects extracted by reference, the object definition is written to the mapping file which requires a customization to map

the objects in the HCL Workload Automation source environment to the environment where the workload application will be deployed.

The mapping file can be viewed and edited with a text editor. It is organized in sections and contains comments to assist you in assigning the correct values to the elements.

As an alternative, when the import process is performed from the **wappman** command line, you can optionally request that the mapping file is automatically modified according to rules defined using regular expressions and specified in one of the following ad-hoc files:

- *workload applicationname*_BasicRegExpMapping.UTF8.rules
- *workload applicationname*_AdvancedRegExpMapping.UTF8.rules

These files are produced by the export process and, if used, must be properly customized. For additional details, see Using regular expressions to modify the mapping file on page 535.

The following table outlines all of the objects that can be contained in a workload application or that are referenced by another element in the workload application and how the export process manages them.

Table 73. Objects extracted during the export process

Object type	What is exported to the definitions file?	What requires customization in the mapping file?	What happens during the import?
Folder	Reference only	Folder name. Map the original folder name to either the root, or to an existing folder in the destination environment.	The folder is not imported. A definition is not imported because the folder is already defined in the target environment, however, the folder names need to be mapped.
Job stream	Object definition including the job stream start condition, if specified.	Workstation Job stream name Job alias Job stream start condition, if specified	Job stream is created in the database. **Restriction:** If the job stream has a dependency on a job stream or job external to the workload application, the mapping file contains a reference to the name of the external job stream or job and the relative workstation definition, however, the definitions file does not contain the job stream or job definition. The name in the mapping file must be mapped to an existing job stream or job in the target

Table 73. Objects extracted during the export process (continued)

Object type	What is exported to the definitions file?	What requires customization in the mapping file?	What happens during the import?
			environment to successfully import the workload application.
Job	Object definition	Workstation Job name Affinity Password variables found in the JSDL	Job is created in the database. If the job has a dependency on a job that is defined in a job stream external to the workload application, the mapping file contains a reference to te following objects: the name of the external job stream, the job defined int he external job stream, the workstation definitions of the job stream, and the workstation definitions of the job. However, the definitions file does not contain the job definition. The name in the mapping file must be mapped to an existing job stream or job in the target environment to successfully import the workload application. Affinity relationships cause jobs to run on the same workstation. The workstation on which the first job runs is chosen dynamically and the affine job or jobs run on the same workstation. The jobs must belong to the same job stream. When a job with an affinity is exported, the name of the job is added to the mapping file. Variables in the JSDL use the format ${password:ws#user}. Only workstations are generically represented. The user field is copied as is to the target environment. Variables should be used for user names.
Run cycle variable tables Job stream variable tables	Object definition	Table name Variable name	Variable table is created in the database. Workstation and default variable tables are extracted by reference and written to the map file. The value associated to the variable can be modified, but not variable names.

Table 73. Objects extracted during the export process (continued)

Object type	What is exported to the definitions file?	What requires customization in the mapping file?	What happens during the import?
			Avoid associating the default variable table to job streams and run cycles.
User	Nothing. Neither an object definition nor a reference is made in the reference file.		The user must exist in the target environment. Variables should be used to refer to users to make the workload application more flexible for reuse.
Calendar	Object definition	Calendar name	Calendar is created in the database
Run cycle	Object definition	Run cycle name	Run cycle is created in the database
Run cycle group	Object definition	Run cycle group name	Run cycle group is created in the database
Internetwork dependencies	Referenced job or job stream is not exported since it belongs to a different engine.	Name of the network agent workstation that handles the follows dependencies between the local network and the remote network.	Internetwork dependency is added to the job or job stream.
External dependencies	The referenced job or job stream is exported only if it belongs to the workload application template.	If the referenced job or job stream does not belong to the workload application template, assign a name to the job, or job stream that corresponds to a job or job stream that already exists in the target environment.	External dependency added to the job or job stream
Resources	Object definition	Workstation Resource name	Resources are created in the database
Global prompts	Object definition	Variables used in the definition	Global prompts are created in the database. Variables can be used. Since they are resolved using the default table, the variable used in a global

Table 73. Objects extracted during the export process (continued)

Object type	What is exported to the definitions file?	What requires customization in the mapping file?	What happens during the import?
			prompt can be mapped to a variable in the target environment.
Workstations	Reference only	Name	Not imported. Workstations are extracted to the definition file as a reference. A definition is not imported because the workstations are already defined in the target environment, however, their names do need to be mapped.
Workstation class	Reference only	Name	Not imported. Workstation classes are extracted to the definition file as a reference. A definition is not imported because the workstation classes are already defined in the target environment, however, their names do need to be mapped.
Variable	Object definition	Value	Imported. Variables are used in several places in a job stream definition. A reference is added to the definitions file.
Event rules	Object definition	Name	Event rules are created in the database for the workstations and job streams included in the template. **Note:** Any scheduling objects referenced in the **Actions** and **Events** defined in the event rule that do not belong to the user's security domain are not imported into the

Table 73. Objects extracted during the export process (continued)

Object type	What is exported to the definitions file?	What requires customization in the mapping file?	What happens during the import?
			target environment and the import operation fails.

Example

The following are excerpts from some of the files contained in the compressed file, created by the export of the workload application, that are used to ready the files for deployment in the target environment.

Table 74. Resolving the mapping file

Definitions file

```
<model:JobStream carryforward=
  "true" draft="false" folder=
   "$FOLDER_/APPS/APP1$/"
 iskey="false" limit="55" name=
  "$JOBSTREAM_BADPBN34_JS1_1I$"
 onrequest="false" priority="100"
 workstation="$WORKSTATION_
  BADPBN34_WC1$">
     <model:runcycles/>
     <model:matching>
       <model:sameDay/>
     </model:matching>
     <model:restrictions/>
     <model:dependencies/>
     <model:jobs>
     <model:job confirmed="false"
definition=
   "$WORKSTATION_MDM112097$
  #$JOB_BADPBN34_J1$"
 isCritical="false" iskey="false"
  name="$JOB_BADPBN34_J1$"
priority="10">
       <model:restrictions/>
       <model:dependencies>
        <model:predecessor target=
         "$WORKSTATION_BADPBN34_
         WC1$#$JOBSTREAM_BADPBN34_
         JS2$.@">
          <model:matching>
           <model:previous>
          </model:matching>
       </model:predecessor>
      </model:dependencies>
```

Mapping file

```
#Folder names
#Replace the value with the
name of a Folder that already
exists in the target environment.
#
FOLDER_/=/
FOLDER_/APPS/APP1=/APPS/APP1
#
#Job stream names
#Job stream names in this
section refer to job streams
that will be created during
the import process.
#Assign a name for the job
stream that does not already
exist in the target environment.
#
JOBSTREAM_BADPBN34_JS1_1I=
  BADPBN34_JS1_1I
#
#Workstation names
#Replace the value with the name
 of a workstation that already
 exists in target environment
#Refer to the MAIN_TEMPLATE_
  SourceEnv_reference.txt file
  for details about the
  workstation.
#
#This workstation is of type
  Manager
WORKSTATION_MDM112097=MDM112097
#This workstation is of type
```

Reference information file

```
CPUNAME $WORKSTATION_MDM112097$
  DESCRIPTION "Sample master domain
   manager"
  OS UNIX
  NODE MDM112097.romelab.it.ibm.com
   TCPADDR 35111
  TIMEZONE Europe/Rome
  DOMAIN MASTERDM
  FOR MAESTRO
    TYPE MDM
    AUTOLINK ON
    BEHINDFIREWALL ON
    FULLSTATUS ON
    SERVER A
END
```

Table 74. Resolving the mapping file

(continued)

Definitions file	Mapping file	Reference information file
</model:job> </model:jobs> </model:JobStream>	Agent WORKSTATION_MDM112097_1= MDM112097_1 #This workstation is of type Broker WORKSTATION_MDM112097_DWB= MDM112097_DWB	

The reference information file indicates that the workstation named, MDM112097, is of type, master domain manager, running on a UNIX operating system. The definitions file contains references to the name of the workstation so, the entry in the mapping file, `WORKSTATION_MDM112097=MDM112097`, must be updated. Replace `MDM112097` with the name of a workstation that already exists in the target environment, and has the same characteristics as those outlined in the reference information file.

Definitions file

```
<screen><model:JobStream carryforward="true" draft="false" folder="$FOLDER_/APPS/APP1$/"
iskey="false" limit="55" iskey="false" limit="55" name="$JOBSTREAM_BADPBN34_JS1_1I$" onrequest="false"
priority="100" workstation="$WORKSTATION_BADPBN34_WC1$"> model:runcycles/> <model:matching> <model:sameDay/
> </model:matching> <model:restrictions/> <model:dependencies/> <model:jobs> <model:job confirmed="false"
defintion="$WORKSTATION_MDM112097$#$JOB_BADPBN34_J1$" isCritical="false" iskey="false" name="$JOB_BADPBN34_J1$"
priority="10"> <model:restrictions/> <model:dependencies> <model:predecessor target="$WORKSTATION_BADPBN34_WC1$#
$JOBSTREAM_ BADPBN34_JS2$.@"> <model:matching> <model:previous> </model:matching> </model:predecessor>
</model:dependencies> </model:job> </model:jobs>< /model:JobStream> </screen>
```

Using regular expressions to modify the mapping file

You can optionally request that the mapping file produced by the export process is automatically modified by the import process, according to rules defined using regular expressions.

The mapping file produced by the export process of a workload application contains a list of elements, some of which are dependent on the topology of the environment in which it is used. These elements need to be customized to match the target environment. When the import process is performed from the **wappman** command line, you can optionally request that the mapping file is automatically modified according to rules defined using regular expressions and specified in one of the following ad-hoc files:

***workload applicationname*_BasicRegExpMapping.UTF8.rules**

This file contains rules defined using basic regular expressions that include only '*' and '?' wildcard characters.

***workload applicationname*_AdvancedRegExpMapping.UTF8.rules**

This file contains rules defined using advanced regular expressions according to Java standards. For additional details about advanced regular expressions, see the related documentation.

These files are produced by the export process and, if used, must be properly customized.

Select the file that better fits your needs and customize the regular expressions according to the names that the objects will have in the target environment. The import process, performed from the **wappman** command line, then applies the defined rules to modify the value of the elements included in the mapping file.

Each file is organized in sections, one for each type of object that can be contained in a workload application. For each section, you can find comments including examples of rules, defined using regular expressions, that apply to the specific object. For example:

```
[JOBSTREAM]
#*_DEV=*_PRD
#A24Y?=B42X?
```

Uncomment the provided examples or add your own rules. For example:

```
[JOBSTREAM]
*_DEV=*_PRD
A24Y?=B42X?
J*=A*
```

Each rule is composed of two regular expressions. The regular expression on the left defines the search pattern, the regular expression on the right defines the replacing value of each matching element. During the import process, both the search and the replace actions are performed on the value of the elements included in the mapping file.

If an object of your workload application matches more than one entry in the file, only the rule included in the last matching entry is applied. For example:

```
According to the above rules:
- the job stream JS1_DEV is renamed as AS1_DEV
- the job stream CCB_DEV is renamed as CCB_PRD
- the job stream A24YK is renamed as B42XK
```

Each line in the mapping file specifies the names to be assigned to objects when importing the workload application template. You can use the -**translationRules** parameter to specify a file to override these names using regular expressions. The compressed file representing the workload application template contains two files, named *wat_name*_`BasicRegExpMapping.UTF8.rules` and *wat_name*_`AdvancedRegExpMapping.UTF8.rules` with examples about how to build the `translationRules` file.

Consider the following example, which imports a workload application into a new environment, using the regular expressions specified in the *translation_rules_file* to assign names to the new objects:

```
wappman -import definition_xml_file mapping_properties_file -translationRules translation_rules_file
```

If the *mapping_properties_file* contains the `JOB_TEST_DECOMPRESSION=TEST_DECOMPRESSION` string, and the *translation_rules_file* contains a rule such as `[JOB]^(TEST_)(.+)=$2`, the resulting job in the target environment is named DECOMPRESSION.

Deploying a workload application

Deploying a workload application is a two-step process beginning with customizing the mapping file and then importing the mapping file and definitions file into the new HCL Workload Automation environment.

About this task

1. Extract the content of the workload application template from the compressed file created by the export operation.
2. Customize the mapping file. For each object listed in the mapping file, either assign the name of an existing object in the target environment, or rename it with the name you want the object to have in the target environment. For information about modifying the mapping file, see Resolving the mapping file on page 529 . You can optionally modify the mapping file according to rules defined using regular expressions and specified in ad-hoc files. These files are produced by the export process and, if used, must be properly customized. For information about using regular expressions, see Using regular expressions to modify the mapping file on page 535.
3. When importing the workload application template into the target environment, you have the option to import the objects contained in the workload application template and maintain an association to the workload application, or you can import only the contents of the workload application, eliminating any ties to the workload application so that the objects can be managed freely in the target environment. Select the objects you do not want to import. If you choose to skip an object, an object with the same name must be present in the target environment, otherwise an error message is returned.
   The import operation can be performed either from the command line, submitting the **wappman -import** command, or from the Dynamic Workload Console, selecting **Design > Import Workload Application**.
   Result
   See WAPPMAN on page 538 for more details about the command usage and syntax. See the topic about importing a workload applicationtemplate from theDynamic Workload Console in the *Dynamic Workload Console User's Guide*.

Results

All of the HCL Workload Automation objects defined in the *definition_xml_file* are created in the target environment, if they do not already exist. You update the mapping file to resolve references to objects that need to be customized to reflect the target environment where the workload application will be deployed.

What to do next

You can subsequently update a workload application with a replace operation. The behavior of the replace operation depends on whether or not the objects to be replaced are tied to a workload application. If the objects are tied to a workload application then, when an updated version of the workload application template is reimported into the target environment, the objects in the target environment are updated if they are still present in the database, created if they are no longer present, and deleted if they are no longer present in the updated workload application template. If the objects are not tied to a workload application or, if the objects are tied to a workload application different from the one specified with the replace operation, then the objects are not replaced and an error message is issued. For more information about the replace operation see the WAPPMAN on page 538.

There are two ways in which the workload application can be updated or modified:

Modifying the template in the source environment

An updated version of the template can be deployed again into the target environment. Any objects already present in the HCL Workload Automation database of the target environment are replaced with the updated versions, any objects that do not already exist in the target environment are created, and objects are deleted from the target environment if the object definition has been removed from the updated workload application. The same mapping file used to originally deploy the workload application can be used to update it, customizing any new objects being deployed with the update.

Modifying the instance in the target environment

After a workload application has been deployed, you can modify the contents of the workload application, such as, adding a new job to a job stream, modifying a job definition, or removing a job or job stream. However, since the contents are still tied to workload application, if it is replaced with a revised workload application template, then the changes are not maintained.

If, instead, only the contents of the workload application template were imported, then since they are not tied to the workload application, they can be modified freely. A subsequent replace operation using the -contents option will replace the objects that are present and maintain any other changes that were made since the import.

WAPPMAN

Imports, replaces, deletes, exports, displays and lists a workload application.

Authorization

You must have ***add*** access if you import a new workload application. If the object already exists in the database you must have ***modify*** access to the object.

The justification is propagated to all objects created, updated, or deleted.

Syntax

wappman [*connection_parameters*]
[-u]|[-V]
| **{-import|-replace}** <*definition_xml_file*> <*mapping_properties_file*>
[-**translationRules** <*translation_rules_file*>]
[-**workloadApplicationName** *[folder/]*<*workload_application_name*>]
[-contents]
-list
-delete *[folder/]*<*workload_application_name*> **[-noContents]**
-export *[folder/]*<*workload_application_template*> [-path <*export_path*>]
[-zip <*export_zip_name*>]
-display *[folder/]*<*workload_application_name*>

Arguments

[*connection_parameters*]

A list of or a custom file containing the connection parameters to be used to connect to the server where you want to import, list, delete, export or display the workload application. You can use connection parameters to override the values specified in the `useropts` and `localopts` files. To determine which connection properties to use, a check is made in the following order:

1. Parameters specified in the command string itself.
2. Parameters specified in the custom properties file.
3. The `useropts` file.
4. The `localopts` file.
5. The `jobmanager.ini` file.

Valid values include:

[-file <*custom_properties_file*>]

The custom properties file where you can specify connection parameters that override the values specified in the `useropts`, `localopts`, and `jobmanager.ini` files. Connection parameters specified in the custom properties file must have the following syntax:

```
HOST=<hostname>
PORT=<port>
PROTOCOL=<http/https>
USERNAME=<username>
PASSWORD=<password>
```

[-host <*hostname*>]

The name of the host that you want to access by using wappman command line.

[-port <*port_number*>]

The TCP/IP port number used to connect to the specified host.

[-protocol {http | https}]

The protocol used to connect to the specified host.

[-jwt *JSON Web Token*]

Specify the JWT to be used for authentication between the master domain manager and agents. You can retrieve the token from the Dynamic Workload Console. This parameter is mutually exclusive with the **username** and **password** parameters. The JWT authentication applies to the commands you launch in the shell where you specify the JWT.

[-username *user_name*]

An HCL Workload Automation user with sufficient privileges to perform the operation. This parameter is mutually exclusive with the **jwt** parameter.

[-password *password*]

The password of the HCL Workload Automation user. This parameter is mutually exclusive with the **jwt** parameter.

Note: If host, port, and protocol parameters are specified in a file, all of them must be specified in the same file.

-u

Displays command usage information and exits.

-V

Displays the command version and exits.

{-import|-replace} <*definition_xml_file*> <*mapping_properties_file*> [-translationRules <*translation_rules_file*>] [-workloadApplicationName <*workload_application_name*>] [-contents]

The name of the definitions file and mapping file to use when importing or replacing a workload application. The operation of importing a workload application is the same as deploying a workload application.

-import

After customizing the mapping file, the mapping file together with the definitions file are passed as input to the **wappman -import** command to deploy the workload application in an HCL Workload Automation environment different from the source environment where the workload application template was originally created.

-import -contents

Imports only the objects contained in the workload application template and not the workload application itself. Any association to the workload application is not created.

-replace

The behavior of the replace operation depends on whether or not the objects to be replaced are tied to a workload application.

If the objects are tied to a workload application then, when an updated version of the workload application template is reimported into the target environment, the objects in the target environment are updated if they are still present in the database, created if they are no longer present, and deleted if they are no longer present in the updated workload application template.

If the objects are not tied to a workload application or, if the objects are tied to a workload application different from the one specified with the replace operation, then the objects are not replaced and an error message is issued.

The replace operation fails if an object external to the workload application references an object in the workload application and with the replace operation the referenced object is deleted because it is no longer present in the updated workload application.

If references were made from the workload application to an external element, the reference is deleted with the replace action.

The same mapping file used to originally deploy the workload application can be used to update it, making any necessary changes to reflect the updated workload application. The mapping file together with the definitions file are passed as input to the **wappman -replace** command.

-replace -contents

The behavior of the replace operation depends on whether or not the objects to be replaced are tied to a workload application. If the objects already exist and are not tied to a workload application, then the objects are replaced, but the workload application is not created. If the objects do not exist in the target environment, then they are created without any ties to the workload application. If the objects already exist and are tied to a workload application, then the replace operation fails and an error message is issued.

Additional optional parameters that you can specify are:

-translationRules

The name of the file containing rules defined using regular expressions that **wappman -import|-replace** command will use to customize the mapping file. For more information about using reular expressions to customize the mapping file, see Using regular expressions to modify the mapping file on page 535.

-workloadApplicationName

The name of the workload application, in case you want to import it with a new name.

-list

Lists all of the workload applications present in the environment.

-delete <*workload_application_name*>

Deletes the specified workload application and all the objects that were added to the environment when the workload application was originally deployed.

-delete <*workload_application_name*> -noContents

The operation deletes the specified workload application, but maintains the objects that were originally tied to it. There is more flexibility in updating the objects as there is no longer an association to the workload application.

-export <*workload_application_template*>

Creates the compressed file, named workload application template, containing all the files and information required to deploy the workload application into the target environment.

-path

The file path where the workload application template is exported.

-zip

The name of the workload application template zip file.

-display <*workload_application_name*>

Displays the contents of the specified workload application.

Displays all of the objects contained in the workload application that were created during the import process.

Comments

You can skip objects when importing a workload application template by adding the SKP prefix before the object you want to skip. If you skip an object, an object with the same name must be present in the target environment, otherwise an error message is returned.

Each line in the mapping file specifies the objects to be imported in the new environment. To avoid importing a specific object, modify the *mapping_properties_file* by adding the **SKP** prefix before each object you want to skip.

Consider a mapping file containing two objects, as follows:

```
JOBSTREAM_TESTJS=TESTJS
JOB_TESTJOB1=TESTJOB1
```

If the `JOB_TESTJOB1` job already exists in the target environment, modify the *mapping_properties_file* by adding the **SKP** prefix before the `JOB_TESTJOB1` job, as follows:

```
JOBSTREAM_TESTJS=TESTJS
SKPJOB_TESTJOB1=TESTJOB1
```

Storing scheduling objects in folders is supported starting from version 9.5 and later. When importing a workload application from a product version earlier than 9.5, all imported objects are stored in the root folder. Ensure you have write access to the root folder before starting the import.

Users can decide to maintain an audit trail recording any changes they perform and the related justifications. To enable the justification option, set up in a system shell the HCL Workload Automation environment variables listed below before running the **wappman** commands:

IWS_DESCRIPTION

Specify the description to be recorded for each change performed by commands in the shell. The maximum length for this value is 512 characters. A warning message displays if you exceed the maximum and excess characters are truncated.

IWS_CATEGORY

Specify the category to be recorded for each change performed by commands in the shell. The maximum length for this value is 128 characters. A warning message displays if you exceed the maximum and excess characters are truncated.

IWS_TICKET

Specify the ticket to be recorded for each change performed by commands in the shell. The maximum length for this value is 128 characters. A warning message displays if you exceed the maximum and excess characters are truncated.

To set up the environment, proceed as follows, depending on your operating system:

On Windows operating systems

type one or more of the following commands, depending on the variable you want to define:

```
set IWS_CATEGORY='my category'
set IWS_DESCRIPTION='my desc'
set IWS_TICKET=12345
```

On UNIX operating systems

type one or more of the following commands, depending on the variable you want to define:

```
export IWS_CATEGORY='my category'
export IWS_DESCRIPTION='my desc'
export IWS_TICKET=12345
```

After setting the environment variables, all the commands you enter in the shell inherit the values you defined for the shell. The auditing information from the shell is stored as is in the Tivoli Common Reporting auditing reports. While the information you provide in the Dynamic Workload Console is guided using a pull-down menu, the information you provide from the command line is not filtered.

For more information about the justification option Tivoli Common Reporting auditing reports, see the section about keeping track of changes and Tivoli Common Reporting in *Dynamic Workload Console User's Guide*.

Logs and traces

You can configure the wappman command line by modifying the parameters in the `CLI.ini` file located in the path *TWA_home*`/TWS/ITA/cpa/config`. This file contains parameters for the message logs and trace files related to workload applications. You should only change the parameters if advised to do so in the HCL Workload Automation documentation or requested to do so by HCL Software Support.

Example

Examples

To import a workload application into a new environment, run:

```
wappman -import <definition_xml_file> <mapping_properties_file>
```

To import only the contents of a workload application into a new environment, without any ties to the workload application, run:

```
wappman -import <definition_xml_file> <mapping_properties_file> -contents
```

To replace an existing workload application, run:

```
wappman -replace <definition_xml_file> <mapping_properties_file>
```

To replace objects that are not tied to any workload applications in an environment, run:

```
wappman -replace <definition_xml_file> <mapping_properties_file> -contents
```

To delete a workload application, run:

```
wappman -delete <workload_application_name>
```

To free objects from any ties to a workload application, run:

```
wappman -delete <workload_application_name> -noContents
```

To list all the workload applications available in an environment, run:

```
wappman -list
```

To display a specific workload application, run:

```
wappman -display <workload_application_name>
```

To export a specific workload application from a remote host, by using a custom properties file, run:

```
wappman -export <workload_application_name> -path <export_path>
  -host <remote_host> l <custom_properties_file>
```

See also

For information about defining a workload application template in the source environment, see the section about creating a workload application template in *Dynamic Workload Console User's Guide*.

For more information about managing a workload application in the target environment, see the section about deploying a workload application in *Dynamic Workload Console User's Guide*.

Chapter 12. Managing objects in the plan - conman

The HCL Workload Automation production plan environment is managed using the **conman** command-line program. The **conman** program is used to start and stop processing, alter and display the `Symphony` production plan, and control workstation linking in a network. It can be used from the master domain manager and from any fault-tolerant agent in the HCL Workload Automation network.

This chapter is divided into the following sections:

- SETUPCONMAN Setting up the conman command-line program on page 545
- Running commands from conman on page 552
- JOBSELECT Selecting jobs in commands on page 556
- JSSELECT Selecting job streams in commands on page 571
- CONMAN COMMANDS on page 582

SETUPCONMAN Setting up the conman command-line program

About this task

The **conman** command-line program manages the production plan environment.

You can use the **conman** program from the master domain manager and from any fault-tolerant agent workstation in the HCL Workload Automation network. On dynamic agents, the following conman commands are supported:

- listsucc. For more information, see LISTSUCC on page 626.
- rerunsucc. For more information, see RERUNSUCC on page 639.

You can decide to maintain an audit trail recording any changes they perform and the related justifications. To enable the justification option, set up in a system shell the HCL Workload Automation environment variables listed below before running any **conman** commands:

IWS_DESCRIPTION

Specify the description to be recorded for each change performed by commands in the shell. The maximum length for this value is 512 characters. A warning message displays if you exceed the maximum and excess characters are truncated.

IWS_CATEGORY

Specify the category to be recorded for each change performed by commands in the shell. The maximum length for this value is 128 characters. A warning message displays if you exceed the maximum and excess characters are truncated.

IWS_TICKET

Specify the ticket to be recorded for each change performed by commands in the shell. The maximum length for this value is 128 characters. A warning message displays if you exceed the maximum and excess characters are truncated.

For more information about the justification option, see the section about keeping track of changes in Dynamic Workload Console User's Guide.

You can customize the logging and tracing level for conman commands by setting the relevant values for the **tws.loggers.trcConman.level** property, available in the `Conman.ini`.

The `Conman.ini` file is located in the following path:

On Windows™ operating systems

TWA_home`\TWS\ITA\cpa\config\template`

On UNIX™ operating systems

TWA_DATA_DIR`/ITA/cpa/config/template`

Setting up the conman environment

About this task

This section gives you the information about the setup you can choose for your **conman** environment.

Note: On Windows® systems, before running **conman** ensure the code-page and fonts are correctly set in the DOS shell to avoid bad character conversion. For additional information about the required settings, see the *Internationalization notes* section in the Release Notes .

Terminal output

The output to your computer is determined by the shell variables defined in the `tws_env` script, which you run before running command line commands. If any of the following variables is not set, the standard shell variables, ***LINES*** and ***COLUMNS***, are used. The variables can be set as follows:

MAESTROLINES

Specifies the number of lines per screen. The default is **-1**. At the end of each screen page, **conman** does not pause at the end of a page. If ***MAESTROLINES*** (or ***LINES***) is set to a positive number, **conman** prompts to continue.

Use of MAESTROLINES is recommended since the LINES variable is a shell operating system variable and in most operating systems it is automatically reset by the operating system itself.

MAESTROCOLUMNS

Specifies the number of characters per line. The following options are available:

- Less than 120
- Equal to or more than 120

MAESTRO_OUTPUT_STYLE

Specifies how object names are displayed. If set to **LONG**, full names are displayed. If set to any value other than **LONG**, long names are truncated to eight characters followed by a plus sign (+).

Offline output

The **;offline** option in **conman** commands is used to print the output of a command on a specified file. When you include it, the following shell variables control the output:

MAESTROLP

Specifies the destination file of a command's output. Set it to one of the following:

> *outputfilename*

Redirects output to a file and overwrites the contents of that file. If the file does not exist, it is created. For example:

```
MAESTROLP='> /tmp/outputfilename.txt'
```

>> *outputfilename*

Redirects output to a file and appends the output to the end of that file. If the file does not exist, it is created.

| *command*

Pipes output to a system command or process. The system command is run whether or not output is generated.

|| *command*

Pipes output to a system command or process. The system command is not run if there is no output.

The default value for ***MAESTROLP*** is **| lp -tCONLIST** which pipes the command output to the printer and places the title “CONLIST" in the printout's banner page.

MAESTROLPLINES

Specifies the number of lines per page. The default is **60**.

MAESTROLPCOLUMNS

Specifies the number of characters per line. The default is **132**.

The variables must be exported before running **conman**.

Selecting the conman prompt on UNIX®

About this task

The **conman** command prompt is, by default, a percent sign (%). This is defined in the *TWS_home*`/localopts` file. The default command prompt is a dash (-). To select a different prompt, edit the **conman** prompt option in the `localopts` file and change the dash. The prompt can be up to 10 characters long, not including the required trailing pound sign (#).

```
#--------------------------------------------------------------------------
# Custom format attributes
#
date format =               1         # The possible values are 0-ymd, 1-mdy,
2-dmy, 3-NLS.
composer prompt =           -
conman prompt =             %
switch sym prompt =        <n>%
#--------------------------------------------------------------------------
```

RUNCONMAN Running the conman program

About this task

To configure the environment for using conman, set the *PATH* and *TWS_TISDIR* variables by running one of the following scripts:

On UNIX® operating systems:

- `. ./`*TWS_home*`/tws_env.sh`

On Windows® operating systems:

- *TWS_home*`\tws_env.cmd`

Then use the following syntax to run commands from the conman user interface:

conman [*custom_parameters*][*connection_parameters*] ["*command*[**&**[*command*]...] [**&**]"]

where:

custom_parameters

Sets the working directory or current folder using the following syntax:

```
-cf /foldername
```

The current directory from where commands are submitted. The default current directory is the root (/).

-file *customPropertiesFile*

The custom properties file where you can specify connection parameters or custom parameters that override the values specified in the `useropts`, `localopts` and `jobmanager.ini` files. Connection parameters specified in the custom properties file must have the following syntax:

```
HOST=hostname
   PORT=port
   PROTOCOL=http/https
   PROXY=proxyName
   PROXYPORT=proxyPortNumber
   PASSWORD=password
```

```
TIMEOUT=seconds
USERNAME=username
CURRENT FOLDER=/foldername
```

connection_parameters

If you are using **conman** from the master domain manager, the connection parameters were configured at installation and do not need to be supplied, unless you do not want to use the default values.

If you are using **conman** from the command line client on another workstation, the connection parameters might be supplied by one or more of these methods:

- Stored in the `localopts` file
- Stored in the `useropts` file
- Supplied to the command in a parameter file
- Supplied to the command as part of the command string

For an overview of these options see the section about setting up options for using the user interfaces in *User's Guide and Reference*. For full details of the configuration parameters see the topic on configuring the command-line client access in the *Administration Guide*.

Note: If you are using conman from the command-line client on another workstation, for the following commands the conman command line connects to the server by using an HTTPS connection:

- Rerunsucc
- Listsucc

In this case, the command-line client assembles the full set of connection parameters in the following order:

1. Parameters specified in the command string itself
2. Parameters specified in the custom properties file
3. The useropts file
4. The localopts file
5. The jobmanager.ini file

Valid values include:

[-host *hostname*]

The name of the host that you want to access by using wappman command line.

[-port *port_number*]

The TCP/IP port number used to connect to the specified host.

[-protocol {*http* | *https*}]

The protocol used to connect to the specified host.

[-proxy *proxyName*]

The name of the proxy server used when accessing a command-line client.

[-proxyport *proxyPortNumber*]

The TCP/IP port number of the proxy server used when accessing using a command-line client.

[-jwt *JSON Web Token*]

Specify the JWT to be used for authentication between the master domain manager and agents. You can retrieve the token from the Dynamic Workload Console. This parameter is mutually exclusive with the **username** and **password** parameters. The JWT authentication applies to the commands you launch in the shell where you specify the JWT.

[-username *user_name*]

An HCL Workload Automation user with sufficient privileges to perform the operation. This parameter is mutually exclusive with the **jwt** parameter.

[-password *password*]

The password of the HCL Workload Automation user. This parameter is mutually exclusive with the **jwt** parameter.

[-timeout *seconds*]

The timeout in seconds when accessing using a command-line client. The default is 3600 seconds.

If host, port, and protocol parameters are specified in a file, all of them must be specified in the same file.

You can invoke the conman command-line both in *batch* and in *interactive* mode.

When running conman in *interactive* mode, you at first launch the conman command-line program and then, from the conman command-line prompt you run commands one at a time, for example:

```
conman -username  admin2  -password  admin2pwd
   ss @+state=hold;deps
   dds sked5;needs=2 tapes
```

You can also launch the same command using JWT instead of username and password:

```
conman -jwt aòsdfjasldfkjasdflòjahsdfjlkasd
   ss @+state=hold;deps
   dds sked5;needs=2 tapes
```

When running conman in *batch* mode, you first launch the conman command-line program specifying as input parameter the command to be issued. Once the command is processed, the conman command-line program exits, for example

```
conman"sj&sp"
```

When issuing commands from conman in batch mode make sure you enclose the commands between double quotation marks. The following are examples of using batch mode to issue more than one command from within conman:

- conman runs the **sj** and **sp** commands, and then quits:

```
conman "sj&sp"
```

- conman runs the **sj** and **sp** commands, and then prompts for a command:

```
conman "sj&sp&"
```

- conman reads commands from the file **cfile**:

```
conman  cfile
```

- commands from the file **cfile** are piped to conman:

```
cat cfile | conman
```

Note:

- On Windows workstations, if the User Account Control (UAC) is turned on and the UAC exception list does not contain the `cmd.exe` file, open the DOS command prompt shell with the `"Run As Admnistrator"` option to run composer on your workstation as a generic user different from Administrator or HCL Workload Automation user.
-

  If you install an HCL Workload Automation component, either agent or server, in the `C:\` or `C:\Program Files\` directories, the **twsuser** or any user other than the Administrator might encounter issues when running commands from the command-line interface due to security settings. For example, the resource command-line interface might fail to run commands, and the conman command might display error or warning messages. To prevent these issues, open the command prompt with the **Run as administrator** option to run commands.

Control characters

You can enter the following control characters to interrupt **conman**.

Control+c

conman stops running the current command at the next step that can be interrupted, and returns a command prompt.

Control+d

conman quits after running the current command, on UNIX® workstations only.

Running system commands

About this task

When you enter a system command using a pipe or a system command prefix (: or !), it is run by a child process. The child process's effective user ID is set to the ID of the user running **conman** to prevent security breaches.

User prompting

About this task

When you use wildcard characters to select the objects to be acted upon by a command, **conman** prompts for confirmation after finding each matching object. Responding with **yes** allows the action to be taken, and **no** skips the object without taking the action.

When you run **conman** interactively, the confirmation prompts are issued at your computer. Pressing the **Return** key in response to a prompt is interpreted as a **no** response. Prompting can be disabled by including the **;noask** option in a command.

Although no confirmation prompts are issued when **conman** is not running in interactive mode, it acts as though the response had been **no** in each case, and no objects are acted on. It is important, therefore, to include the **;noask** option on commands when **conman** is not run in interactive mode.

Running commands from conman

About this task

conman commands consist of the following elements:

commandname selection arguments

where:

commandname

Specifies the command name.

selection

Specifies the object or set of objects to be acted upon. Most scheduling objects support the use of folders. The objects can be organized into a tree-like structure of folders similar to a file system.

arguments

Specifies the command arguments.

When submitting commands involving folders, you can specify either a relative or absolute path to the folder. If no folder is specified, then the command is performed using the current folder which is set to root ("/") by default. If you generally work from a specific folder, then you can use the chfolder command to navigate to folders and sub-folders. The chfolder

command changes the working directory or current folder, so that you can use relative folder paths when submitting commands.

See CHFOLDER on page 457 for detailed information about the ways you can modify the current folder.

The following is an example of a **conman** command:

```
sj sked1(1100 03/05/2023).@+state=hold~priority=0;info;offline
```

where:

sj

The abbreviated form of the **showjobs** command.

sked1(1100 03/05/2023).@+state=hold~priority=0

Selects all jobs in the job stream **sked1(1100 03/05/2023)** that are in the HOLD state with a priority other than zero.

;info;offline

Arguments for the **showjobs** command.

For **conman** commands that operate on jobs and job streams where job and job stream specifications include '**#**' special character, the '**=**' value delimiter must precede the object selection.

See the following example:

```
cj=site3#apwkly(0900 02/19/23).report
```

where:

cj

The abbreviated form of the **cancel job** command.

site3#apwkly(0900 02/19/23).report

Selects job *report* in job stream *apwkly(0900 02/19/23)* on workstation *site3*.

Wildcards

The following wildcard characters are permitted:

@

Replaces one or more alphanumeric characters.

?

Replaces one alphanumeric character.

%

Replaces one numeric character.

When monitoring or filtering scheduling objects in folders, the position of wildcards in the string indicates the object being searched or filtered. The following examples clarify these rules:

[folder/]workstationname#/[folder/]jobstreamname

Search for all job streams with the specified name on the specified workstation. For both job streams and workstations, you can specify the folder in which the object is defined, if any.

/@/@#/@/@

Search for all objects defined in all folders for all workstations defined in all folders. The first statement ("/@/" in the example) indicates the folder in which the workstation is defined, the second statement ("@#", in the example) indicates the workstation name, the third statement ("/@/", in the example) indicates the folder in which the scheduling object is defined, the fourth statement ("@", in the example) indicates the scheduling object.

@#/@/@.@

Filters all jobs in job streams defined in all folders.

@#@.@

Filters on all jobs in job streams defined in the root (/) folder.

@#/@

Filters on all job streams defined in the root (/) folder.

/@/@#/@/@+state=cancl

Filters on all jobs that are in **canceled** state.

Delimiters and special characters

Table 75: Delimiters and special characters for conman on page 554 lists characters having special meanings in **conman** commands:

Table 75. Delimiters and special characters for conman

Char.	Description
&	Command delimiter. See SETUPCONMAN.
+	A delimiter used to select objects for commands. It adds an attribute the object must have. For example: `sked1(1100 03/05/2023).@~priority=0`
~	A delimiter used to select objects for commands. It adds an attribute the object must not have. For example: `sked1(1100 03/05/2023).@~priority=0`
;	Argument delimiter. For example: `;info;offline`
,	Repetition and range delimiter. For example: `state=hold,sked,pend`

Table 75. Delimiters and special characters for conman (continued)

Char.	Description
=	Value delimiter. For example: `state=hold`
:!	Command prefixes that pass the command on to the system. These prefixes are optional; if **conman** does not recognize the command, it is passed automatically to the system. For example: `!ls or :ls`

*****	Comment prefix. The prefix must be the first character on a command line or following a command delimiter. For example:
	`*comment`
	or
	`sj& *comment`

>	Redirects command output to a file and overwrites the contents of that file. If the file does not exist, it is created. For example: `sj> joblist`
>>	Redirects command output to a file and appends the output to the end of that file. If the file does not exist, it is created. For example: `sj >> joblist`
|	Pipes command output to a system command or process. The system command is run whether or not output is generated. For example: `sj| grep ABEND`
||	Pipes command output to a system command or process. The system command is not run if there is no output. For example: `sj || grep ABEND`

Conman commands processing

The **conman** program performs the commands that change the status of objects, such as start or stop for a workstation, and the commands that modify objects in the plan in an *asynchronous* way. This means that you might notice a delay between the time you submit the command and the time the information stored in the `Symphony` file is updated with the result of the command.

This occurs because the **conman** program does not update the information stored in the `Symphony` file; **conman** submits the commands to **batchman** which is the only process which can access and update the information contained in the `Symphony` file. For this reason you need to wait for **batchman** to process the request of modifying the object issued by **conman** and

then to update the information about the object stored in the `Symphony` file before seeing the updated information in the output of the ***showobj*** command.

Any changes made using the conman program that affect the Symphony file are also applied to the replicated plan information in the database.

For example, if you request to delete a dependency using the **conman deldep** command, **conman** submits the **deldep** command by posting an event in the `Mailman.msg` mailbox. The **mailman** process gets the information about the deletion request from `Mailman.msg` and puts it in the `Intercom.msg` mailbox on the workstation that owns the resource you delete the dependency from. On each workstation, **batchman** receives the events in its `Intercom.msg` mailbox and processes them in the same order as they were received. If **batchman** is busy for any reason, events carrying requests to perform **conman** commands continue being queued in the `Intercom.msg` file waiting to be read and processed by **batchman**.

In addition, when **batchman** processes the event, the operator is not notified. As a result, you could delete a dependency and it might appear that it had not been deleted because **batchman** was too busy to immediately perform the requested operation. If you run the command again, the deletion might have already been successful, even though a message saying that the command has been successfully forwarded to **batchman** is displayed in the **conman** prompt.

JOBSELECT Selecting jobs in commands

About this task

For commands that operate on jobs, the target jobs are selected by means of attributes and qualifiers. The job selection syntax is shown below, and described on the following pages.

Syntax

[*jobstream_[folder/]workstation***#**][*folder/*]{*jobstreamname*(*hhmm*[*date*]).*jobname*} [{**+** | **~**}*jobqualifier*[...]]

or

[*job_workstation#*]*jobnumber* [{**+** | **~**}*jobqualifier*[...]

or

[*jobstream_[folder/]workstation***#**]*jobstream_id.job* [{**+** | **~**]*jobqualifier*[...]]**;schedid**

or:

netagent::[*jobstream_workstation***#**][*folder/*]{*jobstreamname*(*hhmm*[*date*]).*jobname* | *jobstream_id.jobname***;schedid**}

Arguments

[folder/]workstation

When used with *jobstream.job*, this specifies the name of the workstation on which the job stream runs. When used with *jobnumber*, it specifies the workstation on which the job runs. Except when also using `schedid`, wildcard characters are permitted. This argument might be required depending on the workstation where you launch the command, as follows:

- If you launch the command on the workstation where the target jobs have run, the *workstation* argument is optional.
- If you launch the command on a hosted workstation, the *workstation* argument is required. Hosted workstations are:
  - extended agents
  - agents
  - pools
  - dynamic pools

[folder/]jobstreamname

Specifies the name of the job stream in which the job runs. Wildcard characters are permitted.

(*hhmm* [*date*])

Indicates the time and date the job stream instance is located in the preproduction plan. The value *hhmm* corresponds to the value assigned to the **schedtime** keyword in the job stream definition if no **at** time constraint was set. After the job stream instance started processing, the value of ***hhmm* [*date*]** is set to the time the job stream started. The use of wildcards in this field is not allowed. When issuing inline **conman** commands from the shell prompt enclose the **conman** command in double quotes " ". For example, run this command as follows:

```
conman "sj my_workstation#my_js(2101 02/23).@"
```

jobstream_id

Specifies the unique job stream identifier. See JSSELECT for more information on job stream identifiers.

schedid

Indicates that the job stream identifier is used in selecting the job stream.

[folder/]jobname

Specifies the name of the job. Wildcard characters are permitted.

jobnumber

Specifies the job number.

jobqualifier

See the following section, Job qualifiers on page 558.

netagent

Specifies the name of the HCL Workload Automation network agent that interfaces with the remote HCL Workload Automation network containing the target job. The two colons (::) are a required delimiter. Wildcard characters are permitted. For more information refer to the topic on managing internetwork dependencies in *User's Guide and Reference*.

Note: HCL Workload Automation helps you to identify the correct job stream instance when the job stream selection provides an ambiguous result if more than one instance satisfy your selection criteria. For example when more than one instances of `WK1#J1` are included in the production plan and so the job stream selection provides an ambiguous result the following prompt is automatically generated to allow you to choose the correct instance:

```
Process WK1#J1[(0600 03/04/19),(0AAAAAAAAAAAABTB)]
        (enter "y" for yes, "n" for no)?y
Command forwarded to batchman for WK1#J1[(0600 03/04/19),(0AAAAAAAAAAAABTB)]
Process WK1#J1[(1010 03/04/19),(0AAAAAAAAAAAABTC)]
        (enter "y" for yes, "n" for no)?n
```

In the output only the job stream instance scheduled on `(0600 03/04/19)` and with identifier `0AAAAAAAAAAAABTB` is selected to run the command.

Job qualifiers

Job qualifiers specify the attributes of jobs to be acted on by a command. They can be prefixed by + or ~. If a job qualifier is preceded by + then the jobs containing that specific attribute are selected for running the command. If a job qualifier is preceded by ~ then the jobs containing that specific attribute are excluded from running the command.

Job qualifier keywords can be abbreviated to as few leading characters as needed to uniquely distinguish them from each other.

at[=*time* | *lowtime*, | ,*hightime* | *lowtime*,*hightime*]

Selects or excludes jobs based on the time specified in the **at** dependency.

If **at** is used alone and it is preceded by + then the jobs selected are those containing an **at** dependency.

If **at** is used alone and it is preceded by ~ then the jobs selected are those not containing an **at** dependency.

time

Specifies the time as follows:

hhmm[**+***n* **days** | *date*] [**timezone**|**tz** *tzname*]

where:

hhmm

The hour and minute.

+*n* days

The next occurrence of *hhmm* in *n* number of days.

date

The next occurrence of *hhmm* on *date*, expressed as *mm/dd[/yy]*.

timezone|**tz** ***tzname***

The name of the time zone of the job. See the topic on managing time zones in *User's Guide and Reference* for valid names.

time conforms to the following rules:

- When *hhmm* is earlier than the current time, the start time is tomorrow; when *hhmm* is later than the current time, the start time is today.
- When *hhmm* is greater than 2400, it is divided by 2400. Of the division result, the whole part represents the number of + *days*, while the decimal part represents the time.

lowtime

Specifies the lower limit of a time range, expressed in the same format as *time*. Jobs are selected that are scheduled to start after this time.

hightime

Specifies the upper limit of a time range, expressed in the same format as *time*. Jobs are selected that are scheduled to start before this time.

lowtime,hightime

You can specify both the upper and lower limits of a time range. If you add this limit with + symbol, all the jobs that are scheduled to start within this time range are selected. If you add this limit with ~ symbol, all the jobs that are not scheduled to start within this time range and all the jobs that do not have an **at** parameter are selected. The time limit must be specified in **hhmm** format.

confirmed

Selects or excludes jobs that were scheduled using the **confirm** keyword.

critical

Selects or excludes jobs that were flagged with the **critical** keyword in a job stream definition.

critnet

Selects or excludes jobs that are flagged as **critical** or are predecessors of critical jobs. Hence, it applies to all the jobs that have a critical start time set.

The critical start time of a critical job is calculated by subtracting its estimated duration from its deadline. The critical start time of a predecessor is calculated by subtracting its estimated duration from the critical start time of its successor. Within a critical network, critical start times are calculated starting from the critical job and working backwards along the line of its predecessors.

deadline[=*time* | *lowtime*, | ,*hightime* | *lowtime*,*hightime*]

Specifies the time within which a job must complete.

If **deadline** is used alone and it is preceded by + then the jobs selected are those containing an **deadline** dependency.

If **deadline** is used alone and it is preceded by ~ then the jobs selected are those not containing an **deadline** dependency.

hhmm[**+***n* **days** | *date*] [**timezone**|**tz** *tzname*]

hhmm

The hour and minute.

***+n* days**

An offset in days from the scheduled deadline time.

date

The next occurrence of *hhmm* on *date*, expressed as *mm/dd[/yy]*.

timezone|tz *tzname*

Specifies the time zone to be used when computing the deadline. See the topic on managing time zones in *User's Guide and Reference* for time zone names. The default is the time zone of the workstation on which the job or job stream is launched.

lowtime

Specifies the lower limit of a time range, expressed in the same format as *time*. Selected jobs have a scheduled deadline not earlier than this time.

hightime

Specifies the upper limit of a time range, expressed in the same format as *time*. Selected jobs have a scheduled deadline not later than this time.

lowtime,hightime

You can specify both the upper and lower limits of a time range. If you add this limit with + symbol, all the jobs that are scheduled to start within this time range are selected. If you add this limit with ~ symbol, all the jobs that are not scheduled to start within this time range and all the jobs that do not have a **deadline** parameter are selected. The time limit must be specified in **hhmm** format.

every[=*rate* | *lowrate*, | ,*highrate* | *lowrate*,*highrate*]

Selects or excludes jobs based on whether or not they have a repetition rate.

rate

Specifies the scheduled run rate, expressed as hours and minutes (*hhmm*).

lowrate

Specifies the lower limit of a rate range, expressed in the same format as *rate*. Jobs are selected that have repetition rates equal to or greater than this rate.

highrate

Specifies the upper limit of a rate range, expressed in the same format as *rate*. Jobs are selected that have repetition rates less than or equal to this rate.

If **every** is used alone and it is preceded by + then the jobs selected are those containing any repetition rate.

If **every** is used alone and it is preceded by ~ then the jobs selected are those not containing any repetition rate.

finished[=*time* | *lowtime*, | ,*hightime* | *lowtime*,*hightime*]

Selects or excludes jobs based on whether or not they have finished.

time

Specifies the exact time the job finished, expressed as follows:

hhmm [*date*] [**timezone|tz** *tzname*]

hhmm

The hour and minute.

date

The next occurrence of *hhmm* on date, expressed as *mm/dd[/yy]*.

timezone|tz *tzname*

The name of the time zone of the job. See the topic on managing time zones in *User's Guide and Reference* for valid names.

lowtime

Specifies the lower limit of a time range, expressed in the same format as *time*. Jobs are selected that finished at or after this time.

hightime

Specifies the upper limit of a time range, expressed in the same format as *time*. Jobs are selected that finished at or before this time.

If **finished** is used alone and it is preceded by + then the jobs selected are the jobs that have finished running.

If **finished** is used alone and it is preceded by ~ then the jobs selected are the jobs that have not finished running.

follows=[*netagent*::][*workstation#*][*folder/*]{*jobstreamname*(*hhmm* [*mm/dd*[*/yy*]])[*.job* | @] | *jobstream_id.job*;schedid}| *job*[;nocheck][,...]

Selects or excludes jobs based on whether or not they have a `follows` dependency.

netagent

Specifies the name of the HCL Workload Automation network agent that interfaces with the remote HCL Workload Automation network containing the prerequisite job. Wildcard characters are permitted. For more information refer to the topic on managing internetwork dependencies in *User's Guide and Reference*.

workstation

Specifies the name of the workstation on which the prerequisite job runs. Wildcard characters are permitted.

[folder/]

Specifies the name of the folder where the prerequisite job runs. If no folder is specified, the root folder ("/") is used by default.

jobstreamname

Specifies the name of the job stream in which the prerequisite job runs. Wildcard characters are permitted. If you enter *jobstreamname***.@**, you specify that the target job follows all jobs in the job stream.

job

Specifies the name of the prerequisite job. When entered without a *jobstreamname*, it means that the prerequisite job is in the same job stream as the target job. Do not specify the job name using wildcard characters for a follows dependency.

jobstream_id

Specifies the unique job stream identifier. See JSSELECT for more information on job stream identifiers.

schedid

This keyword, if present, applies to all the job streams identifiers specified in the clause and indicates that for all the job streams specified you are using the *jobstream_id*s and not the *jobstreamname*s. If you want to select some job streams using the *jobstream_id* and some job streams using the *jobstreamname*, you must use two different **follows** clauses, one containing the job streams identified by the *jobstreamname* without the **schedid** keywords, and the other containing the job streams identified by the *jobstream_id* with the **schedid** keyword.

nocheck

Is valid only for the submission commands and used in conjunction with the**schedid** keyword. The **nocheck** keyword indicates that **conman** does not have to check for the existence of the prerequisite job *jobstream_id.job* in the `Symphony` file. It is assumed that *jobstream_id.job* exists, in case it does not exist **batchman** prints a warning message in the `stdlist`.

If **follows** is used alone and it is preceded by + then the jobs are selected if they contain `follows` dependencies.

If **follows** is used alone and it is preceded by ~ then the jobs are selected if they contain no `follows` dependency.

logon=*username*

Select jobs based on the user names under which they run. If *username* contains special characters it must be enclosed in quotes ("). Wildcard characters are permitted.

needs[=[*workstation#*]*resourcename*]

Selects or excludes jobs based on whether or not they have a resource dependency.

workstation

Specifies the name of the workstation on which the resource is defined. Wildcard characters are permitted.

resourcename

Specifies the name of the resource. Wildcard characters are permitted.

If **needs** is used alone and it is preceded by + then the jobs are selected if they contain `resource` dependencies.

If **needs** is used alone and it is preceded by ~ then the jobs are selected if they contain no `resource` dependency.

opens[=[*workstation#*]*filename*[(*qualifier*)]]

Select jobs based on whether or not they have a `file` dependency. A `file` dependency occurs when a job or job stream is dependent on the existence of one or more files before it can begin running.

workstation

Specifies the name of the workstation on which the file exists. Wildcard characters are permitted.

filename

Specifies the name of the file. The name must be enclosed in quotes (") if it contains special characters other than the following: alphanumerics, dashes (-), slashes (/), backslashes (\), and underscores (_). Wildcard characters are permitted.

qualifier

A valid test condition. If omitted, jobs are selected or excluded without regard to a qualifier.

If **opens** is used alone and it is preceded by + then the jobs are selected if they contain `file` dependencies.

If **opens** is used alone and it is preceded by ~ then the jobs are selected if they contain no `file` dependency.

priority=*pri* | *lowpri*, | ,*highpri* | *lowpri*,*highpri*

Selects or excludes jobs based on their priorities.

pri

Specifies the priority value. You can enter **0** through **99**, **hi** or **go**.

lowpri

Specifies the lower limit of a priority range. Jobs are selected with priorities equal to or greater than this value.

highpri

Specifies the upper limit of a priority range. Jobs are selected with priorities less than or equal to this value.

prompt[=*promptname* | *msgnum*]

Selects or excludes jobs based on whether or not they have a `prompt` dependency.

promptname

Specifies the name of a global prompt. Wildcard characters are permitted.

msgnum

Specifies the message number of a local prompt.

If **prompt** is used alone and it is preceded by + then the jobs are selected if they contain `prompt` dependencies.

If **prompt** is used alone and it is preceded by ~ then the jobs are selected if they contain no `prompt` dependency.

recovery=*recv-option*

Selects or excludes jobs based on their recovery options.

recv-option

Specifies the job recovery option as **stop**, **continue**, or **rerun**.

scriptname=*filename*

Selects or excludes jobs based on their executable file names.

filename

Specifies the name of an executable file. The name must be enclosed in quotes (") if it contains special characters other than the following: alphanumerics, dashes (-), slashes (/), backslashes (\), and underscores (_). Wildcard characters are permitted.

started[=*time* | *lowtime*, | ,*hightime* | *lowtime*,*hightime*]

Selects or excludes jobs based on whether or not they have started.

If **started** is used alone and it is preceded by +, then only the jobs that have started running at this time are selected.

If **started** is used alone and it is preceded by ~, then only the jobs that have started running at or after this time and that are still running are selected.

time

Specifies the exact time the job started, expressed as follows:

hhmm [*date*] [**timezone**|**tz** *tzname*]

hhmm

The hour and minute.

date

The next occurrence of *hhmm* on date, expressed as *mm*/*dd*[/*yy*].

timezone|**tz** ***tzname***

The name of the time zone of the job. See the topic on managing time zones in *User's Guide and Reference* for valid names.

lowtime

Specifies the lower limit of a time range, expressed in the same format as *time*. Only jobs that started at or after this time are selected.

hightime

Specifies the upper limit of a time range, expressed in the same format as *time*. Only jobs that started at or before this time are selected.

lowtime,hightime

You can specify both the upper and lower limits of a time range. If you add this limit with + symbol, all the jobs that are scheduled to start within this time range are selected. If you add this limit with ~ symbol, all the jobs that are not scheduled to start within this time range and all the jobs that do not have a **started** parameter are selected. The time limit must be specified in **hhmm** format.

state=*state*[,...]

Selects or excludes jobs based on their states.

state

Specifies the current state of the job. Valid job states are as follows:

ABEND

The job ended with a nonzero exit code.

ABENP

An **abend** confirmation was received, but the job has not completed.

ADD

The job is being submitted.

CANCL

For internetwork dependencies only. The remote job or job stream has been cancelled.

DONE

The job completed in an unknown state.

ERROR

For internetwork dependencies only, an error occurred while checking for the remote status.

EXEC

The job is running. The + flag written beside the EXEC status means that the job is managed by the local **batchman** process.

EXTRN

For internetwork dependencies only, the status is unknown. An error occurred, a rerun action was just performed on the job in the `EXTERNAL` job stream, or the remote job or job stream does not exist.

FAIL

Unable to launch the job.

FENCE

The job's priority value is below the fence.

HOLD

The job is awaiting dependency resolution.

INTRO

The job is introduced for launching by the system. The + flag written beside the INTRO status means that the job is managed by the local **batchman** process.

PEND

The job completed, and is awaiting confirmation.

READY

The job is ready to launch, and all dependencies are resolved.

SCHED

The job's **at** time has not arrived.

SUCC

The job completed with an exit code of zero.

SUCCP

A **succ** confirmation was received, but the job has not completed.

WAIT

The job is in the WAIT state (extended agent only).

until[=*time* | *lowtime*, | ,*hightime* | *lowtime*,*hightime*]

Selects or excludes jobs based on their scheduled end time.

If **until** is used alone and it is preceded by + then the jobs are selected if they have an **until** time specified.

If **until** is used alone and it is preceded by ~ then the jobs are selected if they have no **until** time specified.

time

Specifies the scheduled end time expressed as follows:

hhmm[**+***n* **days** | *date*] [**timezone**|**tz** *tzname*]

hhmm

The hour and minute.

+*n* days

The next occurrence of *hhmm* in *n* number of days.

date

The next occurrence of *hhmm* on *date*, expressed as *mm/dd[/yy]*.

timezone|**tz** ***tzname***

The name of the time zone of the job. See the topic on managing time zones in *User's Guide and Reference* for valid names.

lowtime

Specifies the lower limit of a time range, expressed in the same format as *time*. Jobs are selected that have scheduled end times on or after this time.

hightime

Specifies the upper limit of a time range, expressed in the same format as *time*. Jobs are selected that have scheduled end times on or before this time.

To select a list of expected objects, ***hightime*** requires the **date** option with the operating system time. It does not work if the current operating system time is set later than the **until** *hightime*

value. To display job streams in the **ready** or **hold** state with **until** set to 12:00, you must add the **date** value to **until**=,1200.

lowtime,hightime

You can specify both the upper and lower limits of a time range. If you add this limit with + symbol, all the jobs that are scheduled to start within this time range are selected. If you add this limit with ~ symbol, all the jobs that are not scheduled to start within this time range and all the jobs that do not have a **until** parameter are selected. The time limit must be specified in **hhmm** format.

JSSELECT Selecting job streams in commands

About this task

For commands that operate on job streams, the target job streams are selected by specifying attributes and qualifiers.

Because ***scheddateandtime*** is specified in minutes, the combination of the **jobstreamname** and the *scheddateandtime* time might not be unique. For this reason the **jobstream_id** has been made available to the user, either for display purposes or to perform actions against a specific instance of a job stream.

The job stream selection syntax is shown below, and described on the following pages. You can choose one of the two syntaxes described.

Syntax

[*[folder/]workstation***#**][*folder/]jobstreamname*(*hhmm*[*date*]) [{**+** | **~**}*jobstreamqualifier*[...]]

[*[folder/]workstation***#**]*jobstream_id* **;schedid**

Arguments

For all objects which can be defined in a folder, such as jobs, job streams, workstations, and so on, you can optionally specify the folder where the object is defined. If no folder is specified, the root folder is used by default.

[folder/]workstation

Specifies the name of the workstation on which the job stream runs. Except when also using `schedid`, wildcard characters are permitted.

[folder/]jobstreamname

Specifies the name of the job stream. Wildcard characters are permitted.

(*hhmm* [*date*])

Indicates the time and date the job stream instance is located in the preproduction plan. This value corresponds to the value assigned to the **schedtime** keyword in the job stream definition if no **at** time constraint was set. After the job stream instance started processing the value of ***hhmm* [*date*]** is set to the time the job stream started. The use of wildcards in this field is not allowed. When issuing in line **conman** commands from the shell prompt enclose the **conman** command in double quotation marks **(")**. For example, run this command as follows:

```
conman "ss my_workstation#my_js(2101 02/23)"
```

jobstreamqualifier

See "Job Stream Qualifiers below.

jobstream_id

Specifies the unique alphanumerical job stream identifier automatically generated by the planner and assigned to that job stream. It is mainly used by internal processes to identify that instance of the job stream within the production plan, but it can often be used also when managing the job stream from the **conman** command-line program by specifying the **;schedid** option.

schedid

Indicates that the job stream identifier is used in selecting the job stream.

Note: HCL Workload Automation helps you to identify the correct job stream instance when the job stream selection provides an ambiguous result if more than one instance satisfy your selection criteria. For example when more than one instances of `WK1#J1` are included in the production plan and so the job stream selection provides an ambiguous result the following prompt is automatically generated to allow you to choose the correct instance:

```
Process WK1#J1[(0600 03/04/19),(0AAAAAAAAAAAABTB)]
        (enter "y" for yes, "n" for no)?y
Command forwarded to batchman for WK1#J1[(0600 03/04/19),(0AAAAAAAAAAAABTB)]
Process WK1#J1[(1010 03/04/19),(0AAAAAAAAAAAABTC)]
        (enter "y" for yes, "n" for no)
```

In the output only the job stream instance scheduled on `(0600 03/04/19)` and with identifier `0AAAAAAAAAAAABTB` is selected to run the command.

Job stream qualifiers

at[=*time* | *lowtime*, | ,*hightime* | *lowtime*,*hightime*]

Selects or excludes job streams based on the scheduled start time.

If **at** is used alone and it is preceded by + then the job streams selected are those containing an **at** dependency.

If **at** is used alone and it is preceded by ~ then the job streams selected are those not containing an **at** dependency.

time

Specifies the scheduled start time expressed as follows:

hhmm[**+***n* **days** | *date*] [**timezone**|**tz** *tzname*]

hhmm

The hour and minute.

+*n* days

The next occurrence of *hhmm* in *n* number of days.

date

The next occurrence of *hhmm* on *date*, expressed as *mm/dd[/yy]*.

timezone|**tz** ***tzname***

The name of the time zone of the job stream. See the topic on managing time zones in *User's Guide and Reference* for valid names.

lowtime

Specifies the lower limit of a time range, expressed in the same format as *time*. Job streams aref selected that have scheduled start times on or after this time.

hightime

Specifies the upper limit of a time range, expressed in the same format as *time*. Job streams are selected that have scheduled start times at or before this time.

lowtime,hightime

You can specify both the upper and lower limits of a time range. If you add this limit with + symbol, all the jobs that are scheduled to start within this time range are selected. If you add this limit with ~ symbol, all the jobs that are not scheduled to start within this time range and all the jobs that do not have an **at** parameter are selected. The time limit must be specified in **hhmm** format.

carriedforward

Selects job streams that were carried forward if preceded by +, excludes job streams that were carried forward if preceded by ~.

carryforward

If preceded by + selects job streams that were scheduled using the **carryforward** keyword; if preceded by ~ excludes job streams that were scheduled using the **carryforward** keyword.

finished[=*time* | *lowtime*, | ,*hightime* | *lowtime*,*hightime*]

Selects or excludes job streams based on whether or not they have finished.

If **finished** is used alone and it is preceded by + then the jobs streams selected are the jobs that have finished running.

If **finished** is used alone and it is preceded by ~ then the jobs streams selected are the jobs that have not finished running.

time

Specifies the exact time the job streams finished, expressed as follows:

hhmm [*date*] [**timezone**|**tz** *tzname*]

hhmm

The hour and minute.

date

The next occurrence of *hhmm* on date, expressed as *mm*/*dd*[/*yy*].

timezone|**tz** ***tzname***

The name of the time zone of the job stream. See the topic on managing time zones in *User's Guide and Reference* for valid names.

lowtime

Specifies the lower limit of a time range, expressed in the same format as *time*. Job streams are selected that finished at or after this time.

hightime

Specifies the upper limit of a time range, expressed in the same format as *time*. Job streams are selected that finished at or before this time.

lowtime,hightime

You can specify both the upper and lower limits of a time range. If you add this limit with + symbol, all the jobs that are scheduled to start within this time range are selected. If you add this limit with ~ symbol, all the jobs that are not scheduled to start within this time range and all the jobs that do not have a **finished** parameter are selected. The time limit must be specified in **hhmm** format.

follows=[*netagent*::][*workstation*#][*folder/*]{*jobstreamname*(*hhmm* [*mm*/*dd*[/*yy*]])[.*job* | @] | *jobstream_id*.*job*;schedid}| *job*[;nocheck] [,...]

Selects or excludes job streams based on whether or not they have a `follows` dependency.

netagent

Specifies the name of the network agent that interfaces with the remote HCL Workload Automation network containing the prerequisite job or job stream. Wildcard characters are permitted. For more information about network agents, refer to the topic on managing internetwork dependencies in *User's Guide and Reference*.

workstation

Specifies the name of the workstation on which the prerequisite job or job stream runs. Wildcard characters are permitted.

[folder/]

Specifies the name of the folder where the prerequisite job stream runs. If no folder is specified, the root folder ("/") is used by default.

jobstreamname

Specifies the name of the prerequisite job stream. Wildcard characters are permitted.

job

Specifies the name of the prerequisite job. Wildcard characters are permitted.

jobstream_id

Specifies the unique job stream identifier. See above for more information on job stream identifiers.

schedid

This keyword, if present, applies to all the job streams identifiers specified in the clause and indicates that for all the job streams specified you are using the *jobstream_id*s and not the *jobstreamname*s. If you want to select some job streams using the *jobstream_id* and some job streams using the *jobstreamname*, you must use two different **follows** clauses, one containing the job streams identified by the *jobstreamname* without the **schedid** keywords, and the other containing the job streams identified by the *jobstream_id* with the **schedid** keyword.

nocheck

Is valid only for the submission commands and used in conjunction with the**schedid** keyword. The **nocheck** keyword indicates that **conman** does not have to check for the existence of the prerequisite job *jobstream_id.job* in the `Symphony` file. It is assumed that *jobstream_id.job* exists, in case it does not exist **batchman** prints a warning message in the `stdlist`.

If **follows** is used alone and it is preceded by + then the jobs streams are selected if they contain `follows` dependencies.

If **follows** is used alone and it is preceded by ~ then the jobs streams are selected if they contain no `follows` dependency.

limit[=*limit* | *lowlimit*, | ,*highlimit* | *lowlimit*,*highlimit*]

Selects or excludes job streams based on whether or not they have a job limit.

limit

Specifies the exact job limit value.

lowlimit

Specifies the lower limit of range. Job streams are selected that have job limits equal to or greater than this limit.

highlimit

Specifies the upper limit of a range. Job streams are selected that have job limits less than or equal to this limit.

If **limit** is used alone and it is preceded by + then the jobs streams are selected if they have any job limit.

If **limit** is used alone and it is preceded by ~ then the jobs streams are selected if they have no job limit.

needs[=[*workstation#*]*resourcename*]

Selects or excludes job streams based on whether or not they have a resource dependency.

workstation

Specifies the name of the workstation on which the resource is defined. Wildcard characters are permitted.

resourcename

Specifies the name of the resource. Wildcard characters are permitted.

If **needs** is used alone and it is preceded by + then the jobs streams are selected if they contain `resource` dependencies.

If **needs** is used alone and it is preceded by ~ then the jobs streams are selected if they contain no `resource` dependency.

opens[=[*workstation#*]*filename*[(*qualifier*)]]

Selects or excludes job streams based on whether or not they have a file dependency. A file dependency occurs when a job or job stream is dependent on the existence of one or more files before it can begin running.

workstation

Specifies the name of the workstation on which the file exists. Wildcard characters are permitted.

filename

Specifies the name of the file. The name must be enclosed in quotes (") if it contains special characters other than the following: alphanumerics, dashes (-), slashes (/), backslashes (\), and underscores (_). Wildcard characters are permitted.

qualifier

A valid test condition. If omitted, job streams are selected or excluded without regard to a qualifier.

If **opens** is used alone and it is preceded by + then the jobs streams are selected if they contain `file` dependencies.

If **opens** is used alone and it is preceded by ~ then the jobs streams are selected if they contain no `file` dependency.

priority=*pri* | *lowpri*, | ,*highpri* | *lowpri*,*highpri*

Selects or excludes job streams based on their priorities.

pri

Specifies the priority value. You can enter **0** through **99**, **hi** or **go**.

lowpri

Specifies the lower limit of a priority range. Job streams are selected with priorities equal to or greater than this value.

highpri

Specifies the upper limit of a priority range. Job streams are selected with priorities less than or equal to this value.

prompt[=*promptname* | *msgnum*]

Selects or excludes job streams based on whether or not they have a prompt dependency.

promptname

Specifies the name of a global prompt. Wildcard characters are permitted.

msgnum

Specifies the message number of a local prompt.

If **prompt** is used alone and it is preceded by + then the jobs streams are selected if they contain `prompt` dependencies.

If **prompt** is used alone and it is preceded by ~ then the jobs streams are selected if they contain no `prompt` dependency.

started[=*time* | *lowtime*, | ,*hightime* | *lowtime*,*hightime*]

Selects or excludes job streams based on whether or not they have started.

If **started** is used alone and it is preceded by + then the jobs streams selected are the jobs that have started running.

If **started** is used alone and it is preceded by ~ then the jobs streams selected are the jobs that have not started running.

time

Specifies the exact time the job stream started, expressed as follows:

hhmm [*date*] [**timezone**|**tz** *tzname*]

hhmm

The hour and minute.

date

The next occurrence of *hhmm* on date, expressed as *mm*/*dd*[/*yy*].

timezone|tz *tzname*

The name of the time zone of the job stream. See the topic on managing time zones in *User's Guide and Reference* for valid names.

lowtime

Specifies the lower limit of a time range, expressed in the same format as *time*. Job streams are selected that started at or after this time.

hightime

Specifies the upper limit of a time range, expressed in the same format as *time*. Job streams are selected that started at or before this time.

lowtime,hightime

You can specify both the upper and lower limits of a time range. If you add this limit with + symbol, all the jobs that are scheduled to start within this time range are selected. If you add this limit with ~ symbol, all the jobs that are not scheduled to start within this time range and all the jobs that do not have a **started** parameter are selected. The time limit must be specified in **hhmm** format.

state=*state*[,...]

Selects or excludes job streams based on their states.

state

Specifies the current state of the job stream. Valid job stream states are as follows:

ABEND

The job stream ended abnormally.

ADD

The job stream has just been submitted.

EXEC

The job stream is running.

EXTRN

For internetwork dependencies only. This is the state of the EXTERNAL job stream containing jobs referencing to jobs or job streams in the remote network.

HOLD

The job stream is awaiting dependency resolution.

READY

The job stream is ready to launch, and all dependencies are resolved.

STUCK

Execution is interrupted. No jobs are launched without operator intervention.

SUCC

The job stream completed successfully.

until[=*time* | *lowtime*, | ,*hightime* | *lowtime*,*hightime*]

Selects or excludes job streams based on the scheduled end time.

If **until** is used alone and it is preceded by + then the jobs streams selected are those containing any scheduled end time.

If **until** is used alone and it is preceded by ~ then the jobs streams selected are those not containing any scheduled end time.

time

Specifies the scheduled end time expressed as follows:

hhmm[**+***n* **days** | *date*] [**timezone**|**tz** *tzname*]

hhmm

The hour and minute.

+*n* days

The next occurrence of *hhmm* in *n* number of days.

date

The next occurrence of *hhmm* on *date*, expressed as *mm/dd[/yy]*.

timezone|**tz** ***tzname***

The name of the time zone of the job stream. See the topic on managing time zones in *User's Guide and Reference* for valid names.

lowtime

Specifies the lower limit of a time range, expressed in the same format as *time*. Job streams are selected that have scheduled end times on or after this time.

hightime

Specifies the upper limit of a time range, expressed in the same format as *time*. Job streams are selected that have scheduled end times on or before this time.

lowtime,hightime

You can specify both the upper and lower limits of a time range. If you add this limit with + symbol, all the jobs that are scheduled to start within this time range are selected. If you add this limit with ~ symbol, all the jobs that are not scheduled to start within this time range and all the jobs that do not have an **until** parameter are selected. The time limit must be specified in **hhmm** format.

Conman return codes

Conman return codes

Conman return codes management

When you run a conman command, the command line can show an outcome return code. To find the return code, perform the following action:

On Windows Operating systems:

```
echo %ERRORLEVEL%
```

On UNIX Operating systems:

```
echo $?
```

The conman command line provides the following return codes for the submit sched and submit job commands:

0

The command completed successfully.

10

The submit sched encountered an error.

11

The submit job encountered an error.

All other commands always return the 0 return code.

CONMAN COMMANDS

You can use the **conman** program from the master domain manager and from any fault-tolerant agent workstation in the HCL Workload Automation network. On dynamic agents, the following conman commands are supported:

- listsucc. For more information, see LISTSUCC on page 626.
- rerunsucc. For more information, see RERUNSUCC on page 639.

Command names and keywords can be entered in either uppercase or lowercase characters, and can be abbreviated to as few leading characters as are needed to uniquely distinguish them from each other. Some of the command names also have specific short forms.

Note: The workstation types in the following table have these meanings:

- M - master domain managers and backup masters
- F - domain managers and fault-tolerant agents
- S - standard agents (you can only display files on a standard agent)

Table 76. List of conman commands

Command	Short Form	Description	Type
adddep { job | sched }	**adj | ads**	Adds job or job stream dependencies.	F
altjob	**aj**	Modifies a job in the plan before it runs.	F
altpass		Alters a user object definition password.	F
altpri	**ap**	Alters job or job stream priorities.	F
cancel { job | sched }	**cj | cs**	Cancels a job or a job stream.	F
checkhealthstatus	**chs**	Invokes **chkhltst** service to check if mailbox can be successfully read by **mailman** or if there are errors in the mailbox header.	MFS
chfolder	**cf**	Changes the working directory.	F
confirm	**conf**	Confirms job completion.	F
console	**cons**	Assigns the HCL Workload Automation console.	FS
continue	**cont**	Ignores the next error.	FS
deldep { job | sched }	**ddj | dds**	Deletes job or job stream dependencies.	F
deployconf	**deploy**	Gets the latest monitoring configuration for the event monitoring engine on the workstation.	FS
display { file | job | sched }	**df | dj | ds**	Displays files, jobs, and job streams.	FS
exit	**e**	Exits **conman**.	FS
fence	**f**	Sets HCL Workload Automation job fence.	F
help([1])	**h**	Displays command information.	FS
kill	**k**	Stops an executing job.	F
limit { cpu | sched }	**lc | ls**	Changes a workstation or job stream job limit.	F
link	**lk**	Opens workstation links.	FS
listfolder	**lf**	Lists folders.	F
listsym	**lis**	Displays a list of `Symphony` log files.	F
listsucc		Lists the successors of a job.	F

Table 76. List of conman commands (continued)

Command	Short Form	Description	Type
recall	**rc**	Displays prompt messages.	F
redo	**red**	Edits the previous command.	FS
release { job | sched }	**rj | rs**	Releases job or job stream dependencies.	F
reply	**rep**	Replies to prompt message.	F
rerun	**rr**	Reruns a job.	F
rerunsucc		Reruns a job and runs its successors.	F
resetFTA	**N/A**	Recovers a corrupt Symphony file on the specified fault-tolerant agent	T
resource	**res**	Changes the number of resource units.	F
setsym	**set**	Selects a Symphony log file.	F
showcpus	**sc**	Displays workstation and link information.	FS
showdomain	**showd**	Displays domain information.	FS
showfiles	**sf**	Displays information about files.	F
showjobs	**sj**	Displays information about jobs.	F
showprompts	**sp**	Displays information about prompts.	F
showresources	**sr**	Displays information about resources.	F
showschedules	**ss**	Displays information about job streams.	F
shutdown	**shut**	Stops HCL Workload Automation production processes.	FS
start		Starts HCL Workload Automation production processes.	FS
startappserver		Starts the WebSphere Application Server Liberty process	DM
starteventprocessor	**startevtp**	Starts the event processing server.	M([2])
startmon	**startm**	Starts the `monman` process that turns on the event monitoring engine on the agent.	FS
status	**stat**	Displays HCL Workload Automation production status.	FS
stop		Stops HCL Workload Automation production processes.	FS

Table 76. List of conman commands (continued)

Command	Short Form	Description	Type
stop ;progressive		Stops HCL Workload Automation production processes hierarchically.	
stopappserver	**stopapps**	Stops the WebSphere Application Server Liberty process	DM
stopeventprocessor	**stopevtp**	Stops the event processing server.	M([2])
stopmon	**stopm**	Stops the event monitoring engine on the agent.	FS
submit { docommand | file | job | sched }	**sbd |** **sbf |** **sbj |** **sbs**	Submits a command, file, job, or job stream.	FS([3])
switcheventprocessor	**switchevtp**	Switches the event processing server from master domain managers to backup masters or vice versa.	M
switchmgr	**switchm**	Switches the domain manager.	F
system-command		Sends a command to the system.	FS
tellop	**to**	Sends a message to the console.	FS
unlink		Closes workstation links.	FS
version	**v**	Displays **conman**'s command line program banner.	FS

1. Not available on supported Windows® operating system.
2. Includes workstations installed as backup masters but used as ordinary fault-tolerant agents.
3. You can use **submit job** (**sbj**) and **submit sched** (**sbs**) on a standard agent by using the connection parameters or specifying the settings in the `useropts` file when invoking the **conman** command line.

Note: In the commands, the terms *sched* and *schedule* refer to *job streams*, and the term *CPU* refers to *workstation*.

Other HELP topics:

Command	Description
COMMANDS	List of conman commands
JOBSELECT	Information about selecting and qualifying jobs.
JOBSTATES	List of job states.
JSSELECT	Information about selecting and qualifying job streams.

Command	Description
JSSTATES	List of job stream states.
MANAGEBACKLEVEL	Managing jobs and job streams from back-level agents.
RUNCONMAN	How to run conman.
SETUPCONMAN	Describes how to setup to use conman.
SPECIALCHAR	Describes the wildcards, delimiters and other special characters you can use.

ADDDEP JOB

Adds dependencies to a job.

You must have ***adddep*** access to the job. To include `needs` and `prompt` dependencies, you must have ***use*** access to the resources and global prompts.

Syntax

{**adddep job** | **adj**} **=** *jobselect*
;*dependency*[**;**...]
[**;noask**]

Arguments

jobselect

See JOBSELECT for information on selecting jobs in commands.

dependency

The type of dependency. Specify one of the following. Wildcard characters are not permitted.

at=*hhmm*[**timezone** | **tz** *tzname*][**+***n* **days** | *mm/dd[/yy]*] | [**absolute** | **abs**]

confirmed

deadline=*time* [**timezone**|**tz** *tzname*][**+***n* **day**[**s** | *mm/dd[/yy]*]

every=*rate*

follows=[*netagent*::][*workstation*#]{*jobstreamname*[*hhmm* [*mm/dd[/yy]*]][.*job* | @] | *jobstream_id.job*;**schedid**}|*job*[,...] [**if** '*condition_name*[| *condition_name*][| ...]']

follows=[*netagent*::][*workstation*#]{*jobstreamname*[*hhmm* [*mm/dd[/yy]*]][.*job* | @] | *jobstream_id.job*;**schedid**}|*job*[,...] [**if** '*condition_name*[| *condition_name*][| ...]']

Note: Internetwork dependencies do not support folders, therefore, the network agent workstation, and the jobs and job streams running on them, cannot be defined in a folder different from the root (/). Folders are supported on all other workstation types as follows:

```
[follows {[[folder/]workstation#][folder/]jobstreamname[.jobname]
```

follows=[[*folder/*]*workstation***#**][*folder/*]{*jobstreamname*[*hhmm* [*mm/dd[/yy]*]][*.job* | **@**] | *jobstream_id.job***;schedid**}| *job*[**,**...] [**if** '*condition_name*[| *condition_name*][| ...]']

The *condition_name* variable indicates the name of the condition defined in the job definition. Conditions must be separated by | and enclosed between single quotes. Conditions can be **status conditions**, based on job status, or other **output conditions**, based on a mapping expression such as a return code, output variables, or output found in a job log. In each **follows** statement, you can specify only one dependency type: either status or output conditions. At submission time, you can add status or output conditions, but no joined dependencies.

maxdur=[*hhhmm*] [**onmaxdur** *action*]

mindur=[*hhhmm*] [**onmindur** *action*]

needs=[*num*] [*[folder/]workstation***#**][*folder/]resource*[**,**...]

opens=[*[folder/]workstation***#**]"*filename*"[**(***qualifier***)**][**,**...]

priority[**=***pri* | **hi** | **go**]

prompt="[**:** | **!**]*text*" | *[folder/]promptname*[**,**...]

until *time* [**timezone**|**tz** *tzname*][**+***n* **day**[**s**]] | [**absolute** | **abs**] [**;onuntil** *action*]

noask

Specifies not to prompt for confirmation before taking action on each qualifying job.

Note:

1. If you add twice a dependency on a job stream to a job, both dependencies are treated.
2. When using the **deadline** keyword, ensure the **bm check deadline** option is set to a value higher than 0 in the `localopts` configuration file on the workstation you are working on. You can define the **bm check deadline** option on each workstation on which you want to be aware of the deadline expiration, or, if you want to obtain up-to-date information about the whole environment, define the option on the master domain manager. Deadlines for critical jobs are evaluated automatically, independently of the **bm check deadline** option.

For more information about the **bm check deadline** option, see the section about localopts details in *Administration Guide*.

3. If you add a dependency to a job after it has completed, it is not evaluated. However, any subsequent reruns of the job will process the dependency correctly.

Comments

If you do not specify a value for `priority`, the job reverts to its original scheduled priority. If you do not specify a workstation in `follows`, `needs`, or `opens`, the default is the workstation on which the job runs.

You cannot use this command to add a `resource` or a `prompt` as dependencies unless they are already referenced by a job or a job stream in the `Symphony` file.

Example

Examples

In the following example, you want to add a resource dependency of two tapes to a job. To explain all the potentialities of the folder support, in the example the following objects are defined in a folder:

- the workstation on which the job stream is defined (`/wksfolder2/mywks2`)
- job stream containing the job (`folderA/jsAA(0900 02/19/18)`)
- the resources on which the dependency is defined (`/resfolder1/tapes`)
- the workstation on which the two tapes are defined (`/wksfolder1/mywks1`)

To add a resource dependency consisting of two tapes defined in the `resfolder1` folder and belonging to workstation `mywks1` defined in `wksfolder1` folder to a job named `job3` and belonging to the `jsAA` job stream, which is defined in `folderA` folder belonging to `mywks2` workstation defined in `wksfolder2` folder.

```
adj /wksfolder2/mywks2#/folderA/jsAA(0900 02/19/18).job3 ; needs=2 /wksfolder1/mywks1#/resfolder1/tapes
```

To add an external follows dependency on job `JOB022` in job stream `MLN#/FOLDER1/SCHED_02(0600 02/24/18)` from `JOBA` in job stream `MLN#/FOLDER2/NEW_TEST(0900 02/19/18)`, run the following command:

```
adj=MLN#/FOLDER2/NEW_TEST(0900 02/19/18).JOBA ; follows
      MLN#/FOLDER1/SCHED_02(0600 02/24/18).JOB022
```

To add a file dependency, and an **until** time to job `j6` in job stream `JS2(0900 02/19/18)`, run the following command:

```
adj=WK1#JS2(0900 02/19/18).j6 ; opens="/usr/lib/prdata/file5"(-s %p) ; until=2330
```

To kill job `PAYROLL_JOB` in job stream `ABSENCES_JS` when it has run for more than 9 hours and 1 minute, run the following command:

```
adj DUBAI#ABSENCES_JS.PAYROLL_JOB ;maxdur=901 ;onmaxdur kill
```

See also

From the Dynamic Workload Console you can perform the same task as follows:

1. In the Welcome page, select **Monitor your workload**, or in the navigation bar at the top of the page, click **Monitoring & Reporting > Orchestration Monitoring > Monitor Workload**.
2. In **Object Type**, select **Workstation/Job/Jobstream**.
3. Select an engine.
4. From the **Workload monitoring actions** drop-down list, select **Check health status**
5. You can click on**ok** to confirm or **Cancel** to cancel the **Check health status**
6. From the **Query** drop-down list, select **All Jobs in plan** or another task to monitor jobs.
7. Click **Run** to run the monitoring task.
8. From the table containing the list of jobs, select the job to which you want to add a dependency and click **Dependencies...**.
9. In the Dependencies panel, expand a dependency section and click the button corresponding to the dependency action you want to add.

ADDDEP SCHED

Adds dependencies to a job stream.

You must have ***adddep*** access to the job stream. To include `needs` and `prompt` dependencies, you must have ***use*** access to the resources and global prompts.

Syntax

{**adddep sched** | **ads**} **=** *jstreamselect*
;*dependency*[**;**...]
[**;noask**]

Arguments

jstreamselect

See JSSELECT for information on selecting job streams in commands.

dependency

The type of dependency. Specify one of the following. Wildcard characters are not permitted.

at=*hhmm*[**timezone** | **tz** *tzname*][**+***n* **days** | *mm/dd[/yy]*] | [**absolute** | **abs**]

Note: If *hhmm* is a number greater than 2400, the number is divided by 2400, and the integer result is the number of days to move forward, like the **+***n* **days** option.

carryforward

deadline=*time* [**timezone**|**tz** *tzname*][**+***n* **day**[**s** | *mm/dd[/yy]*]

follows=[*netagent*::][*workstation#*]{*jobstreamname*[*hhmm* [*mm/dd[/yy]*]][.*job* | **@**] | *jobstream_id.job*;**schedid**}| *job*[**,**...] [**if**'*condition_name*[| *condition_name*][| ...]']

follows=[*netagent*::][*workstation#*]{*jobstreamname*[*hhmm* [*mm/dd[/yy]*]][.*job* | **@**] | *jobstream_id.job*;**schedid**}| *job*[**,**...] [**if** '*condition_name*[| *condition_name*][| ...]']

Note: Internetwork dependencies do not support folders, therefore, the network agent workstation, and the jobs and job streams running on them, cannot be defined in a folder different from the root (/). Folders are supported on all other workstation types as follows:

```
[follows {[folder/]workstation#][folder/]jobstreamname[.jobname]
```

follows=[[*folder/*]*workstation#*][*folder/*]{*jobstreamname*[*hhmm* [*mm/dd[/yy]*]][.*job* | **@**] | *jobstream_id.job*;**schedid**}| *job*[**,**...] [**if** '*condition_name*[| *condition_name*][| ...]']

The *condition_name* variable indicates the name of the condition defined in the job definition. Conditions must be separated by | and enclosed between single quotes. Conditions can be **status conditions**, based on job status, or other **output conditions**, based on a mapping expression such as a return code, output variables, or output found in a job log. In each **follows** statement, you can specify only one dependency type: either status or output conditions. At submission time, you can add status or output conditions, but no joined dependencies.

limit=*limit*

needs=[*num*] [*[folder/]workstation#][folder/]resource*[**,**...]

opens=[*[folder/]workstation#*]"*filename*"[**(***qualifier***)**][**,**...]

priority[**=***pri* | **hi** | **go**]

prompt="[**:** | **!**]*text*" | *[folder/]promptname*[**,**...]

until *time* [**timezone**|**tz** *tzname*][**+***n* **day**[**s**] | [**absolute** | **abs**]] [**;onuntil** *action*]

noask

Specifies not to prompt for confirmation before taking action on each qualifying job stream.

Note:

1. If you add a dependency on a job stream to another job stream twice, only one dependency is considered.
2. When using the **deadline** keyword, ensure the **bm check deadline** option is set to a value higher than 0 in the `localopts` configuration file on the workstation you are working on. You can define the **bm check deadline** option on each workstation on which you want to be aware of the deadline expiration, or, if you want to obtain up-to-date information about the whole environment, define the option on the master domain manager. Deadlines for critical jobs are evaluated automatically, independently of the **bm check deadline** option.

For more information about the **bm check deadline** option, see the section about localopts details in *Administration Guide*.

3. If you add a dependency to a job stream after it has completed, it is not evaluated. However, any subsequent reruns of the job stream will process the dependency correctly.

Comments

- If you do not specify a value for `priority`, the job stream reverts to its original scheduled priority.
- If you do not specify a value for **limit**, the value defaults to 0.
- If you do not specify a workstation in `follows`, `needs`, or `opens`, the default is the workstation on which the job stream runs.
- You cannot use this command to add a `resource` or a `prompt` as dependencies unless they already exists in the production plan. To see which resource and prompts exist in the plan refer to SHOWRESOURCES and SHOWPROMPTS.
- A job stream can have a maximum of 40 dependencies. Once submitted with 40 dependencies, you may only remove existing ones; adding new dependencies is not allowed—even if the total count drops below 40 after deletion. While removed dependencies are correctly excluded from the submitted job stream, new dependencies cannot be added unless the original entries are explicitly deleted from the database.

Example

Examples

To add a prompt dependency to job stream `sked9(0900 02/19/06)`, stored in folder **myfolder**, run the following command:

```
ads myfolder/sked9(0900 02/19/06) ; prompt=msg103
```

To add an external follows dependency to job `JOBB` in job stream `CPUA#SCHED_02(0600 02/24/06)` and a job limit to job stream `CPUA#TEST(0900 02/19/06)`, run the following command:

```
ads=CPUA#TEST(0900 02/19/06) ; follows CPUA#SCHED_02(0600 02/24/06).JOBB; limit=2
```

See also

From the Dynamic Workload Console you can perform the same task as follows:

1. In the navigation bar at the top, click **Monitoring and Reporting > Orchestration Monitor**.
2. Select an engine.
3. In **Object Type**, select **Workstation/Job Stream/Job**.
4. From the **Query** drop-down list, select **All Job Streams in plan** or another task to monitor job streams.
5. Click **Run** to run the monitoring task.

6. From the table containing the list of job streams, select the job stream to which you want to add a dependency and click the **Dependencies...**.
7. In the Dependencies panel, expand a dependency section and click the button corresponding to the dependency action you want to add.

ALTJOB

Modify a job in the plan before it runs.

You must have ***submit*** access to the job.

Syntax

{**altjob** | **aj**} *jobselect*
[**;streamlogon**|**logon=***new_logon*]
[**;docommand=***"new_command"*|**;script=***"new_script"*]
[**;noask**]

Arguments

jobselect

See JOBSELECT for information about selecting jobs in commands. Wild cards are supported.

streamlogon|**logon=*new_logon***

Specifies that the job must run under a new user name in place of the original user name.

docommand=*"new_command"*

Specifies the new command that the job must run in place of the original command. This argument is mutually exclusive with the **script** argument.

script=*"new_script"*

Specifies the new script that the job must run in place of the original script. This argument is mutually exclusive with the **docommand** argument.

noask

Specifies not to prompt for confirmation before taking action on each qualifying job.

Comments

With **altjob conman** command, you can make changes to the job definition after it has already been submitted into the plan, while maintaining the original definition in the database. This can also be done from either the Job Stream Graphical View or the job monitoring view of the Dynamic Workload Console.

Note: When you edit the definition of a job in the plan that contains variables, the job runs and completes, but is unable to resolve the variables with their value.

For information about **jobinfo**, see the jobinfo topic in the chapter about the Utility commands in *User's Guide and Reference*.

See also

From the Dynamic Workload Console you can perform the same task as follows:

1. In the Welcome page, select **Monitor your workload**, or in the navigation bar at the top of the page, click **Monitoring & Reporting > Orchestration Monitoring > Monitor Workload**.
2. Select an engine.
3. In **Object Type**, select **Workstation/Job/Jobstream**.
4. From the **Query** drop-down list, select **All Jobs in plan** or another task to monitor jobs.
5. Click **Run** to run the monitoring task.
6. From the table containing the list of jobs, select a job and click **Edit Job**.

ALTPASS

Alters the password of a user object in the current production plan.

You must have ***altpass*** access to the user object.

Syntax

altpass
[*[folder/]workstation*#]
username
[;"*password*"]

Arguments

[folder/]workstation

Specifies the workstation on which the user is defined. Use the upper case for this field even though you used the mixed case when specifying the *workstation* in the user definition. For more information refer to the topic about the definition of users in *User's Guide and Reference*. Do not specify this field if the user belongs to a Windows domain managed by active directory. The default is the workstation on which you are running **conman**.

username

Specifies the name of a user. Use the same user name specified in the HCL Workload Automation database and note that they are case-sensitive. For more information, see *User's Guide and Reference*.

password

Specifies the new password. It must be enclosed in double quotation marks. To indicate no password for the user, use two consecutive double quotation marks ("").

The maximum length for passwords is **30 bytes**. However, 30 bytes does not always equal 30 characters, as some characters may require multiple bytes, depending on the chosen encoding.

The following characters are not supported by conman altpass, when changing the user password in the plan, and when using REST APIs v1:

- `\` – Backslash
- `/` – Slash (or forward slash)
- `&` – Ampersand
- `+` – Plus
- `=` – Equals
- `#` – Hash (or number sign)
- `[ ]` – Space

However, you can define passwords with special characters using one of the following methods:

Using REST API v2

This API version fully supports the use of special characters in password values.

Modifying the User Definition (Model):

You can directly modify the user password in the user definition in the model. Because the optman option **enAddUser /au** is set to YES by default, the modified password is immediately updated in the current plan. For more information about **enAddUser /au**, see the topic about global options in *Administration Guide*.

Comments

If you do not specify a *password*, **conman** prompts for a password and a confirmation. The password is not displayed as it is entered and should not be enclosed in quotes. Note that the change is made only in the current production plan, and is therefore temporary. To make a permanent change see the topic about the definition of Windows users in *User's Guide and Reference*.

Example

Examples

To change the password of user `Jim` on workstation `mis5`, stored in folder `myfolder`, to `mynewpw`, run the following command:

```
altpass myfolder/MIS5#JIM;"mynewpw"
```

To change the password of user `jim` on workstation `Mis5` to `mynewpw` without displaying the password, run the following command:

```
altpass MIS5#JIM
password: xxxxxxxx
confirm: xxxxxxxx
```

To change the password of user `Jim`, defined in an active directory managed Windows domain named `twsDom`, to `mynewpw`, run the following command:

```
altpass TWSDOM\JIM;"mynewpw"
```

See also

From the Dynamic Workload Console you can perform the same task as described in:

the *Dynamic Workload Console User's Guide*, section about Changing user password in the plan.

ALTPRI

Alters the priority of a job or job stream.

You must have ***altpri*** access to the job or job stream.

Syntax

{**altpri** | **ap**} *jobselect* | *jstreamselect*
[**;***pri*]
[**;noask**]

Arguments

jobselect

See JOBSELECT for information about selecting jobs in commands.

jstreamselect

See JSSELECT for information about selecting job streams in commands.

pri

Specifies the priority level. You can enter a value of **0** through **99**, **hi**, or **go**.

noask

Specifies not to prompt for confirmation before taking action on each qualifying job or job stream.

Example

Examples

To change the priority of the `balance` job in job stream `glmonth(0900 02/19/06)`, run the following command:

```
ap glmonth(0900 02/19/06).balance;55
```

To change the priority of job stream `glmonth(0900 02/19/06)`, run the following command:

```
ap glmonth(0900 02/19/06);10
```

See also

From the Dynamic Workload Console you can perform the same task as follows:

1. In the navigation bar at the top, click **Monitoring and Reporting > Orchestration Monitor**.
2. Select an engine.
3. In **Object Type**, select **Job** or **Job Stream**.
4. From the **Query** drop-down list, select a query to monitor jobs or job streams.
5. Click **Run** to run the monitoring task.
6. From the table containing the list of results, select the job or job stream whose priority you want to change and click **More Actions > Priority** tab.
7. In the displayed panel, set the new priority and click **OK**.

CANCEL JOB

Windows™ Cancels a job.

You must have ***cancel*** access to the job.

Syntax

{**cancel job** | **cj**} *jobselect*
  [**;pend**]
  [**;noask**]

Arguments

jobselect

See JOBSELECT for information about selecting jobs in commands.

pend

Cancels the job only after its dependencies are resolved.

noask

Specifies not to prompt for confirmation before taking action on each qualifying job.

Comments

If you cancel a job before it is launched, it does not launch. If you cancel a job after it is launched, it continues to run. If you cancel a job that is running and it completes in the ABEND state, no automatic job recovery steps are attempted.

If you do not use the **;pend** option, jobs and job streams that are dependent on the cancelled job are released immediately from the dependency.

If you include the **;pend** option, and the job has not been launched, cancellation is deferred until all of the dependencies, including an **at** time, are resolved. Once all the dependencies are resolved, the job is cancelled and any jobs or job streams that are dependent on the cancelled job are released from the dependency. During the period the cancel is deferred, the notation **[Cancel Pend]** is listed in the Dependencies column of the job in a **showjobs** display.

If you include the **;pend** option and the job has already been launched, the option is ignored, and any jobs or job streams that are dependent on the cancelled job are immediately released from the dependency.

You can use the **rerun** command to rerun jobs that have been cancelled, or that are marked **[Cancel Pend]**. You can also add and delete dependencies on jobs that are marked **[Cancel Pend]**.

To immediately cancel a job that is marked **[Cancel Pend]**, you can either enter a **release** command for the job or enter another **cancel** command without the **;pend** option.

For jobs with expired **until** times, the notation **[Until]** is listed in the Dependencies column in a **showjobs** display, and their dependencies are no longer evaluated. If such a job is also marked **[Cancel Pend]**, it is not cancelled until you release or delete the **until** time, or enter another **cancel** command without the **;pend** option.

To stop evaluating dependencies, set the priority of a job to zero with the **altpri** command. To resume dependency evaluation, set the priority to a value greater than zero.

Note: In the case of internetwork dependencies, cancelling a job in the `EXTERNAL` job stream releases all local jobs and job streams from the dependency. Jobs in the `EXTERNAL` job stream represent jobs and job streams that have been specified as internetwork dependencies. The status of an internetwork dependency is not checked after a **cancel** is performed. For more information see the topic on managing internetwork dependencies in *User's Guide and Reference*.

Example

Examples

To cancel job `report` in job stream `apwkly(0900 02/19/06)` on workstation `site3`, run the following command:

```
cj=site3#apwkly(0900 02/19/06).report
```

To cancel job `setup` in job stream `mis5(1100 02/10/06)`, if it is not in the ABEND state, run the following command:

```
cj mis5(1100 02/10/06).setup~state=abend
```

To cancel job `job3` in job stream `sked3(0900 02/19/03)` only after its dependencies are resolved, run the following command:

```
cj sked3(0900 02/19/06).job3;pend
```

See also

From the Dynamic Workload Console you can perform the same task as follows:

1. In the Welcome page, select **Monitor your workload**, or in the navigation bar at the top of the page, click **Monitoring & Reporting > Orchestration Monitoring > Monitor Workload**.
2. Select an engine.
3. In **Object Type**, select **Workstation/Job/Jobstream**.
4. Click **Run** to run the monitoring task.
5. From the table containing the list of jobs, select a job and click **More Actions > Cancel**.

CANCEL SCHED

Cancels a job stream.

You must have ***cancel*** access to the job stream.

Syntax

{**cancel sched** | **cs**} *jstreamselect*
[**;pend**]
[**;noask**]

Arguments

jstreamselect

See JSSELECT for information about selecting job streams in commands.

pend

Cancels the job stream only after its dependencies are resolved.

noask

Specifies not to prompt for confirmation before taking action on each qualifying job stream.

Comments

If you cancel a job stream before it is launched, it does not launch. If you cancel a job stream after it is launched, the jobs that have started complete, but no other jobs are launched.

If you do not use the **;pend** option, jobs and job streams that are dependent on the cancelled job stream are released immediately from the dependency.

If you use the **;pend** option and the job stream has not been launched, cancellation is deferred until all of its dependencies, including an **at** time, are resolved. Once all dependencies are resolved, the job stream is cancelled and any dependent jobs or job streams are released from the dependency. During the period the **cancel** is deferred, the notation **[Cancel Pend]** is listed in the Dependencies column of a **showschedules** display.

If you include the **;pend** option and the job stream has already been launched, any remaining jobs in the job stream are cancelled, and any dependent jobs and job streams are released from the dependency.

To immediately cancel a job stream marked **[Cancel Pend]**, either enter a **release** command for the job stream or enter another **cancel** command without the **;pend** option.

To stop evaluating dependencies, set the job stream's priority to zero with the **altpri** command. To resume dependency evaluation, set the priority to a value greater than zero.

If the cancelled job stream contains jobs defined with the **every** option, only the last instance of such jobs is listed as canceled in a **showjobs** display.

Example

Examples

To cancel job stream `sked1(1200 02/17/23)` on workstation `site2`, run the following command:

```
cs=site2#sked1(1200 02/17)
```

To cancel job stream `mis2(0900 02/19/23)` if it is in the STUCK state, run the following command:

```
cs mis2(0900 02/19)+state=stuck
```

See also

From the Dynamic Workload Console you can perform the same task as follows:

1. In the navigation bar at the top, click **Monitoring and Reporting > Orchestration Monitor**.
2. Select an engine.
3. In **Object Type**, select **Workstation/Job Stream/Job**.
4. From the **Query** drop-down list, select **All Job Streams in plan** or another task to monitor job streams.
5. Click **Run** to run the monitoring task.
6. Select a job stream and click **More Actions > Cancel**.

CHECKHEALTHSTATUS

Invokes **chkhltst** service to verify the connectivity between the domain manager and workstations. It checks that the Symphony file is not corrupted, the mailbox files can be successfully read by **mailman**, without errors in the mailbox header, and that the mailbox is not full. This command can be useful to diagnose the reason for an unlinked workstation and to get suggestions about how to recover the problem.

Syntax

{**checkhealthstatus** | **chs**} [*[folder/]workstation*]

Arguments

[folder/]workstation

Specifies the workstation on which to run the **chkhltst** service. If *workstation* is not specified, the service is launched locally.

Example

Examples

To check the health status of the *site1* workstation, stored in folder `myfolder`, launch the following command:

```
checkhealthstatus myfolder/site1
```

See also

From the Dynamic Workload Console you can perform the same task as follows:

1. In the navigation bar at the top, click **Monitoring and Reporting > Orchestration Monitor**.
2. Select an engine.
3. From the drop-down menu, select **Workstation**.
4. From the **Query** drop-down list, select a query to monitor workstations.
5. From workstations table, select the workstation whose connectivity you want to check and click **More Actions > Check Health Status.**.
6. Click **Ok** to proceed with the **Check Health Status** action.

CHFOLDER

Use to navigate folders in the plan. This command changes the working directory or current directory, which is set to root ("/") by default, so that you can use relative folder paths when submitting commands.

Authorization

To **change** folder, you must have ***display*** access to the folder.

Syntax

{**chfolder** | **cf**} *foldername*

Arguments

foldername

Is the name and path where the folder is located and to which you want to navigate.

Comments

When submitting commands involving folders, you can specify either a relative or absolute path to the folder. If no folder is specified, then the command is performed using the current folder. If you generally work from a specific folder, then you can use the chfolder command to set a new current folder. The chfolder command changes the working directory or current folder, which is set to root ("/") by default, so that you can use relative folder paths when submitting commands.

There are additional ways you can modify the current folder. The following methods for setting the current folder are also ordered by precedence, for example, the value specified in the customer properties file overrides the current folder setting in the `localopts` file.

CustomParameters

You can pass **chfolder | cf** as a parameter together with a conman command from the conman command line. For example, to show job streams in the folder named `/TEMPLATE/`, then submit the following from the conman command line:

```
conman -cf /TEMPLATE/ ss @#@
```

You can obtain the same result by changing the current folder in conman first, then submitting the command:

```
%cf /TEMPLATE/
```

```
%ss @#@
```

UNISON_CURRENT_FOLDER

Set the value of the current folder using the UNISON_CURRENT_FOLDER environment variable.

Custom properties file

You can use the **CURRENT FOLDER** parameter to set the value of the current folder in the custom properties file that can be passed to conman. The parameters in the custom properties file override the values specified in the `useropts` and `localopts` files. For example, set the parameter as follows:

```
CURRENT FOLDER = /foldername
```

See RUNCONMAN Running the conman program on page 548 for information about the custom properties file.

`useropts`

You can set the current folder in the `useropts` file. The `useropts` file contains values for `localopts` parameters that you can personalize for individual users. Set the current folder as follows:

```
current folder = /foldername
```

`localopts`

You can set the current folder in the localopts file as follows:

```
------------------------------------
# Current Folder
#
current folder = /mycurrentfolder
```

The maximum length for the full folder path (that is, the path name including the parent folder and all nested sub-folders) is 800 characters, while each folder name supports a maximum of 256 characters.

Example

Examples

To change the current folder to "TEST", run:

```
chfolder /TEST
```

CONFIRM

Confirms the completion of a job that was scheduled with the **confirmed** keyword. By default, evaluation on the job is performed when the job completes. However, if you confirm the job while it is running, the confirmation overrides the evaluation performed at job completion time. You can also override the evaluation of the output conditions: for example, if you set one or more output conditions to true (using confirm SUCC), the specified output conditions are set to true and any other conditions in the job are set to false.

You must have ***confirm*** access to the job.

Syntax

{**confirm** | **conf**} *jobselect*
;{**succ** | **abend**}
[**;IF** '*output_condition_name*[, *output_condition_name*]
[, ...]'] [**;noask**]

Arguments

jobselect

See JOBSELECT for information about selecting jobs in commands.

succ

Confirms that the job ended successfully.

abend

Confirms that the job ended unsuccessfully.

output_condition_name

Confirms the SUCC or ABEND status for one or more specified output conditions. Any conditions not specified are set to not satisfied. This setting overrides any other evaluation.

noask

Specifies not to prompt for confirmation before taking action on each qualifying job.

Comments

Changing the state of a job from ABEND to SUCC does not require that the **confirmed** keyword be used to schedule the job. For more information about job confirmation, see the topic about the **confirmed** keyword in *User's Guide and Reference*. For more information about `EXTERNAL` jobs, see the topic on managing internetwork dependencies in *User's Guide and Reference*.

The following table shows the effect of the **confirm** command on the various states of jobs, with or without output conditions:

Table 77. State change after confirm command

Initial Job State	State after confirm ;succ	State after confirm ;abend
READY	No effect, with or without output conditions	No effect, with or without output conditions
HOLD	No effect, with or without output conditions	No effect, with or without output conditions
EXEC	SUCC_P, with or without output conditions	ABEND_P, with or without output conditions
ABENP	**SUCCP**, with or without output conditions	**without output conditions** No effect. **with output conditions** ABENP and selected output conditions are set to satisfied.
SUCCP	No effect, with or without output conditions	No effect, with or without output conditions
PEND	**without output conditions** SUCC **with output conditions** SUCC and selected output conditions are set to satisfied.	**without output conditions** ABEND **with output conditions** ABEND and selected output conditions are set to satisfied.
DONE	**SUCC**, with or without output conditions.	**ABEND**, with or without output conditions.
SUCC	**without output conditions** The operation is not supported. **with output conditions** SUCC and selected output conditions are set to satisfied.	**without output conditions** The operation is not supported. **with output conditions** The operation is not supported.
ABEND	**SUCC**, with or without output conditions.	**without output conditions** No effect.

Table 77. State change after confirm command (continued)

Initial Job State	State after confirm ;succ	State after confirm ;abend
		with output conditions ABEND and selected output conditions are set to satisfied.
SUPPR	**without output conditions** SUCC **with output conditions** SUCC and selected output conditions are set to satisfied.	The operation is not supported, with or without output conditions.
FAIL	The operation is not supported, with or without output conditions.	The operation is not supported, with or without output conditions.
SCHED	No effect, with or without output conditions	No effect, with or without output conditions
ERROR (for shadow jobs only)	**SUCC**, with or without output conditions	**ABEND**, with or without output conditions
any job in the **EXTERNAL** job stream	**SUCC**, with or without output conditions	**ABEND**, with or without output conditions

Example

Examples

To issue a **succ** confirmation for job `job3` in job stream `misdly(1200 02/17/23)`, run the following command:

```
confirm misdly(1200 02/17/23).job3;succ
```

To issue an **abend** confirmation for job number **234**, run the following command:

```
confirm 234;abend
```

To issue a **;succ** confirmation for job `job4` and set `MYOUTPUTCOND` to **true** in the `daily(1130 02/17/2023)` job stream, run the following command:

```
confirm daily(1130 02/17/2023).job4;succ if MYOUTPUTCOND
```

The following example shows the effect of the **confirm** command on the status of the `79765613` job and its output conditions. The `79765613` job completed in SUCC status:

1. Type the **showjobs;info** command to retrieve information about the job:

   ```
   sj 79765613;info
   ```

   The following sample output is a subset of the information you obtain by running the command:

   ```
   Workstation      Job Stream       SchedTime       Job
   NC050239        #DUBAIJS1         1908 10/23      (NC050239_1#)JS2

                                                  oc: OUT1 false "RC=2"
                                                  oc: OUT2 true "RC=1500"
   ```

   The first output condition, `OUT1` is **false** and the second one, `OUT2`, is **true**.

2. Type the **confirm** command to confirm the ABEND status on condition `OUT1`:

   ```
   confirm 79765613 ABEND; IF OUT1
   ```

3. Type the **showjobs;info** command again. The following sample output is a subset of the information you obtain by running the command:

   ```
   Workstation      Job Stream       SchedTime       Job
   NC050239        #DUBAIJS1         1908 10/23      (NC050239_1#)JS2

                                                 oc: OUT1 true "RC=2"
                                                 oc: OUT2 false "RC=1500"
   ```

   The first output condition, `OUT1` has now changed to **true** and the second one, `OUT2`, has changed to **false**.

CONSOLE

Assigns the HCL Workload Automation console and sets the message level.

You must have ***console*** access to the workstation.

Syntax

{**console** | **cons**}
  [**sess** | **sys**]
  [**;level=***msglevel*]

Arguments

sess

Sends HCL Workload Automation console messages and prompts to standard output.

sys

Stops sending HCL Workload Automation console messages and prompts to standard output. This occurs automatically when you exit **conman**.

msglevel

The level of HCL Workload Automation messages that are sent to the console. Specify one of the following levels:

-1

This is the value the product automatically assigns if you modify any of the arguments for the console and you do not reassign any value to `msglevel`. With this value the product sends all the messages generated by all agents and for all operations to the console.

0

No messages. This is the default on fault-tolerant agents.

1

Exception messages such as operator prompts and job abends.

2

Level 1, plus job stream successful messages.

3

Level 2, plus job successful messages. This is the default on the master domain manager.

4

Level 3, plus job launched messages.

Comments

If you enter a **console** command with no options, the current state of the console is displayed.

By default, HCL Workload Automation control processes write console messages and prompts to standard list files. In UNIX®, you can also have them sent to the **syslog** daemon.

Example

Examples

To begin writing console messages and prompts to standard output and change the message level to **1**, run the following command:

```
console sess;level=1
```

To stop writing console messages and prompts to standard output and change the message level to **4**, run the following command:

```
cons sys;l=4
```

To display the current state of the console, run the following command:

```
cons
Console is #J675, level 2, session
```

675 is the process ID of the user's shell.

CONTINUE

Ignores the next command error.

Syntax

{**continue** | **cont**}

Comments

This command is useful when commands are entered non-interactively. It instructs **conman** to continue running commands even if the next command, following **continue**, results in an error.

Example

Examples

To have **conman** continue with the **rerun** command even if the **cancel** command fails, run the following command:

```
conman "cont&cancel=176&rerun job=sked5(1200 02/17/23).job3"
```

DELDEP JOB

Deletes dependencies from a job.

You must have ***deldep*** access to the job.

Syntax

{**deldep job** | **ddj**} *jobselect*
;*dependency*[**;**...]
[**;noask**]

Arguments

jobselect

See JOBSELECT for information about selecting jobs in commands.

dependency

The type of dependency. Specify at least one of the following. You can use wildcard characters in *workstation*, *jstream*, *job*, *resource*, *filename*, and *promptname*.

at[**=***time* | *lowtime* | *hightime* | *lowtime*,*hightime*]

confirmed

deadline[**=***time*[**timezone** | **tz** *tzname*][**+***n* **days** | *mm/dd[/yy]*]]

every

follows=[*netagent*::][*workstation#*]{*jobstreamname*[*hhmm* [*mm/dd[/yy]*]][.*job* | @] | *jobstream_id.job*;**schedid**}| *job*[,...] [**if** '*condition_name*[| *condition_name*][| ...]']

Note: Internetwork dependencies do not support folders, therefore, the network agent workstation, and the jobs and job streams running on them, cannot be defined in a folder different from the root (/). Folders are supported on all other workstation types as follows:

```
[follows {[[folder/]workstation#][folder/]jobstreamname[.jobname]
```

follows=[[*folder/*]*workstation#*][*folder/*]{*jobstreamname*[*hhmm* [*mm/dd[/yy]*]][.*job* | @] | *jobstream_id.job*;**schedid**}| *job*[,...] [**if** '*condition_name*[| *condition_name*][| ...]']

The *condition_name* variable indicates the name of the condition defined in the job definition. Conditions must be separated by | and enclosed between single quotes. Conditions can be **status conditions**, based on job status, or other **output conditions**, based on a mapping expression such as a return code, output variables, or output found in a job log. In each follows statement, you can specify only one dependency type: either status or output conditions. At submission time, you can delete status or output conditions. If the conditional dependency belongs to a join, if the number of conditions that must be met is different from ALL, the number is automatically reduced by one.

maxdur=[*hhhmm*] [**onmaxdur** *action*]

mindur=[*hhhmm*] [**onmindur** *action*]

needs[**=**[*num*] [*[folder/]workstation#*][*folder/]resource*[**,**...]]

opens[**=**[*[folder/]workstation#*]"*filename*"[**(***qualifier***)**][**,**...]]

priority

prompt[**="**[**:** | **!**]*text*" | *promptname*[**,**...]]

until[**=***time* [**timezone**|**tz** *tzname*][**+***n* **day**[**s**]] [**;onuntil** *action*]]

noask

Specifies not to prompt for confirmation before taking action on each qualifying job.

Comments

If you delete `priority`, the job reverts to its original scheduled priority. When you delete an `opens` dependency, you can include only the base file name and **conman** performs a case-insensitive search for matching files, ignoring the directory names. Dependencies on all matching files are deleted.

Deleted dependencies no longer remain in effect when running the **rerun** command.

To delete all the `follows` dependencies from the jobs contained in a specific job stream, specify the `follows` keyword as:

```
follows=job_stream_name
```

Do not use a wildcard in this case (such as `follows=job_stream_name.@` because the command will be rejected.

Example

Examples

To delete a resource dependency from job `job3` in job stream `sked9(0900 02/19/06)`, run the following command:

```
ddj sked9(0900 02/19/06).job3 ; needs=2 tapes
```

To delete all external follows dependency from job stream `CPUA#TEST(0900 02/19/06)`, run the following command:

```
ddj=CPUA#TEST(0900 02/19/06).JOBA ; follows
```

See also

From the Dynamic Workload Console you can perform the same task as follows:

1. From the **Monitoring and Reporting** menu, click **Orchestration Monitor**.
2. Select an engine.
3. In **Object Type**, select **Workstation/Job/Jobstream**.
4. From the **Query** drop-down list, select **All Jobs in plan** or another task to monitor jobs.
5. Click **Run** to run the monitoring task.
6. From the table containing the list of jobs, select the job from which you want to remove a dependency and click **Dependencies...**.
7. In the Dependencies panel, select the dependency you want to delete and click **Delete**.

DELDEP SCHED

Deletes dependencies from a job stream.

You must have ***deldep*** access to the job stream.

Syntax

{**deldep sched** | **dds**} *jstreamselect*
;*dependency*[**;**...]
[**;noask**]

Arguments

jstreamselect

See JOBSELECT for information about selecting jobs in commands.

dependency

The type of dependency. Specify at least one of the following. You can use wildcard characters in *workstation*, *jstreamname*, *jobname*, *resource*, *filename*, and *promptname*, with the exception of *workstation* when used in a follows dependency.

at[**=***time* | *lowtime* | *hightime* | *lowtime*,*hightime*]

carryforward

deadline[**=***time*[**timezone** | **tz** *tzname*][**+***n* **days** | *mm/dd[/yy]*]]

follows=[*netagent***::**][*workstation***#**]{*jobstreamname*[*hhmm* [*mm/dd[/yy]*]][*.job* | **@**] | *jobstream_id.job***;schedid**}| *job*[**,**...] [**if** '*condition_name*[| *condition_name*][| ...]']

Note: Internetwork dependencies do not support folders, therefore, the network agent workstation, and the jobs and job streams running on them, cannot be defined in a folder different from the root (/). Folders are supported on all other workstation types as follows:

```
[follows {[[folder/]workstation#][folder/]jobstreamname[.jobname]
```

follows=[[*folder/*]*workstation***#**][*folder/*]{*jobstreamname*[*hhmm* [*mm/dd[/yy]*]][*.job* | **@**] | *jobstream_id.job***;schedid**}| *job*[**,**...] [**if** '*condition_name*[| *condition_name*][| ...]']

The *condition_name* variable indicates the name of the condition defined in the job definition. Conditions must be separated by | and enclosed between single quotes. Conditions can be **status conditions**, based on job status, or other **output conditions**, based on a mapping expression such as a return code, output variables, or output found in a job log. In each follows statement, you can specify only one dependency type: either status or output conditions. At submission time, you can delete status or output conditions. If the conditional dependency belongs to a join, if the number of conditions that must be met is different from ALL, the number is automatically reduced by one.

limit

needs[**=**[*num*] [*[folder/]workstation***#**][*folder/]resource*[**,**...]]

opens[**=**[*[folder/]workstation***#**]"*filename*"[**(***qualifier***)**][**,**...]]

priority

prompt[**="[:** | **!**]*text*" | *promptname*[**,**...]]

until[**=***time* [**timezone**|**tz** *tzname*][**+***n* **day**[**s**]] [**;onuntil** *action*]]

noask

Specifies not to prompt for confirmation before taking action on each qualifying job stream.

Comments

If you delete `priority` , the job reverts to its original scheduled priority. When you delete an `opens` dependency, you can include only the base file name, and **conman** performs a case-insensitive search for matching files, ignoring the directory names. Dependencies on all matching files are deleted.

Deleted dependencies no longer remain in effect when running the **rerun** command.

Example

Examples

To delete a `resource` dependency from job stream `sked5(0900 02/19/22)`, run the following command:

```
dds sked5(0900 02/19/22);needs=2 tapes
```

To delete all `follows` dependencies from job stream `sked3(1000 04/19/06)`, run the following command:

```
dds sked3(1000 04/19/22);follows
```

See also

1. In the navigation bar at the top, click **Monitoring and Reporting > Orchestration Monitor**.
2. Select an engine.
3. In **Object Type**, select **Workstation/Job Stream/Job**.
4. From the **Query** drop-down list, select **All Job Streams in plan** or another task to monitor job streams.
5. Click **Run** to run the monitoring task.
6. From the table containing the list of job streams, select the job streams from which you want to remove a dependency and click **Dependencies...**.
7. In the Dependencies panel, select the dependency you want to delete and click **Delete**.

DEPLOYCONF

Downloads the latest monitoring configuration for the event monitoring engine on a workstation.

Syntax

{**deployconf** | **deploy**} [*domain***!**]*[folder/]workstation*

Arguments

domain

Specifies the name of the destination domain for the operation. Wildcard characters are not permitted.

If you do not include *domain*, the default domain is the one in which **conman** is running.

[folder/]workstation

Specifies the name of the workstation to which the configuration is to be deployed. Wildcard characters are not permitted.

Comments

Use this command to deploy to one workstation at a time.

If the existing configuration is already up-to-date, the command has no effect.

Permission to `start` actions on `cpu` objects is required in the security file to be enabled to run this command.

Example

Examples

To deploy the latest monitoring configuration for the event monitoring engine on workstation `mis5`, stored in folder `myfolder`, run the following command:

```
deployconf myfolder/MIS5
```

DISPLAY

Displays a job file or a job stream definition.

If you specify a file by name, you must have read access to the file. For job files and job stream definitions, you must have ***display*** access to the job or job stream.

Syntax

{**display file** | **df**} *filename* [**;offline**]

{**display job** | **dj**} *jobselect* [**;offline**]

{**display sched** | **ds**} *jstreamselect*
[**valid** {**at** *date* | **in** *date date*}
[**;offline**]

Arguments

filename

Specifies the name of the file, usually a job script file. The name must be enclosed in quotes (") if it contains characters other than the following: alphanumeric characters, dashes (-), slashes (/), backslashes (\), and underscores (_). Wildcard characters are permitted. The file must be accessible from your login workstation. Use this option is you want to show only the content of the job script file.

jobselect

The job whose job file is displayed. See JOBSELECT for information about selecting jobs in commands. The job file must be accessible from your login workstation. This keyword applies only to path and filename of the script file of jobs defined with the **scriptname** option.

jstreamselect

The job stream whose definition is displayed. See JSSELECT for information about selecting job streams in commands.

valid

Specifies the day or the interval of days during which the job stream instances to be displayed must be active. This means that the validity interval of those job stream instances must contain the time frame specified in

valid argument. The format used for *date* depends on the value assigned to the ***date format*** variable specified in the `localopts` file. If not specified the selected instance is the one valid today.

offline

Sends the output of the command to the **conman** output device. For information about this device, see the offline output topic in SETUPCONMAN.

Example

Examples

To display the file `c:\maestro\jclfiles\arjob3`, run the following command:

```
df c:\apps\maestro\jclfiles\arjob3
```

To display the script file for job `createpostreports` in job stream `FINALPOSTREPORTS` offline, run the following command:

```
dj FINALPOSTREPORTS(2359 02/14/23).CREATEPOSTREPORTS
```

This is a sample output of this command:

```
M235062_99#FINALPOSTREPORTS(2359 02/14/23).CREATEPOSTREPORTS /opt/TWA/TWS/
                                                        CreatePostReports
   #!/bin/sh
  ######################################################################
  # Licensed Materials - Property of IBM* and HCL**
  # 5698-WSH
  # (C) Copyright IBM Corp. 1998, 2016 All rights reserved.
  # (C) Copyright HCL Technologies Ltd. 2016, 2026 All rights reserved.
  # * Trademark of International Business Machines
  # ** Trademark of HCL Technologies Limited
  ######################################################################
  #@(#) $Id: CreatePostReports.sh,v 1.0

  ##
  ## CreatePostReports message catalog definitions.
  ##

  ##
  ## message set id
  ##
  MAE_CREATEPOSTREPORTS_SET=226
  MAE_COPYRIGHT_SET=234

  ##
  ....
  ...
  ....
  #
  # End
  #
```

To display the job stream definition for job stream `mod`, run the following command:

```
ds mod
```

This is a sample output of this command:

```
Job Stream Name   Workstation       Valid From  Updated On  Locked By
 ----------------  ----------------  ----------  ----------  ----------------
MOD               M235062_99        03/30/2023  04/04/2023  -

SCHEDULE M235062_99#MOD VALIDFROM 03/30/2023
ON RUNCYCLE SCHED1_PREDSIMPLE VALIDFROM 07/18/2023 "FREQ=DAILY;INTERVAL=1"
( AT 1111 )
CARRYFORWARD
FOLLOWS M235062_99#SCHED_FIRST1.@
FOLLOWS M235062_99#SCHED_FIRST.JOB_FTA
PRIORITY 66
:
M235062_99#JOBMDM
 SCRIPTNAME "/usr/acct/scripts/gl1" STREAMLOGON root
 DESCRIPTION "general ledger job1"
 TASKTYPE UNIX
 RECOVERY STOP
 PRIORITY 30
NEEDS 16 M235062_99#JOBSLOTS
PROMPT PRMT3

B236153_00#JOB_FTA
FOLLOWS M235062_99#SCHED_FIRST1.@
FOLLOWS M235062_99#SCHED_FIRST.JOB_FTA
PRIORITY 66
:
M235062_99#JOBMDM
 SCRIPTNAME "/usr/acct/scripts/gl1" STREAMLOGON root
 DESCRIPTION "general ledger job1"
 TASKTYPE UNIX
 RECOVERY STOP
 PRIORITY 30
NEEDS 16 M235062_99#JOBSLOTS
PROMPT PRMT3

B236153_00#JOB_FTA
 DOCOMMAND "echo pippo" STREAMLOGON root
 DESCRIPTION "general ledger job1"
 TASKTYPE UNIX
 RECOVERY STOP
 FOLLOWS JOBMDM
END
```

See also

From the Dynamic Workload Console you can perform the same task as described in:

the Dynamic Workload Console User's Guide, section about Listing object definitions in the database.

For more information about how to create and edit scheduling objects, see:

the Dynamic Workload Console User's Guide, section about Designing your Workload.

EXIT

Exits the **conman** command line program.

Syntax

{**exit** | **e**}

Comments

When you are in help mode in UNIX®, this command returns **conman** to command-input mode.

Example

Examples

To exit the **conman** command-line program, run the following command:

```
exit
```

or

```
e
```

FENCE

Changes the job fence on a workstation. Jobs are not launched on the workstation if their priorities are less than or equal to the job fence.

You must have ***fence*** access to the workstation.

Syntax

{**fence** | **f**} *workstation*
;*pri*
[**;noask**]

Arguments

workstation

Specifies the workstation name. The default is your login workstation.

pri

Specifies the priority level. You can enter **0** through **99**, **hi**, **go**, or **system**. Entering **system** sets the job fence to zero.

noask

Specifies not to prompt for confirmation before taking action on each qualifying workstation.

Comments

The job fence prevents low priority jobs from being launched, regardless of the priorities of their job streams. It is possible, therefore, to hold back low priority jobs in high priority job streams, while allowing high priority jobs in low priority job streams to be launched.

When you first start HCL Workload Automation following installation, the job fence is set to zero. When you change the job fence, it is carried forward during preproduction processing to the next day's production plan.

To display the current setting of the job fence, use the **status** command.

Example

Examples

To change the job fence on workstation `site4`, run the following command:

```
fence site4;20
```

To change the job fence on the workstation on which you are running **conman**, run the following command:

```
f ;40
```

To prevent all jobs from being launched by HCL Workload Automation on workstation `tx3`, run the following command:

```
f tx3;go
```

To change the job fence to zero on the workstation on which you are running **conman**, run the following command:

```
f ;system
```

See also

From the Dynamic Workload Console you can perform the same task as follows:

1. In the navigation bar at the top, click **Monitoring and Reporting > Orchestration Monitor**.
2. Select an engine.
3. From the drop-down menu, select **Workstation**.
4. From the **Query** drop-down list, select a query to monitor workstations.
5. Click **Run** to run the monitoring task.
6. In the displayed panel, set the new priority and click **OK**.

HELP

Displays help information about commands. Not available in Windows®.

Syntax

{**help** | **h**} {*command*|*keyword*}

Arguments

command

Specifies the name of a **conman** or system command. For **conman** commands, enter the full command name; abbreviations and short forms are not supported. For commands consisting of two words, enter the first word, and help for all versions of the command is displayed. For example, entering **help display** displays information about the **display file**, **display job**, and **display sched** commands.

keyword

You can also enter the following keywords:

COMMANDS

Lists all conman commands.

SETUPCONMAN

Describes how to setup to use conman.

RUNCONMAN

How to run conman.

SPECIALCHAR

Describes the wildcards, delimiters and other special characters you can use.

JOBSELECT

Lists information about selecting jobs for commands.

JOBSTATES

Lists information about job states.

JSSELECT

Lists information about selecting job streams for commands.

JSSTATES

Lists information about job stream states.

MANAGEBACKLEVEL

Managing jobs and job streams from back-level agents.

Example

Examples

To display a list of all **conman** commands, run the following command:

```
help commands
```

To display information about the **fence** command, run the following command:

```
help fence
```

To display information about the **altpri job** and **altpri sched** commands, run the following command:

```
h altpri
```

To display information about the special characters you can use, run the following command:

```
h specialchar
```

KILL

Stops a job that is running. In UNIX®, this is accomplished with a UNIX® **kill** command. You must have ***kill*** access to the job.

Syntax

{**kill** | **k**} *jobselect*
[**;noask**]

Arguments

jobselect

See JOBSELECT for information about selecting jobs in commands.

noask

Specifies not to prompt for confirmation before taking action on each qualifying job.

Comments

The **kill** operation is not performed by **conman**; it is run by an HCL Workload Automation production process, so there might be a short delay.

Killed jobs end in the ABEND state. Any jobs or job streams that are dependent on a killed job are not released. Killed jobs can be rerun.

Example

Examples

To kill the job `report` in job stream `apwkly(0600 03/05/06)` on workstation `site3`, run the following command:

```
kill site3#apwkly(0600 03/05/06).report
```

To kill job number `124` running on workstation `geneva`, run the following command:

```
kill geneva#124
```

See also

From the Dynamic Workload Console you can perform the same task as follows:

1. In the Welcome page, select **Monitor your workload**, or in the navigation bar at the top of the page, click **Monitoring & Reporting > Orchestration Monitoring > Monitor Workload**.
2. Select an engine.
3. In **Object Type**, select **Workstation/Job/Jobstream**.
4. From the **Query** drop-down list, select **All Jobs in plan** or another task to monitor jobs.
5. Click **Run** to run the monitoring task.
6. From the table containing the list of jobs, select the job you want to kill and click **More Actions > Kill** .

LIMIT CPU

Changes the limit of jobs that can run simultaneously on a workstation. You must have ***limit*** access to the workstation.

Syntax

{**limit cpu** | **lc** } *[folder/]workstation*
;*limit*
[**;noask**]

Arguments

[folder/]workstation

Specifies the name of the workstation. Wildcard characters are permitted. The default is your login workstation.

limit

Specifies the how many jobs can run concurrently on the workstation. Supported values are from **0** to **1024** and **system**.

If you set limit cpu to **0**:

- For a job stream in the **READY** state, only jobs with **hi** and **go** priority values can be launched on the workstation.
- For a job stream with a **hi** or **go** priority value, all jobs with a priority value other than 0 can be launched on the workstation.

If you set limit cpu to **system**, there is no limit to the number of concurrent jobs on the workstation. For the extended agent, the limit to SYSTEM sets the job limit to 0.

noask

Specifies not to prompt for confirmation before taking action on each qualifying workstation.

Comments

To display the current job limit on your login workstation, use the **status** command.

When you first start HCL Workload Automation following installation, the workstation job limit is set to zero, and must be increased before any jobs are launched. When you change the limit, it is carried forward during preproduction processing to the next day's production plan.

HCL Workload Automation attempts to launch as many jobs as possible within the job limit. There is a practical limit to the number of processes that can be started on a workstation. If this limit is reached, the system responds with a message indicating that system resources are not available. When a job cannot be launched for this reason, it enters the **fail** state. Lowering the job limit can prevent this from occurring.

Example

Examples

To change the job limit on the workstation on which you are running **conman**, run the following command:

```
lc ;12
```

To change the job limit on workstation `rx12`, stored in folder `myfolder`, run the following command:

```
lc myfolder/rx12;6
```

To set to 10 the job limit on all the workstations belonging to the domain and to child domains, run the following command:

```
lc @!@;10
```

See also

From the Dynamic Workload Console you can perform the same task as follows:

1. In the navigation bar at the top, click **Monitoring and Reporting > Orchestration Monitor**.
2. Select an engine.
3. From the drop-down menu, select **Workstation**.
4. From the **Query** drop-down list, select a query to monitor workstations.
5. Click **Run** to run the monitoring task.
6. From the table containing the list of workstations, select a workstation and click **More Actions > Limit...**.
7. Set the limit as required.

LIMIT SCHED

Changes the **limit** set in the definition of a job stream. For additional information on setting a limit in a job stream definition, refer to the topic on the "limit" job stream keyword in *User's Guide and Reference*. You must have ***limit*** access to the job stream.

Syntax

{**limit sched** | **ls** } *jstreamselect*
;*limit*
[**;noask**]

Arguments

jstreamselect

See JSSELECT for information about selecting job streams in commands.

limit

Specifies the job limit. You can enter **0** through **1024**.

noask

Specifies not to prompt for confirmation before taking action on each qualifying job stream.

Example

Examples

To change the job limit on all job streams that include `sales` in their name, run the following command:

```
ls sales@;4
```

To change the job limit on job stream `CPUA#Job1`, run the following command:

```
ls=CPUA#apwkly;6
```

See also

From the Dynamic Workload Console you can perform the same task as follows:

1. In the navigation bar at the top, click **Monitoring and Reporting > Orchestration Monitor**.
2. Select an engine.
3. In **Object Type**, select **Workstation/Job Stream/Job**.
4. Click **Run** to run the monitoring task.
5. Select a job stream and click **More Actions > Limit...**.
6. Set the limit as required.

LINK

Opens communication links between workstations. In an HCL Workload Automation network, fault-tolerant and standard agents are linked to their domain managers, and domain managers are linked to their parent domain managers. Extended agents are not linked; they communicate through a host.

You must have ***link*** access to the target workstation.

The command requires that another workstation be present in your environment in addition to the master domain manager.

Syntax

{**link** | **lk**} [*domain***!**][*folder/*]*workstation*
  [**;noask**]

Arguments

domain

Specifies the name of the domain in which links are opened. Wildcard characters are permitted.

This argument is useful when linking more than one workstation in a domain. For example, to link all the agents in domain `stlouis`, use the following command:

```
lk stlouis!@
```

The domain is not needed if you do not include wildcard characters in *workstation*.

If you do not include *domain*, and you include wildcard characters in *workstation*, the default domain is the one in which **conman** is running.

[folder/]workstation

Specifies the name of the workstation to be linked. Wildcard characters are permitted.

This command is not supported on remote engine workstations.

noask

Specifies not to prompt for confirmation before taking action on each qualifying workstation.

Comments

If the **autolink** option is set to **on** in a workstation definition, its link is opened automatically each time HCL Workload Automation is started. If **autolink** is set to **off**, you must use **link** and **unlink** commands to control linking. For information about **autolink** see the topic on workstation definitions in *User's Guide and Reference*.

Assuming that a user has **link** access to the workstations being linked, the following rules apply:

- A user running **conman** on the master domain manager can link any workstation in the network.
- A user running **conman** on a domain manager other than the master can link any workstation in its own domain and subordinate domains. The user cannot link workstations in peer domains.
- A user running **conman** on an agent can link any workstation in its local domain provided that the workstation is a domain manager or host. A peer agent in the local domain cannot be linked.
- To link a subordinate domain while running **conman** in a higher domain, it is not necessary that the intervening links be open.

Example

Examples

A figure showing a network and a table of the links in it opened by "link" commands run by users in various locations in the network can be found in the topic on the "link" command in *User's Guide and Reference*.

See also

From the Dynamic Workload Console you can perform the same task as follows:

1. In the navigation bar at the top, click **Monitoring and Reporting > Orchestration Monitor**.
2. Select an engine.
3. From the drop-down menu, select **Workstation**.
4. From the **Query** drop-down list, select a query to monitor workstations.
5. Click **Run** to run the monitoring task.
6. From the table containing the list of workstations, select a workstation and click **Link**.

LISTFOLDER

Lists folders defined in the plan, either in the root or in a folder.

Authorization

If the ***enListSecChk*** global option is set to **yes** on the master domain manager, then to list a folder, you must have either ***list*** access, or ***list*** and ***display*** access.

Syntax

{**listfolder** | **lf**} *foldername*

Arguments

foldername

If only a forward slash ("/") is specified, then all folders in the root are listed.

If a folder name follows, then all folders contained in the specified folder are listed, excluding the specified folder. Wildcard characters are permitted.

To list all folders in an entire tree structure, specify the ampersand character ("@")

Example

Examples

To list all folders in the root, run:

```
listfolder /
```

To list all folders contained in the folder named "Test", run:

```
listfolder /Test/
```

To list all folders and sub-folders contained in the folder named "Test", run:

```
listfolder /Test/@
```

LISTSYM

Lists the production plan (Symphony files) already processed.

When used from a fault-tolerant agent command line, this command shows the latest Symphony file, saved as `MSymOldBackup`.

Syntax

{**listsym** | **lis**} [**trial** | **forecast**]
[**;offline**]

Arguments

trial

Lists trial plans.

forecast

Lists forecast plans.

offline

Sends the output of the command to the **conman** output device. For information about this device, see the offline output topic in SETUPCONMAN.

Results

Job Stream Date

The date used to select the job streams to run.

Actual Date

The date **batchman** began running the `Symphony` file.

Start Time

The time **batchman** began running the `Symphony` file.

Log Date

The date the plan (`Symphony` file) was logged by the **stageman** command.

Run Num

The run number assigned to the plan (`Symphony` file). This is used internally for HCL Workload Automation network synchronization.

Size

The number of records contained in the `Symphony` file.

Log Num

The log number indicating the chronological order of log files. This number can be used in a **setsym** command to switch to a specific log file.

Filename

The name of the log file assigned by the **stageman** command.

Example

Examples

To list the production plan files, run the following command:

```
listsym
```

this is a sample output for the command:

```
Job Stream   Actual Start      Log      Run          Log
Date         Date   Time       Date     Num    Size   Num   Filename
07/18/24   07/18/24 10:58   07/18/24     36     41     1  M202407181101 (Exp)
07/18/24   07/18/24 10:58   07/18/24     35     43     2  M202407181058 (Exp)
07/18/24   07/18/24 10:56   07/18/24     34     53     3  M202407181057 (Exp)
07/18/24   07/18/24 10:55   07/18/24     33     43     4  M202407181056 (Exp)
07/18/24   07/18/24 10:54   07/18/24     32     52     5  M202407181055 (Exp)
07/18/24   07/18/24 10:48   07/18/24     31     61     6  M202407181054 (Exp)
07/18/24   07/18/24 10:47   07/18/24     30     40     7  M202407181048 (Exp)
07/18/24   07/18/24 10:46   07/18/24     29     45     8  M202407181047 (Exp)
07/18/24   07/18/24 10:45   07/18/24     28     30     9  M202407181045 (Exp)
07/18/24   07/18/24 10:24   07/18/24     27    281    10  M202407181044 (Exp)
07/18/24   07/18/24 10:21   07/18/24     26    157    11  M202407181024 (Exp)
07/18/24   07/18/24 10:17   07/18/24     25    113    12  M202407181021 (Exp)
07/18/24   07/18/24 10:16   07/18/24     24     38    13  M202407181017 (Exp)
```

To view the latest production plan file that was processed on a fault-tolerant agent, run the following command from the fault-tolerant agent **conman** command line program :

```
listsym
```

this is a sample output for the command:

```
Job Stream  Actual Start       Log      Run           Log
Date        Date    Time       Date     Num    Size    Num   Filename
07/18/24   07/18/24 10:17   07/18/24     25    113    12  MSymOldBackup (Exp)
```

To list files containing trial plans, run the following command:

```
listsym trial
```

this is a sample output for the command:

```
Job Stream   Actual Start      Log      Run          Log
Date         Date   Time       Date     Num    Size   Num   Filename
07/18/24   07/18/24 10:21   07/18/24     26    157    11  M202407181024 (Exp)
07/18/24   07/18/24 10:17   07/18/24     25    113    12  M202407181021 (Exp)
07/18/24   07/18/24 10:16   07/18/24     24     38    13  M202407181017 (Exp)
```

To list the files containing the forecast plans, run the following command:

```
listsym forecast
```

This is a sample output for the command:

```
Job Stream   Actual Start      Log      Run          Log
Date         Date   Time       Date     Num    Size   Num   Filename
07/18/24   07/18/24 09:40   07/18/24      7     56    30  M202407180941 (Exp)
```

See also

In the Dynamic Workload Console:

1. From the navigation bar, click **Planning > Workload Forecast > Manage Available plans**.
2. Select an engine.
3. Click a plan type or write a plan filename
4. Click **Display Plans List**.

LISTSUCC

Lists the successors of a job.

You must have ***rerun*** access to the job. You also need to enable plan data replication in the database. For more information, see the section about replicating plan data in the database in *User's Guide and Reference*.

Syntax

listsucc *jobselect*

Arguments

jobselect

See JOBSELECT for information about selecting jobs in commands.

Comments

If the user running the command is not authorized to see and rerun all the successors of the failed job, the list being displayed contains only the successors he is allowed to see. An error message is displayed, stating there are some additional successors he is not authorized to see or run.

The maximum number of successor jobs that can be returned is 1.000. To change this value, edit the **com.hcl.tws.conn.plan.rerun.successors.maxjobs** property in the `TWSConfig.properties` file, located in `*TWA_DATA_DIR*/usr/servers/engineServer/resources/properties/TWSConfig.properties`. To make this change effective, restart the master domain manager. When you run the command, the parent job is returned in the list of successors, but it does not count towards the total number of successor jobs listed. For example, if you set the **com.hcl.tws.conn.plan.rerun.successors.maxjobs** property to ten, and the total number of successors of your parent job is ten, a total of eleven jobs will be returned. This happens because the parent job is also listed, because it is scheduled to be rerun with its successors.

The action is always performed on the last rerun instance of the specified job. Also if you specify the job number of an intermediate job in the rerun sequence, the action is performed on the last job in the rerun sequence.

The Sched Time column in the command output is expressed in the timezone of the workstation where the job ran.

Example

Examples

When you launch the listsucc WXA_VMDM#FINAL.STARTAPPSERVER command, the following output is displayed:

```
Successors in the same job stream
Workstation  Job stream        Job                Status  Sched Time  Sched Id          Messages
------------ ---------------  ------------------ -----  ----------  ----------------  ----------------
WXA_VMDM     FINAL             STARTAPPSERVER     HOLD    2359 03/17  0AAAAAAAAAAAAADY  Invalid status
WXA_VMDM     FINAL             MAKEPLAN           HOLD    2359 03/17  0AAAAAAAAAAAAADY  Invalid status
WXA_VMDM     FINAL             SWITCHPLAN         HOLD    2359 03/17  0AAAAAAAAAAAAADY  Invalid status


Successors in other job streams
Workstation  Job stream        Job                Status  Sched Time  Sched Id          Messages
```

```
------------ -----------------  ------------------ ------  ----------  ----------------  ----------------
WXA_VMDM     FINALPOSTREPORTS  CREATEPOSTREPORTS  HOLD    2359 03/17  0AAAAAAAAAAAAAEH  Invalid status
WXA_VMDM     FINALPOSTREPORTS  CHECKSYNC          HOLD    2359 03/17  0AAAAAAAAAAAAAEH  Invalid status
WXA_VMDM     FINALPOSTREPORTS  UPDATESTATS        HOLD    2359 03/17  0AAAAAAAAAAAAAEH  Invalid status
```

The Messages column shows the value **Invalid status** when for a job it is not possible to rerun all the successor jobs.

See also

From the Dynamic Workload Console you can perform the same task as follows:

1. In the Welcome page, select **Monitor your workload**, or in the navigation bar at the top of the page, click **Monitoring & Reporting > Orchestration Monitoring > Monitor Workload**.
2. Select an engine.
3. In **Object Type**, select **Workstation/Job/Jobstream**.
4. From the **Query** drop-down list, select **All Jobs in plan** or another task to monitor jobs.
5. Click **Run** to run the monitoring task.
6. From the table containing the list of jobs, select a job and click **Rerun with successors...**.

RECALL

Displays prompts that are waiting for a response.

You must have ***display*** access to the prompts.

Syntax

{**recall** | **rc**} [*[folder/]workstation*]
[**;offline**]

Arguments

[folder/]workstation

Specifies the name of the workstation on which the prompt was issued. If you do not specify a workstation, only prompts for the login workstation and global prompts are displayed.

offline

Sends the output of the command to a file or command specified in **conman**. For more information, see the offline output topic in SETUPCONMAN.

Results

State

The state of the prompt. The state of pending prompts is always ASKED.

Message or Prompt

For named prompts, the message number, the name of the prompt, and the message text. For unnamed prompts, the message number, the name of the job or job stream, and the message text.

Example

Examples

To display pending prompts issued on the workstation on which you are running **conman**, run the following command:

```
recall
```

or:

```
rc
```

To display pending prompts on workstation `site3`, stored in folder `myfolder`, run the following command:

```
rc myfolder/site3
```

To display pending prompts on all workstations and have the output sent to a file or command, run the following command:

```
rc @;offline
```

See also

From the Dynamic Workload Console you can perform the same task as follows:

1. In the navigation bar at the top, click **Monitoring and Reporting > Orchestration Monitor**.
2. Select an engine.
3. In **Object Type**, select **Prompt**.
4. In the **Query** drop-down list, select **All Prompts in plan**, which will list all prompts regardless of their status, or create and select another task.
5. Click **Run** to run the monitoring task.

REDO

Edits and reruns the previous command.

Syntax

{**redo** | **red**}

Context

When you run the **redo** command, **conman** displays the previous command, so that it can be edited and rerun. Use the spacebar to move the cursor under the character to be modified, and enter the following directives.

Directives

d[*dir*]

Deletes the character above the **d**. This can be followed by other directives.

i*text*

Inserts text before the character above the **i**.

r*text*

Replaces one or more characters with text, beginning with the character above the **r**. Replace is implied if no other directive is entered.

>*text*

Appends text to the end of the line.

>d[*dir* | *text*]

Deletes characters at the end of the line. This can be followed by another directive or text.

>r*text*

Replaces characters at the end of the line with text.

Directive Examples

ddd

Deletes the three characters above the **d**s.

iabc

Inserts **abc** before the character above the **i**.

rabc

Replaces the three characters, starting with the one above the **r**, with **abc**.

abc

Replaces the three characters above **abc** with **abc**.

d diabc

Deletes the character above the first **d**, skips one character, deletes the character above the second **d**, and inserts **abc** in its place.

>abc

Appends **abc** to the end of the line.

>ddabc

Deletes the last two characters in the line, and inserts **abc** in their place.

>rabc

Replaces the last three characters in the line with **abc**.

Example

Examples

To insert a character, run the following command:

```
redo
setsm 4
```

```
    iy
setsym 4
```

To replace a character, run the following command:

```
redo
setsym 4
       5
setsym 5
```

RELEASE JOB

Releases jobs from normal and time dependencies. Conditional dependencies are not released. Conditional dependencies must be released explicitly or confirmed in **SUCC** status to ensure a correct release.

You must have ***release*** access to the job.

Syntax

{**release job** | **rj**} *jobselect*
[**;***dependency*[**;**...]]
[**;noask**]

Arguments

jobselect

Specifies the job or jobs to be released. See JOBSELECT for information about selecting jobs in commands.

dependency

The type of dependency. You can specify one of the following. You can use wildcard characters in *workstation, jstreamname, jobname, resource, filename,* and *promptname*.

at[**=***time* | *lowtime* | *hightime* | *lowtime*,*hightime*]

confirmed

deadline[**=***time*[**timezone** | **tz** *tzname*][**+***n* **days** | *mm/dd[/yy]*]]

every

follows=[*netagent*::][*workstation*#]{*jobstreamname*[*hhmm* [*mm*/*dd[/yy]*]][.*job* | **@**] | *jobstream_id*.*job***;schedid**}| *job*[**,**...] [**if** '*condition_name*[| *condition_name*][| ...]'][**from join** *join_name*]

Network agent workstations do not support folders, therefore neither the network agent nor the jobs or job streams running on them can be defined in folders. Folders are supported on all other workstation types, as follows:

```
 [follows {[folder/][workstation#]
[folder/]jobstreamname[.jobname]
```

The *condition_name* variable indicates the name of the condition defined in the job definition. Conditions must be separated by | and enclosed between single quotes. Conditions can be **status conditions**, based on job status, or other **output conditions**, based on a mapping expression such as a return code, output variables, or output found in a job log. In each follows statement, you can specify only one dependency type: either status or output conditions. At submission time, you can release status or output conditions. If the conditional dependency belongs to a join, if the number of conditions that must be met is different from ALL, the number is automatically reduced by one.

needs[**=**[*num*] [*[folder/]workstation***#**]*[folder/]resource*[**,**...]]

opens[**=**[*[folder/]workstation***#**]"*filename*"[**(***qualifier***)**][**,**...]]

priority

prompt[**="[:** | **!]***text*" | *[folder/]promptname*[**,**...]]

until[**=***time* [**timezone**|**tz** *tzname*][**+***n* **day**[**s**]] [**;onuntil** *action*]]

noask

Specifies not to prompt for confirmation before taking action on each qualifying job.

Comments

The command applies only to jobs that are in the HOLD state; that is, jobs that are waiting for the resolution of a dependency. Note also that the dependency is released only for the current run of the job and not for future reruns (the permanent release from a dependency can be obtained with the `deldep` command).

When you release an `opens` dependency, you can include only the base file name, and **conman** performs a case-insensitive search for matching files, ignoring the directory names. Dependencies on all matching files are released.

For `needs` dependencies, the released job is given the required number of units of the resource, even though they might not be available. This can cause the available units in a **showresources** to display a negative number.

When you release a job from a `priority` dependency, the job reverts to its original scheduled priority.

Released dependencies remain in effect when running the **rerun** command.

Example

Examples

To release job `job3` in job stream `ap(1000 03/05/18)`, stored in folder **myfolder**, from all of its dependencies, run the following command:

```
rj myfolder/ap(1000 03/05/18).job3
```

To release all jobs on workstation `site4` from their dependencies on a prompt named `glprmt`, run the following command:

```
rj=site4#@.@;prompt=glprmt
```

See also

From the Dynamic Workload Console you can perform the same task as follows:

1. In the Welcome page, select **Monitor your workload**, or in the navigation bar at the top of the page, click **Monitoring & Reporting > Orchestration Monitoring > Monitor Workload**.
2. Select an engine.
3. In **Object Type**, select **Workstation/Job/Jobstream**.
4. From the **Query** drop-down list, select **All Jobs in plan** or another task to monitor jobs.
5. Click **Run** to run the monitoring task.
6. From the table containing the list of jobs, select one or more jobs and click **Release Dependencies**.

RELEASE SCHED

Releases job streams from dependencies.

You must have ***release*** access to the job stream.

Syntax

{**release sched** | **rs**} *jstreamselect*
[**;***dependency*[**;**...]]
[**;noask**]

Arguments

jstreamselect

See JSSELECT for information about selecting job streams in commands.

dependency

The type of dependency. Specify one of the following. You can use wildcard characters in *workstation, jstream, job, resource, filename,* and *promptname*.

at[**=***time* | *lowtime* | *hightime* | *lowtime*,*hightime*]

carryforward

deadline[**=***time*[**timezone** | **tz** *tzname*][**+***n* **days** | *mm/dd[/yy]*]]

follows=[*netagent***::**][*workstation***#**]{*jobstreamname*[*hhmm* [*mm*/*dd[/yy]*]][*.job* | **@**] | *jobstream_id*.*job***;schedid**}| *job*[**,**...] [**if** '*condition_name*[| *condition_name*][| ...]'][**from join** *join_name*]

Network agent workstations do not support folders, therefore neither the network agent nor the jobs or job streams running on them can be defined in folders. Folders are supported on all other workstation types, as follows:

```
 [follows {[folder/][workstation#]
[folder/]jobstreamname[.jobname]
```

The *condition_name* variable indicates the name of the condition defined in the job definition. Conditions must be separated by | and enclosed between single quotes. Conditions can be **status conditions**, based on job status, or other **output conditions**, based on a mapping expression such as a return code, output variables, or output found in a job log. In each follows statement, you can specify only one dependency type: either status or output conditions. At submission time, you can release status or output conditions. If the conditional dependency belongs to a join, if the number of conditions that must be met is different from ALL, the number is automatically reduced by one.

limit

needs[**=**[*num*] [*[folder/]workstation***#**]*[folder/]resource*[**,**...]]

opens[**=**[*[folder/]workstation***#**]"*filename*"[**(***qualifier***)**][**,**...]]

priority

prompt[**="[:** | **!]***text*" | *[folder/]promptname*[**,**...]]

until[**=***time* [**timezone**|**tz** *tzname*][**+***n* **day**[**s**]] [**;onuntil** *action*]]

noask

Specifies not to prompt for confirmation before taking action on each qualifying job stream.

Comments

When deleting an `opens` dependency, you can include only the base file name, and **conman** performs a case-insensitive search for matching files, ignoring the directory names. Dependencies on all matching files are released.

For `needs` dependencies, the released job stream is given the required number of units of the resource, even though they might not be available. This can cause the available units in a **showresources** to display a negative number.

When you release a job stream from a `priority` dependency, the job stream reverts to its original priority.

In certain circumstances, when you have submitted a **deldep** command, the command might have succeeded even though it is again forwarded to **batchman**. For more information, see the "Conman commands processing" topic in SPECIALCHAR.

Example

Examples

To release job stream instance with *jobstream_id* `0AAAAAAAAAAABSE`, stored in folder **myfolder**, from all of its dependencies, run the following command:

```
rs myfolder/0AAAAAAAAAAABSE; schedid
```

To release job stream `sked5(1105 03/07/06)` from all of its `opens` dependencies, run the following command:

```
rs sked5(1105 03/07/06);opens
```

To release all job streams on workstation `site3` from their dependencies on job stream `sked23` contained in the `TEST` folder, on workstation `main`, run the following command:

```
rs=site3#@;follows=main#/TEST/sked23
```

where TEST is the folder where the job stream is located.

See also

From the Dynamic Workload Console you can perform the same task as follows:

1. In the navigation bar at the top, click **Monitoring and Reporting > Orchestration Monitor**.
2. Select an engine.
3. In **Object Type**, select **Workstation/Job Stream/Job**.
4. From the **Query** drop-down list, select **All Job Streams in plan** or another task to monitor job streams.
5. Click **Run** to run the monitoring task.
6. From the table containing the list of job streams, select one or more job streams and click **Release Dependencies**.

REPLY

Replies to a job or job stream prompt.

You must have ***reply*** access to the named or global prompt. To reply to an unnamed prompt, you must have ***reply*** access to prompts, and ***reply*** access to the associated job or job stream.

Syntax

{**reply** | **rep**}
{ *promptname* | [*[folder/]workstation***#**]*msgnum*}
;*reply*
[**;noask**]

Arguments

promptname

Specifies the name of a global prompt. Wildcard characters are permitted.

[folder/]workstation

Specifies the name of the workstation on which an unnamed prompt was issued.

msgnum

Specifies the message number of an unnamed prompt. You can display message numbers with the **recall** and **showprompts** commands.

reply

Specifies the reply, either **Y** for yes or **N** for no.

noask

Specifies not to prompt for confirmation before taking action on each qualifying prompt.

Comments

If the reply is **Y**, dependencies on the prompt are satisfied. If the reply is **N**, the dependencies are not satisfied and the prompt is not reissued.

Prompts can be replied to before they are issued. You can use the **showprompts** command to display all prompts, whether or not they have been issued.

Example

Examples

To reply **Y** to the global prompt `arpmt`, run the following command:

```
reply arprmt;y
```

To reply **N** to message number **24** on workstation `site4`, run the following command:

```
rep site4#24;n
```

RERUN

Reruns a job.

You must have ***rerun*** access to the job.

To use **streamlogon**|**logon**, **docommand**, or **script** arguments, you must have ***submit*** access to the job.

To use the **from** argument, you must have ***submitdb*** access to the job.

Syntax

{**rerun** | **rr**} *jobselect*
[[**;from=**[*[folder/]wkstat***#**]*job*
[**;at=***time*]
[**;pri=***pri*]] |
[[**;streamlogon**|**logon=***new_logon*]
[**;docommand=***"new_command"*|**;script=***"new_script"*]] |
[**;step=***step*]]
[**;sameworkstation=**]
[**;noask**]

Arguments

jobselect

See JOBSELECT for information about selecting jobs in commands. Wildcards are supported.

from=[*[folder/]wkstat#*]*job*

Specifies the name of a job defined in the database whose job file or command will be run in place of the job specified by *jobselect*. You can rerun jobs also in the SUPPR state, as long as they do not belong to job streams that are in the cancelled or suppressed state.

wkstat#

Specifies the name of the workstation on which the **from** job runs. The default is the workstation on which **conman** is running.

job

Specifies the name of the **from** job definition The following types of job names are not permitted:

- The names of jobs submitted using the **submit file** and **submit docommand** commands.
- The alias names of jobs submitted using the **submit job** command.

This argument is mutually exclusive with streamlogon|logon and docommand|script arguments.

The recovery options, if any, are partly inherited from the original job definition and partly retrieved from the **from** job. Table 78: Recovery options retrieval criteria on page 636 describes the criteria by which recovery options in the original and **from** job definition are retrieved.

Table 78. Recovery options retrieval criteria

recovery option	Inherited from original job	Retrieved from "from" job
stop	No	Yes
continue	No	Yes
rerun	Yes	No
repeatevery	Yes	No
for	Yes	No
after	Yes	No
abendprompt	Yes	No

To use the **from** argument, you must have access to the database from the computer on which you are running **conman**

at=*time*

Specifies the rerun job's start time, expressed as follows:

*hhmm [**timezone**|**tz** tzname]* [**+***n* **days** | *date*]

where:

hhmm

The hour and minute.

+*n* days

The next occurrence of *hhmm* in *n* number of days.

date

The next occurrence of *hhmm* on *date*, expressed as *mm/dd[/yy]*.

timezone|tz *tzname*

The name of the time zone of the job. See the chapter called "Managing time zones" in *User's Guide and Reference* for valid names.

pri=*pri*

Specifies the priority to be assigned to the rerun job. If you do not specify a priority, the job is given the same priority as the original job.

sameworkstation

If the parent job ran on a workstation that is part of a pool or a dynamic pool, you can decide whether it must rerun on the same workstation or on a different one. This is because the workload on pools and dynamic pools is assigned dynamically based on a number of criteria and the job might be rerun on a different workstation.

When you rerun the job manually, the **sameworkstation** setting you define with the **rerun** command is applied only to the instance you rerun, and is ignored in any subsequent reruns of that instance. For example, also if the job you rerun already contains rerun information (defined with the **rerun**, **repeatevery**, and **for** arguments in the job definition) the **sameworkstation** setting you define with the **rerun** command applies only to the specific instance you rerun. For the subsequent reruns, the setting defined in the job definition is used. .

step=*step*

Specifies that the job is rerun using this name in place of the original job name. See Comments on page 638 for more information.

streamlogon|logon=*new_logon*

Specifies that the job is rerun under a new user name in place of the original user name. This argument applies only to completed jobs. This argument is mutually exclusive with the **from** argument.

docommand=*"new_command"*

Specifies the new command that the rerun job runs in place of the original command. This argument is mutually exclusive with the **script** and **from** arguments. This argument applies only to completed jobs.

script=*"new_script"*

Specifies the new script that the rerun job runs in place of the original script. This argument is mutually exclusive with the **docommand** and **from** arguments. This argument applies only to completed jobs.

noask

Specifies not to prompt for confirmation before taking action on each qualifying job.

Comments

You can rerun jobs that are in the SUCC, FAIL, or ABEND state. A rerun job is placed in the same job stream as the original job, and inherits the original job's dependencies. If you rerun a repetitive (**every**) job, the rerun job is scheduled to run at the same rate as the original job.

Note: You can issue **rerun** for jobs in the `EXTERNAL` job stream that are in the ERROR state. Jobs in the `EXTERNAL` job stream represent jobs and job streams that have been specified as internetwork dependencies. The job state is initially set to **extrn** immediately after a **rerun** is run, and **conman** begins checking the state.

Note: When you rerun a job using **docommand**, or **script** arguments, if the job contains variables, the job reruns and completes, but is unable to resolve the variables with their value.

When **;from** is used, the name of the rerun job depends on the value of the Global Option **enRetainNameOnRerunFrom**. If the option is set to **Y**, rerun jobs retain the original job names. If the option is set to **N**, rerun jobs are given the **from** job names. For more information, see *Administration Guide*.

In **conman** displays, rerun jobs are displayed with the notation **>>rerun as**. To refer to a rerun job in another command, such as **altpri**, you must use the original job name.

When a job is rerun with the **;step** option, the job runs with *step* in place of its original name. Within a job script, you can use the **jobinfo** command to return the job name and to run the script differently for each iteration. For example, in the following UNIX® script, the **jobinfo** command is used to set a variable named **STEP** to the name that was used to run the job. The **STEP** variable is then used to determine how the script is run.

```
...
MPATH=`maestro`
STEP=`$MPATH/bin/jobinfo job_name`
if [$STEP = JOB3]
  then
  ...
  STEP=JSTEP1
  fi
if [$STEP = JSTEP1]
  then
  ...
  STEP=JSTEP2
  fi
if [$STEP = JSTEP2]
  then
  ...
  fi
...
```

In **conman** displays, jobs rerun with the **;step** option are displayed with the notation **>>rerun step**.

For information about **jobinfo**, see the jobinfo topic in the chapter about the Utility commands in *User's Guide and Reference*.

Example

Examples

To rerun job `job4` in job stream `sked1` on workstation `main`, run the following command:

```
rr main#sked1.job4
```

To rerun job `job5` in job stream `sked2` using the job definition for job `jobx` where the job's **at** time is set to 6:30 p.m. and its priority is set to **25**, run the following command:

```
rr sked2.job5;from=jobx;at=1830;pri=25
```

To rerun job `job3` in job stream `sked4` using the job name `jstep2`, run the following command:

```
rr sked4.job3;step=jstep2
```

See also

From the Dynamic Workload Console you can perform the same task as follows:

1. In the Welcome page, select **Monitor your workload**, or in the navigation bar at the top of the page, click **Monitoring & Reporting > Orchestration Monitoring > Monitor Workload**.
2. Select an engine.
3. In **Object Type**, select **Workstation/Job/Jobstream**.
4. From the **Query** drop-down list, select **All Jobs in plan** or another task to monitor jobs.
5. Click **Run** to run the monitoring task.
6. From the table containing the list of jobs, select a job and click **Rerun...**.

RERUNSUCC

Reruns a job and its successors. You can choose whether you want to rerun all successors in the same job stream of the failed job (internal successors), or all successors, both in the same job stream and in external job streams (internal and external successors).

You must have ***rerun*** access to the job. You also need to enable plan data replication in the database. For more information, see the section about replicating plan data in the database in *User's Guide and Reference*.

Syntax

rerunsucc *jobselect* **[;internal][;all]**

Arguments

jobselect

See JOBSELECT for information about selecting jobs in commands.

;internal

Specifies that all successors of the failed job (parent job) in the same job stream must be rerun. Any successors in external job streams are not rerun.

;all

Specifies that all successors of the failed job (parent job) both in the same job stream and in external job streams must be rerun.

Comments

If you enter the command without any options, a list of the internal and external successors with the related status is returned. A message is displayed asking to confirm whether you want to rerun only the internal successors or both the internal and external successors.

If the user running the command is not authorized to see and rerun all the successors of the failed job, the list being displayed contains only the successors he is allowed to see. An error message is displayed, stating there are some additional successors he is not authorized to see or run.

The maximum number of successor jobs that can be returned is 1,000. To change this value, edit the **com.hcl.tws.conn.plan.rerun.successors.maxjobs** property in the `TWSConfig.properties` file, located in `*TWA_DATA_DIR*/usr/servers/engineServer/resources/properties/TWSConfig.properties`. To make this change effective, restart the master domain manager. When you run the command, the parent job is returned in the list of successors, but it does not count towards the total number of successor jobs listed. For example, if you set the **com.hcl.tws.conn.plan.rerun.successors.maxjobs** property to ten, and the total number of successors of your parent job is ten, a total of eleven jobs will be returned. This happens because the parent job is also listed, because it is scheduled to be rerun with its successors.

The rerun action is always performed on the last rerun instance of the specified job. Also if you specify the job number of an intermediate job in the rerun sequence, the action is performed on the last job in the rerun sequence.

You can rerun job successors only if they are in specific states. For example, successors in intermediate states, such as EXEC, WAIT, INTRO, cannot be rerun. See Table 79: Successors status on page 640 for a complete list.

Table 79. Successors status

Status	Expected behavior
WAIT	An error is returned and the rerun operation is not performed
INTRO	An error is returned and the rerun operation is not performed
EXEC	An error is returned and the rerun operation is not performed
EXTERNAL	An error is returned and the rerun operation is not performed
ABENDP/SUCCP	An error is returned and the rerun operation is not performed
READY	An error is returned and the rerun operation is not performed

Table 79. Successors status (continued)

Status	Expected behavior
PEND	An error is returned and the rerun operation is not performed
SUPPR (job stream)	An error is returned and the rerun operation is not performed
HOLD	The predecessor of the job in HOLD status is rerun, but the rerun sequence stops at the job in HOLD status
BOUND	The predecessor of the job in BOUND status is rerun, but the rerun sequence stops at the job in BOUND status
FENCE	The predecessor of the job in FENCE status is rerun, but the rerun sequence stops at the job in FENCE status
SUPPR (job)	The rerun operation is performed
SUCC	The rerun operation is performed
CANCEL	The rerun operation is performed

Example

Examples

To return a list of all successors of the failed job (parent job) with the related status, type the following command:

```
Rerunsucc MDM94FP1#RequestInfo.UpdateData
```

An output similar to the following is returned:

```
        Successors in the same job stream:
        MDM94FP1#RequestInfo.UpdateFunction                        SUCC
        SUCCMDM04FP1#RequestInfo.NotifyOfTheRequestReived          SUCC

        Successors to be rerun in another job stream:
        MDM94FP1#BatchProcessing.UpdateFunction                    SUCC
        MDM94FP1#BatchProcessing.EvaluateRisk                      SUCC
        ….
        MDM94FP1#ReportProcessing.RunReport           SUCC
        Do you want to run all successors, both internal  and external? Y
```

To run the command in batch mode and rerun all internal successors without confirmation by the user, type the following command:

```
Rerunsucc MDM94FP1#RequestInfo.UpdateData;internal
```

An output similar to the following is displayed:

```
Successors in the same job stream:

        MDM94FP1#RequestInfo.UpdateFunction                         SUCC
```

```
        MDM04FP1#RequestInfo.NotifyOfTheRequestReceived      SUCC
        ……
```

To run the command in batch mode and rerun all successors, both internal and external, without confirmation by the user, type the following command:

```
Rerunsucc MDM94FP1#RequestInfo.UpdateData;all
```

An output similar to the following is displayed:

```
MDM94FP1#RequestInfo.UpdateFunction                        SUCC
        MDM04FP1#RequestInfo.NotifyOfTheRequestReceived             SUCC
        MDM94FP1#BatchProcessing.UpdateFunction                     SUCC
        MDM94FP1#BatchProcessing.EvaluateRisk                       SUCC
        MDM94FP1#ReportProcessing.RunReport                         SUCC
        …..
        ……
```

See also

From the Dynamic Workload Console you can perform the same task as follows:

1. In the Welcome page, select **Monitor your workload**, or in the navigation bar at the top of the page, click **Monitoring & Reporting > Orchestration Monitoring > Monitor Workload**.
2. Select an engine.
3. In **Object Type**, select **Workstation/Job/Jobstream**.
4. From the **Query** drop-down list, select **All Jobs in plan** or another task to monitor jobs.
5. Click **Run** to run the monitoring task.
6. From the table containing the list of jobs, select a job and click **More Actions > Rerun with successors...**. A dialog is displayed listing all successors, both internal and external in two separate tables. In the dialog you can choose whether you want to rerun all successors in the same job stream or all successors both in the same job stream and in any other job streams.

RESETFTA

Generates an updated `Sinfonia` file and sends it to a fault-tolerant agent on which the `Symphony` file has corrupted.

Note: Complete removal and replacement of the `Symphony` file causes some loss of data, for example events on job status, or the contents of the Mailbox.msg message and the tomaster.msg message queues. If state information about a job was contained in those queues, that job is rerun. It is recommended that you apply this command with caution.

In the process, the following files are moved to the *TWA_home*/TWS/tmp directory:

- Appserverbox.msg
- clbox.msg
- Courier.msg

- Intercom.msg
- Mailbox.msg
- Monbox.msg
- Moncmd.msg
- Symphony
- Sinfonia

Before the command is performed, an information message is displayed to request confirmation and ensure the command is not issued by mistake. If one of the target files cannot be moved because it is being used by another process (for example, the mailman process is still running) the operation is not performed and an error message is displayed.

Authorization

You must have **RESETFTA** access to the fault-tolerant agent you want to reset.

Syntax

resetFTA cpu

Arguments

cpu

Is the fault-tolerant agent to be reset.

This command is not available in the Dynamic Workload Console.

Example

Examples

To reset the fault-tolerant agent with name omaha, run the following command:

```
resetFTA omaha
```

RESOURCE

Changes the number of total units of a resource.

You must have ***resource*** access to the resource.

Syntax

{**resource** | **reso**} [*[folder/]workstation***#**]
*[folder/]resource***;***num*
[**;noask**]

Arguments

[folder/]workstation

Specifies the name of the workstation on which the resource is defined. The default is the workstation on which **conman** is running.

[folder/]resource

Specifies the name of the resource.

num

Specifies the total number of resource units. Valid values are **0** through **1024**.

noask

Specifies not to prompt for confirmation before taking action on each qualifying resource.

Example

Examples

To change the number of units of resource `tapes` to **5**, run the following command:

```
resource tapes;5
```

To change the number of units of resource `jobslots` on workstation `site2`, stored in folder `myfolder`, to **23**, run the following command:

```
reso myfolder/site2#jobslots;23
```

See also

From the Dynamic Workload Console you can perform the same task as follows:

1. In the Welcome page, select **Monitor your workload**, or in the navigation bar at the top of the page, click **Monitoring & Reporting > Orchestration Monitoring > Monitor Workload**.
2. Select an engine.
3. In **Object Type**, select **Resource**.
4. From the **Query** drop-down list, select **All Resources in plan** or another task to monitor resources.
5. Click **Run** to run the monitoring task.
6. From the table of results, select a resource and click **Change Units...**.

SETSYM

Selects a production plan archive file. Subsequent display commands show the contents of the archived production plan. You cannot modify the information in a production plan archive file.

Syntax

{**setsym** | **set**} [**trial** | **forecast**] [*filenum*]

Arguments

trial

Lists trial plans.

forecast

Lists forecast plans.

filenum

Specifies the number of the production plan archive file. If you do not specify a log file number, the pointer returns to zero, the current production plan (`Symphony`). Use the **listsym** command to list archive file numbers.

Example

Examples

To select production plan archive file **5**, run the following command:

```
setsym 5
```

To select the current production plan (`Symphony` file), run the following command:

```
set
```

See also

In the Dynamic Workload Console:

1. From the navigation bar, click **Planning > Workload Forecast > Manage Available Plans**.
2. Select an engine.
3. Click **Archived plans** or provide a plan filename.
4. Click **Display Plans List**.

SHOWCPUS

Displays information about workstations and links.

The displayed information is updated only while HCL Workload Automation (batchman) is running on the workstations. If **batchman** is up or down is confirmed on screen by the `Batchman LIVES` or `Batchman down` message when you issue the `conman start` command.

You must have ***list*** access to the object being shown if the ***enListSecChk*** option was set to **yes** on the master domain manager when the production plan was created or extended.

Syntax

{**showcpus** | **sc**} [[*domain*!][*folder/*]*workstation*]
[;info|;link]

[**;offline**]
[**;showid**]

{**showcpus** | **sc**} [[*domain***!**][*folder/*]*workstation*] [**;getmon**]

Arguments

domain

Specifies the name of a domain. The default is the domain in which the command is run.

[folder/]workstation

Specifies the name of a workstation and optionally, the folder in which is it defined. The default is the workstation where the command is run. When no domain and no workstation are specified, the output can be the following:

- The following command displays all the workstations that are in the domain of the workstation where the command was run, plus all the connected domain managers if the workstation is a domain manager.

  ```
  conman "sc"
  ```

- The following command displays all the workstations that are in the domain of the workstation where the command was run, without the connected domain managers.

  ```
  conman "sc @"
  ```

info

Displays information in the **info** format.

link

Displays information in the **link** format.

offline

Sends the output of the command to the **conman** output device. For information about this device, see the offline output topic in SETUPCONMAN.

showid

Displays a unique identifier that identifies a workstation, resource or prompt. These objects are no longer identified in the plan solely by their names, but also by the folder in which they are defined. The name and folder association is mapped to a unique identifier. For example, for workstations, the **this_cpu** option is the unique identifier of the workstation in the `localopts` file. You can verify the unique identifier for workstations, resources, and prompts by submitting the composer list command, together with the `;showid` filter, or by submitting the conman command, `showcpus`, `showresources`, or `showprompts`, in combination with the `;showid` filter. See the related example in the **Examples** section.

Identifying workstations by their unique identifier avoids the problem that occurred in previous versions when objects were renamed in the plan. For example, if an object was renamed and then carried forward to a new production plan, references to the old object name were lost. With the implementation of the unique identifier, this will no longer occur and dependencies will be correctly resolved.

When deleting a workstation, if the workstation is still in the plan, then another workstation cannot be renamed with the name of the deleted workstation for the number of days specified by the global option folderDays. However, a brand new workstation can be created with the name of the deleted workstation. This behavior applies only to dynamic agents, pools, and dynamic pools. The default value is 10 days.

getmon

Returns the list of event rules defined for the monitor running on the specified workstation in the following format:

```
<rule_name>::<eventProvider>#<eventType>:<scope>
```

The rule scope is automatically generated information about rule attributes, such as the workstations where it is used, a job or file name, and so on.

The header of the output contains also the time stamp of when the rule configuration package was last generated.

Note: This option is not valid on dynamic workload broker workstations (or dynamic domain managers). In this case, you can retrieve the information about the active rules defined in these workstations in the *TWA_home*\TWS\monconf\TWSObjectsMonitor.cfg file on the master domain manager.

Results

When the `getmon` parameter is not used, the output of the command is produced in three formats, **standard**, **info**, and **link**. The default value is **standard**. The meaning of the characters displayed depends on the type of format you select.

When the `getmon` parameter is used, the list of rules is provided as separate output.

Example

Examples

1. To display all workstations in all folders, run the following command:

   ```
   sc /@/@
   ```

2. To display all workstations, in all folders, including the unique identifier for the workstation in the plan, run the following command:

   ```
   sc @;showid
   ```

   The following is a sample of the output for this command. The unique identifier for workstations defined in the root folder is identical to the workstation name (CPUID). For workstations defined in a folder different from the root, the

unique identifier is different from the workstation name (CPUID). In this output sample, "`>>/MERGERS/AP/WINFTA`", refers to the folder path where the `WINFTA` workstation is defined:

```
CPUID              RUN NODE        LIMIT FENCE DATE     TIME  STATE       METHOD  DOMAIN
AP-MERGERS-WIN      35 *UNIX MASTER   10     0 10/01/23 23:59    I J  M EA        MASTERDM
 {AP-MERGERS-WIN    }
AP-MERGERS-LNX86    35  UNIX AGENT    10     0 10/01/23 23:59  LBI J  M           MASTERDM
 {AP-MERGERS-LNX86}
AP-MERGERS-LNX36    35  OTHR BROKER   10     0 10/01/23 23:59  LTI JW             MASTERDM
 {AP-MERGERS-LNX36}
MASTERAGENTS        35  OTHR POOL     10     0 10/01/23 23:59  LBI J              MASTERDM
 {MASTERAGENTS    }
>>/MERGERS/AP/
WINFTA              35  UNIX FTA      10     0 10/01/23 23:59  LTI JW M           MASTERDM
 {0AAA5D7ZC7BY24A6}
```

3. To display information about the workstation on which you are running **conman** in the **info** format, run the following command:

   ```
   showcpus ;info
   ```

   A sample output for this command is:

   ```
   CPUID         VERSION   TIME ZONE              INFO
   MASTER         10.2.8  US/Pacific             Linux 2.6.5-7.191-s390 #1 SM
   FTA1           10.2.8                         Linux 2.4.9-e.24 #1 Tue May
   ```

4. To display link information for all workstations, run the following command:

   ```
   sc @!@;link
   ```

   A sample output is the following:

   ```
   CPUID       HOST      FLAGS  ADDR  NODE
   MASTER     MASTER     AF T  51099  9.132.239.65
   FTA1       FTA1       AF T  51000  CPU235019
   FTA2       FTA2       AF T  51000  9.132.235.42
   BROKER1    MASTER     A  T  51111  9.132.237.17
   ```

5. To display information about the workstation, run the following command:

   ```
   showcpus
   ```

   If you run this command in an environment when the primary connection of the workstation with its domain or higher manager is not active, you receive the following output:

   ```
   CPUID   RUN    NODE     LIMIT FENCE  DATE     TIME    STATE    METHOD DOMAIN
   MASTER  360 *WNT  MASTER   10    0 03/05/2023 1348   I J   E          MASTERDM
   FTA1    360  WNT  FTA      10    0 03/05/2023 1348 FTI JW M           MASTERDM
   FTA2    360  WNT  FTA      10    0 03/05/2023 1348 FTI JW M           MASTERDM
   FTA3    360  WNT  MANAGER  10    0 03/05/2023 1348 LTI JW M           DOMAIN1
   FTA4    360  WNT  FTA      10    0 03/05/2023 1348 F I J  M           DOMAIN1
   FTA5    360  WNT  FTA      10    0 03/05/2023 1348   I J  M           DOMAIN1
   SA1     360  WNT  S-AGENT  10    0 03/05/2023 1348 F I J  M           DOMAIN1
   XA_FTA4 360  OTHR X-AGENT  10    0 03/05/2023 1348 L I J  M           DOMAIN1
   FTA6    360  WNT  MANAGER  10    0 03/05/2023 1348 F I J  M           DOMAIN2
   FTA7    360  WNT  FTA      10    0 03/05/2023 1349 F I J  M           DOMAIN2
   FTA7    360  WNT  FTA      10    0 03/05/2023 1349 F I J  M           DOMAIN2
   BROKER  360  OTHR BROKER   10    0 03/05/2023 1349 LTI JW             MASTERDM
   ```

If you run this command in an environment when the primary connection of the workstation with its domain or higher manager is active and at least one secondary connection is not active, you receive the following output:

```
CPUID   RUN     NODE     LIMIT FENCE   DATE    TIME    STATE     METHOD DOMAIN
MASTER  360 *WNT  MASTER   10    0  03/05/2023 1348    I J   E          MASTERDM
FTA1    360  WNT  FTA      10    0  03/05/2023 1348  FTI JW M           MASTERDM
FTA2    360  WNT  FTA      10    0  03/05/2023 1348  FTI JW M           MASTERDM
FTA3    360  WNT  MANAGER  10    0  03/05/2023 1348  FTI JW M           DOMAIN1
FTA4    360  WNT  FTA      10    0  03/05/2023 1348  F I J  M           DOMAIN1
FTA5    360  WNT  FTA      10    0  03/05/2023 1348  L I    M           DOMAIN1
SA1     360  WNT  S-AGENT  10    0  03/05/2023 1348  F I J  M           DOMAIN1
XA_FTA4 360  OTHR X-AGENT  10    0  03/05/2023 1348  L I J  M           DOMAIN1
FTA6    360  WNT  MANAGER  10    0  03/05/2023 1348  F I J  M           DOMAIN2
FTA7    360  WNT  FTA      10    0  03/05/2023 1349  F I J  M           DOMAIN2
```

If you run this command in an environment when the primary connection of the workstation with its domain or higher manager and all secondary connections are active, you receive the following output:

```
CPUID   RUN     NODE     LIMIT FENCE   DATE    TIME     STATE    METHOD DOMAIN
MASTER  360 *WNT  MASTER   10    0  03/05/2023 1348    I J    E          MASTERDM
FTA1    360  WNT  FTA      10    0  03/05/2023 1348  FTI JW M           MASTERDM
FTA2    360  WNT  FTA      10    0  03/05/2023 1348  FTI JW M           MASTERDM
FTA3    360  WNT  MANAGER  10    0  03/05/2023 1348  FTI JW M           DOMAIN1
FTA4    360  WNT  FTA      10    0  03/05/2023 1348  F I J  M           DOMAIN1
FTA5    360  WNT  FTA      10    0  03/05/2023 1348  F I    M           DOMAIN1
SA1     360  WNT  S-AGENT  10    0  03/05/2023 1348  F I J  M           DOMAIN1
XA_FTA4 360  OTHR X-AGENT  10    0  03/05/2023 1348  L I J  M           DOMAIN1
FTA6    360  WNT  MANAGER  10    0  03/05/2023 1348  F I J  M           DOMAIN2
FTA7    360  WNT  FTA      10    0  03/05/2023 1349  F I J  M           DOMAIN2
```

6. To get a list of active rule monitors on the workstation named `CPU1`, stored in folder `myfolder`, run this command:

   ```
   sc CPU1 getmon
   ```

   You get the following output:

   ```
   Monitoring configuration for CPU1:
   ******************************************
   *** Package Date : 04/22/2023 12:00 GMT ***
   ******************************************
   Rule1::FileMonitor#FileCreated:Workstation=CPU1,CPU2;File=”\tmp\filename”
   Rule2::FileMonitor#ModificationCompleted:Workstation=CPU1,CPU3;File=”\staging\orders”
   Rule3::TWSObjectsMonitor#JobSubmit:JobKey=CPU1#JS1.Job1
   Rule5::TWSObjectsMonitor#JobLate:JobKey=CPU1#JS1.Job1
   ```

See also

From the Dynamic Workload Console you can perform the same task as follows:

1. In the navigation bar at the top, click **Monitoring and Reporting > Orchestration Monitor**.
2. Select an engine.
3. From the drop-down menu, select **Workstation**.
4. From the **Query** drop-down list, select a query to monitor workstations.
5. Click **Run** to run the monitoring task.

Standard format

CPUID

The name of the workstation to which this information applies.

RUN

The run number of the `Symphony` file .

NODE

The node type and workstation type. Node types are as follows:

- UNIX®
- WNT
- OTHER
- ZOS
- IBM i

Workstation types are as follows:

- MASTER
- MANAGER
- FTA
- S-AGENT
- X-AGENT
- AGENT
- POOL
- D-POOL
- REM-ENG

LIMIT

The HCL Workload Automation job limit.

FENCE

The HCL Workload Automation job fence.

DATE TIME

The date and time HCL Workload Automation started running the current production plan (`Symphony` file).

STATE

Displays the following information:

- The state of the workstation's links and processes. Up to five characters are displayed as follows. The explanation of the characters is divided based on the character scope:

```
[L|F] [T|H|X|B] [I] [J] [W|H|X] [M] [E|e] [D] [A|R]
```

where:

L

The primary link is open (linked) to its domain or upper manager.

If the workstation is of type agent or remote engine, this flag indicates that the workstation is connected to the workload broker server.

If the workstation is of type pool or dynamic pool, this flag indicates that the workload broker workstation the pool or dynamic pool is registered to is linked to its domain or upper manager.

F

The workstation is fully linked through primary and all secondary connections. This flag appears only if the ***enSwfaultTol*** global option is set to *YES* using the **optman** command line on the master domain manager and it indicates that the workstation is directly linked to its domain manager and to all its full backup domain managers. For information on how to use the **optman** command line, refer to *Administration Guide*.

T

This flag is displayed if the fault-tolerant agent is directly linked to the domain manager from where you run the command.

H

The workstation is linked through its host.

X

The workstation is linked as an extended agent (x-agent).

B

The workstation communicates through the workload broker server.

I

If the workstation is of type agent, MASTER, MANAGER, FTA, S-AGENT, X-AGENT, this flag indicates that **jobman** program has completed startup initialization.

If the workstation is of type agent, pool or dynamic pool, this flag indicates that the agent is correctly initialized.

If the workstation is of type remote engine, this flag indicates that the communication between the remote engine workstation and the remote engine is correctly initialized.

J

If the workstation is of type agent MASTER, MANAGER, FTA, S-AGENT, X-AGENT, this flag indicates that **jobman** program is running.

If the workstation is of type agent, this flag indicates that JobManager is running. Because no monitoring is performed on dynamic pool workstations, for this workstation type the J character is always shown.

If the workstation is of type pool, this flag indicates that the JobManager process is running on at least one agent registered to the pool.

If the workstation is of type remote engine, this flag indicates that the ping command to the remote engine is successful.

W

The workstation is linked via TCP/IP using the **writer** process.

If the workstation running **conman** is directly linked to the remote workstation, you see the flag W because the local mailman is linked to the remote writer process.

```
LTI JW
```

If the workstation running **conman** is not directly linked to the remote workstation, you do not see the flag W because the local mailman is not directly linked to the remote writer process.

```
L I J
```

For more details about the **writer** process, the topic about network processes in the *Administration Guide*.

Note: If the workstation running **conman** is the extended agent's host, the state of the extended agent is

```
LXI JX
```

If the workstation running **conman** is not the extended agent's host, the state of the extended agent is

```
LHI JH
```

- The state of the monitoring agent. Up to three characters are displayed as follows:

```
[M] [E|e] [D]
```

where:

M

The `monman` process is running. This flag is displayed for all the workstations in the network when the event-driven workload automation feature is enabled (global option `enEventDrivenWorkloadAutomation` is set to `yes`), with the exception of those workstations where `monman` was manually stopped (using either **conman** or the Dynamic Workload Console).

E

The event processing server is installed and running on the workstation.

e

The event processing server is installed on the workstation but is not running.

D

The workstation is using an up-to-date package monitoring configuration. This flag is displayed for the workstations on which the latest package of event rules was deployed (either manually with the `planman deploy` command or automatically with the frequency specified by the `deploymentFrequency` global option).

- The state of the WebSphere Application Server Liberty. A one-character flag is displayed, if the application server is installed:

  ```
  [A|R]
  ```

  where:

  A

  WebSphere Application Server Liberty was started.

  R

  WebSphere Application Server Liberty is restarting.

  The flag is blank if the application server is down or if it was not installed.

METHOD

The name of the access method specified in the workstation definition. For extended agents only.

DOMAIN

The name of the domain in which the workstation is a member.

Info format

CPUID

The name of the workstation to which this information applies.

VERSION

The version of the HCL Workload Automation agent installed on the workstation.

TIMEZONE

The time zone of the workstation. It is the same as the value of the TZ environment variable. For an extended agent, this is the time zone of its host. For a remote engine workstation, this is the time zone of the remote engine.

INFO

An informational field. For all the workstation types except the extended agent and the broker workstations it contains the operating system version and the hardware model. For extended agents and remote engine workstations, no information is listed. For remote engine workstation it shows Remote Engine.

Link format

CPUID

The name of the workstation to which this information applies.

HOST

The name of the workstation acting as the host to a standard agent or extended agent. For domain managers and fault-tolerant agents, this is the same as CPUID. For standard agent and broker workstations, this is the name of the domain manager. For extended agents, this is the name of the host workstation.

FLAGS

The state of the workstation properties. Up to five characters are displayed as follows:

```
[A] [B] [F] [s] [T]
```

A

Autolink is turned on in the workstation definition.

B

This flag is used only in end-to-end environment and it indicates if the **deactivate job launching** flag is disabled.

F

Full Status mode is turned on in the workstation definition.

s

The ID of **mailman** server for the workstation.

T

The link is defined as TCP/IP.

ADDR

The TCP/IP port number for the workstation.

NODE

The node name of the workstation.

SHOWDOMAIN

Displays domain information.

The displayed information is updated only as long as HCL Workload Automation (batchman) is running. Whether batchman is up or down is confirmed on screen by the `Batchman LIVES` or `Batchman down` message when you issue the `conman start` command.

You must have ***list*** access to the object being shown if the ***enListSecChk*** option was set to **yes** on the master domain manager when the production plan was created or extended.

Syntax

{**showdomain** | **showdom** | **sd**} [*domain*]
[**;info**]
[**;offline**]

Arguments

domain

Specifies the name of the domain. The default is the domain in which **conman** is running. Wildcard characters are permitted.

info

Displays information in the **info** format.

offline

Sends the output of the command to the **conman** output device. For information about this device, see the offline output topic in SETUPCONMAN.

Results

The output of the command is produced in two formats, **standard**, and **info**.

Example

Examples

To display information about the domain `masterdm`, run the following command:

```
showdomain masterdm
```

A sample output is the following:

```
DOMAIN            MANAGER          PARENT
*MASTERDM        *MASTER
```

To display the member workstations in all domains in the **info** format, run the following command:

```
showdomain @;info
```

a sample output is the following:

```
DOMAIN           MEMBER-CPUs      CPU-Type
 MASTERDM        *MASTER           MASTER
```

```
DOM1            FTA1            MANAGER
DOM2            FTA2            MANAGER
```

See also

From the Dynamic Workload Console you can perform the same task as follows:

1. In the navigation bar at the top, click **Monitoring and Reporting > Orchestration Monitor**.
2. Select an engine.
3. In **Object Type**, select **Domain**.
4. From the **Query** drop-down list, select a task to monitor domains.
5. Click **Run** to run the monitoring task.

Standard format

DOMAIN

The name of the domain to which this information applies.

MANAGER

The name of the domain manager.

PARENT

The name of the parent domain.

Info format

DOMAIN

The name of the domain to which this information applies.

MEMBER-CPUS

The names of the workstations in the domain.

CPU-TYPE

The type of each workstation: MASTER, MANAGER, FTA, S-AGENT, X-AGENT, or BROKER.

SHOWFILES

Displays information about file dependencies. A file dependency occurs when a job or job stream is dependent on the existence of one or more files before it can begin running.

The displayed information is updated only as long as HCL Workload Automation (batchman) is running. Whether batchman is up or down is confirmed on screen by the `Batchman LIVES` or `Batchman down` message when you issue the `conman start` command.

Syntax

{**showfiles** | **sf**} [[*[folder/]workstation#*]*file*]
[**;***state*[**;**...]]
[**;keys**]
[**;offline**]

{**showfiles** | **sf**} [[*[folder/]workstation***#**]*file*]
[**;***state*[**;**...]]
[**;deps**[**;keys** | **info** | **logon**]]
[**;offline**]

Arguments

[folder/]workstation

Specifies the name of the workstation on which the file exists. The default is the workstation on which **conman** is running. Wildcard characters are permitted.

file

Specifies the name of the file. The name must be enclosed in quotes (") if it contains characters other than the following: alphanumerics, dashes (-), slashes (/), backslashes (\), and underscores (_). The default is to display all file dependencies. Wildcard characters are permitted.

state

Specifies the state of the file dependencies to be displayed. The default is to display file dependencies in all states. The states are as follows:

yes

File exists and is available.

no

File is unavailable, or does not exist.

?

Availability is being checked.

<blank>

The file has not yet been checked, or the file was available and used to satisfy a job or job stream dependency.

keys

Displays a single column list of the objects selected by the command.

deps

Displays information in the **deps** format. Use **keys**, **info**, or **logon** to modify the display.

offline

Sends the output of the command to the **conman** output device. For information about this device, see the offline output topic in SETUPCONMAN.

Results

The output of the command is produced in three formats: **standard**, **keys**, and **deps**. The arguments **keys**, **info**, and **logon** modify the **deps** display.

Example

Examples

To display the status of a file dependency for `d:\apps\mis\lib\data4`, run the following command:

```
showfiles d:\apps\mis\lib\data4
```

To display **offline** the status of all file dependencies on all workstations stored in all folders in the `deps` format, run the following command:

```
sf /@/@#@;deps;offline
```

To display the status of all file dependencies on all workstations in the `deps` format, run the following command:

```
sf @#@;deps
```

A sample output is the following:

```
                                                (Est) (Est)
Workstation Job Stream SchedTime  Job      State Pr Start Elapse ReturnCode Dependencies

MASTER#/test/^LFILEJOB^ Dependencies are:
 MASTER     #LFILEJOB  0600 11/26 ******** READY 10
                                LFILEJOB HOLD  10 (11/26)                ^LFILEJOB^

MASTER#/usr/home/me10_99/`/usr/home/me10_99/bin/parms FILE_JS1` Dependencies are:
 MASTER     #FILE_JS1  0600 11/26 ******** HOLD  10 (11/26)                parms FILE_JS1`
                                FILE_JS1 HOLD  10 (11/26)

MASTER#/usr/home/me10_99/`/usr/home/me10_99/bin/parms FILE_JOB1` Dependencies are:
 MASTER     #FILE_JOB1 0600 11/26 ******** READY 10
                                FILE_JB1 HOLD  10 (11/26)                parms FILE_JB1`
```

See also

From the Dynamic Workload Console you can perform the same task as follows:

1. In the navigation bar at the top, click **Monitoring and Reporting > Orchestration Monitor**.
2. Select an engine.
3. In **Object Type**, select **File**.
4. From the **Query** drop-down list, select a task to monitor files.
5. Click **Run** to run the monitoring task.

Standard format

Exists

The state of the file dependency.

File Name

The name of the file.

Keys format

Files are listed with one file on each line. Directory names are not included. Each file is listed in the following format:

```
workstation#file
```

Deps format

Files are listed followed by the dependent jobs and job streams. Jobs are listed in the standard **showjobs** format. Job streams are listed in the standard **showschedules** format.

Deps;keys format

Jobs and job streams that have `file` dependencies are listed with one on each line, in the following format:

```
workstation#jstream[.job]
```

Deps;info format

Files are listed, followed by the dependent jobs and job streams. Jobs are listed in the **showjobs;info** format. Job streams are listed in the standard **showschedules** format.

Deps;logon format

Files are listed followed by the dependent jobs and job streams. Jobs are listed in the **showjobs;logon** format. Job streams are listed in the standard **showschedules** format.

SHOWJOBS

Displays information about jobs.

For information about how to use wildcards to filter jobs and the folders within which they are defined, see Wildcards on page 553.

The displayed information is updated only as long as HCL Workload Automation (batchman) is running. Whether batchman is up or down is confirmed on screen by the `Batchman LIVES` or `Batchman down` message when you issue the `conman start` command.

You must have ***list*** access to the object being shown if the ***enListSecChk*** option was set to **yes** on the master domain manager when the production plan was created or extended.

Syntax

{**showjobs** | **sj**} [*jobselect*]
[**;keys** | **info** | **step** | **logon** | **crit** | **keys retcod**]
[**;short** | **single**]
[**;offline**]
[**;showid**]

{**showjobs** | **sj**} [*jobselect*]
[**;keys**]
[**;extended**]
[**;short** | **single**]
[**;offline**]
[**;showid**]

{**showjobs** | **sj**} [*jobselect*]
[**;deps**[**;keys** | **info** | **logon**]]
[**;short** | **single**]
[**;offline**]
[**;showid**]
[**;props**]

{**showjobs** | **sj**} [*jobselect* |
[*[folder/]workstation#*]*jobnumber.hhmm*]
[**;stdlist**[**;keys**]]
[**;short** | **single**]
[**;offline**]
[**;showid**]
[**;props**]

Arguments

crit

Displays information in the **crit** format.

deps

Displays information in the "deps" format; that is, the jobs used in *follows* dependencies are listed followed by the dependent jobs and job streams. Jobs are listed in the basic `showjobs` format. Job streams are listed in the basic `showschedules` format. Use "keys", "info", or "logon" to modify the "deps" display.

hhmm

The time the job started. Use this, together with the **stdlist** and **single** arguments, to display a specific instance of the job.

info

Displays information in the **info** format.

Note: When displaying the output for a job, the job definition is not displayed correctly in USERJOBS and the output conditions are not displayed. However, the correct information can be seen in the stdlist.

jobnumber

The job number.

jobselect

See JOBSELECT for information about selecting jobs in commands.

keys

Displays a single column list of the objects selected by the command.

logon

Displays information in the **logon** format.

offline

Sends the output of the command to the **conman** output device. For information about this device, see the offline output topic in SETUPCONMAN.

props

Displays the following information about the specified job instance, you must have display access to the props of the specified job instance being shown:

General Information

- Job
- Workstation Folder
- Workstation
- Task
- Task Type
- Job Stream
- Job Stream Workstation Folder
- Job Stream Workstation
- Scheduled Time
- Priority
- Login
- Monitored
- Requires Confirmation
- Interactive
- Critical

Runtime Information

- Actual Workstation
- Status
- Internal Status
- Not Satisfied Dependencies
- Job Number
- Rerun Options
- Information
- Promoted
- Return Code
- Return Code Mapping Expression
- Successful output conditions related to a SUCC job status
- Other output conditions

Time Information

- Actual Start
- Earliest Start
- Latest Start
- Latest Start Action
- Maximum Duration
- Maximum Duration Action
- Minimum Duration
- Minimum Duration Action
- Critical Latest Start
- Deadline
- Repeat Range
- Actual Duration
- Estimated Duration
- Confidence Interval

Recovery Information

- Action
- Message
- Job Definition
- Workstation Folder
- Workstation
- Retry after
- Number of attempts
- Current attempt
- Run on same workstation

Extra Information

This section shows additional properties specific for shadow jobs and jobs defined by JSDL. For shadow jobs it contains the following information:

For distributed shadow jobs:

- Remote Job Scheduled Time
- Remote Job
- Remote Job Stream
- Remote Job Stream Workstation

For z/OS shadow jobs:

- Remote Job Scheduled Time
- Remote Job
- Remote Job Workstation
- Remote Job Error Code



Note: When displaying the output for a job, the job definition is not displayed correctly in USERJOBS and the output conditions are not displayed. However, the correct information can be seen in the stdlist.

retcod

Displays the return code for the job. This argument must be used in conjunction with the **keys** argument, for example:

```
%sj @; keys retcod
```

short

Shortens the display for **every** and **rerun** jobs to include only the following:

- The first iteration
- Jobs in different states
- Exactly matched jobs



Note: This field shows the specific properties if the job is a shadow job or a job defined by JSDL.

showid

Displays for each job stream the job stream identifier.

single

Selects only the parent job in a chain that can include reruns, repetitions, and recovery jobs. The job must be identified by job number in *jobselect*. This is useful with the **stdlist** option.

stdlist

Displays information in the **stdlist** format. Use the **keys** argument to modify the display.

Note: Information on archived jobs is not retrievable using the **stdlist** option.

step

Displays information in the **step** format.

workstation

The name of the workstation on which the job runs. Wildcard characters are permitted.

Comments

If a job fails because the agent is not available, the job is automatically restarted and set to the READY status, waiting for the agent to connect again. As soon as the agent connects again, the job is submitted.

Results

The output of the **showjobs** command is produced in eight formats: **standard**, **keys**, **info**, **step**, **logon**, **deps**, **crit**, and **stdlist**. The **keys**, **info**, **crit**, and **logon** arguments modify the displays.

Example

Examples

- To display the status of all jobs in the `acctg` job stream defined in the folder `echo` on workstation `site3`, stored in folder `myfolder`, you can run the `showjobs` command in one of these two formats:

  ```
  showjobs /myfolder/site3#/echo/acctg.@
  ```

  or:

  ```
  showjobs /myfolder/site3#/echo/acctg
  ```

- To display the status of job `JBA` belonging to job stream `TEST1` on workstation `CPUA`, on which you are running **conman**, and ask to show the job stream identifier for the job stream, run the following command:

  ```
  sj CPUA#TEST1(0900 02/19/18).JBA
  ```

  A sample output for this command is the following:

  ```
  Workstation Job Stream SchedTime Job State Pr Start Elapse ReturnCode Dependencies

  CPUA        #TEST1    0900 02/19 *** HOLD   0(02/19)                {02/20/18}; -TEST-
                                   JBA HOLD  66(14:30)                J2(0600 02/24/18).JB1
  ```

  - The **at** dependency is shown as `(14:30)` in the Start column and the follows dependency from the job `J2(0600 02/24/18).JB1` for job `JOBA` is shown in the Dependencies column.
  - In the Dependencies column the date enclosed in braces, `{02/20/15}`, indicates that the job stream instance has been carried forward and the date indicates the day when the job stream instance was added to the production plan for the first time.

- The following output example displays the status of all jobs, including predecessors and, in particular, job, `JOBVACS`, defined in job stream, `JSHOLIDAYS1`, that has a conditional dependency on predecessor job, `JOBCHECKCALC`, that specifies that `JOBVACS` runs if `JOBCHECKCAL` goes into either `ABEND` or `FAIL` state:

```
S_MDM      #JSHOLIDAYS  0600 09/23 ******************************** READY 10  (00:01)
                         (S_AGT#)JOBCHECKCAL                        HOLD  19
                         (S_AGT#)JOBVACS                            HOLD  10

JOBCHECKCAL IF ABEND | FAIL
```

- To display the status of jobs belonging to job stream `JSDOC` on workstation `site3`, on which you are running **conman**, and ask to show the job stream identifier for the job stream, run the following command:

```
%sj JSDOC.@;showid
```

  A sample output for this command is the following:

```
Workstation Job Stream SchedTime Job State Pr Start Elapse ReturnCode Dependencies

site3        #JSDOCOM 0600 11/26 *** SUCC  10 11/26 00:01            {0AAAAAAAAAAAACRZ}
                                 JDOC SUCC  10 11/26 00:01       0    #J25565
```

  The job stream identifier `0AAAAAAAAAAAACRZ` for job stream `JDOCOM` is shown in the Dependencies column.

  Note: The time or date displayed in the **Start** column is converted in the time zone set on the workstation where the job stream is to run.

- To display the status of jobs belonging to job stream `JSDOCOM` on workstation `site3`, and ask to show the information about the user ID under which the job runs, run the following command:

```
sj site3#JSDOCOM.@;logon
```

  A sample output for this command is the following:

```
Workstation Job Stream SchedTime  Job    State Job#    Logon   ReturnCode
site3       #JSDOCOM  0600 11/26
                                  JDOCOM SUCC  #J25565 me10_99     0
```

- To display the status of all jobs in the HOLD state on all workstations, in the **deps** format, run the following command:

```
sj @#@.@+state=hold;deps
```

  a sample output is the following:

```
Workstation Job Stream SchedTime Job  State Pr Start Elapse RetCode Dependencies

CPUA#JS2.JOBB Dependencies are:
```

```
CPUA        #JS21     0900 02/19 ***** HOLD  0(02/19)              {02/20/18}; -TEST- JOBA HOLD
 66(14:30)
                                                                                JS22(0600
 02/24/18).JOBB

CPUA#JS25.JOBC Dependencies are:


CPUA       #JS25      0600 02/24 ***** HOLD 10(02/24)              {02/20/18}
                                 jobaa HOLD 10(02/24)(00:01)       TEST1; JOBC          TEST2; JOB1
                                                                                JS18(0600 02/24/18).@

CPUA#JS25.JOB1 Dependencies are:


CPUA       #JS25      0600 02/24 ***** HOLD 10(02/24)               {02/20/18}
                                 JOBC  HOLD 10(02/24)(00:01)        JOB1
                                 jobaa HOLD 10(02/24)(00:01)        TEST1; JOBC         TEST2; JOB1
JS18(0600 02/24/18).@
```

- To display the log from the standard list files for the job `JOBC` in the job stream `JS25(0600 09/24/23)` on workstation `CPUA`, running in a UNIX® environment, run the following command:

  ```
  sj CPUA#JS25 (0600 09/24/23).JOBC;stdlist
  ```

  The output is the following:

  ```
  ================================================================
  = JOB : CPUA#JS25[(0600 09/24/23),(0AAAAAAAAAAAABQM)].JOBC
  = USER : mdm93mdm
  = JCLFILE : ls
  = TWSRCMAP :
  = AGENT : CPUA
  = Job Number: 987278608
  = Sun Sep 24 17:06:27 CEST 2023
  ================================================================
  AE
  CAP
  IMShared
  TWA
  WebSphere

  ================================================================
  = Exit Status : 0
  = SC STATUS_OK : true
  = OC OUTPUTCOND2 : false
  = OC OUTPUTCOND1 : true
  = System Time (Seconds) : 0
  Elapsed Time (hh:mm:ss) : 00:00:01
  = User Time (Seconds)   : 0
  = Job CPU usage (ms) : 20
  = Job Memory usage (kb) : 1272
  = Sun Sep 24 15:16:25 CEST 2023
  ```

  where:

Exit Status

Is the status of the job when it completed.

OC <*output_condition_name*>

The result of the evaluation of the output conditions that when satisfied, determine which successor job runs. Output conditions that are satisfied display `true`, and output conditions that are not satisfied display `false`. Successful output conditions are represented by the `sc` flag in the ;stdlist output.

System Time

Is the time the kernel system spent for the job.

Elapsed Time

Is the elapsed time for the job.

User Time

Is the time the system user spent for the job.

Note: The **System Time** and **User Time** fields are used only in UNIX®. Their values in Windows® are always set to **0**. This is because, in Windows®, the **joblnch.exe** process runs in a very short time, which can be considered null.

For an explanation of the Estimated Duration and Confidence Interval job properties, see the section about the logman command in *User's Guide and Reference*.

- To display the properties of the job with job number `227137038`, run the following command:

```
sj 227137038;props
```

A sample output for this command is the following:

```
sj SMA1964199;props
General Information
 Job = JOBAUTO
 Workstation = NC005090_1
 Task =
 <?xml version="1.0" encoding="UTF-8"?>
 <jsdl:jobDefinition xmlns:jsdl="http://www.ibm.com/xmlns/prod/scheduling/1.0/jsdl"
   xmlns:jsdle=
     "http://www.ibm.com/xmlns/prod/scheduling/1.0/jsdle">
   <jsdl:application name="executable">
   <jsdle:executable interactive="false">
   <jsdle:script>ping -n 180 localhost</jsdle:script>
   </jsdle:executable>
   </jsdl:application>
   </jsdl:jobDefinition>
 Task Type = executable
 Job Stream = SMA1964199
 Job Stream Workstation = NC005090_1
 Scheduled Time = 11/20/2023 16:57:00 TZ CET
 Priority = 50
```

```
 Login =
 Monitored = No
 Requires Confirmation = No
 Interactive = No
 Critical = No

Runtime Information
 Status = Running
 Internal Status = EXEC
 Not Satisfied Dependencies = 0
 Job Number = 227137038
 Rerun Options =
 Information =
 Promoted = No
 Return Code =
 Success Output Conditions
 STATUS_OK = true “RC=0”
Other Output Conditions
 STATUS_ERR1 = false “RC=2”
 STATUS_ERR2 = false “RC=3”

Time Information
 Actual Start = 11/20/2023 16:57:36 TZ CET
 Earliest Start =
 Latest Start =
 Latest Start Action =
 Maximum Duration =
 Maximum Duration Action =
 Minimum Duration =
 Minimum Duration Action =
 Critical Latest Start =
 Deadline =
 Repeat Range =
 Actual Duration =
 Estimated Duration = 00:03:02 (hh:mm:ss)
 Confidence Interval = 00:00:00 (hh:mm:ss)

Recovery Information
 Action = Stop
 Message =
 Job Definition =
 Workstation =
 Retry after =
 Number of attempts=
 Current attempt =
 Run on same workstation =

Extra Information
 PID = 3132
```

- The following example displays the status of the job `dbseload` with a return code of **7** and a state of SUCCESSFUL:

```
$conman sj workstation#DAILY_DB_LOAD
 ##################################################################
  # Licensed Materials - Property of IBM* and HCL**
  # 5698-WSH
  # (C) Copyright IBM Corp. 1998, 2016 All rights reserved.
  # (C) Copyright HCL Technologies Ltd. 2016, 2026 All rights reserved.
```

```
  # * Trademark of International Business Machines
  # ** Trademark of HCL Technologies Limited
  ################################################################
Installed for user "tme10_99".
Locale LANG set to the following: "en"
Scheduled for (Exp) 02/20/18 (#35) on CPUA.
Batchman LIVES. Limit:50,Fence:0,Audit Level:0
sj workstation#DAILY_DB_LOAD
(Est) (Est)
CPU Schedule Job State Pr Start
Elapse Dependencies Return Code
WORKSTATION #DAILY_DB_LOAD ***************************** SUCC 10 22:11
00:04
DATASPLT SUCC 10 22:11
00:01 #J17922 0
DATAMRGE ABEND 10 22:12
00:01 #J17924 1
CHCKMRGE SUCC 10 22:12
00:01 #J17926 0
DATACLNS SUCC 10 22:12
00:01 #J17932 0
DATARMRG SUCC 10 22:13
00:01 #J18704 0
DBSELOAD SUCC 10 22:13
00:01 #J18706 7
DATAREPT SUCC 10 22:13
00:01 #J18712 0
DATARTRN SUCC 10 22:14
00:01 #J18714 0
$
```

- The following example displays the return code for a specific job named `workstation#daily_db_load.dbseload`:

```
$ conman sj workstation#daily_db_load.dbseload\;keys\;retcod
  ################################################################
  # Licensed Materials - Property of IBM* and HCL**
  # 5698-WSH
  # (C) Copyright IBM Corp. 1998, 2016 All rights reserved.
  # (C) Copyright HCL Technologies Ltd. 2016, 2026 All rights reserved.
  # * Trademark of International Business Machines
  # ** Trademark of HCL Technologies Limited
  ################################################################
Installed for user "tme10_99".
Locale LANG set to the following: "en"
Scheduled for (Exp) 02/20/19 (#35) on CPUA.
Batchman LIVES. Limit:50,Fence:0,Audit Level:0
sj workstation#daily_db_load.dbseload;keys;retcod 8
$
```

  The *retcod* feature when integrated into a script can become quite powerful.

- The following example shows a job stream containing a job (named RE_ACCOUNTS_JOB) which ends in ABEND state. After the parent RE_ACCOUNTS_JOB job fails, the recovery job starts. When the recovery job completes in success state, the parent job waits a minute, then reruns. The sequence is repeated for three times, if the parent job

keeps failing. If the parent job completes successfully, the rerun sequence is interrupted at the first successful run of the parent job:

```
SCHEDULE NC053009#RE_JS_04
ON RUNCYCLE RC1 "FREQ=DAILY;INTERVAL=1"
:
NC053009_1#JOB_2
 FOLLOWS RE_ACCOUNTS_JOB
NC053009_1#RE_ACCOUNTS_JOB
END

NC053009_1#RE_ACCOUNTS_JOB
 DOCOMMAND "sleep 60; exit 5"
 STREAMLOGON bankmngr
 TASKTYPE UNIX
 RECOVERY RERUN REPEATEVERY 0001 FOR 3
  AFTER NC053009_1#RECOVERY_JOB
```

- The following is the output of the sj command at the end of the sequence:

```
NC053009        #RE_JS_04         0000 01/24 ************************************STUCK 10 17:40
                          (NC053009_1#)JOB_2                              HOLD  10           RE_ACCOUNTS_JOB
                          (NC053009_1#)RE_ACCOUNTS_JOB             ABEND 10 17:40  00:01        5 #J277948077
 >>recovery               (NC053009_1#)RECOVERY_JOB                SUCC  10 17:42  00:01        0 #J277948082
 >>rerun 1 of 3           (NC053009_1#)RE_ACCOUNTS_JOB             ABEND 10 17:44  00:01        5 #J277948085
 >>recovery               (NC053009_1#)RECOVERY_JOB                SUCC  10 17:45  00:01        0 #J277948087
 >>rerun 2 of 3           (NC053009_1#)RE_ACCOUNTS_JOB             ABEND 10 17:47  00:01        5 #J277948089
 >>recovery               (NC053009_1#)RECOVERY_JOB                SUCC  10 17:48  00:01        0 #J277948090
 >>rerun 3 of 3           (NC053009_1#)RE_ACCOUNTS_JOB             ABEND 10 17:50  00:01        5 #J277948092
                                                                                                [Recovery]
```

- To show job streams defined in the folder named FOLDJS_API on all workstations in all folders using an absolute path, submit the following command:

```
conman ss /@/@#/FOLDJS_API/@
```

The following is the output of this command:

```
%ss /@/@#/FOLDJS_API/@
                                                    (Est)  (Est)    Jobs  Sch
Workstation      Job Stream       SchedTime   State Pr Start  Elapse   #  OK Lim
>>               /FOLDJS_API/
EU-HWS-LNX127   #JS_API_132A  1155 07/31      READY 10                 0   0
EU-HWS-LNX127   #JS_API_132B  1543 07/31      ABEND 10 15:44  00:00    1   0
/US/
US-HWS-WIN10    #JS_API_144C  0000 07/31      HOLD  20                 1   0
```

Optionally, you can obtain this same result, by submitting the command using a relative path. To submit the command using a relative path, use the -cf option to change the current folder from root (/) to /FOLDJS_API/ as follows:

```
conman -cf /FOLDJS_API/ ss /@/@#@
```

An additional way to obtain the same result is to change the current folder in conman first, then submit the command:

```
%cf /FOLDJS_API/
```

```
%ss /@/@#@
```

See also

From the Dynamic Workload Console you can perform the same task as follows:

1. In the navigation bar at the top, click **Monitoring and Reporting > Orchestration Monitor**.
2. Select an engine.
3. In **Object Type**, select **Workstation/Job/Jobstream**.
4. From the **Query** drop-down list, select **All Jobs in plan** or another task to monitor jobs.
5. Click **Run** to run the monitoring task.

Standard format

CPU

The workstation on which the job runs.

Schedule

The name of the job stream.

SchedTime

The time and date when the job was scheduled to run in the plan.

Job

The name of the job. The following notation may precede a job name:

>> rerun as

A job that was rerun with the **rerun** command, or as a result of automatic recovery.

>> rerun *rerun_number* of *rerun_total*

A job that is part of a rerun sequence and its position within the sequence

>> rerun step

A job that was rerun with the **rerun ;step** command.

>> every run

The second and subsequent runs of an every job.

>> recovery

The run of a recovery job.

State

The state of the job or job stream. Job states are as follows:

ABEND

The job ended with a non-zero exit code.

ABENP

An **abend** confirmation was received, but the job is not completed.

ADD

The job is being submitted.

CANCL

For internetwork dependencies only. The remote job or job stream has been cancelled.

DONE

The job completed in an unknown state.

ERROR

For internetwork dependencies only, an error occurred while checking for the remote status.

EXEC

The job is running.

EXTRN

For internetwork dependencies only, the status is unknown. An error occurred, a rerun action was just performed on the job in the `EXTERNAL` job stream, or the remote job or job stream does not exist.

FAIL

Unable to launch the job.

FENCE

The priority of the job is below the fence.

HOLD

The job is awaiting dependency resolution.

INTRO

The job is introduced for launching by the system.

PEND

The job completed, and is awaiting confirmation.

READY

The job is ready to launch, and all dependencies are resolved.

SCHED

The **at** time set for the job has not been reached.

SUCC

The job completed with an exit code of zero.

SUCCP

A SUCC confirmation was received, but the job is not completed.

SUPPR

The job is suppressed because the condition dependencies associated to its predecessors are not satisfied.

WAIT

The job is in the WAIT state (extended agent).

Job stream states are as follows:

ABEND

The job stream ended with a nonzero exit code.

ADD

The job stream was added with operator intervention.

CANCELP

The job stream is pending cancellation. Cancellation is deferred until all of the dependencies, including an at time, are resolved.

ERROR

For internetwork dependencies only, an error occurred while checking for the remote status.

EXEC

The job stream is running.

EXTRN

For internetwork dependencies only. This is the state of the EXTERNAL job stream containing jobs referencing to jobs or job streams in the remote network.

HOLD

The job stream is awaiting dependency resolution.

READY

The job stream is ready to launch and all dependencies are resolved.

STUCK

Execution of the job stream was interrupted. No jobs are launched without operator intervention.

SUCC

The job stream completed successfully.

SUPPR

The job stream is suppressed because the condition dependencies associated to its predecessors are not satisfied.

Pr

The priority of the job stream or job. A plus sign (+) preceding the priority means the job has been launched.

(Est)Start

The start time of the job stream or job. Parentheses indicate an estimate of the start time. If the command is performed on the same day when the job is scheduled to run, the **Start** parameter displays a time as (Est)Start. If the command is performed on a day different from the day when the job is scheduled to run, the **Start** parameter displays a date as (Est)Start. For example if you have the following job whose start time occurs on the same day when the job is scheduled to run:

```
SCHEDULE MASTERB1#JS_B
ON RUNCYCLE RULE1 "FREQ=DAILY;"
AT 1700
:
MASTERB1#JOB1
 AT 1800
END
```

You receive the following output:

```
%sj @#@
                                          (Est)  (Est)
CPU      Schedule SchedTime  Job   State Pr Start  Elapse RetCode Deps
MASTERB1#JS_B     1700 08/18 ***** HOLD  10(17:00)
                             JOB1  HOLD  10(18:00)
```

For example if you have the following job whose start time occurs on a day different from the day when the job is scheduled to run:

```
SCHEDULE MASTERB1#JS_A
ON RUNCYCLE RULE1 "FREQ=DAILY;"
AT 0400
:
MASTERB1#JOB_A
 AT 0500
END
```

You receive the following output:

```
%sj @#@
                                          (Est)  (Est)
CPU      Schedule SchedTime  Job   State Pr Start  Elapse RetCode Deps
MASTERB1#JS_A     0400 08/19 ***** HOLD  10(08/19)
                             JOB_A HOLD  10(08/19)
```

(Est)Elapse

The run time of the job stream or job. Parentheses indicate an estimate based on logged statistics.

dependencies

A list of job dependencies and comments. Any combination of the following can be listed:

- For a `follows` dependency, a job stream or job name is displayed.

  If the job or job stream is a pending predecessor, its name is followed by a `[P]`.

  In case of an orphaned dependency an `[O]` is displayed.

For conditional dependencies, the name of the predecessor job or job stream is displayed followed by one or more output conditions in the format, IF <*condition_name*> ... where *condition_name* can represent the execution status of the predecessor job or job stream, the job or job stream status, or other conditions based on the output or outcome of the predecessor job. When there is more than one condition specified, the conditions are separated by the pipe (|) symbol. The following is what appears in the showjob output in the Dependencies column for a predecessor job, JOBL1, with several output conditions set on it. Whichever condition is satisfied determines which successor job runs:

```
JOBL1 IF EXEC
  | STATUS_OK | STATUS_ERR12
```

When more than one output condition is aggregated or joined, for example, when 2 conditions out of 2 need to be satisfied before the successor job can run, then the output is displayed as follows:

```
       JOIN MYJOIN      2 OF
JOBL1 IF EXEC
JOBL2 IF ABEND
```

For more information on pending predecessors and orphaned dependencies refer to the topic on managing external follows dependencies for jobs and job streams in the chapter on "Managing the production cycle" in *User's Guide and Reference*.

- For an `opens` dependency, the file name is displayed. If the file resides on an extended agent and its name is longer than 25 characters, only the last 25 characters are displayed.
- For a `needs` dependency, a resource name enclosed in hyphens (-) is displayed. If the number of units requested is greater than one, the number is displayed before the first hyphen.
- For a **deadline** time, the time preceded by an angle bracket (<) is displayed.
- For an **every** rate, the repetition rate preceded by an ampersand (&) is displayed.
- For an **until** time, the time preceded by an angle bracket (<) is displayed.
- For a **maximum duration** time that is exceeded, **[MaxDurationExceeded]** is displayed in addition to the setting maxdur=*hhh:mm*.
- For a **maximum duration** time that is exceeded, and for which the **onmaxdur** *action* is set to `Kill`, **[KillSubmitted]** is displayed.
- For a **maximum duration** time that is exceeded, and for which the **onmaxdur** *action* is set to `Continue`, **[Continue]** is displayed.
- For a **minimum duration** time that is not reached and for which a job completes with success, [MinDurationNotReached] is displayed in addition to the setting mindur=*hhh:mm*.
- For a **minimum duration** time that is not reached, and for which the **onmindur** *action* is set to `Continue`, **[Continue]** is displayed.
- For a **minimum duration** time that is not reached, and for which the **onmindur** *action* is set to `Abend`, **[Abended]** is displayed.
- For a **minimum duration** time that is not reached, and for which the **onmindur** *action* is set to `Confirm`, **[ConfirmSubmitted]** is displayed.
- For a `prompt` dependency, the prompt number is displayed in the format *#num*. For global prompts, the prompt name follows in parentheses.

- For running jobs, the process identification number (PID) is displayed in the format **#J***nnnnn*.
- Jobs submitted on UNIX® using the HCL Workload Automation **at** and **batch** commands are labeled **[Userjcl]**.
- When reporting time dependencies the **showjobs** command shows in the **Start** column:
  - Only the time *hh:mm* if the day when the time dependencies is set matches with the day when the **showjobs** command is run.
  - Only the date *MM/DD* if the day when the time dependencies is set does not match with the day when the **showjobs** command is run.
- Cancelled jobs are labeled **[Cancelled]**.
- Jobs cancelled with **;pend** option are labeled **[Cancel Pend]**.
- Jobs with expired **until** times, including jobs cancelled with **;pend** option, are labeled **[Until]**.
- **[Recovery]** means that operation intervention is required.
- **[Confirmed]** means that confirmation is required because the job was scheduled using the **confirm** keyword.
- **[Script]** applies to end-to-end networks only; it means that this job has a centralized script and that has not yet been downloaded to the agent.

In the **Dependencies** column is also listed the name of the actual workstation where the job ran. This detail is available only if the job has started and has run on a pool workstation. This information can be useful, for example when you need to determine your license consumption and therefore need to know on which workstation in the pool the job actually ran.

For more information about licensing, see the section about licensing in the Administration Guide.

Keys format

Job names are listed with job stream ID in the following format:

```
workstation#jstream hhmm mm/dd.job
```

for example:

```
HCL Orchestration Command Line Interface(windows)/OCLI 1.1.2.0
Licensed Materials - Property of HCL
(c) Copyright HCL Technologies Ltd. 2019, 2024.
13e5b4d0-7c61-42e2-b2e9-be125f1322f2.JOB1
90832d47-c4b2-4d6e-9b01-8930bdb6202d.JOB2
```

keys;extended

If you add the **extended** option with **keys**, the following information is also displayed:

- Workstation name
- Job stream name
- The scheduled date and time to run the job.

Info format

CPU

The workstation on which the job runs.

Schedule

The name of the job stream.

SchedTime

The time and date when the job was scheduled to run in the plan.

Job

The name of the job. The following notation might precede a job name:

>> rerun as

A job that was rerun with the **rerun** command, or as a result of automatic recovery.

>> rerun *rerun_number* of *rerun_total*

A job that is part of a rerun sequence and its position within the sequence

>> rerun step

A job that was rerun with the **rerun ;step** command.

>> every run

The second and subsequent runs of an every job.

>> recovery

The run of a recovery job.

Job File

The name of the script or executable file of the job. Long file names might wrap, causing incorrect paging. To avoid this, pipe the output to **more**.

Opt

The job recovery option, if any. The recovery options are **RE** for rerun, **CO** for continue, and **ST** for stop.

Job

The name of the recovery job, if any.

Prompt

The number of the recovery prompt, if any.

For example:

```
conman "sj;info | more
```

produces a sample output like the following:

```
 --------Restart---------
CPU   Schedule SchedTime  Job      JobFile               Opt  Job Prompt
M235062+#SCHED_22 1010 03/06
                    JOBMDM   /usr/acct/scripts/gl1
                  (B236153+#)JOB_FTA  echo job12
M235062+#SCHED_22 0600 03/07
                    JOBMDM   /usr/acct/scripts/gl1
                  (B236153+#)JOB_FTA  echo job12
M235062+#FINAL    2359 02/13
                    STARTAPPSERVER  /opt/HCL/TWA/TWS/../wastools/startWas.sh
                                                                CO
                    MAKEPLAN  /opt/HCL/TWA/TWS/MakePlan TWSRCMAP:(RC=0) OR (RC=4)
                    SWITCHPLAN  /opt/HCL/TWA/TWS/SwitchPlan
M235062+#FINALPOSTREPORTS    2359 02/13
                    CHECKSYNC /opt/HCL/TWA/TWS/CheckSync
                    CREATEPOSTREPORTS  /opt/HCL/TWA/TWS/CreatePostReports
                                                                CO
                    UPDATESTATS  /opt/HCL/TWA/TWS/UpdateStats
                                                                CO
M235062+#SCHED12  1010 03/06
                    JOBMDM   /usr/acct/scripts/gl1
                  (B236153+#)JOB_FTA  echo job12
```

The following example displays the status of condition dependencies set on job, `JOB_WAGES`, in job stream, `PAYROLL`. The output conditions are displayed with "`sc`" to identify successful conditions and `"oc`" to identify other output conditions. For each condition name and value pair, a value is assigned to identify whether the condition was satisfied `true`, not satisfied `false` or not yet evaluated `N/A`. :

```
%sj RENOIR#PAYROLL.WAGES;info
-------- Restart ---------
Workstation    Job Stream     SchedTime  Job     JobFile     Opt  Job     Prompt#
RENOIR         #PAYROLL       0000 08/10
                                JOB_WAGES         dir
                                                sc: STATUS_OK true “RC=0”
                                                   oc: STATUS_ERR1 false “RC=2”
                                                   oc: STATUS_ERR2 false “RC=3”
```

Step format

This format is not supported in Windows®.

CPU

The workstation on which the job runs.

Schedule

The name of the job stream.

SchedTime

The time and date when the job was scheduled to run in the plan.

Job

The name of the job. The following notation might precede a job name:

>> rerun as

A job that was rerun with the **rerun** command, or as a result of automatic recovery.

>> rerun *rerun_number* of *rerun_total*

A job that is part of a rerun sequence and its position within the sequence

>> repeated as

The second and subsequent runs of an **every** job.

State

The state of the job or job stream. See "Standard Format" for information about state.

Return code

The return code of the job.

Job#

The process identification number displayed as **#J***nnnnn*.

Step

A list of descendant processes that are associated with the job. For extended agent jobs, only host processes are listed.

Logon format

CPU

The workstation on which the job runs.

Schedule

The name of the job stream.

SchedTime

The time and date when the job was scheduled to run in the plan.

Job

The name of the job. The following notation might precede a job name:

>> rerun as

A job that was rerun with the **rerun** command, or as a result of automatic recovery.

>> rerun *rerun_number* of *rerun_total*

A job that is part of a rerun sequence and its position within the sequence

>> repeated as

The second and subsequent runs of an **every** job.

State

The state of the job or job stream. See "Standard Format" for information about state.

Return code

The return code of the job.

Job#

The process identification number displayed as **#J***nnnnn*.

Logon

The user name under which the job runs.

On Windows operating systems you can have one of the following formats:

user name

Where *username* is the name of the Windows user.

domain\username

Where *domain* is the Windows® domain of the user and the *username* is the name of the Windows user.

username@internet_domain

Where *username@internet_domain* is the name of a system user in an e-mail address format. The username is followed by the "at sign" followed by the name of the Internet domain with which the user is associated.

Note: Insert the escape character '\' before the '@' character in the `username@internet_domain` value in the logon field. For example if you are using the `administrator@bvt.com` user in the logon field, use the following syntax:

```
..................... ; logon=administrator\@bvt.com
```

Stdlist format

A standard list file is created automatically by **jobmon** in Windows® or **jobman** in UNIX®, for each job that **jobmon** and **jobman** launches. You can display the contents of the standard list files using **conman**. A standard list file contains:

- Header and trailer banners.
- Echoed commands.
- The stdout output of the job.
- The stderr output of the job.

To specify a particular date format to be used in the standard list files, change the HCL Workload Automation date format before creating the standard list files. You do this by modifying the date locale format.

Depending on your environment, change the date locale format by performing the steps listed below:

- In UNIX®, set the LANG variable in the environment when **netman** starts. If the LANG variable is not set, the operating system locale is set by default to "C".
- In Windows®, perform the following steps:
  1. Go to Control `Panel→Regional` Options and set your locale (location).
  2. Right-click **My Computer**, go to Properties, click **Advanced**, go to Environment Variables and set the LANG variable as a system variable.
  3. Shut down and restart the system.

The standard list files for the selected jobs are displayed.

Stdlist;keys format

The names of the standard list files for the selected jobs are listed, one on each line.

Crit format

CPU

The workstation on which the job runs.

Schedule

The name of the job stream.

SchedTime

The time and date when the job was scheduled to run in the plan.

Job

The name of the job. The following notation might precede a job name:

>> rerun as

A job that was rerun with the **rerun** command, or as a result of automatic recovery.

>> rerun *rerun_number* of *rerun_total*

A job that is part of a rerun sequence and its position within the sequence

>> repeated as

The second and subsequent runs of an **every** job.

State

The state of the job or job stream. See “Standard Format” for information about state.

Pr

The priority of the job stream or job. A plus sign (+) preceding the priority means the job has been launched.

(Est)Start

The start time of the job stream or job. Parentheses indicate an estimate of the start time. If the start time is more than 24 hours in the past or future, the date is listed instead of the time.

(Est)Elapse

The run time of the job stream or job. Parentheses indicate an estimate based on logged statistics.

CP

Indicates if the job is flagged as critical (**C**) and/or promoted (**P**).

CritStart

The latest time a job can start without impacting the deadlines of mission critical successors.

For example, the result of the following generic command:

```
%sj @#@;crit
```

is:

```
                                            (Est)  (Est)        Crit
CPU     Schedule SchedTime  Job      State Pr Start  Elapse  CP   Start

MYCPU_F+#JSA     1600 03/05 ******** HOLD  10
                            JOBA1    HOLD  10                CP   1759 03/05
                            JOBA2    HOLD  10                     1758 03/05
                            JOBA3    HOLD  10                     1757 03/05
                            JOBA4    HOLD  10                C    1659 03/05
```

Note that:

- The **C** flag applies only to jobs defined as critical in their job stream definition. It is set at plan or submit time.
- The **P** flag applies to both critical jobs and to their predecessors (which are jobs that are not defined as critical but might nonetheless impact the timely completion of a successor critical job). It is set at execution time if the job was promoted.
- Both critical jobs and critical predecessors have a critical start time.

  The scheduler calculates the critical start time of a critical job by subtracting its estimated duration from its deadline. It calculates the critical start time of a critical predecessor by subtracting its estimated duration from the critical start time of its next successor. Within a critical network the scheduler calculates the critical start time of the critical job first and then works backwards along the chain of predecessors. These calculations are reiterated as many times as necessary until the critical job has run.

Deps format

Jobs used in `follows` dependencies are listed followed by the dependent jobs and job streams. Jobs are listed in the standard **showjobs** format. Job streams are listed in the standard **showschedules** format.

Deps;keys format

Jobs and job streams that have `follows` dependencies are listed, one on each line.

Deps;info format

Jobs used in `follows` dependencies are listed, followed by the dependent jobs and job streams. Jobs are listed in the **showjobs;info** format. Job streams are listed in the standard **showschedules** format.

Deps;logon format

Jobs used in `follows` dependencies are listed, followed by the dependent jobs and job streams. Jobs are listed in the **showjobs;logon** format. Job streams are listed in the standard **showschedules** format.

SHOWPROMPTS

Displays information about prompts.

The displayed information is updated only as long as HCL Workload Automation (batchman) is running. Whether batchman is up or down is confirmed on screen by the `Batchman LIVES` or `Batchman down` message when you issue the `conman start` command.

You must have ***list*** access to the object being shown if the ***enListSecChk*** option was set to **yes** on the master domain manager when the production plan was created or extended.

Syntax

{**showprompts** | **sp**} [*[folder/]promptselect*]
  [**;keys**]
  [**;offline**]
[**;showid**]

{**showprompts** | **sp**} [*[folder/]promptselect*]
  [**;deps**[**;keys** | **info** | **logon**]]
  [**;offline**]
  [**;showid**]

Arguments

promptselect

[*[folder/]promptname* | [*[folder/]workstation#*]*msgnum*][**;***state*[**;**...]]

[folder/]promptname

Specifies the name of a global prompt. Wildcard characters are permitted.

[folder/]workstation

Specifies the name of the workstation on which an unnamed prompt is issued. The default is the workstation on which **conman** is running.

msgnum

Specifies the message number of an unnamed prompt.

state

Specifies the state of prompts to be displayed. The states are as follows:

YES

The prompt was replied to with **y**.

NO

The prompt was replied to with **n**.

ASKED

The prompt was issued, but no reply was given.

INACT

The prompt has not been issued.

keys

Displays a single column list of the objects selected by the command.

deps

Displays information in the **deps** format. Use **keys**, **info**, or **logon** to modify the display.

info

Displays information in the **info** format.

logon

Displays information in the **logon** format.

offline

Sends the output of the command to the **conman** output device. For information about this device, see the offline output topic in SETUPCONMAN.

showid

Displays a unique identifier that identifies a prompt. Prompts are no longer identified in the plan solely by their names, but also by the folder in which they are defined. The name and folder association is mapped to a unique identifier. You can verify the unique identifier for prompts by submitting the composer list command, together with the `;showid` filter, or by submitting the conman command, `showprompts`, in combination with the `;showid` filter.

Identifying prompts by their unique identifier avoids the problem that occurred in previous versions when objects were renamed in the plan. For example, if an object was renamed and then carried forward to a new

production plan, references to the old object name were lost. With the implementation of the unique identifier, this will no longer occur and dependencies will be correctly resolved.

When deleting a folder, a prompt, or resource, if there are still objects in the plan that reference these objects, then another folder, prompt, or resource cannot be renamed with the name of the deleted folder, prompt or resource for the number of days specified by folderDays. However, a brand new folder, prompt, or resource can be created with the name of the deleted object. See the folderDays global option in the *Administration Guide*

Note: Prompt numbers assigned to both global and local prompts change when the production plan is extended.

Results

The output of the command is produced in three formats: **standard**, **keys**, and **deps**. The arguments **keys**, **info**, and **logon** modify the **deps** display.

Example

Examples

To display the status of all `prompt` issued on the workstation on which you are running **conman**, run the following command:

```
showprompts
```

a sample is the following:

```
State Message or Prompt
ASKED 1(PRMT3) !continue?
INACT 3(CPUA#SCHED_12[(0600 03/12/18),
      (0AAAAAAAAAAAABST)]) Are you ready to process job1?
INACT 5(CPUA#SCHED_12[(1010 03/12/18),(0AAAAAAAAAAAABSU)])
      Are you ready to process job2?
INACT 7(CPUA#SCHED_22[(0600 03/12/18),(0AAAAAAAAAAAABTR)])
      Are you ready to process job3?
```

To display the status of all `mis` `prompt`s that have been issued, in the **deps** format, run the following command:

```
sp mis@;asked;deps
```

To display the status of `prompt` number 7 on workstation `CPUA`, stored in folder `myfolder`, run the following command:

```
sp myfolder/CPUA#7
```

The output of the command is:

```
INACT 7(myfolder/CPUA#SCHED_22[(0600 03/12/18),(0AAAAAAAAAAAABTR)])
      Are you ready to process job3?
```

See also

From the Dynamic Workload Console you can perform the same task as follows:

1. In the navigation bar at the top, click **Monitoring and Reporting > Orchestration Monitor**.
2. Select an engine.

3. In **Object Type**, select **Prompt**.
4. In the **Query** drop-down list, select **All Prompts in plan**, which will list all prompts regardless of their status, or create and select another task.

Standard format

State

The state of the prompt.

Message or Prompt

For named prompts, the message number, the name, and the text of the prompt. For unnamed prompts, the message number, the name of the job or job stream, and the text of the prompt.

Keys format

The prompts are listed one on each line. Named prompts are listed with their message numbers and names. Unnamed prompts are listed with their message numbers, and the names of the jobs or job streams in which they appear as dependencies.

Deps format

Prompts used as dependencies are listed, followed by the dependent jobs and job streams. Jobs are listed in the standard **showjobs** format. Job streams are listed in the standard **showschedules** format.

Deps;keys format

Jobs and job streams that have `prompt` dependencies are listed one on each line.

Deps;info format

Prompts used as dependencies are listed, followed by the dependent jobs and job streams. Jobs are listed in the **showjobs;info** format. Job streams are listed in the standard **showschedules** format.

Deps;logon format

Prompts used as dependencies are listed, followed by the dependent jobs and job streams. Jobs are listed in the **showjobs;logon** format. Job streams are listed in the standard **showschedules** format.

SHOWRESOURCES

Displays information about resources.

The displayed information is updated only as long as HCL Workload Automation (batchman) is running. Whether batchman is up or down is confirmed on screen by the `Batchman LIVES` or `Batchman down` message when you issue the `conman start` command.

You must have ***list*** access to the object being shown if the ***enListSecChk*** option was set to **yes** on the master domain manager when the production plan was created or extended.

Syntax

{**showresources** | **sr**} [[*[folder/]workstation#*][*folder/]resourcename*]
[**;keys**]
[**;offline**]
[**;showid**]

{**showresources** | **sr**} [[*[folder/]workstation#*][*folder/]resourcename*]
[**;deps**[**;keys** | **info** | **logon**]]
[**;offline**]
[**;showid**]

Arguments

[folder/]workstation

Specifies the name of the workstation on which the resource is defined. The default is the workstation on which **conman** is running.

[folder/]resourcename

Specifies the name of the resource. Wildcard characters are permitted.

keys

Displays a single column list of the objects selected by the command.

deps

Displays information in the **deps** format. Use **keys**, **info**, or **logon** to modify the display.

info

Displays information in the **info** format.

logon

Displays information in the **logon** format.

offline

Sends the output of the command to the **conman** output device. For information about this device, see the offline output topic in SETUPCONMAN.

showid

Displays a unique identifier that identifies a resource. Resources are no longer identified in the plan solely by their names, but also by the folder in which they are defined. The name and folder association is mapped to a unique identifier. You can verify the unique identifier for resources by submitting the composer list command,

together with the `;showid` filter, or by submitting the conman command, `showresources`, in combination with the `;showid` filter.

Identifying resources by their unique identifier avoids the problem that occurred in previous versions when objects were renamed in the plan. For example, if an object was renamed and then carried forward to a new production plan, references to the old object name were lost. With the implementation of the unique identifier, this will no longer occur and dependencies will be correctly resolved.

When deleting a folder, a prompt, or resource, if there are still objects in the plan that reference these objects, then another folder, prompt, or resource cannot be renamed with the name of the deleted folder, prompt or resource for the number of days specified by folderDays. However, a brand new folder, prompt, or resource can be created with the name of the deleted object. See the folderDays global option in the *Administration Guide*

Results

The output of the command is produced in three formats: **standard**, **keys**, and **deps**. The arguments **keys**, **info**, and **logon** modify the **deps** display.

Example

Examples

To display information about all resources on the workstation on which you are running **conman**, run the following command:

```
showresources
```

A sample output is:

```
CPU#Resource      Total Available    Qty UsedBy
CPUA #JOBSLOTS    16        16       No holders of this resource
```

To display information about the `jobslots` resource on workstation `CPUA` stored in folder `myfolder`,in the **deps** format, run the following command:

```
sr myfolder/CPUA#JOBSLOTS;deps
```

A sample output is the following:

```
                                          (Est)  (Est)
Workstation Job Stream SchedTime Job  State Pr Start Elapse RetCode Dependencies

CPUA       #JOBSLOTS Dependencies are:


  FTAA     #SCHED_F+ 0600 03/04 ****** HOLD 55(03/04)              [03/04/25];#33
                        (CPUA#)JOBMDM HOLD 30(03/04)              #1(PRMT3);-16 JOBSLOTS-

  FTAA     #SCHED_F+ 1010 03/04 ****** HOLD 55(03/04)              [03/04/25];#34
                        (CPUA#)JOBMDM HOLD 30(03/04)              #1(PRMT3);-16 JOBSLOTS-

  FTAA     #SCHED_F+ 0600 03/05 ****** HOLD 55(03/05)              [03/04/25];#35
                        (CPUA#)JOBMDM HOLD 30(03/05)              #1(PRMT3);-16 JOBSLOTS-
```

See also

From the Dynamic Workload Console you can perform the same task as follows:

1. In the Welcome page, select **Monitor your workload**, or in the navigation bar at the top of the page, click **Monitoring & Reporting > Orchestration Monitoring > Monitor Workload**.
2. Select an engine.
3. In **Object Type**, select **Resource**.
4. From the **Query** drop-down list, select **All Resources in plan** or another task to monitor resources.
5. Click **Run** to run the monitoring task.

Standard format

CPU

The workstation on which the resource is defined.

Resource

The name of the resource.

Total

The total number of defined resource units.

Available

The number of resource units that have not been allocated.

Qty

The number of resource units allocated to a job or job stream.

Used By

The name of the job or job stream.

Keys format

The resources are listed one on each line.

Deps format

Resources used as dependencies are listed, followed by the dependent jobs and job streams. Jobs are listed in the standard **showjobs** format. Job streams are listed in the standard **showschedules** format.

Deps;keys format

Jobs and job streams that have `resource` dependencies are listed one on each line.

Deps;info format

Resources used as dependencies are listed, followed by the dependent jobs and job streams. Jobs are listed in the **showjobs;info** format. Job streams are listed in the standard **showschedules** format.

Deps;logon format

Resources used as dependencies are listed, followed by the dependent jobs and job streams. Jobs are listed in the **showjobs;logon** format. Job streams are listed in the standard **showschedules** format.

SHOWSCHEDULES

Displays information about job streams.

For information about how to use wildcards to filter job streams and the folders within which they are defined, see Wildcards on page 553.

The displayed information is updated only as long as HCL Workload Automation (batchman) is running. Whether batchman is up or down is confirmed on screen by the `Batchman LIVES` or `Batchman down` message when you issue the `conman start` command.

You must have ***list*** access to the object being shown if the ***enListSecChk*** option was set to **yes** on the master domain manager when the production plan was created or extended.

Syntax

{**showscheds** | **ss**} [*jstreamselect*]
  [**;keys**]
  [**;offline**]
  [**;showid**]

{**showscheds** | **ss**} [*jstreamselect*]
  [**;deps**[**;keys** | **info** | **logon**]]
  [**;offline**]
  [**;showid**]

Arguments

jstreamselect

See JSSELECT for information about selecting job streams in commands.

keys

Displays a single column list of the objects selected by the command.

deps

Displays information in the "deps" format; that is, the job streams used in *follows* dependencies are listed followed by the dependent jobs and job streams. Jobs are listed in the basic `showjobs` format. Job streams are listed in the basic `showschedules` format. Use "keys", "info", or "logon" to modify the "deps" display.

info

Displays information in the **info** format.

logon

Displays information in the **logon** format.

offline

Sends the output of the command to the **conman** output device. For information about this device, see the offline output topic in SETUPCONMAN.

showid

Displays for each job stream the job stream identifier.

Results

The output of the command is produced in three formats: standard, **keys**, and **deps**. The arguments **keys**, **info**, and **logon** modify the **deps** display. The list displayed in the output of the command does not include jobs that were rerun in previous scheduling processes, but the total shown at the end does.

Example

Examples

To display the status of job stream `CLEM_DOCOM` defined in the folder `PROD`, on workstation `SITE3` defined in the folder `EU`, and ask for the job stream identifier, run the following command:

```
%ss /EU/SITE3#/PROD/JS_DOCOM ;showid
```

A sample output of this command is the following:

```
                                       (Est) (Est)  Jobs  Sch
Workstation Job Stream SchedTime State Pr Start Elapse #  OK Lim
>>          /PROD/
site3      #JS_DOCOM  0600 11/26 SUCC  10 11/26 00:01  1   1      {0AAAAAAAAAAAACRZ}
```

To display the status of all job streams defined in all folders in the HOLD state on the workstation on which you are running **conman**, run the following command:

```
showschedules /@/@+state=hold
```

A sample output for this command is the following:

```
                                         (Est)  (Est)  Jobs   Sch
Workstation Job Stream SchedTime  State Pr  Start  Elapse #  OK  Lim
>>          /PROD/
site3       #FILE_JS1  0600 11/26 HOLD  10 (11/26)        1   0       parms FILE_JS1`
```

To display the status of all job streams with name beginning with `sched` on workstation `CPUA` in the **deps;info** format, run the following command:

```
ss CPUA#sched@;deps;info
```

A sample output is the following:

```
   --------Restart---------
  CPU   Schedule SchedTime Job    JobFile              Opt   Job       Prompt

CPUA  #JS_FIRST1[(0600 03/10/18),(0AAAAAAAAAAAABVY)] Dependencies are:
  CPUA#MOD 0212 03/10
                           JOBMDM /usr/scripts/gl1(B236153+#)JOB_FTA1  echo Start gl1?
  CPUA#MOD 0251 03/10
                           JOBMDM /usr/scripts/gl2(B236153+#)JOB_FTA2  echo Start gl2?
```

To display **offline** the status of all job streams in the ABEND state on all workstations, run the following command:

```
ss @#@+state=abend;off
```

To display the status of all job streams defined in all folder and on all workstations defined in all folders, run the following command:

```
%ss /@/@#/@/@
```

This is a sample output for the command. The output includes a job stream with two conditional dependencies. The first, job stream JS_CALC, has an external conditional dependency on a predecessor job JOB12P in the JS_REPORT job stream. If JOB12P satisfies the condition represented by ON STATUS_OK, then the successor runs. The second, job stream JS_CALC has an external conditional dependency on job stream JS_TAX on which two output conditions are set: IF ABEND | SUPPR.

```
                                        (Est)  (Est)    Jobs  Sch
Workstation Job Stream  SchedTime  State Pr  Start  Elapse  #  OK  Lim
>>          /PROD/
site3       #JS_DOCOM   0600 09/20 SUCC  10  09/20  00:01   1   1
site3       #JS_SCRIPT  0600 09/20 SUCC  10  09/20  00:03   1   1
site2       #JS_PRED1   1000 09/20 SUCC  10  09/20  00:01   1   1
site3       #JS_SCRIPT1 0600 09/20 ABEND 10  09/20  00:01   1   0
site3       #LFILEJOB   0600 09/20 READY 10                  1   0
site1       #RES_100    0600 09/20 SUCC  10  09/20  00:09   1   1
site3       #FILE_JS1   0600 09/20 HOLD  10 (09/20)         1   0        parms FILE_JS1`
site3       #FILE_JOB   0600 09/20 SUCC  10  09/20  00:01   1   1
site3       #JS_CALC    0000 09/20 HOLD  10 (09/20)         1   0
JS_REPORT (0000 09/20/25).JOB12P IF STATUS_OK
JS_TAX (0000 09/20/25).@ IF ABEND | SUPPR
```

See also

From the Dynamic Workload Console you can perform the same task as follows:

1. In the navigation bar at the top, click **Monitoring and Reporting > Orchestration Monitor**.
2. Select an engine.
3. In **Object Type**, select **Workstation/Job Stream/Job**.
4. From the **Query** drop-down list, select **All Job Streams in plan** or another task to monitor job streams.
5. Click **Run** to run the monitoring task.

Standard format

CPU

The workstation on which the job stream runs.

Schedule

The name of the job stream.

SchedTime

The time and date when the job stream was scheduled to run in the plan.

State

The state of the job stream. The states are as follows:

ADD

The job stream was added with operator intervention.

ABEND

The job stream ended with a nonzero exit code.

CANCELP

The job stream is pending cancellation. Cancellation is deferred until all of the dependencies, including an at time, are resolved.

ERROR

For internetwork dependencies only, an error occurred while checking for the remote status.

EXEC

The job stream is running.

EXTRN

For internetwork dependencies only. This is the state of the EXTERNAL job stream containing jobs referencing to jobs or job streams in the remote network.

HOLD

The job stream awaiting dependency resolution.

READY

The job stream is ready to launch and all dependencies are resolved.

STUCK

Job stream execution was interrupted. No jobs are launched without operator intervention.

SUCC

The job stream completed successfully.

SUPPR

The job stream is suppressed because the condition dependencies associated to its predecessors are not satisfied.

Pr

The priority of the job stream.

(Est)Start

The start time of the job stream or job. Parentheses indicate an estimate of the start time. If the command is performed on the same day when the job stream is scheduled to run, the **Start** parameter displays a time as (Est)Start. If the command is performed on a day different from the day when the job stream is scheduled to run, the Start parameter displays a date as (Est)Start. For example if you have the following job stream whose start time occurs on the same day when the job stream is scheduled to run:

```
SCHEDULE MASTERB1#JS_B
ON RUNCYCLE RULE1 "FREQ=DAILY;"
AT 1800
:
MASTERB1#JOB1
END
```

You receive the following output:

```
%ss @#@
                                    (Est)  (Est)    Jobs   Sch
CPU      Schedule SchedTime  State Pr Start  Elapse   #  OK  Lim
MASTERB1#JS_B     1800 08/18 HOLD  10(18:00)          1   0
```

For example if you have the following job stream whose start time occurs on a day different from the day when the job stream is scheduled to run:

```
SCHEDULE MASTERB1#JS_A
ON RUNCYCLE RULE1 "FREQ=DAILY;"
AT 0500
:
MASTERB1#JOB1
END
```

You receive the following output:

```
%ss @#@
                                    (Est)  (Est)  Jobs  Sch
CPU      Schedule SchedTime  State Pr Start  Elapse #  OK Lim
MASTERB1#JS_A     0500 08/19 HOLD  10(08/19)        1   0
```

(Est)Elapse

The run time of the job stream. Parentheses indicate an estimate based on logged statistics.

Jobs #

The number of jobs in the job stream.

Jobs OK

The number of jobs that have completed successfully.

Sch Lim

The job stream's job limit. If one is not listed, no limit is in effect.

dependencies

A list of job stream dependencies and comments. Any combination of the following may be listed:

- For a `follows` dependency, a job stream or job name is displayed. If the job or job stream is a pending predecessor, its name is followed by a `[P]`.
- For conditional dependencies, the name of the predecessor job stream is displayed followed by one or more output conditions in the format, IF <*condition_name*> ... where *condition_name* can represent the execution status of the predecessor job stream, the job stream status, or other conditions based on the output or outcome of the predecessor job stream. When there is more than one condition specified, the conditions are separated by the pipe (|) symbol. The following is what appears in the showschedules output in the Dependencies column for a predecessor job stream, JOBSTREAML1, with an ABEND and FAIL status condition set. Depending on whether JOBSTREAML1 completes in ABEND or FAIL status determines which successor job runs:

  ```
  JOBSTREAML1(0000 09/15/15).JOBL1 IF ABEND | FAIL
  ```

- For an `opens` dependency, the file name is displayed. If the file resides on an extended agent, and its name is longer than 25 characters, only the last 25 characters are displayed.
- For a `needs` dependency, a resource name enclosed in hyphens (-) is displayed. If the number of units requested is greater than one, the number is displayed before the first hyphen.
- For an **until** time, the time preceded by an angled bracket (<).
- For a `prompt` dependency, the prompt number displayed as *#num*. For global prompts, the prompt name in parentheses follows.
- Cancelled job streams are labeled **[Cancelled]**.
- Job streams cancelled with the **;pend** option are labeled **[Cancel Pend]**.
- For a **deadline** time, the time preceded by an angle bracket (<) is displayed.
- Job streams that contain the **carryforward** keyword are labeled **[Carry]**.
- For job streams that were carried forward from the previous production plan, the original name and date are displayed in brackets.
- When reporting time dependencies the **showschedules** command shows in the **Start** column:
  - Only the time *hh:mm* if the day when the time dependencies is set matches with the day when the **showschedules** command is run.
  - Only the date *mm/dd* if the day when the time dependencies is set does not match with the day when the **showschedules** command is run.



Note: The time or date displayed in the **Start** column is converted in the time zone set on the workstation where the job stream is to run.

Keys format

The job streams are listed one on each line.

Deps format

Job streams used as dependencies are listed, followed by the dependent jobs and job streams. Jobs are listed in the standard **showjobs** format. Job streams are listed in the standard **showschedules** format.

Deps;keys format

Job streams that have `follows` dependencies are listed one on each line.

Deps;info format

Job streams used in as dependencies are listed, followed by the dependent jobs and job streams. Jobs are listed in the **showjobs;info** format. Job streams are listed in the standard **showschedules** format.

Deps;logon format

Job streams used in as dependencies are listed, followed by the dependent jobs and job streams. Jobs are listed in the **showjobs;logon** format. Job streams are listed in the standard **showschedules** format.

SHUTDOWN

Unconditionally stops all the HCL Workload Automation production processes and services on Windows workstations, including **batchman**, **jobman**, **netman**, **mailman**, **appservman**, all **mailman** servers, and all **writer** processes.

Even though this command does stop the **appservman** service, it does not stop the WebSphere Application Server Liberty services. To stop WebSphere Application Server Liberty services, run the **stopappserver** command. For more information, see the section about **stopappserver** in *User's Guide and Reference*.

The **shutdown** command does not stop the **tokensrv** service.

Note: This command is not supported on remote engine workstations.

You must have ***shutdown*** access to the workstation.

Syntax

{**shutdown** | **shut**} [**;wait**]

Arguments

wait

Waits until all processes have stopped before prompting for another command.

Comments

The **shutdown** command stops the processes only on the workstation on which **conman** is running. To restart **netman** only, run the **StartUp** command. For information about the **StartUp** command, see the "Startup" topic in the chapter on the "Utility commands" in *User's Guide and Reference*. To restart the entire process tree, run the following **conman** commands:

```
start
startappserver
startmon
```

You must run a **conman unlink @** command before executing a **shutdown** command.

Example

Examples

To shut down production on the workstation on which you are running **conman**, run the following command:

```
unlink @
shutdown
```

To shut down production on the workstation on which you are running **conman** and wait for all processes to stop, run the following command:

```
unlink@;noask
shut ;wait
```

START

Starts HCL Workload Automation production processes, except for the event monitoring engine and WebSphere Application Server Liberty (see STARTAPPSERVER and STARTMON to learn about the commands that start these processes).

Note: Make sure **conman start** is not issued while either **JnextPlan** or **stageman** runs.

You must have ***start*** access to the workstation.

Syntax

start [*domain***!**][*folder/*]*workstation*
[;mgr]
[;noask]
[;demgr]

Arguments

domain

Specifies the name of the domain in which workstations are started. Wildcard characters are permitted.

This argument is useful when starting more than one workstation in a domain. For example, to start all the agents in domain **stlouis**, use the following command:

```
start stlouis!@
```

If *domain* is omitted, and *workstation* contains wildcard characters, the default domain is the one in which **conman** is running.

[folder/]workstation

Specifies the name of the workstation to be started. Wildcard characters are permitted.

This command is not supported on remote engine workstations.

mgr

This can be entered only on the workstation on which **conman** is running. It starts the local workstation as the domain manager. The workstation becomes the new domain manager and the current domain manager becomes a fault-tolerant agent. This form of the command usually follows a **stop** command.

Note: The preferred method of switching a domain manager is to use a **switchmgr** command. See SWITCHMGR for more information.

noask

Specifies not to prompt for confirmation before taking action on each qualifying workstation.

demgr

This option prevents the opening of external connections during the transition time between when an agent starts as an old domain manager, and when the **switchmgr** command is run, depriving the agent of the domain manager function. This option is run automatically, but until the old domain manager has processed the **switchmgr** event (in the case, for example, of delayed restart or restart after repairing a damaged agent), the *demgr* option **must** be used to start the old domain manager from the local command line.

Comments

The **start** command is used at the start of each production period to restart HCL Workload Automation following preproduction processing. At that time it causes the autolinked fault-tolerant agents and standard agents to be initialized and started automatically. Agents that are not autolinked are initialized and started when you run a **link** command.

Assuming the user has ***start*** access to the workstations being started, the following rules apply:

- A user running **conman** on the master domain manager can start any workstation in the network.
- A user running **conman** on a domain manager other than the master can start any workstation in that domain and subordinate domains. The user cannot start workstations in peer domains.
- A user running **conman** on an agent can start workstations that are hosted by that agent.

Example

Examples

A figure showing a network and a table of the workstations started by "start" commands run by users in various locations in the network can be found in the topic on the "start" command in *User's Guide and Reference*.

See also

From the Dynamic Workload Console you can perform the same task as follows:

1. From the drop-down menu, select **Workstation**.
2. From the **Query** drop-down list, select a query to monitor workstations.
3. From the table containing the list of workstations, select a workstation and click **Start**.

STARTAPPSERVER

Starts the WebSphere Application Server Liberty on the workstation.

Syntax

startappserver [*domain*!][*folder/*]*workstation*
[**;wait**]

Arguments

domain

Specifies the name of the domain of the workstation. Because workstations have unique names, the domain is not needed when starting the WebSphere Application Server Liberty on a specific workstation. Wildcard characters are permitted.

If *domain* is omitted, and *workstation* contains wildcard characters, the default domain is the one in which **conman** is running.

[folder/]workstation

Specifies the name of the workstation where you want to start the monitoring engine. Wildcard characters are permitted. If no domain and workstations are specified, the action is on the local workstation.

wait

Waits until WebSphere Application Server Liberty has started before prompting for another command.

Comments

Permission to `start` actions on `cpu` objects is required in the security file to be enabled to run this command.

WebSphere Application Server Liberty can also be started with the `StartUp` utility command. For more information, see the "Startup" topic in the chapter on the "Utility commands" in *User's Guide and Reference*.

STARTEVENTPROCESSOR

Starts the event processing server on the master domain manager, backup master, or on a workstation installed as a backup master that functions as a plain fault-tolerant agent.

Syntax

{**starteventprocessor** | **startevtp**} [*domain***!**]*workstation*

Arguments

domain

Specifies the name of the domain of the workstation.

[folder/]workstation

Specifies the name of the workstation where you want to start the event processing server. Wildcard characters are not permitted.

Comments

You can omit the workstation name if you run the command locally.

Permission to `start` actions on `cpu` objects is required in the security file to be enabled to run this command.

See also

From the Dynamic Workload Console you can perform the same task as follows:

1. In the navigation bar at the top, click **Monitoring and Reporting > Orchestration Monitor**.
2. Select an engine.
3. From the drop-down menu, select **Workstation**.
4. From the **Query** drop-down list, select a query to monitor workstations.
5. Click **Run** to run the monitoring task.
6. From the table containing the list of workstations, select a workstation and click **More Actions > Start Event Processor**.

STARTMON

Starts the monman process that turns on the event monitoring engine on the workstation.

Syntax

{**startmon** | **startm**} [*domain***!**][*folder/]workstation*
[**;noask**]

Arguments

domain

Specifies the name of the domain of the workstation. Because workstations have unique names, the domain is not needed when starting the monitoring engine on a specific workstation. Wildcard characters are permitted.

If *domain* is omitted, and *workstation* contains wildcard characters, the default domain is the one in which **conman** is running.

[folder/]workstation

Specifies the name of the workstation where you want to start the monitoring engine. Wildcard characters are permitted.

noask

Specifies not to prompt for confirmation before taking action on each qualifying workstation.

Comments

Permission to `start` actions on `cpu` objects is required in the security file to be enabled to run this command.

See also

From the Dynamic Workload Console you can perform the same task as follows:

1. In the navigation bar at the top, click **Monitoring and Reporting > Orchestration Monitor**.
2. Select an engine.
3. From the drop-down menu, select **Workstation**.
4. From the **Query** drop-down list, select a query to monitor workstations.
5. Click **Run** to run the monitoring task.
6. From the table containing the list of workstations, select a workstation and click **More Actions > Start Event Monitoring**.

STATUS

Displays the **conman** banner and the HCL Workload Automation production status.

Syntax

{**status** | **stat**}

Results

Following the word **schedule** on the second line of output, the production plan (`Symphony` file) mode is shown in parentheses. The **Def** or **Exp** information can appear. **Def** means that the production plan is in non-expanded mode, and **Exp** means it is in expanded mode. The mode of the production plan is determined by the setting of the global option ***expanded version***.

Example

Examples

The following example displays the status of the current production plan.

```
%status
################################################################
  # Licensed Materials - Property of IBM* and HCL**
  # 5698-WSH
  # (C) Copyright IBM Corp. 1998, 2016 All rights reserved.
  # (C) Copyright HCL Technologies Ltd. 2016, 2026 All rights reserved.
  # * Trademark of International Business Machines
  # ** Trademark of HCL Technologies Limited
  ################################################################.
Job stream (Exp) 11/26/19 (#34) on site3.
Batchman LIVES. Limit:19, Fence:0, Audit Level:0
```

STOP

Stops HCL Workload Automation production processes. To stop the **netman** process, use the **shutdown** command. You must have ***stop*** access to the workstation.

Syntax

stop [*domain*!][*folder/]workstation*
  [;wait]
  [;noask]

Arguments

domain

Specifies the name of the domain in which workstations are stopped. Because workstations have unique names, the domain is not needed when stopping a specific workstation. Wildcard characters are permitted.

This argument is useful when stopping more than one workstation in a domain. For example, to stop all the agents in domain **stlouis**, use the following command:

```
stop stlouis!@
```

If *domain* is omitted, and *workstation* contains wildcard characters, the default domain is the one in which **conman** is running.

[folder/]workstation

Specifies the name of the workstation to be stopped. Wildcard characters are permitted.

This command is not supported on remote engine workstations.

wait

Specifies not to accept another command until all processes have stopped.

noask

Specifies not to prompt for confirmation before taking action on each qualifying workstation.

Comments

If the **stop** command cannot be applied to a distant workstation (for example, if the TCP/IP path is not available), the command is stored locally in a `pobox` file, and is sent to the workstation when it becomes linked.

Assuming the user has ***stop*** access to the workstations being stopped, the following rules apply:

- A user running **conman** on the master domain manager can stop any workstation in the network.
- A user running **conman** on a domain manager other than the master can stop any workstation in that domain and subordinate domains. The user cannot stop workstations in peer domains.
- A user running **conman** on an agent can stop any workstation in the local domain.

When you issue a **stop @** command on a domain manager, a local **conman stop** command runs on the remote CPUs. The command starts running on the lowest stations in the network hierarchy, then finally runs on the domain manager. However, the `Symphony` file is not updated before the CPUs go down. Therefore, if you issue a **conman sc@!@** command from any CPU, the resulting information might be an up to date picture of the states of the CPUs, even of the domain manager.

Example

Examples

A figure showing a network and a table of the workstations stopped by "stop" commands run by users in various locations in the network can be found in the topic on the "stop" command in *User's Guide and Reference*.

STOP ;PROGRESSIVE

Stops HCL Workload Automation production processes hierarchically when you have defined at least one workstation as BEHINDFIREWALL in an HCL Workload Automation network. Similar to the stop @!@ command, but more effective in improving plan performance. The command does not run from the domain in which the command was initially issued for each subordinate domain, but runs at each hierarchical level.

Note: This command is not supported on remote engine workstations.

You must have ***stop*** access to the workstation.

Syntax

stop ;progressive

Comments

When you issue the command on a domain manager, all workstations in that domain are stopped and to run on any subordinate domains. The command continues to run in this hierarchical manner, the domain manager stops workstations in the same domain, stops itself, and then continues to run on subordinate domains.

Example

Examples

A figure showing a network and a table of the workstations stopped by using the "stop;progressive" command at selected workstations in the network can be found in the topic on the "stop" command in *User's Guide and Reference*.

STOPAPPSERVER

Stops the WebSphere Application Server Liberty on the workstation.

Syntax

{**stopappserver** | **stopapps**} [*domain***!**][*folder/]workstation*
[**;wait**]

Arguments

domain

Specifies the name of the domain of the workstation. Because workstations have unique names, the domain is not needed when stopping the WebSphere Application Server Liberty on a specific workstation. Wildcard characters are permitted.

If *domain* is omitted, and *workstation* contains wildcard characters, the default domain is the one in which **conman** is running.

[folder/]workstation

Specifies the name of the workstation where you want to stop the monitoring engine. Wildcard characters are permitted. If no domain and workstations are specified, the action is on the local workstation.

wait

Waits until WebSphere Application Server Liberty has stopped before prompting for another command.

Comments

Permission to `stop` actions on `cpu` objects is required in the security file to be enabled to run this command.

On Windows systems refrain from using Windows services to stop WebSphere Application Server Liberty. If you use Windows services, the *appserverman* process, which continues to run, will start WebSphere Application Server Liberty again. Use this command instead.

To be able to run conman stopappserver command, complete the following two customization procedure to provide the user credentials to WebSphere Application Server Liberty:

- Customize the `Attributes for conman (CLI in version 8.4) connections` section in the `localopts` file by specifying the details of the connector or of the master domain manager.

  You must also:

1. Create (or customize if already present) the `useropts` file manually, adding the `USERNAME` and `PASSWORD` attributes for the user who will run stopappserver. Make sure the `useropts` file name is entered in the `USEROPTS` key in the `Attributes for conman (CLI) connections` section. See the section about setting user options in *Administration Guide* for further details.
2. Encrypt the password in the `useropts` file simply by running conman.

On Windows systems, you can also use the Shutdown utility command. For more information, see the "Shutdown" topic in the chapter on the "Utility commands" in *User's Guide and Reference* .

STOPEVENTPROCESSOR

Stops the event processing server.

Syntax

{**stopeventprocessor** | **stopevtp**} [*domain***!**][*[folder/]workstation*]

Arguments

domain

Specifies the name of the domain of the workstation.

workstation

Specifies the name of the master domain manager, backup master, or workstation installed as a backup master that functions as a plain fault-tolerant agent where you want to stop the event processing server. Wildcard characters are not permitted.

You can omit the workstation name if you run the command locally.

Comments

This command cannot be issued in an asynchronous way.

If you issue the command from a workstation other than the one where the event processor is configured, the command uses the command-line client, so the user credentials for the command-line client must be set correctly.

Permission to `stop` actions on `cpu` objects is required in the security file to be enabled to run this command.

See also

From the Dynamic Workload Console you can perform the same task as follows:

1. In the navigation bar at the top, click **Monitoring and Reporting > Orchestration Monitor**.
2. Select an engine.
3. From the drop-down menu, select **Workstation**.
4. From the **Query** drop-down list, select a query to monitor workstations.

5. Click **Run** to run the monitoring task.
6. From the table containing the list of workstations, select a workstation and click **More Actions > Stop Event Processor**.

STOPMON

Stops the event monitoring engine on the workstation.

Syntax

{**stopmon** | **stopm**} [*domain***!**][*folder/*]*workstation*
[**;wait**]
[**;noask**]

Arguments

domain

Specifies the name of the domain of the workstation. Because workstations have unique names, the domain is not needed when stopping the monitoring engine on a specific workstation. Wildcard characters are permitted.

If *domain* is omitted, and *workstation* contains wildcard characters, the default domain is the one in which **conman** is running.

[folder/]workstation

Specifies the name of the workstation where you want to stop the monitoring engine. Wildcard characters are permitted.

wait

Specifies not to accept another command until the monitoring engine has stopped.

noask

Specifies not to prompt for confirmation before taking action on each qualifying workstation.

Comments

The monitoring engine is restarted automatically when the next production plan is activated (on Windows also when HCL Workload Automation is restarted) unless you disable the `autostart monman` local option.

The command is asynchronous, unless you specify the `wait` keyword.

Permission to `stop` actions on `cpu` objects is required in the security file to be enabled to run this command.

See also

From the Dynamic Workload Console you can perform the same task as follows:

1. In the navigation bar at the top, click **Monitoring and Reporting > Orchestration Monitor**.
2. Select an engine.
3. From the drop-down menu, select **Workstation**.
4. From the **Query** drop-down list, select a query to monitor workstations.
5. Click **Run** to run the monitoring task.
6. From the table containing the list of workstations, select a workstation and click **More Actions > Stop Event Monitoring**.

SUBMIT DOCOMMAND

Submits a command to be launched as a job.

To run this command, in the security file you must have ***submit*** access for the job with the name specified in its database definition and, if you use the ***alias*** keyword, also with the name specified with this keyword. In addition, if you use the ***recoveryjob*** keyword, you must have ***submit*** access for the job specified with that keyword.

To include `needs` and `prompt` dependencies, you must have ***use*** access to the resources and global prompts.

If you submit the job from a workstation other than the master domain manager, you must be connecting as a user that:

- has proper credentials defined in the "useropts" file (see the topic on Setting up options for using the user interfaces in the chapter on "Configuring the job environment" in the *User's Guide and Reference*) to connect to the master domain manager through WebSphere Application Server Liberty
- is authorized to perform submit commands in the security file stored on the master domain manager

Syntax

{**submit docommand** = | **sbd**} [*[folder/]workstation#*]"*cmd*"
[**;alias**[**=***name*]]
[**;into=**[*[folder/]workstation***#**]
{*jobstream_id***;schedid** |[*folder/*]*jobstreamname*
([*hhmm*[*date*]])}]
[**;***joboption*[**;**...]]

Arguments

[folder/]workstation

Specifies the name of the workstation on which the job will be launched. Wildcard characters are permitted, in which case, the job is launched on all qualifying workstations. The default is the workstation on which **conman** is running. You cannot specify a domain or workstation class.

Note: Because of a limitation in the way Windows® manages the equal (=) sign in the shell environment, you must mask the equal (=) sign as follows '\='\ when submitting Windows® commands using **submit docommand**. For example, to set the local variable **var1** to *hello* you must issue the following command:

```
%sbd "set var1\"=\"hello"
```

cmd

Specifies a valid system command of up to 255 characters. The entire command must be enclosed in quotes ("). The command is treated as a job, and all job rules apply.

alias=*name*

Specifies a unique name to be assigned to the job. If you enter the **alias** keyword without specifying a name, a name is constructed using up to the first six alphanumeric characters (in upper case) of the command, depending on the number of characters in the command, followed by a ten digit random number. If there are blanks in the command, the name is constructed using up to the first six alphanumeric characters before the blank. For example, if the command is "**rm apfile**", the generated name will be similar to **RM0123456789**. If the command is longer than six alphanumeric characters such as, "**wlsinst**", the generated name will be **wlsins0396578515**.

If you do not include **alias** the first time you submit the command, a job name is constructed using up to 255 characters of the command name. If you submit a command a second time from the same workstation, the **alias** keyword is mandatory and must be unique for each command submission.

into=[*folder/*]*jobstream_instance*

Identifies the job stream instance into which the job will be placed for launching and optionally, the path to the folder where the job stream is located. If [*folder/*] is omitted, then the root folder is assumed. Select the job stream instance as follows:

[*[folder/]workstation***#**][*folder/*]*jobstreamname*([*hhmm*[*date*]])

or

[*[folder/]workstation***#**]*jobstream_id* **;schedid**

If **into** is not used, the job is added to a job stream named **JOBS**.

joboption

Specify any of the following:

at=*hhmm* [**timezone**|**tz** *tzname*] [**+***n* **days** | *mm/dd[/yy]*] | [**absolute** | **abs**]

confirmed

critical

deadline=*time* [**timezone**|**tz** *tzname*][**+***n* **day**[**s** | *mm/dd[/yy]*]

every=*rate*

follows=[*netagent***::**][*workstation***#**]{*jobstreamname*[*hhmm* [*mm/dd[/yy]*]][*.job* | **@**] | *jobstream_id.job***;schedid**}| *job*[**,**...] [**if** '*condition_name*[| *condition_name*][| ...]']

Note: Internetwork dependencies do not support folders, therefore, the network agent workstation, and the jobs and job streams running on them, cannot be defined in a folder different from the root (/). Folders are supported on all other workstation types as follows:

```
[follows {[[folder/]workstation#][folder/]jobstreamname[.jobname]
```

follows=[[*folder/*]*workstation*#][*folder/*]{*jobstreamname*[*hhmm* [*mm/dd[/yy]*]][*.job* | **@**] | *jobstream_id.job*;**schedid**}| *job*[**,**...] [**if** '*condition_name*[| *condition_name*][| ...]']

The *condition_name* variable indicates the name of the condition defined in the job definition. Conditions can be **status conditions**, based on job status, or other **output conditions**, based on a mapping expression such as a return code, output variables, or output found in a job log. In each follows statement, you can specify only one dependency type: either status or output conditions. At submission time, you can add status or output conditions, but no joined dependencies. Network agent workstations do not support folders, therefore neither the network agent nor the jobs or job streams running on them can be defined in folders.

wait

The time in seconds HCL Workload Automation waits before performing a second check on the dependency if the object specified in the dependency does not exist. If the object specified in the dependency was created during the **wait** interval, the submission is performed, otherwise the job is not submitted.

nocheck

Ignores the dependency. If the object specified in the dependency does not exist, the submission is performed anyway.

Note: The **;nocheck** argument is not supported in internetwork dependencies.

interactive

Note: This keyword can be used in Windows® environments only.

maxdur=*time*[**onmaxdur** *action*]

mindur=*time*[**onmindur** *action*]

logon=*user*.

needs=[*num*] [*[folder/]workstation*#][*folder/]resource*[**,**...]

opens=[*[folder/]workstation*#]"*filename*"[**(***qualifier***)**][**,**...]

priority=[*pri* | **hi** | **go**]

prompt="[**:** | **!**]*text*" | *promptname*[**,**...]

recovery=stop | **continue** | **rerun**

recoveryjob=[*[folder/]workstation#*]*jobname*

The name of a recovery job different from the one (if present) specified in the job definition in the database.

after[*[folder/]workstation#*][*folder/*]*jobname*

abendprompt "*text*"

until *time* [**timezone**|**tz** *tzname*][**+***n* **day**[**s**] | [**absolute** | **abs**]] [**;onuntil** *action*]

The default value for ***joboption*** is the user on the workstation from which the command is being run.

Using local parameters

You can use local parameters as values with the following keywords:

- `cmd`
- `opens`
- `logon`
- `prompt`
- `abendprompt`

Local parameters are defined and managed with the "parms" utility command (see the topic on "parms" in the chapter on "Using the utility commands" in *User's Guide and Reference*) in a local database on the workstation where the job is run. The parameters are resolved on the workstation while the submit command is in execution.

Comments

Jobs submitted in production from the **conman** command line are not included in the preproduction plan and so they cannot be taken into account when identifying external follows dependencies predecessors.

If you do not specify a *workstation* with `follows`, `needs`, `opens`, or `into`, the default is the workstation of the job.

The scheduler classifies follows dependencies as *internal* when they are specified only by their job name within the job stream. It classifies them as *external* when they are specified in the *workstationName#[folder/]jobStreamName.jobName* format.

When you submit the object into a job stream and add a follows dependency that shares the same job stream name (for example, you submit the object into job stream `schedA` and define a follows dependency on `schedA.job2`), the dependency is treated as an *external* follows dependency.

Example

Examples

To submit a **rm** command into the job stream `JOBS` with a `follows` dependency, run the following command:

```
submit docommand = "rm apfile";follows sked3
```

To submit a **sort** command with the alias **sortit** and place the job in the job stream `reports` with an **at** time of 5:30 p.m., run the following command:

```
sbd "sort < file1 > file2";alias=sortit;into=reports;at=1730
```

To submit **chmod** commands on all workstations with names beginning with `site`, run the following command:

```
sbd site@#"chmod 444 file2";alias
```

See also

From the Dynamic Workload Console you can perform the same task as described in:

the Dynamic Workload Console User's Guide, section about Submitting ad hoc jobs.

SUBMIT FILE

Submits a file to be launched as a job.

To run this command, in the security file you must have ***submit*** access for the job with the name specified in its database definition and, if you use the ***alias*** keyword, also with the name specified with this keyword. In addition, if you use the ***recoveryjob*** keyword, you must have ***submit*** access for the job specified with that keyword.

To include `needs` and `prompt` dependencies, you must have ***use*** access to the resources and global prompts.

If you submit the job from a workstation other than the master domain manager, you must be connecting as a user that:

- Has proper credentials defined in the "useropts" file (see the topic on Setting up options for using the user interfaces in the chapter on "Configuring the job environment" in the *User's Guide and Reference*) to connect to the master domain manager through WebSphere Application Server Liberty
- Is authorized to perform submit commands in the security file stored on the master domain manager

Syntax

{**submit file** = | **sbf**} "*filename*"
  [**;alias**[**=***name*]]
  [**;into=**[*[folder/]workstation***#**]{*jobstream_id*
;schedid |[*folder/*]*jobstreamname*([*hhmm*[*date*]])}]
  [**;***joboption*[**;**...]]
  [**;noask**]

Arguments

filename

Specifies the name of the file, up to 255 characters. Wildcard characters are permitted. The name must be enclosed in quotes (") if it contains characters other than alphanumeric characters, dashes (-), slashes (/), and underscores (_). See the examples.

alias=*name*

Specifies a unique name to be assigned to the job. If you enter the **alias** keyword without specifying a name, a name is constructed using up to the first six alphanumeric characters (in upper case) of the file name, depending on the number of characters in the file name, followed by a ten digit random number. For example, if the file name is `jclttx5`, the generated name will be similar to `JCLTTX0123456789`.

If you do not include **alias**, a filename is constructed using up to 255 alphanumeric characters of the file's base name, in upper case.

In either of the above cases, if the file name does not start with a letter, you are prompted to use **alias=** *name*.

If you submit a file a second time from the same workstation, the **alias** keyword is mandatory and must be unique for each file submission.

into=[*folder/*]*jobstream_instance*

Identifies the job stream instance, and optionally the folder in which it is defined, into which the job will be placed for launching. If [*folder/*] is omitted, then the root folder is assumed. Select the job stream instance as follows:

[*[folder/]workstation***#**][*folder/*]*jobstreamname*([*hhmm*[*date*]])

or

[*[folder/]workstation***#**]*jobstream_id* **;schedid**

If **into** is not used, the job is added to a job stream named **JOBS**.

joboption

Specify one of the following:

at=*hhmm* [**timezone**|**tz** *tzname*] [**+***n* **days** | *mm/dd[/yy]*] | [**absolute** | **abs**]

confirmed

critical

deadline=*time*[**timezone** | **tz** *tzname*][**+***n* **days** | *mm/dd[/yy]*]

every=*rate*

follows=[*netagent*::][*workstation***#**]{*jobstreamname*[*hhmm* [*mm/dd[/yy]*]][*.job* | **@**] | *jobstream_id.job***;schedid**}| *job*[**,**...] [**if** '*condition_name*[| *condition_name*][| ...]']

Note: Internetwork dependencies do not support folders, therefore, the network agent workstation, and the jobs and job streams running on them, cannot be defined in a folder different from the root (/). Folders are supported on all other workstation types as follows:

```
[follows { [[folder/]workstation# ] [folder/ ]jobstreamname [ .jobname ]
```

follows=[[*folder/*]*workstation*#][*folder/*]{*jobstreamname*[*hhmm* [*mm/dd[/yy]*]][*.job* | @] | *jobstream_id.job*;**schedid**}| *job*[,...] [**if** '*condition_name*[| *condition_name*][| ...]']

The *condition_name* variable indicates the name of the condition defined in the job definition. Conditions can be **status conditions**, based on job status, or other **output conditions**, based on a mapping expression such as a return code, output variables, or output found in a job log. In each follows statement, you can specify only one dependency type: either status or output conditions. At submission time, you can add status or output conditions, but no joined dependencies.

wait

The time in seconds HCL Workload Automation waits before performing a second check on the dependency if the object specified in the dependency does not exist. If the object specified in the dependency was created during the **wait** interval, the submission is performed, otherwise the job is not submitted.

nocheck

Ignores the dependency. If the object specified in the dependency does not exist, the submission is performed anyway.

Note: The **;nocheck** argument is not supported in internetwork dependencies.

interactive

Note: This keyword can be used in Windows® environments only.

logon=*user*

maxdur=*time*[**onmaxdur** *action*]

mindur=*time*[**onmindur** *action*]

needs=[*num*] [*[folder/]workstation*#][*folder/]resource*[,...]

opens=[*[folder/]workstation*#]"*filename*"[(*qualifier*)][,...]

priority=[*pri* | **hi** | **go**]

prompt="[**:** | **!**]*text*" | *[folder/]promptname*[,...]

recovery=stop | **continue** | **rerun**

recoveryjob=[*[folder/]workstation*#][*folder/*]*jobname*

The name of a recovery job, and optionally the folder containing the recovery job, different from the job (if present) specified in the job definition in the database.

after [*[folder/]workstation#*][*folder/*]*jobname*

abendprompt "*text*"

until *time* [**timezone**|**tz** *tzname*][**+***n* **day**[**s**] | [**absolute** | **abs**]] [**;onuntil** *action*]

noask

Specifies not to prompt for confirmation before taking action against each qualifying file.

Using local parameters

You can use local parameters as values with the following keywords:

- `opens`
- `logon`
- `prompt`
- `abendprompt`

Local parameters are defined and managed with the "parms" utility command (see the topic on "parms" in the chapter on "Using the utility commands" in *User's Guide and Reference*) in a local database on the workstation where the job is run. The parameters are resolved on the workstation while the submit command is running.

Comments

Jobs submitted in production from the **conman** command line are not included in the preproduction plan and so they cannot be taken into account when identifying external follows dependencies predecessors.

If you do not specify a workstation with `follows`, `needs`, `opens`, or `into`, the default is the workstation on which **conman** is running.

The scheduler classifies follows dependencies as *internal* when they are specified only by their job name within the job stream. It classifies them as *external* when they are specified in the *workstationName#[folder/]jobStreamName.jobName* format.

When you submit the object into a job stream and add a follows dependency that shares the same job stream name (for example, you submit the object into job stream `schedA` and define a follows dependency on `schedA.job2`), the dependency is treated as an *external* follows dependency. Because the scheduler uses the `sameday` matching criteria to resolve external dependencies, dependencies originated in this way are never added the first time the object is submitted.

Example

Examples

To submit a file into the job stream `jobs` (the job name is `myjcl`), run the following command:

```
submit file = d:\jobs\lib\daily\myjcl
```

where the `;into` sequence was omitted.

To submit a file, with a job name of `misjob4`, into the job stream `missked` located in the root folder, run the following command:

```
sbf /usr/lib/mis/jcl4;alias=misjob4;into=missked ;needs=2 slots
```

The job needs two units of the `slots` resource.

To submit a file, with a job name of `misjob4`, into the job stream `missked` that is located in the `europe` folder, run the following command:

```
sbf /usr/lib/mis/jcl4;alias=misjob4;into=/europe/missked
```

To submit all files that have names beginning with `back` into the job stream `bkup`, run the following command:

```
sbf "/usr/lib/backup/back@";into=bkup
```

To submit file `tws_env.cmd`, whose path contains a blank, on a Windows workstation run:

- In interactive mode:

  ```
  sbf "\"C:\Program Files\HCL\TWS\lucaMDM\tws_env.cmd\"";alias=MYJOB
  ```

  Being in Windows, the double quotation marks (") must be escaped by the `"\` character sequence.
- In command line mode:

  ```
  conman sbf "\"\\\"C:\Program Files\HCL\TWS\lucaMDM\tws_env.cmd\\\"\"";alias=MYJOB
  ```

  Being in Windows, and running the command externally from the conman environment, the escape sequence becomes longer.

where `"\"` is the escape character for the blank in the file path.

SUBMIT JOB

Submits a job to be launched.

To run this command, in the security file you must have ***submit*** (***submitdb***) access for the job with the name specified in its database definition and, if you use the ***alias*** keyword, also with the name specified with this keyword. In addition, if you use the ***recoveryjob*** keyword, you must have ***submit*** access for the job specified with that keyword.

Note that if you have security ***submitdb*** rights only, you are limited to submit jobs defined in the database. You cannot submit ad-hoc jobs.

To include `needs` and `prompt` dependencies, you must have ***use*** access to the resources and global prompts.

If you submit the job from a workstation other than the master domain manager, you must be connecting as a user that:

- Has proper credentials defined in the "useropts" file (see the topic on Setting up options for using the user interfaces in the chapter on "Configuring the job environment" in the *User's Guide and Reference*) to connect to the master domain manager through WebSphere Application Server Liberty
- Is authorized to perform **submit** commands in the security file stored on the master domain manager

If you submit a shadow job, see the section about defining and managing cross dependencies in *User's Guide and Reference* for more details.

Syntax

{**submit job** = | **sbj** = } [*workstation#*][*folder/*]*jobname*
[**;alias**[**=***name*]]
[**;into=**[*[folder/]workstation***#**]{*jobstream_id*
;schedid | [*folder/*]*jobstreamname*([*hhmm*[*date*]])}]
[**;***joboption*[**;**...]]
[**;vartable=***tablename*]
[**;noask**]

Arguments

[folder/]workstation

Specifies the name of the workstation on which the job will be launched. Wildcard characters are permitted, in which case, the job is launched on all qualifying workstations. The default is the workstation on which `conman` is running. You cannot specify a domain or workstation class.

[folder/]jobname

Specifies the name of the job, and optionally, the name of the folder in which it is included. Wildcard characters are permitted, in which case, all qualifying jobs are submitted. If the job is already in the production plan, and is being submitted into the same job stream, you must use the **alias** argument to assign a unique name.

alias=*name*

Specifies a unique name to be assigned to the job in place of *jobname*. If you enter the **alias** keyword without specifying a name, a name is constructed using the first five alphanumeric characters of *jobname* followed by a ten digit random number. The name is always upshifted. For example, if *jobname* is `jcrttx5`, the generated name will be similar to `JCRTT1234567890`.

into=[*folder/*]*jobstream_instance*

Identifies the job stream instance, and optionally the folder in which it is defined, into which the job will be placed for launching. When a job definition defined in a folder is submitted into a job stream, then in the plan, the folder associated to the job becomes that of the job stream. The folder in which the job definition was originally defined is no longer considered. If a job stream instance is not specified, then the default job stream, JOBS, is used. The folder associated to the JOBS job stream is always the root (/). Select the job stream instance as follows:

[*[folder/]workstation***#**][*folder/*]*jobstreamname*([*hhmm*[*date*]])

or

[*[folder/]workstation*#]*jobstream_id* **;schedid**

If **into** is not used, the job is added to a job stream named **JOBS**.

joboption

Specify one of the following:

at=*hhmm* [**timezone**|**tz** *tzname*] [**+***n* **days** | *mm/dd[/yy]*] | [**absolute** | **abs**]

confirmed

critical

deadline=*time*[**timezone** | **tz** *tzname*][**+***n* **days** | *mm/dd[/yy]*]

every=*rate*

follows=[*netagent*::][*workstation*#]{*jobstreamname*[*hhmm* [*mm/dd[/yy]*]][.*job* | @] | *jobstream_id.job***;schedid**}| *job*[,...] [**if** '*condition_name*[| *condition_name*][| ...]']

Note: Internetwork dependencies do not support folders, therefore, the network agent workstation, and the jobs and job streams running on them, cannot be defined in a folder different from the root (/). Folders are supported on all other workstation types as follows:

```
[follows {[[folder/]workstation#][folder/]jobstreamname[.jobname]
```

follows=[[*folder/*]*workstation*#][*folder/*]{*jobstreamname*[*hhmm* [*mm/dd[/yy]*]][.*job* | @] | *jobstream_id.job***;schedid**}| *job*[,...] [**if** '*condition_name*[| *condition_name*][| ...]']

The *condition_name* variable indicates the name of the condition defined in the job definition. Conditions can be **status conditions**, based on job status, or other **output conditions**, based on a mapping expression such as a return code, output variables, or output found in a job log. In each follows statement, you can specify only one dependency type: either status or output conditions. At submission time, you can add status or output conditions, but no joined dependencies.

wait

The time in seconds HCL Workload Automation waits before performing a second check on the dependency if the object specified in the dependency does not exist. If the object specified in the dependency was created during the **wait** interval, the submission is performed, otherwise the job is not submitted.

nocheck

Ignores the dependency. If the object specified in the dependency does not exist, the submission is performed anyway.

Note: The **;nocheck** argument is not supported in internetwork dependencies.

maxdur=*time*[**onmaxdur** *action*]

mindur=*time*[**onmindur** *action*]

needs=[*num*] [*[folder/]workstation#][folder/]resource*[**,**...]

opens=[*[folder/]workstation#*]"*filename*"[**(***qualifier***)**][**,**...]

priority=[*pri* | **hi** | **go**]

prompt="[**:** | **!**]*text*" | *promptname*[**,**...]

recovery=stop | **continue** | **rerun**

recoveryjob= [*[folder/]workstation#*][*folder/]jobname*

The name of a recovery job, and optionally the folder within which it is contained, different from the job (if present) specified in the job definition in the database.

after [*[folder/]workstation#*][*folder/]jobname*

abendprompt "*text*"

until *time* [**timezone**|**tz** *tzname*][**+***n* **day**[**s**] | [**absolute** | **abs**]] [**;onuntil** *action*]

vartable=*tablename*

Specifies the name of the "variable table", if different than the default one, where the "variables" you intend to use are defined. For more information about "variable tables" and "variables", see the topic on defining scheduling objects in the chapter on "Defining objects in the database" in "*User's Guide and Reference*".

Remember:

- With this command, you can use variable substitution for the following keywords:
  - `opens`
  - `prompt`
  - `abendprompt`
- Enclose the variable between carets (^), and then enclose the entire string between quotation marks. If the variable contains a portion of a path, ensure that the caret characters are not immediately preceded by a backslash (\) because, in that case, the `\^` sequence could be



wrongly interpreted as an escape sequence and resolved by the parser as caret character. If necessary, move the backslash into the definition of the variable between carets.
- Variables specified in the job definition with the ${variablename} format are not resolved. For more information, see the section about variable and parameter definition in *User's Guide and Reference*.

If you submit a job containing variables defined in a variable table that is not the default variable table and you do not specify the variable table in the run-cycle, job stream, or workstation, the variables are not resolved. See the section about customizing your workload using variable tables in *User's Guide and Reference*.

noask

Specifies not to prompt for confirmation before taking action against each qualifying job.

Comments

Jobs submitted in production from the **conman** command line are not included in the preproduction plan and so they cannot be taken into account when identifying external follows dependencies predecessors.

If you do not specify a workstation with `follows`, `needs`, `opens`, or `into`, the default is the workstation of the job.

at specifies at which time the job can be submitted. If the **at** keyword is used, then the job cannot start before the time set with this keyword. Note that if the master domain manager of your network runs with the `enLegacyStartOfDayEvaluation` and `enTimeZone` options set to `yes` to convert the `startOfDay` time set on the master domain manager to the local time zone set on each workstation across the network, you must add the **absolute** keyword to make it work.

The scheduler classifies follows dependencies as *internal* when they are specified only by their job name within the job stream. It classifies them as *external* when they are specified in the *workstation#[folder/] jobstreamName.jobName* format.

When you submit the object into a job stream and add a follows dependency that shares the same job stream name (for example, you submit the object into job stream `schedA` and define a follows dependency on `schedA.job2`), the dependency is treated as an *external* follows dependency. Because the scheduler uses the `sameday` matching criteria to resolve external dependencies, dependencies originated in this way are never added the first time the object is submitted.

Example

Examples

To submit the `test` jobs into the job stream `JOBS`, run the following command:

```
sbj = test
```

To submit a job with an alias of `rptx4` and place the job in the job stream `reports` with an **at** time of 5:30 p.m., run the following command:

```
sbj = rjob4;alias=rptx4;into=reports;at=1730
```

To submit the `test` job in the job stream `ORDERS` that is found in the folder named `ONLINE`, run the following command:

```
sbj = test;into=/ONLINE/orders
```

To submit job `txjob3` on all workstations whose names begin with `site`, run the following command:

```
sbj = site@#txjob3;alias
```

See also

From the Dynamic Workload Console you can perform the same task as described in:

the *User's Guide and Reference*, section about Submitting predefined jobs.

SUBMIT SCHED

Submits a job stream for processing.

To run this command, in the security file you must have ***submit*** access for the job stream with the name specified in its database definition and, if you use the ***alias*** keyword, also with the name specified with this keyword. To include `needs` and `prompt` dependencies, you must have ***use*** access to the resources and global prompts.

The `submit schedule` command uses the credentials set in the `useropts` file belonging to the *TWS_user* who installed that workstation.

If you submit the job stream from a workstation other than the master domain manager, you must be connecting as a user that:

- has proper credentials defined in the "useropts" file (see the topic on Setting up options for using the user interfaces in the chapter on "Configuring the job environment" in the *User's Guide and Reference*) to connect to the master domain manager through WebSphere Application Server Liberty
- is authorized to perform submit commands in the security file stored on the master domain manager

Syntax

{**submit sched** = | **sbs** = } [*[folder/]workstation#*][*folder/*]*jstreamname*
[**;alias**[**=***name*]]
[**;***jstreamoption*[**;**...]]
[**;vartable=***tablename*]
[**;noask**]

Arguments

[folder/]workstation

Specifies the name of the workstation on which the job stream will be launched. Wildcard characters are permitted, in which case, the job stream is launched on all qualifying workstations. The default is the workstation on which **conman** is running. You cannot specify a domain or workstation class.

[*folder/*]

Specifies the name of the folder in which the job stream is defined.

jstreamname

Specifies the name of the job stream. Wildcard characters are permitted, in which case, all qualifying job streams are submitted. If the job stream is already in the production plan, you must use the **alias** argument to assign a unique name.

alias=*name*

Specifies a unique name to be assigned to the job stream in place of *jstreamname*. If set, this value corresponds also to the *jobstream_id*. If you enter the **alias** keyword without specifying a name, a name is constructed using the first five alphanumeric characters of *jstreamname* followed by a ten digit random number. The name is always upshifted. For example, if *jstreamname* is `sttrom`, the generated name will be similar to `STTRO1234567890`.

The authorization to submit the schedule is checked in the Security file using the original name not the alias name.

jstreamoption

Enter any of the following (refer to the topic on the Job stream definition keywords in the chapter on "Defining objects in the database" in *User's Guide and Reference* to find which options are mutually exclusive):

[at=*hhmm* [timezone|tz *tzname*] [+*n* days | *date*] [absolute | abs]] | [schedtime=[*hhmm* [*date*] | [+*n* days]]

where:

at specifies at which time the job stream can be launched. If the "at" keyword is used, then the job stream cannot start before the time set with this keyword . Note that if the master domain manager of your network runs with the `enLegacyStartOfDayEvaluation` and `enTimeZone` options set to `yes` to convert the `startOfDay` time set on the master domain manager to the local time zone set on each workstation across the network, you must add the **absolute** keyword to make it work.

schedtime represents the day and time when the job stream is positioned in the plan. If by this time the job stream is free from dependencies, and has no defined **at** time restrictions, it is launched. The value assigned to **schedtime** does not represent a dependency for the job stream. Its value is then displayed in the *SchedTime* columns in the output of the show commands. If an **at** restriction is defined, then the value assigned to **schedtime** is overwritten by the **at** value. When the job stream actually starts, the value assigned to **schedtime** is overwritten by the actual start time of the job stream.

The format used for *date* depends on the value assigned to the ***date format*** variable specified in the `localopts` file.

If no additional time zone is specified, the time zone set on the workstation running the command is assumed.

carryforward

Makes a job stream eligible to be carried forward to the next production plan if it is not completed before the end of the current production plan.

deadline=*time*[timezone | tz *tzname*][+*n* days | *date*]

If no additional time zone is specified, the time zone set on the workstation running the command is assumed.

follows=[*netagent*::][*workstation*#][*folder/*]{*jobstreamname*(*hhmm* [*mmdd[/yy]*]) [.*job* | @] | *jobstream_id*.*job*;schedid}| *job* '[IF *condition_name*[| *condition_name*][| ...]]' [;nocheck][;wait=*time*][,...]

The matching criteria used when submitting job streams in production is different from the way **follows** dependencies are resolved in the preproduction plan. When a job stream, for example `JS_A`, containing a **follows** dependency from a job or a job stream, for example `JS_B`, is submitted from the **conman** command line program, the predecessor instance of `JS_B` is defined following this criterion:

1. The closest instance of `JS_B` preceding `JS_A`.
2. If no preceding instance of `JS_B` exists, then the predecessor instance is the closest instance of `JS_B` following `JS_A`.
3. Otherwise an error is displayed and the command fails if the **;nocheck** keyword is not used.

The predecessor job stream instance is searched among the instances added to the production plan when **JnextPlan** was run and the instances submitted in production with the **sbs** command, including those submitted with an alias.

follows=[*netagent***::**][*workstation***#**]{*jobstreamname*[*hhmm* [*mm/dd[/yy]*]][.*job* | @] | *jobstream_id.job***;schedid**}| *job*[**,**...] [**if** '*condition_name*[| *condition_name*][| ...]']

Note: Internetwork dependencies do not support folders, therefore, the network agent workstation, and the jobs and job streams running on them, cannot be defined in a folder different from the root (/). Folders are supported on all other workstation types as follows:

```
[follows {[[folder/]workstation#][folder/]jobstreamname[.jobname]
```

follows=[[*folder/*]*workstation***#**][*folder/*]{*jobstreamname*[*hhmm* [*mm/dd[/yy]*]][.*job* | @] | *jobstream_id.job***;schedid**}| *job*[**,**...] [**if** '*condition_name*[| *condition_name*][| ...]']

The *condition_name* variable indicates the name of the condition defined in the job definition. Conditions can be **status conditions**, based on job status, or other **output conditions**, based on a mapping expression such as a return code, output variables, or output found in a job log. In each follows statement, you can specify only one dependency type: either status or output conditions. At submission time, you can add status or output conditions, but no joined dependencies.

wait

The time in seconds HCL Workload Automation waits before performing a second check on the dependency if the object specified in the dependency does not exist.

If the object specified in the dependency was created during the **wait** interval, the submission is performed, otherwise the job is not submitted.

nocheck

Ignores the dependency. If the object specified in the dependency does not exist, the submission is performed anyway.

Note: The **;nocheck** argument is not supported in internetwork dependencies.

limit=*joblimit*

Limits the number of jobs that can run simultaneously in a job stream on the same CPU.

needs=[*num*] [*[folder/]workstation#*][*[folder/]resource*[,...]

Defines resources that must be available before a job or job stream is launched. You can use the needs keyword either in a job stream definition or in the definition of the contained jobs, not in both.

opens=[*[folder/]workstation#*]"*filename*"[(*qualifier*)][,...]

Specifies files that must be available before a job or job stream can be launched.

priority=[*pri* | hi | go]

Sets the priority of a job or job stream. By assigning a different priority to jobs or job streams you determine which one starts first, if the dependencies are solved.

prompt="[: | !]*text*" | *promptname*[,...]

Specifies prompts that must be answered affirmatively before a job or job stream is launched.

until *time* [timezone|tz *tzname*][+*n* day[s] | [absolute | abs]] [;onuntil *action*]

Depending on the object definition the until keyword belongs to, specifies the latest time a job stream must be completed or the latest time a job can be launched.

If no additional time zone is specified, the time zone set on the workstation running the command is assumed.

vartable=*tablename*

Specifies the name of the "variable table", if different than the default one, where the "variables" you intend to use are defined. For more information about "variable tables" and "variables", see the topic on defining scheduling objects in the chapter on "Defining objects in the database" in *User's Guide and Reference*.

Remember:

- With this command, you can use variable substitution for the following keywords:
  - `opens`
  - `prompt`



- Enclose the variable between carets (^), and then enclose the entire string between quotation marks. If the variable contains a portion of a path, ensure that the caret characters are not immediately preceded by a backslash (\) because, in that case, the `\^` sequence could be wrongly interpreted as an escape sequence and resolved by the parser as caret character. If necessary, move the backslash into the definition of the variable between carets.

If you submit a job stream with jobs containing variables defined in a variable table that is not the default variable table and you do not specify the variable table in the run-cycle, job stream, or workstation, the variables are not resolved. See the section about customizing your workload using variable tables in *User's Guide and Reference*.

noask

Specifies not to prompt for confirmation before taking action against each qualifying job stream.

Comments

Job streams submitted in production from the **conman** command line are not included in the preproduction plan and so they cannot be taken into account when identifying external follows dependencies predecessors.

If you do not specify a workstation with `follows`, `needs`, or `opens`, the default is the workstation of the job stream.

The scheduler classifies follows dependencies as *internal* when they are specified only by their job name within the job stream. It classifies them as *external* when they are specified in the *workstationName#[folder/]jobStreamName.jobName* format.

When you submit a job stream that includes a job with a follows dependency that shares the same job stream name (for example, job stream `schedA` includes a job named `job6` that has a follows dependency on `schedA.job2`), the dependency is added as an *external* follows dependency. Because the scheduler uses the `sameday` matching criteria to resolve external dependencies, dependencies originated in this way are never added the first time the object is submitted.

Example

Examples

To submit the `adhoc` job stream included in the `payroll` folder on workstation `site1` stored in folder `myfolder`, and flags it as a **carryforward** job stream, run the following command:

```
submit sched = myfolder/site1#PAYROLL/adhoc;carryforward
```

To submit job stream `fox4` with a job limit of **2**, a priority of **23**, and an **until** time of midnight, run the following command:

```
sbs = fox4;limit=2;pri=23;until=0000
```

To submit job stream `sched3` on all workstations with names that start with `site`, run the following command:

```
sbs = site@#sched3
```

See also

From the Dynamic Workload Console you can perform the same task as described in:

the *Dynamic Workload Console User's Guide*, section about Submitting predefined job streams.

SWITCHEVENTPROCESSOR

Switches the event processing server from the master domain manager to the backup master or vice versa.

Note that you can run the event processing server also on a workstation installed as a backup master that runs as a plain fault-tolerant agent.

Syntax

{**switcheventprocessor** | **switchevtp**} *[folder/]workstation*

Arguments

[folder/]workstation

Specifies the name of the master domain manager or of the backup master domain manager where you want to switch the event processing server. Wildcard characters are not permitted.

Comments

If you issue the command from a workstation other than the one where the event processor is configured, the command uses the command-line client, so the user credentials for the command-line client must be set correctly.

In case of backup masters the workstation must have the `full-status` attribute set to `on`.

Permission to `start` and `stop` actions on `cpu` objects is required in the security file to be enabled to run this command.

The correlation state of pending correlation rule instances is lost whenever the server is turned off or migrated. If caching of received events is enabled in the configuration file of the EIF listener, the cached events are lost after the event processor is switched.

Important:

- Before running this command, run planman deploy as a precaution. Do this to make sure that your latest changes or additions to active event rules are deployed before the event processor is switched and so avoid the risk that, because of a time mismatch, the latest updates (sent automatically based on the setup of the deploymentFrequency global option) are received by the old event processor instead of the new one.
- The master and backup masters designated to run the event processor should have their clocks synchronized at all times to avoid inconsistencies in the calculation of the time interval of running event

rules. In fact, if the event processor is switched to a not-synchronized computer, timeout actions in the process of being triggered might undergo unexpected delays. Use a Network Time Protocol (NTP) server to keep all clocks synchronized.

See also

From the Dynamic Workload Console you can perform the same task as follows:

1. In the navigation bar at the top, click **Monitoring and Reporting > Orchestration Monitor**.
2. Select an engine.
3. From the drop-down menu, select **Workstation**.
4. From the **Query** drop-down list, select a query to monitor workstations.
5. Click **Run** to run the monitoring task.
6. From the table containing the list of workstations, select a workstation and click **More Actions > Become Event Processor**.

SWITCHMGR

Switches domain management to and from the following workstation types:

- from a master domain manager to a backup master domain manager.
- from a domain manager to a backup domain manager.
- from a dynamic domain manager to a backup dynamic domain manager.

You must have ***start*** and ***stop*** access to the backup workstation.

The **switchmgr** command must only be used as part of specific procedures for switching domain management capabilities from a domain manager to its backup domain manager either permanently or temporarily. For information about these procedures, see the section about changing a domain manager and changing a master domain manager *Administration Guide*.

Syntax

{**switchmgr** | **switchm**} *domain*;*newmgr*

Arguments

domain

Specifies the domain in which you want to switch managers.

newmgr

Specifies the name of the new domain manager. This must be a workstation in the same domain, and should be defined beforehand as a fault-tolerant agent with Resolve Dependencies and Full Status enabled.

Comments

The command stops a specified workstation and restarts it as the domain manager. All domain member workstations are informed of the switch, and the old domain manager is converted to a fault-tolerant agent in the domain.

The next time `JnextPlan` is run on the old domain manager, the domain acts as though another **switchmgr** command had been run and the old domain manager automatically resumes domain management responsibilities.

Fault-tolerant agents defined with `securitylevel = on` might fail to use the SSL port to connect to the new master domain manager after the switchmgr command is run. In this case do either of the following to let the agent start correctly:

- Unlink and then link the agent from the new master domain manager.
- Use the `securitylevel = force` option on the agent.

Example

Examples

To switch the domain manager to workstation `orca` in the `masterdm` domain, run the following command:

```
switchmgr masterdm;orca
```

To switch the domain manager to workstation `ruby` in the `bldg2` domain, run the following command:

```
switchmgr bldg2;ruby
```

See also

- To use the broker CLI after the switch, define the parameters of the CLIConfig.properties file. For further details, see the section about the command-line configuration file in *User's Guide and Reference*.
- From the Dynamic Workload Console you can perform the same task as follows:
    1. In the navigation bar at the top, click **Monitoring and Reporting > Orchestration Monitor**.
    2. Select an engine.
    3. From the drop-down menu, select **Workstation**.
    4. From the **Query** drop-down list, select a query to monitor workstations.
    5. Click **Run** to run the monitoring task.
    6. From the table containing the list of workstations, select a workstation and click **More Actions > Become Master Domain Manager**.

SYSTEM COMMAND

Runs a system command.

Syntax

[: | **!]** *system-command*

Arguments

system-command

Specifies any valid system command. The prefix (: or !) is required only when a command name has the same spelling as a **conman** command.

Example

Examples

To run a **ps** command in UNIX®, run the following command:

```
ps -ef
```

To run a **dir** command in Windows®, run the following command:

```
dir \bin
```

TELLOP

Sends a message to the HCL Workload Automation console.

Syntax

{**tellop** | **to**} [*text*]

Arguments

text

Specifies the text of the message. The message can contain up to 900 characters.

Comments

If **tellop** is issued on the master domain manager, the message is sent to all linked workstations. If issued on a domain manager, the message is sent to all of the linked agents in its domain and subordinate domains. If issued on a workstation other than a domain manager, the message is sent only to its domain manager if it is linked. The message is displayed only if the console message level is greater than zero. See CONSOLE.

If **tellop** is entered alone, it prompts for the message text. At the prompt, type each line and press the Return key. At the end of the message, type two slashes (//) or a period (.), and press the Return key. You can use the new line sequence (\n) to format messages. Typing **Control+c** at any time will exit the **tellop** command without sending the message.

Example

Examples

To send a message, run the following command:

```
tellop TWS will be stopped at\n4:30 for 15 minutes.
```

To prompt for text before sending a message, run the following command:

```
to
TELLOP>*********************************
TELLOP>*  TWS will be stopped at       *
TELLOP>*  4:30 for 15 minutes.         *
TELLOP>*********************************
TELLOP>//
```

UNLINK

Closes communication links between workstations.

You must have ***unlink*** access to the target workstation.

Syntax

unlink [*domain***!**]*[folder/]workstation*
[;noask]

Arguments

domain

Specifies the name of the domain in which to close links. It is not necessary to specify the domain name of a workstation in the master domain. Wildcard characters are permitted.

Note: You must always specify the domain name when unlinking a workstation not in the master domain.

This argument is useful when unlinking more than one workstation in a domain. For example, to unlink all the agents in domain `stlouis`, use the following command:

```
unlink stlouis!@
```

If you do not specify *domain*, and *workstation* includes wildcard characters, the default domain is the one in which **conman** is running.

[folder/]workstation

Specifies the name of the workstation to be unlinked. Wildcard characters are permitted.

This command is not supported on remote engine workstations.

noask

Specifies not to prompt for confirmation before taking action on each qualifying workstation.

Comments

Assuming that a user has **unlink** access to the workstations being unlinked, the following rules apply:

- A user running **conman** on the master domain manager can unlink any workstation in the network.
- A user running **conman** on a domain manager other than the master can unlink any workstation in its own domain and subordinate domains. The user cannot unlink workstations in peer domains.
- A user running **conman** on an agent can unlink any workstation in its local domain provided that the workstation is either a domain manager or host. A peer agent in the same domain cannot be unlinked.

For additional information see LINK.

Example

Examples

A figure showing a network and a table of the links in it closed by "unlink" commands run by users in various locations in the network can be found in the topic on the "unlink" command in *User's Guide and Reference*.

See also

From the Dynamic Workload Console you can perform the same task as follows:

1. In the navigation bar at the top, click **Monitoring and Reporting > Orchestration Monitor**.
2. Select an engine.
3. From the drop-down menu, select **Workstation**.
4. From the **Query** drop-down list, select a query to monitor workstations.
5. Click **Run** to run the monitoring task.
6. From the table containing the list of workstations, select a workstation and click **Unlink**.

VERSION

Displays the **conman** program banner, inclusive of the version up to the installed fix pack level.

Syntax

{**version** | **v**}

Example

Examples

To display the **conman** program banner, run the following command:

```
%version
```

The output is similar to this:

```
##################################################################
# Licensed Materials - Property of IBM* and HCL**
# 5698-WSH
# (C) Copyright IBM Corp. 1998, 2016 All rights reserved.
# (C) Copyright HCL Technologies Ltd. 2016, 2026 All rights reserved.
```

```
  # * Trademark of International Business Machines
  # ** Trademark of HCL Technologies Limited
  ################################################################
Installed for user "twsuser".
Locale LANG set to the following: "en"
Scheduled for (Exp) 05/20/19 (#8) on LB001542_MASTER.Batchman LIVES.
Limit:55,Fence:0,Audit Level:0
```

Chapter 13. Using Orchestration Query Language

You can use the essential Orchestration Query Language (OQL) keywords and syntax to effectively write queries.

You can monitor the HCL Workload Automation production plan environment, both distributed and z/OS, by using the OQL, which applies to REST API V2 and the Orchestration Monitor of the Dynamic Workload Console. Write queries for items in your database and retrieve required results easily and quickly by using the OQL syntax.

An OQL query begins with an *expression*, which is composed of different *conditions*, known as *queryTerms.* A queryTerm is the minimum condition of a OQL construction, for example jobStreamName = test1.

QueryTerms include three main elements: a *field*, a *comparison_operator*, and a *value*. The fields and the values of the query are case-sensitive.

The field must begin with a letter. After the first letter, the field can contain the following characters:

- **0 - 9**
- **a - z**
- **A - Z**
- **_**
- **-**
- **.**

You can express the *value* in one of the following ways:

- Integers, both positive and negative.
- Strings in single quotation marks (').
- Duration, based on ISO 8601 standard. Duration is expressed in `P[n]Y[n]M[n]DT[n]H[n]M[n]S` format, where `[n]` replaces the value of each time and date element. For example, to express a duration of 4 hours and 20 seconds, the format to use is `PT4H20S`.
- Instant, based on ISO 8601 standard. Instant is expressed in YYYY-MM-DDThh:mm:ss.milliseconds format. For example, to express the instant for December 1st, 2025 at 11 PM, the format to use is 2025-12-01T23:00:00.000Z.
- Identifier (ID) based on Universally Unique Identifier (UUID) standard.

The topic is divided in the following sections:

- OQL syntax keywords and comparison operators on page 733
- OQL syntax fields for workstations on page 734
- OQL syntax fields for job streams on page 737
- OQL syntax fields for jobs on page 745
- OQL syntax fields for resources on page 752
- Query aliases on page 754
- Examples on page 754

OQL syntax keywords and comparison operators

The following table details OQL keywords and comparison operators, both of which are essential for constructing effective queries.

Table 80. OQL syntax keywords

Keywords	Description
AND	Returns the results that satisfy all the conditions separated by **AND**. It is case insensitive.
OR	Returns the results that satisfy at least one of the conditions separated by **OR**. It is case insensitive.
(	Opens a priority expression.
)	Closes a priority expression.
[	Opens a list.
]	Closes a list.
=	Returns all the elements equal to the specified element. Numbers, strings, or one of these keywords can follow the keyword: **true**, **false**, or **null**.
!=	Returns all the elements different from the given one. Numbers, strings, or one of these keywords can follow the keyword: **true**, **false**, or **null**.
<=	Returns all the elements that are less than or equal to the specified keyword. Only numbers or strings can follow.
>=	Returns all the elements that are greater than or equal to the specified keyword. Only numbers or strings can follow.
<	Returns all the elements that are less than the specified keyword. Only numbers or strings can follow.
>	Returns all the elements that are greater than the specified keyword. Only numbers or strings can follow.
IN	Returns items that the specified list includes.
NOT IN	Returns items that the specified list does not include.
LIKE	Returns elements that match the specified pattern. The accepted characters are as follows: • **@**: matches all characters. • **?**: matches a single character in a specific position.
NOT LIKE	Returns elements that do not match the specified pattern. The accepted characters are as follows:

Table 80. OQL syntax keywords (continued)

Keywords	Description
	• **@**: matches all characters. • **?**: matches a single character in a specific position.
ORDER BY	Orders the query results according to the specified fields. You can use the keyword at the end of the construction. By default, the field is sorted in ascending order. The accepted values are as follows: • **ASC**: Sorts the results in ascending order. • **DESC**: Sorts the results in descending order.
,	Separates values inside a list.
.	Specifies a subfield of an item.
true	Represents the Boolean value `true`.
false	Represents the Boolean value `false`.
null	Represents the `null` value.

OQL syntax fields for workstations

The table below provides a comprehensive list and description of OQL fields that apply to workstations.

Note: OQL fields are case-sensitive; therefore, adherence to the exact capitalization and spelling is mandatory.

Unless specified, the fields and the values listed in the table apply both to distributed and z/OS environments.

Table 81. OQL syntax fields for workstation

Field name	Description

activeFlags z/OS	The flags of the workstation. Supported values are:
	`AUTOMATION|WAIT|VIRTUAL|JOB_SETUP|WTO|zCENTRIC|DYNAMIC|STARTED_TASK|PENDING_OFFLINE|CONTROL_ON_SERVERS`
	Example:
	`activeFlags LIKE '@'`

activeStates	The statuses of the workstation. Supported values are: Distributed `LINKED|UNLINKED|FULLY_LINKED|NOT_APPLICABLE`

Table 81. OQL syntax fields for workstation (continued)

Field name	Description

	z/OS
	`FULLY_LINKED|LINKED|UNLINKED|WS_STATUS_ACTIVE|WS_STATUS_OFF|WS_STATUS_FAILED|WS_STATUS_UNKNOWN`
	Example:
	`activeStates IN ['LINKED', 'FULLY_LINKED']`

available z/OS	The workstation is flagged as available. Supported values are `true` or `false`. Example: `available IN ['true']`
description z/OS	The description of the workstation. Example: `description IN ['General Workstation']`
domainName Distributed	The name of the domain within which the workstation is hierarchically organized. Example: `domainName != 'MASTERDM'`
folder Distributed	The name of the folder where the workstation is saved. Example: `folder LIKE '/@A/'`
id	The identifier of the workstation. Example: `id = 'ca15f24a-0e6b-3f1b-9251-854d8d0d3639'`
jobmanUp Distributed	Identifies whether the agent is running or not. Supported values are `true` or `false`. Example: `jobmanUp = true`
name	The name of the workstation. Example: `name = 'wksrome'`

Table 81. OQL syntax fields for workstation (continued)

Field name	Description
remoteEngineType z/OS	The type of remote engine on which the current remote workstation is defined. Supported values are `Z` for z/OS engines, and `D` for distributed engines. Example: `remoteEngineType LIKE '@`

reporting z/OS	The type of reporting. Supported values are:
	`AUTOMATIC|MANUAL_START_STOP|MANUAL_COMPLETION_ONLY|NON_REPORTING`
	Example:
	`reporting LIKE 'AUTOMATIC'`

resource1Info.usedForControl z/OS	The job must be submitted only if the requested resource amount is available. Supported values are `true` or `false`. Example: `resource1Info.usedForControl IN ['true']`
resource2Info.usedForControl z/OS	The job must be submitted only if the requested resource amount is available. Supported values are `true` or `false`. Example: `resource2Info.usedForControl IN ['false']`

resource2Info.usage z/OS	The type of usage defined for resource 2. Supported values are:
	`PLANNING|CONTROL|PLANNING_AND_CONTROL|NEITHER_PLANNING_NOR_CONTROL`
	Example:
	`resource2info.usage IN ['NEITHER_PLANNING_NOR_CONTROL']`

type	The type of workstation. Supported values are:
	Distributed
	`AGENT|POOL|DYNAMIC_POOL|REMOTE_ENGINE|MANAGER|FTA|XA|SA|WORKLOAD_BROKER|DOMAIN_MANAGER`
	z/OS
	`GENERAL|REM_ENG|COMPUTER|PRINTER`

Table 81. OQL syntax fields for workstation (continued)

Field name	Description
	Example: `type IN ['FTA']`

OQL syntax fields for job streams

The table below provides a comprehensive list and description of OQL fields that apply to job streams.

Note: OQL fields are case-sensitive; therefore, adherence to the exact capitalization and spelling is mandatory.

Unless specified, the fields and the values listed in the table apply both to distributed and z/OS environments.

Table 82. OQL syntax fields for job stream

Field name	Description
aliasJobStreamName Distributed	The alias name of the job stream. Example: `aliasJobStreamName LIKE '@rome'`

commonStatus	The external status of the job stream in plan. The supported values are:
	Distributed
	`WAITING|READY|RUNNING|SUCCESSFUL|ERROR|CANCELED|HELD|UNDECIDED|BLOCKED|SUPPRESS`
	z/OS
	`WAITING|READY|RUNNING|SUCCESSFUL|ERROR|CANCELED|HELD|UNDECIDED|SUPPRESS`
	Example:
	`commonStatus IN ['ERROR','BLOCKED']`

compositeStatus.status	The internal status of the job stream in plan. Supported values are:
	Distributed
	`HOLD|READY|FENCE|INTRO|FAIL|EXEC|WAIT|SUPPR|END|SUCC|SUCCP|ABEND|CANCL|CANCLP|ABENP|PEND|SUSP|STUCK|EXTRN`
	z/OS
	`WAIT|COMPLETE|ERROR|STARTED|UNDECIDED|DELETED`

Table 82. OQL syntax fields for job stream (continued)

Field name	Description
	Example: `compositeStatus.status NOT IN ['SUCC','ABEND','FAIL']`

dependencies.dependencyStatus Distributed	The status of the job stream dependency. Supported values are:
	`SATISFIED|UNSATISFIED|UNDECIDED`
	Example:
	`dependencies.dependencyStatus = 'SATISFIED'`

dependencies.jobId Distributed	The identifier of the job instance on which the job stream depends. Example: `dependencies.jobId = 'ca15f24a-0e6b-3f1b-9251-854d8d0d3639'`
dependencies.folderOriginalName Distributed	The name of the folder where the job dependency is saved. Example: `dependencies.folderOriginalName NOT LIKE '@JS_@'`
dependencies.jobStreamName Distributed	For dependencies on jobs, the name of the job stream to which the job belongs. For dependencies on job streams, the name of the job stream. Example: `dependencies.jobStreamName LIKE '?a'`
dependencies.jobName Distributed	The name of the job on which the job stream depends. Example: `dependencies.jobName LIKE '@job1'`
dependencies.jobStreamFolder Distributed	For dependencies on jobs, the name of the folder of the job stream to which the job belongs. For dependencies on job streams, the name of the folder of the dependent job stream. Example:

Table 82. OQL syntax fields for job stream (continued)

Field name	Description
	`dependencies.jobStreamFolder NOT LIKE '/@root/'`
dependencies.jobStreamId Distributed	For dependencies on jobs, the id of the job stream to which the job belongs. For dependencies on job streams, the id of the dependent job stream. Example: `dependencies.jobStreamId = 'ca15f24a-0e6b-3f1b-9251-854d8d0d3639'`
dependencies.jobStreamName Distributed	For dependencies on jobs, the name of the job stream to which the job belongs. For dependencies on job streams, the name of the dependent job stream. Example: `dependencies.jobStreamName LIKE '@js_exec'`
dependencies.jobStreamWorkstation Distributed	For dependencies on jobs, the name of the workstation of the job stream to which the job belongs. For dependencies on job streams, the name of the workstation of the dependent job stream. Example: `dependencies.jobStreamWorkstation LIKE '@wks_rome@'`
dependenciesStats.numberOfDependencies Distributed	The number of dependencies of the job stream. Supported operators are `=` and `!=`. Example: `dependenciesStats.numberOfDependencies = '3'`
dependenciesStats.numberOfFileDependencies Distributed	The number of dependencies of the job stream of type `File`. Supported operators are `=` and `!=`. Example: `dependenciesStats.numberOfFileDependencies = '1'`

Table 82. OQL syntax fields for job stream (continued)

Field name	Description
dependenciesStats.numberOfInternetworkDependencies Distributed	Number of dependencies of the job stream of type `Internetwork dependency`. Supported operators are `=` and `!=`. Example: `dependenciesStats.numberOfInternetworkDependencies != '10'`
dependenciesStats.numberOfJobDependencies Distributed	The number of job dependencies of the job stream. Supported operators are `=` and `!=`. Example: `dependenciesStats.numberOfJobDependencies != '7'`
dependenciesStats.numberOfJobStreamDependencies Distributed	The number of job stream dependencies of the job stream. Supported operators are `=` and `!=`. Example: `dependenciesStats.numberOfJobStreamDependencies = '4'`
dependenciesStats.numberOfPromptDependencies Distributed	The number of dependencies of the job stream of type `Prompt`. Supported operators are `=` and `!=`. Example: `dependenciesStats.numberOfPromptDependencies != '9'`
dependenciesStats.numberOfResourceDependencies Distributed	The number of dependencies of the job stream of type `Resource`. Supported operators are `=` and `!=`. Example: `dependenciesStats.numberOfResourceDependencies = '3'`
dependenciesStats.numberOfUnresolvedDependencies Distributed	The number of dependencies of the job stream that have not been satisfied. Supported operators are `=` and `!=`. Example: `dependenciesStats.numberOfUnresolvedDependencies != '8'`
description Distributed	The information contained in the job stream description field at the moment the job stream enters the plan. Example: `description in ['recoveryteam', 'supportcontact']`

Table 82. OQL syntax fields for job stream (continued)

Field name	Description
folder Distributed	The name of the folder within which the job stream is saved. Example: `folder ='/JSFOLDER'`
folderId Distributed	The identifier of the folder within which the job stream is saved. Example: `folderId = 'be11g38a-0u7m-5f1d-5594-458d8d0d6541'`
id Distributed	The identifier of the job stream. Example: `id = 'ca15f24a-0e6b-3f1b-9251-854d8d0d3639'`
jobs.dependencies.jobStreamId Distributed	The identifier of the job stream to which the dependency of the job belongs. Example: `jobs.dependencies.jobStreamId = 'ca15f26a-0e9b-3cpl-5594-999d8d0d3639'`
jobStreamStats.numberOfAbendedJob Distributed	The number of jobs in `ABEND` status within the job stream. Supported operators are `=` and `!=`. Example: `jobStreamStats.numberOfAbendedJob != '8'`
jobStreamStats.numberOfCurrentNodes Distributed	The number of current nodes within the job stream. Supported operators are `=` and `!=`. Example: `jobStreamStats.numberOfCurrentNodes = '2'`
jobStreamStats.numberOfExecutingJob Distributed	The number of jobs that are currently running within the job stream. Supported operators are `=` and `!=`. Example: `jobStreamStats.numberOfExecutingJob != '6'`
jobStreamStats.numberOfFailedJob Distributed	The number of jobs that are in `FAIL` status within the job stream. Supported operators are `=` and `!=`. Example:

Table 82. OQL syntax fields for job stream (continued)

Field name	Description
	`jobStreamStats.numberOfFailedJob = '1'`
jobStreamStats.numberOfJob Distributed	The total amount of jobs within the job stream. Supported operators are `=` and `!=`. Example: `jobStreamStats.numberOfJob != '3'`
jobStreamStats.numberOfNotRunningJob Distributed	The number of jobs that are currently not running within the job stream. Supported operators are `=` and `!=`. Example: `jobStreamStats.numberOfNotRunningJob = '10'`
jobStreamStats.numberOfSkelJob Distributed	The number of jobs in `SKEL` status within the job stream. Supported operators are `=` and `!=`. Example: `jobStreamStats.numberOfSkelJob = '1'`
jobStreamStats.numberOfSuccessfullJob Distributed	The number of jobs in `SUCCESS` status within the job stream. Supported operators are `=` and `!=`. Example: `jobStreamStats.numberOfSuccessfullJob != '7'`
jobStreamStats.numberOfUndecidedJob Distributed	The number of jobs in unknown status within the job stream. Supported operators are `=` and `!=`. Example: `jobStreamStats.numberOfUndecidedJob = '1'`
jsDefinitionFlags.carriedForward Distributed	The job stream is flagged as `carried forward`. Supported values are `true` or `false`. Example: `jsDefinitionFlags.carriedForward = false`
jsOptions.carryForward Distributed	The job stream is to be carried forward. Supported values are `true` or `false`. Example: `jsOptions.carryForward = true`

Table 82. OQL syntax fields for job stream (continued)

Field name	Description
jsOptions.monitored	The job stream is to be monitored. Supported values are `true` or `false`. Example: `jsOptions.monitored = false`
key Distributed	The complete key that identifies the job stream. Example: `key LIKE '@jskey'`
limit Distributed	The limit value set for the job stream. Example: `limit > '1' AND limit <= '3'`
name	The name of the job stream. Example: `name NOT LIKE '@JS@'`
occurrenceToken z/OS	The unique token that identifies the job stream. `occurrenceToken IN ['e079702a9547afc2']`
originalJobStreamName	The name of the job stream before any change was performed. Example `originalJobStreamName NOT LIKE 'job@'`
origPriority Distributed	The original priority value set for the job stream. Example: `origPriority = '10'`
pendingPredecessor Distributed	The job stream has predecessor instances that are pending. Supported values are `true` or `false`. Example: `pendingPredecessor = false`
priority	The priority value set for the job stream. Example: `priority = '101'`

Table 82. OQL syntax fields for job stream (continued)

Field name	Description
schedTime	The scheduled time of the job stream. Example: `schedTime >= '2025-03-01T00:00:00.000Z'`
timeInfo.actualEndTime	The time at which the job stream ended, that is the real end time. Example: `timeInfo.actualEndTime < '2025-03-02T10:35:22.000Z'`
timeInfo.actualStartTime	The time at which the job stream started, that is the real start time. Example: `timeInfo.actualStartTime > '2025-02-27T11:15:00.000Z'`
timeInfo.elapsedTime Distributed	The elapsed time of the job stream, that is the real duration. Supported operators are `=` and `!=`. Example: `timeInfo.elapsedTime != 'PT6H3S'`
timeInfo.estimatedDuration Distributed	The estimated duration of the job stream. Supported operators are `=` and `!=`. Example: `timeInfo.estimatedDuration = 'PT9H45S'`
timeRestrictions.deadlineTime	The time within which the job stream should complete. Example: `timeRestrictions.deadlineTime != '2025-09-13T09:00:00.000Z'`
timeRestrictions.startTime	The start time set for the job stream. Example: `timeRestrictions.startTime > '2025-04-27T10:15:00.000Z'`
timeRestrictions.untilTime Distributed	The latest time the job stream can be launched. Example:

Table 82. OQL syntax fields for job stream (continued)

Field name	Description
	`timeRestrictions.untilTime != '2025-02-27T11:35:16.000Z'`
workstation	The workstation where the job stream runs. Example: `workstation = '/WSFOLDER/WS'`

OQL syntax fields for jobs

The table below provides a comprehensive list and description of OQL fields that apply to job streams.

Note: OQL fields are case-sensitive; therefore, adherence to the exact capitalization and spelling is mandatory.

Unless specified, the fields and the values listed in the table apply both to distributed and z/OS environments.

Table 83. OQL syntax fields for jobs

Field name	Description

dependencies.dependencyStatus Distributed	The status of the job dependency. Supported values are:
	`SATISFIED|UNSATISFIED|UNDECIDED`
	Example:
	`dependencies.dependencyStatus != 'UNDECIDED'`

dependencies.jobId Distributed	The identifier of the job instance on which the job depends. Example: `dependencies.jobId = 'ca15f24a-0e6b-3f1b-9251-854d8d0d3639'`
dependencies.jobKey.name Distributed	The name of the job dependency. Example: `dependencies.jobKey.name NOT LIKE '@JS_@'`
dependencies.jobStreamFolder Distributed	For dependencies on jobs, the name of the folder of the job stream to which the job belongs. For dependencies on job streams, the name of the folder of the dependent job stream. Example: `dependencies.jobStreamFolder = '/folder/js'`

Table 83. OQL syntax fields for jobs (continued)

Field name	Description
dependencies.jobStreamId Distributed	For dependencies on jobs, the id of the job stream to which the job belongs. For dependencies on job streams, the id of the dependent job stream. Example: `dependencies.jobStreamId = 'ca15f24a-0e6b-3f1b-9251-854d8d0d3639'`
dependencies.jobStreamName Distributed	For dependencies on jobs, the name of the job stream to which the job belongs. For dependencies on job streams, the name of the dependent job stream. Example: `dependencies.jobStreamName LIKE 'J?S@'`
dependencies.jobStreamWorkstation Distributed	For dependencies on jobs, the name of the workstation of the job stream to which the job belongs. For dependencies on job streams, the name of the workstation of the dependent job stream. Example: `dependencies.jobStreamWorkstation LIKE 'wks_@'`
dependenciesStats.numberOfFileDependencies Distributed	The number of dependencies of the job of type `File`. Supported operators are `=` and `!=`. Example: `dependenciesStats.numberOfFileDependencies = '2'`
dependenciesStats.numberOfInternetworkDependencies Distributed	Number of dependencies of the job of type `Internetwork dependency`. Supported operators are `=` and `!=`. Example: `dependenciesStats.numberOfInternetworkDependencies != '4'`
dependenciesStats.numberOfJobDependencies Distributed	The number of job dependencies of the job. Supported operators are `=` and `!=`.

Table 83. OQL syntax fields for jobs (continued)

Field name	Description
	Example: `dependenciesStats.numberOfJobDependencies = '3'`
dependenciesStats.numberOfJobStreamDependencies Distributed	The number of job stream dependencies of the job. Supported operators are `=` and `!=`. Example: `dependenciesStats.numberOfJobStreamDependencies != '8'`
dependenciesStats.numberOfPromptDependencies Distributed	The number of dependencies of the job of type `Prompt`. Supported operators are `=` and `!=`. Example: `dependenciesStats.numberOfPromptDependencies = '1'`
dependenciesStats.numberOfResourceDependencies Distributed	The number of dependencies of the job of type `Resource`. Supported operators are `=` and `!=`. Example: `dependenciesStats.numberOfResourceDependencies != '5'`
dependenciesStats.numberOfUnresolvedDependencies Distributed	The number of dependencies of the job that have not been satisfied. Supported operators are `=` and `!=`. Example: `dependenciesStats.numberOfUnresolvedDependencies = '2'`
folder Distributed	The name of the folder within which the job is saved. Example: `folder LIKE '/@A@'`
folderId Distributed	The identifier of the folder within which the job is saved. Example: `folderId = 'be11g38a-0u7m-5f1d-5594-458d8d0d6541'`
id Distributed	The identifier of the job. Example: `id = 'ca15f24a-0e6b-3f1b-9251-854d8d0d3639'`

Table 83. OQL syntax fields for jobs (continued)

Field name	Description
jobDefinition.isCommand Distributed	The job definition is a command. Supported values are `true` if the job definition is a command, or `false` if the job definition is a script file. Example: `jobDefinition.isCommand = true`
jobDefinition.key.name Distributed	The name of the job definition. Example: `jobDefinition.key.name LIKE '@jobdef1_'`

jobDefinition.recoveryOption Distributed	The recovery options set for the job definition. Supported values are:
	`STOP|CONTINUE|RERUN`
	Example:
	`jobDefinition.recoveryOption = 'CONTINUE'`

jobDefinition.taskType Distributed	The task type of the job definition. Example: `jobDefinition.taskType = 'executable'`
jobDefinition.userLogin Distributed	The username of the job definition. Example: `jobDefinition.userLogin != 'user3'`
jobDefinition.workstation Distributed	The name of the workstation on which the job definition runs. Example: `jobDefinition.workstation NOT LIKE '/wks@'`
jobDefinition.workstationKey Distributed	The name of the workstation on which the job definition runs. Example: `jobDefinition.workstationKey LIKE 'rome1@'`
jobDefinitionFlags.monitored z/OS	The job must be monitored. Supported values are `true` or `false`.

Table 83. OQL syntax fields for jobs (continued)

Field name	Description
jobOptions.critical Distributed	The `critical` option is set for the job. Supported values are `true` or `false`. Example: `jobOptions.critical = true`
jobOptions.every Distributed	The `every` option is set for the job. Supported values are `true` or `false`. Example: `jobOptions.every = false`
jobOptions.noOperation	The `No operation` option is set for the job. Supported values are `true` or `false`. Example: `jobOptions.noOperation = true`
jobOptions.requiresConfirmation	The `Requires confirmation` option is set for the job. Supported values are `true` or `false`. Example: `jobOptions.requiresConfirmation = false`
jobruns.actualWorkstation Distributed	The name of the workstation where the job actually runs. Example: `jobruns.actualWorkstation NOT LIKE '@wks@'`

jobruns.commonStatus	The external status of the job runs. Supported values are:
	Distributed
	`WAITING|READY|RUNNING|SUCCESSFUL|ERROR|CANCELED|HELD|UNDECIDED|BLOCKED|SUPPRESS`
	z/OS
	`WAITING|READY|RUNNING|SUCCESSFUL|ERROR|CANCELED|HELD|UNDECIDED|INTERRUPTED|COMPLETED|SUPPRESS_BY_COND`
	Example:
	`jobruns.commonStatus IN ['WAITING', 'BLOCKED']`

jobruns.compositeStatus.status	The internal status of the job runs. Supported values are: Distributed

Table 83. OQL syntax fields for jobs (continued)

Field name	Description

	`HOLD|READY|FENCE|INTRO|FAIL|EXEC|WAIT|SUPPR|END|SUCC|SUCCP|ABEND|CANCL|CANCLP|ABENP|PEND|SUSP|STUCK|EXTRN`
	z/OS
	`WAIT|ARRVING|READY|NONREPPRED|STARTED|INTRPT|CANCL|UNDCDED|COMPLT|ERROR|SUCCP|SUPPR`
	Example:
	`jobruns.compositeStatus.status NOT IN ['SUPPR']`

jobruns.elapsedTime Distributed	The elapsed time of the job run instance, that is the real duration. Example: `jobruns.elapsedTime = 'PT5H9M'`
jobruns.estimatedDuration Distributed	The estimated duration of the job run instance. Example: `jobruns.estimatedDuration = 'PT9H21M'`
jobruns.jobNumber	The total number of jobs runs. Supported operators are `=` and `!=`. Example: `jobruns.jobNumber = '3'`
jobruns.maxDurationGone Distributed	The duration of the run is higher than the maximum duration value. Supported values are `true` or `false`. Example: `jobruns.maxDurationGone = true`
jobruns.minDurationNotReached Distributed	The duration of the run is lower than the minim duration value. Supported values are `true` or `false`. Example: `jobruns.minDurationNotReached = false`
jobruns.reruntype Distributed	The type of rerun. Supported values are: `RERUN|RECOVERY_RERUN|RERUN_EVERY|RERUN_FROM|RECOVERY|REGULAR|RERUN_STEP` Example:

Table 83. OQL syntax fields for jobs (continued)

Field name	Description
	`jobruns.reruntype IN ['RERUN', 'RECOVERY']`
jobruns.returnCode Distributed	The return code of the job run instance. Example: `jobruns.returnCode = '1'`
jobruns.step Distributed	The alias name of the job run instance. Example: `jobruns.step = 'JOBNAME'`
jobruns.timeInfo.actualEndTime Distributed	The time at which the job run instance ended. Example: `jobruns.timeInfo.actualEndTime = '2025-12-01T23:00:00.000Z'`
jobruns.timeInfo.actualStartTime Distributed	The time at which the job run instance started. Example: `jobruns.timeInfo.actualStartTime != '2025-12-01T23:00:00.000Z'`
jobruns.errorCode z/OS	The error code of the first operation in error status within the job run. Example: `jobruns.errorCode LIKE 'OS@'`
jobs.jobDefinition.workstationName	The name of the workstation where the job definition referenced by the job instance is saved. Example: `jobs.jobDefinition.workstationName NOT LIKE '@A'`
jobStreamId	The identifier of the job stream within which the job is saved. Example: `jobStreamId = 'ca15f24a-0e6b-3f1b-9251-854d8d0d3639'`
jobStreamName	The name of the job stream within which the job is saved. Example: `jobStreamName NOT LIKE 'A?@'`

Table 83. OQL syntax fields for jobs (continued)

Field name	Description
jobStreamWorkstation	The name of the workstation on which runs the job stream within which the job is saved. Example: `jobStreamWorkstation != 'WKS_1'`
name	The name of the job. Example: `name LIKE '@A?@'`
priority	The priority value set for the job. Example: `priority > '9' AND priority <= '100'`
jobruns.timeDependent z/OS	The evaluation of the AT time value of the dependency is active. Supported values are `true` or `false`. Example: `jobruns.timeDependent IN ['false']`

timeRestrictions.untilAction Distributed	The action set for a job when the until time has expired but the job has not yet started. Supported values are:
	`SUPPRESS|CONTINUE|CANCEL`
	Example:
	`timeRestrictions.untilAction = 'SUPPRESS'`

timeRestrictions.untilTime Distributed	The latest time a job can be launched. Example: `timeRestrictions.untilTime = '2025-03-02T10:35:22.000Z'`
workstation	The workstation where the job runs. Example: `workstation LIKE '@wks_rome1@'`

OQL syntax fields for resources

The table below provides a comprehensive list and description of OQL fields that apply to resources.

Note: OQL fields are case-sensitive; therefore, adherence to the exact capitalization and spelling is mandatory.

The filtering of resources is only available in distributed environments.

Table 84. OQL syntax fields for resources

Field name	Description
id	The id of the resource. Example: `id = 'ca15f24a-0e6b-3f1b-9251-854d8d0d3639'`
name	The name of the resource. Example: `name LIKE 'res@'`
folder	The folder of the resource. Example: `folder NOT LIKE '@A'`
workstation	The workstation of the resource. Example: `workstation LIKE '@_1'`
workstationFolder	The folder where the workstation of the resource is saved. Example: `workstationFolder LIKE '/ws_rome@'`
quantity	The entire quantity of the resource. Example: `quantity = '5'`
quantityInUse	The quantity of the resource that is currently in use. Example: `quantityInUse != '2'`
quantityAvailable	The quantity of the resource that is available. Example: `quantityAvailable = '4'`
status	The status of the resource. Supported values are:

Table 84. OQL syntax fields for resources (continued)

Field name	Description

	`AVAILABLE|NOT_AVAILABLE|UNDECIDED`
	Example:
	`status IN ['AVAILABLE']`

Query aliases

OQL supports case-insensitive aliases to simplify query syntax and improve readability. An alias functions as a replacement for a more complex OQL filter.

Currently, the following alias is supported for jobs and job streams:

STARTED

This alias filters jobs and job streams based on whether they have an actual start value. Supported values are `true` or `false`. For example:

```
STARTED = true
```

```
started = false
```

You can create more refined queries by combining the alias with other OQL syntax keywords and fields for more in-depth filtering:

```
name LIKE '@job1' AND folder LIKE '/@A@' AND started = false
```

Examples

This section shows various examples of OQL queries that can be done on jobs, job streams or workstations.

Workstation

1. The following example shows a query that retrieves all the workstations in plan that are either linked and running, or they are a fault-tolerant agent in a domain that is different from `MASTERDM`:

```
(activeStates IN  ['LINKED'] AND jobmanUp = true) OR (type IN ['FTA']  AND  domainName !=
 'MASTERDM')
```

2. The example below shows a query that retrieves all the workstations that have all the following characteristics:
   - They have a name that contains the sequence of letters `NEW`
   - They are either agent, fault-tolerant agent, or domain manager type workstations
   - They are in a domain whose name starts with letter `M`
   - They are in `UNLINKED` state
   - They are not currently running
   - They are contained in a folder whose name ends with letter `A`

```
name LIKE '@NEW@' AND type IN ['AGENT','FTA','DOMAIN_MANAGER'] AND domainName LIKE 'M@' AND
 activeStates IN  ['UNLINKED'] AND jobmanUp = false  AND folder LIKE '/@A/'
```

Job stream

1. The example below shows a query that retrieves either:
   - Job streams whose name does not contain the word `JS` within and whose limit is greater than 1 and less or equal to 3

     Or

   - Job streams whose name contains the letter `A` and that has a priority greater than 9 and less or equal to 100

```
(name NOT LIKE '@JS@' AND (limit > 1 AND limit <= 3)) OR (name LIKE '@A@' AND (priority > 9
 AND priority <= 100))
```

2. The following example shows a query that retrieves either:
   - Job streams that are in **WAITING** and **READY** status and that have a schedule time between March, 1st 2025 at 12 AM and March, 10th 2025 at 11:59 PM

     Or

   - Job streams that are in **ERROR** and **BLOCKED** status and that have a schedule time between January, 1st 2025 at 12 AM and February, 25th 2025 at 11:59 PM

```
(commonStatus IN ['WAITING','READY'] AND (schedTime >= '2025-03-01T00:00:00.000Z' AND
 schedTime <= '2025-03-10T23:59:59.000Z')) OR (commonStatus IN ['ERROR','BLOCKED'] AND
 (schedTime >= '2025-01-01T00:00:00.000Z' AND schedTime <= '2025-02-25T23:59:59.000Z'))
```

3. The following example shows a query that performs a basic search. Using regular plan filters, the query would correspond to `/WSFOLDER/WS#/JSFOLDER/JSNAME`:

```
name = 'JSNAME' AND workstation = '/WSFOLDER/WS' AND folder ='/JSFOLDER'
```

4. The below query retrieves all job streams whose status is either `SUCC`, `ABEND`, or `FAIL`, and whose start time is prior to 2025/02/27 11:15:00:000 UTC:

```
compositeStatus.status IN ['SUCC','ABEND','FAIL'] AND timeInfo.actualStartTime <
 '2025-02-27T11:15:00.000Z'
```

5. The following example shows a query that retrieves all the job streams that contain 3 jobs within, and that either:
   - Have priority value set to 101 and are in **RUNNING** status

     Or

   - Are in a folder and on a workstation whose name starts with `/WS`

```
((priority = 101 AND commonStatus = 'RUNNING') OR (workstation = '/WS@' AND folder = '/'))
 AND jobStreamStats.numberOfJob = 3
```

Job

1. The following example shows a query that retrieves all jobs that:
   - Have a name that starts with the letter `A`
   - Are part of the job stream named `JOBS`

Moreover, the query retrieves jobs that either run on the workstation named `/WORKSTATION`, or that belong to job streams scheduled on `/WORKSTATION`. Results are sorted by name in descending order:

```
name LIKE 'A@'  AND (workstation = '/WORKSTATION' OR jobStreamWorkstation = '/WORKSTATION')
 AND jobStreamName = 'JOBS' ORDER BY name DESC
```

2. The following query retrieves all jobs that are critical and that either are not in `FAIL`, `ABEND`, and `HOLD`, or have been cancelled:

```
(jobruns.compositeStatus.status NOT IN ['FAIL','ABEND','HOLD'] OR
 jobruns.compositeStatus.canceled = true ) AND jobOptions.critical = true
```

3. The below query retrieves all jobs that have a single dependency, excluding those that have completed with either `SUCCESSFUL` or `ERROR` status:

```
dependenciesStats.numberOfJobDependencies = 1 AND jobruns.commonStatus NOT IN
 ['SUCCESSFUL','ERROR']
```

Resource

1. The following example shows a query that retrieves all resources that:
   - Have a name that contains the word `RES`
   - Have the quantity in use value equal to 3
   - Are in `AVAILABLE` status

```
name LIKE '@RES@' AND quantityInUse = '3' AND status in ['AVAILABLE']
```

2. The following query retrieves all resources that have a quantity available value that is either 5 or 9, and that are saved in a folder named `ROME`:

```
(quantityAvailable = '5' OR quantityAvailable = '9' ) AND folder = 'ROME'
```

Chapter 14. Orchestration CLI

The command line interface to run jobs or job streams and to interact with HCL Workload Automation.

When you want to run or update a job or job stream without using the Dynamic Workload Console, you can use Orchestration CLI. You can use the feature to run the context, model, and plan commands. This is a stand-alone feature which you can download and install on its own without the installation of any HCL Workload Automation components. You can connect to remote workstations by configuring the `config.yaml` file.

Related information

Configuring Orchestration CLI on page 757

Running commands from Orchestration CLI on page 762

Orchestration CLI commands on page 764

Configuring Orchestration CLI

The extension for Orchestration CLI is bundled with the product. If required, you can download it as a stand-alone feature also.

Before you begin

You must have completed any one of the following tasks:

Downloaded the latest version of HCL Workload Automation.

or

Downloaded the executable files of Orchestration CLI.

About this task

After you download the required files for Orchestration CLI, you need to configure the `config.yaml` extension file by any source code editor. The `config.yaml` file contains mandatory and optional fields. To create the `config.yaml` file, you must run the Orchestration CLI file using the command prompt.

1. Navigate to the directory where you have downloaded the files for Orchestration CLI.
2. Run Orchestration CLI from the command line.

   Note: For windows, you can also use powershell to run the file.

3. Type `ocli plan` as command, and then press enter to create the `config.yaml` file.

   In Windows operating system, the `config.yaml` file is created at `c:\Users\`*`user_name`*`\.OCLI`.

   In Unix operating systems, the `config.yaml` file is created at `home/`*`user_name`*`/.OCLI`.

4. Open the `config.yaml` file by using any source code editor.
5. Perform the following actions in the `config.yaml` file.

   You must enter valid details for the mandatory fields. If required, you can also specify the optional fields.

Table 85. Mandatory fields

Option	Action
connection	
host	Enter the host name or IP address of the remote engine that you want to use.
port	Enter the port number of the remote engine that you want to use. The default value is 31116.
jwt	When the connection is established for the first time, this field is filled in automatically. For more information, see Authenticating Orchestration CLI using API Keys on page 763.
user	Specify the user name of the user who installed the agent.

Table 86. Optional fields

Option	Actions
lang	You can use this option to specify any one of the supported languages. The following languages are supported in Orchestration CLI: ◦ Brazilian Portuguese (pt_BR) ◦ English (EN, the default value) ◦ French (FR) ◦ German (DE) ◦ Italian (IT) ◦ Japanese (JA) ◦ Korean (KO) ◦ Simplified Chinese (zh_CN) ◦ Spanish (ES) ◦ Traditional Chinese (zh_TW) **Important:** You must ensure that the recent version of powershell is installed on Windows

Option	Actions
	operating system to view the content in the selected language.
dateformat	The date format is specific to the commands you run on Orchestration CLI. The default format is *ymd*. If you do not specify any format the default one is applied.
current_folder	Specify the path to any folder that will be taken as an absolute path, when you run the commands. The default value is "/".
default workstation	You can specify a workstation as default for Orchestration CLI model item definitions. The workstation mentioned is used as the reference, if you do not specify any workstation in an item definition. The default value is **TEST_AGENT**.
local_timezone	You can use this option to specify the timezone. If not, the time zone of the operating system is referred. The supported formats are: ◦ The extended form "area/city". For example, "America/New_York" for EST. ◦ The 3 character notation, for example ECT (European Central Time).
model	
format	You can use this option to set up the format for Orchestration CLI. Specify any one of the following formats. The default value is YAML. ◦ JSON ◦ schedlang ◦ YAML For more information, see Managing multiple formats on page 888.
logging	
level	You can use this option to filter the log messages that are retrieved in the `ocli.log` file. When you set a value, all the messages equal to that severity and above are sorted. The default value is **info**. To view the file, navigate to `C:\Users\user_name\.OCLI\logs`. You

Option	Actions
	can enter any one of the following message types. The message types are listed from lower to higher severity. ◦ **info**: Any additional information which are not tied to the current task or action. ◦ **trace**: Messages that provides information about the application flow. ◦ **debug**: Informational messages that are useful to troubleshoot the application. ◦ **warning**: Messages related to user actions that will have an unexpected or unwanted consequences. ◦ **error**: Messages related to errors or critical failures that may block the users to proceed further until the issue is resolved. ◦ **fatal**: Messages related to errors that will cause the Orchestration CLI to close without saving any data. ◦ **panic**: Messages related to errors that will cause the Orchestration CLI to close without saving any data. The error is similar to a **fatal** error and the only difference is the way the application is closed.
redirect_to	You can use this option to change the location of the log file. The default location is `c:\Users\`*`user_name`*`\.OCLI\logs`.

Option	Actions
	You can use this option to change the location of the log file. The default location is `/home/user_name/.OCLI/logs`.
file_size	You can specify the maximum size of the log file in Mega Bytes(MB). The default value is **10**.
backup_files	You can specify the number of log files which you want to save. The default value is **3**.
connection	
contextroot	You can specify the endpoint that connects with the server. The default value is **/,/twsd/cli**.
protocol	Enter the protocol that you want to use to connect to the remote engine. The default value is **https**.
insecure	Enter *true*, to ignore the host name error on SSL certificate. The default value is *false*, which means HCL Workload Automation checks whether the certificates are valid and trusted.
proxy	Specify the proxy IP address, if you are using a proxy server.
proxyport	Specify the port of the proxy server.
proxyuser	Provide the user name of the proxy account, if authentication is required.
proxypassword	Provide the proxy account password.
timeout	You can use this option to specify the maximum time in milliseconds to display the timeout error message. The default value is **3600**.
show_warnings	If you do not want to receive any warnings when there is an internet connection failure, set the option to **false**. The default value is **true**.
compatibility	
show_copyright_header	If you do not want the copyright content to be displayed when you run a command, set the option to **false**. The default value is **true**.

6. Save the `config.yaml` file.

Consider the following example:

```
host: <your_hostname>
      port: <port_number>
      protocol: https
      insecure: true
      jwt : <token from Dynamic Workload Console>
```

Results

You have successfully configured Orchestration CLI.

Note: Orchestration CLI generates temporary files inside the default temporary folder of the operative system. Ensure all Orchestration CLI users have read and write permissions on that temporary folder.

What to do next

You can run the commands using Orchestration CLI to accomplish specific results. For more information, see Running commands from Orchestration CLI on page 762.

Running commands from Orchestration CLI

After you installed and configured the Orchestration CLI, you can run one or a combination of commands to accomplish specific results.

Before you begin

You must have completed the following tasks:

- Installed and configured the Orchestration CLI in your system. For more information, see Configuring Orchestration CLI on page 757.
- The required permissions to run the commands. For more information, see overview section of each commands mentioned in Orchestration CLI commands on page 764.

About this task

You can use Orchestration CLI to run context, model, and plan commands.

1. Navigate to the directory where you have downloaded the files for Orchestration CLI.
2. Run Orchestration CLI from the command line.

   Note: For windows, you can also use powershell to run the file.

3. Type `ocli context`, `ocli plan`, or `ocli model` in the command line, and then enter the command you want to run.

   Important: You must type `ocli context`, `ocli plan`, or `ocli model` before every command that you want to run. For example,

```
ocli model display workstation
```

Note: For the types of command that you can run in Orchestration CLI, see Orchestration CLI commands on page 764.

4. Press Enter.

Results

You have successfully run the command in Orchestration CLI and the result is displayed.

Authenticating Orchestration CLI using API Keys

When you connect to HCL Workload Automation using Orchestration CLI for the first time, you need to authenticate the connection using API Keys.

Before you begin

You must have completed the following tasks:

- Downloaded the package for Orchestration CLI.
- Created valid credentials in an authentication or authorization provider that uses the open protocol standard OpenID Connect.
- Configured HCL Workload Automation to specify the instance where the authentication or authorization provider is running. For more information, see the topic about configuring a user registry in Planning and Installation.

1. Navigate to the directory where you have downloaded the files for Orchestration CLI.
2. Run Orchestration CLI from the command line.

Note: For windows, you can also use powershell to run the file.

3. Type any Orchestration CLI command.

Example:

```
ocli model list folder @
```

An error message with a web link to create the API Key is displayed.

4. Open the web link using any web browser.

The login page of the authentication provider that you configured is displayed.

5. Enter the username and password for the authorization provider.

   A message is displayed to indicate that the API Key has been successfully generated. This API Key gets automatically updated in the `config.yaml` file.

Results

You have successfully established a connection with HCL Workload Automation from Orchestration CLI.

What to do next

You can fill in other mandatory fields in the `config.yaml` file and if already done, you can run any Orchestration CLI commands to achieve specific results. For more information, see Running commands from Orchestration CLI on page 762.

Orchestration CLI commands

You can use the various command options in Orchestration CLI to achieve specific results. The command options are not case sensitive and you can use either uppercase or lowercase letters.

In HCL Workload Automation, you can use the command line to run context, model, and plan commands. The commands that you can use in Orchestration CLI have abbreviated forms. You can either use the abbreviation or the spelled-out form to complete the task. You can use the following lists of commands in Orchestration CLI:

Context commands

- list on page 771
- new on page 771
- remove on page 773
- set on page 773
- switch on page 774

Model commands

- add on page 775
- delete on page 776
- display on page 779
- extract on page 785
- list on page 788
- listfolder on page 793
- lock on page 793
- mkfolder on page 796
- modify on page 796
- new on page 799
- rename on page 802
- renamefolder on page 803
- replace on page 804

- rmfolder on page 805
- unlock on page 806

Plan commands

- ADDDEP JOB on page 809
- ADDDEP SCHED on page 810
- ALTJOB on page 812
- ALTPRI on page 814
- CANCEL JOB on page 815
- CANCEL SCHED on page 817
- change resource on page 822
- CONFIRM on page 819
- DELDEP JOB on page 823
- DELDEP SCHED on page 824
- FENCE on page 825
- KILL on page 826
- LIMIT CPU on page 827
- LIMIT SCHED on page 828
- LISTFOLDER on page 828
- nop/unnop on page 829
- RELEASE JOB on page 829
- RELEASE SCHED on page 830
- reply on page 831
- rerun on page 833
- showcpu on page 835
- showfile on page 840
- showjobs on page 846
- showprompt on page 860
- show resource on page 866
- showschedules on page 871
- SUBMIT DOCOMMAND on page 876
- SUBMIT JOB on page 878
- SUBMIT SCHED on page 879

Related information

Configuring Orchestration CLI on page 757

Running commands from Orchestration CLI on page 762

Access required to run Orchestration CLI commands

You must have specific permissions to run each command in Orchestration CLI. The level of access to run the command is customizable and you can use the Dynamic Workload Console to manage the roles.

In HCL Workload Automation you can create roles for each user and configure the level of access to perform specific actions. The roles are classified into two, administrative and standard. You must create an administrative and a standard role for each user. The administrative role defines the level of access user must have to manage API keys and register an agent. The standard role defines the level of access user must have to run specific commands. In Dynamic Workload Console, navigate to **Administration > Manage Workload Security > Create new roles** to manage the permissions associated with both the role types. The different permissions associated with Orchestration CLI commands are grouped under the following categories.

- **Design and Monitor Workload**

  Permissions related to model commands.
- **Modify current plan**

  Permissions related to plan commands.
- **Submit Workload**

  Permissions related to job and job stream definitions in plan.
- **Manage Workload Environment**

  Permissions related to workstations.
- **Administrative Tasks**

  Permissions related to administration and plug-ins.

The specific permissions required to run particular commands are given below.

Orchestration CLI model commands	
Command	**Access required on items**
add	*ADD* **Important:** If you want to perform the action on an existing item, you must have the *MODIFY* access and if the item is locked by another user, you must also have the *UNLOCK* access.
delete	*DELETE*
display	*DISPLAY*

Orchestration CLI model commands	
Command	**Access required on items**
	Important: If you want to run the command with the **FULL** parameter to view information about job streams then you must also have *Display* access on job.
draft	*DRAFT*
extract	*DISPLAY* **Important:** In addition to the permission mentioned above, you must also have *MODIFY* access if you want to run the command with the **lock** parameter.
list	*LIST*
listfolder	*LIST*
lock	*MODIFY*
mkfolder	*ADD*
modify	*MODIFY* and *DISPLAY* **Important:** In addition to the permission mentioned above, you must also have the following permissions if you want to run the command to update information about job streams: • *USE* access on jobs. • *USE* access on calendars. Furthermore, if you want to run the command with **FULL** parameter, you must also have the following permissions: • *DISPLAY* access on jobs. • *USE* access on workstation where the job is defined. • *ADD* access on job, If the job definition is new or *MODIFY* access on job if the definition is an existing one.

Orchestration CLI model commands	
Command	**Access required on items**
new	*ADD*
nop	*NOOPDB*
rename	*DELETE* and *ADD*
renamefolder	*DELETE* and *ADD*
replace	• To replace with a new item: *ADD* • To replace an existing item: *MODIFY* • To replace an existing item that is locked by another user: *MODIFY* and *UNLOCK* **Important:** In addition to the permission mentioned above, you must also have the following permissions if you want to run the command to update information about job streams: ◦ *USE* access on jobs. ◦ *USE* access on calendars. Furthermore, if you want to run the command with **FULL** parameter, you must also have the following permissions: ◦ *DISPLAY* access on jobs. ◦ *USE* access on workstation where the job is defined. ◦ *ADD* access on job, If the job definition is new or *MODIFY* access on job if the definition is an existing one.
rmfolder	*DELETE*
undraft	*DRAFT*
unlock	• If the item is locked by the same user: *LIST* • If the item is locked by another user: *UNLOCK*
unnop	*NOOPDB*

Orchestration CLI plan commands	
Command	**Access required on items**
adddep job/sched	*ADDDEP*
altjob	*SUBMIT*
altpass	*MODIFY*
altpri	*altpri*
cancel job/sched	*CANCEL*
confirm	*JOB-CONFIRM*
deldep job/sched	*DELDEP*
fence	*fence*
kill	*JOB-KILL*
limit sched	*limit*
listfolder	*LIST*
release job/sched	*RELEASE*
reply	*reply*
setonline	*LINK*
setoffline	*UNLINK*
showcpu	*DISPLAY*
showjobs	*DISPLAY*
showschedules	*DISPLAY*
submit docommand	You must have the same accesses you need for the submit job command.
submit job	You need specific access to perform actions on jobs and ad-hoc jobs. • To submit an existing job definition, you must have *JOB-SUBMIT* or *JOB-SUBMITDB* access. If you want to submit an existing job definition into an existing job stream, in addition to the access specified above, you must also have the *SCHEDULE-SUBMIT* access. • To submit an existing ad-hoc job definition, you must have the *JOB-SUBMITDB* and *CPU-USE*

Orchestration CLI plan commands	
Command	**Access required on items**
	access. If you want to submit an existing ad hoc job definition into an existing job stream, in addition to the access specified above, you must also have the *SCHEDULE-SUBMIT* access.
submit sched	*SCHEDULE-SUBMIT*

Orchestration CLI plug-in commands	
Commands	**Access required**
delete	*DELETE_PLUGIN*
install	*INSTALL_PLUGIN*
list	*LIST_PLUGIN*
update	*INSTALL_PLUGIN* and *DELETE_PLUGIN*

Multiple contexts and context commands

The term context refers to the set of parameters in the `config.yaml` file that is used to establish an environment to run HCL Workload Automation. You can create multiple contexts in the `config.yaml` file to manage the workload. This helps you to utilize the resources in a more efficient manner.

Overview

You can create multiple contexts with different configuration settings using Orchestration CLI. This feature allows you to manage various workflow scenarios by changing the context. You can use context commands to manage or switch between multiple contexts and to do actions in a specific context. The default context is the one that you create initially until you change it using the commands. Unless you explicitly specify the context in the command, all commands are run using the default context. To manage different contexts, you can use the following set of commands in Orchestration CLI:

- list on page 771
- new on page 771
- remove on page 773
- set on page 773
- switch on page 774

Important: To manage the context parameters more efficiently when you have multiple `config.yaml` files available in your computer, see Managing multiple config.yaml files on page 887.

Restriction: Do not enclose a command and its parameters in double quotes or single quotes. The parser interprets the content inside the quotes as a single command name, and it displays an error message.

list

You can use the list command to display the contexts available in the `config.yaml` file.

Syntax and command line options

You can enter the command as follows:

```
ocli context [list | l]
```

The default context is identified with an asterisk (*) in the list.

Related information

Managing multiple config.yaml files on page 887

new

You can use the new command to create a new context in the `config.yaml` file.

Syntax and command line options

You can enter the command as follows:

```
ocli context new context_name
```

The *context_name* parameter is mandatory. Once you provide the context name a message is displayed to enter URL. You must provide the host and port name separated by a colon (:). These are mandatory values that you must provide to create a context. For more details see the following example.

Example:

1. Follow the below steps to create a context named `TEST` with `localhost` and `8080` as host and port respectively.
   a. Enter the command

   ```
   ocli context new TEST
   ```

   The following message is displayed to add host and port name

   ```
   ocli context new TEST
   Enter url:
   ```

   Note: If you do not provide the environment name the following message is displayed:

```
ocli context new
Enter context name:
```

You must enter a valid name to continue.

b. Enter the host and port name separated by a colon (:)

```
ocli context new TEST
Enter url:
localhost:8080
```

A confirmation message is displayed to indicate the creation of the new context.

```
ocli context new TEST
Enter url:
localhost:8080
You have successfully added the TEST context.
```

Note: If you enter `http` or `https` as port, the default values `80` and `443` are respectively added to the `config.yaml` file.

2. Follow the below steps to create the `LOCAL` context with `localhost` and `9433` as host and port respectively.
   a. Enter the command

```
ocli context new LOCAL
```

The following message is displayed to add host and port name

```
ocli context new LOCAL
Enter url:
```

b. Enter the url as `https://localhost:9433/`:

```
ocli context new TEST
Enter url:
https://localhost:9433/
```

A confirmation message is displayed to indicate the creation of the new context.

```
ocli context new TEST
Enter url:https://localhost:9433/
You have successfully added the LOCAL context.
```

Note: If you enter `http` or `https` as URL and did not add any specific value as port, the default values `80` and `443` are respectively added to the `config.yaml` file.

Related information

Managing multiple config.yaml files on page 887

remove

You can use the remove command to remove an existing context from the `config.yaml` file.

Syntax and command line options

You can enter the command as follows:

```
ocli context [remove | rm] context_name
```

The *context_name* parameter is mandatory.

Example

- Run the following command to remove the `TEST` context from the `config.yaml` file.

  ```
  ocli context remove TEST
  ```

Related information

Managing multiple config.yaml files on page 887

set

You can use the set command to update and manage the parameters or properties of each context in the `config.yaml` file.

Overview

Each context in the `config.yaml` contains a set of parameters or properties that can be configured to set up an environment. You can use the set command to update the values of these parameters. This permits you to customize the behavior of a specific command without modifying the entire `config.yaml` file. You can specify the required parameter value directly in the command, to override the value set in the default or selected `config.yaml` file. This flexibility gives you granular control over the configuration settings for each context, further enhancing the versatility of managing multiple `config.yaml` files and context parameters.

Syntax and command line options

You can enter the command as follows:

```
ocli context set [context_name]
parameter=value
```

The *[parameter=value]* parameter is mandatory.

Important: If you run the command without *[context_name]* parameter, the *[parameter=value]* for the default context is updated.

The details required with each parameter is as follows:

context_name

Specify the context name.

parameter=value

Specify a parameter in the context with a value. In the `config.yaml` file, you have multiple parameters under **logging** and **connection**. If you want to update any values under these parameters, specify the complete level separated by a dot (.) For example if you want to update the value for **port** parameter under **connection**, specify the value as

```
ocli context set connection.port=443
```

Example

1. Run the following command to update the **port** to `443` in `TEST` context.

   ```
   ocli context set TEST connection.port=443
   ```

Related information

Managing multiple config.yaml files on page 887

switch

You can use the switch command to change the default context.

Overview

The switch command helps you to manage the default context when you have multiple contexts available in the `config.yaml` file. You can run the list command to view all the available contexts in the `config.yaml` file, and the default context is identified with an asterisk (*).

Syntax and command line options

You can enter the command as follows:

```
ocli context [switch | sw] context_name
```

The *context_name* parameter is mandatory.

context_name

Specify the name of the context that you want to set as default.

Example

- Run the following command to set the default context to `TEST`:

  ```
  ocli context switch TEST
  ```

Related information

Managing multiple config.yaml files on page 887

Model commands

You can use the model commands to create or modify the item definitions for folders, jobs, job streams, and workstations.

You can use the following list of model commands in Orchestration CLI:

- add on page 775
- delete on page 776
- display on page 779
- draft/undraft on page 785
- extract on page 785
- list on page 788
- listfolder on page 793
- lock on page 793
- mkfolder on page 796
- modify on page 796
- new on page 799
- nop/unnop on page 800
- rename on page 802
- renamefolder on page 803
- replace on page 804
- rmfolder on page 805
- unlock on page 806

Important: To manage the context parameters more efficiently when you have multiple `config.yaml` files available in your computer, see Managing multiple config.yaml files on page 887.

Related information

Temporarily modifying the parameters in the config.yaml file on page 890

add

You can use the add command to add scheduling items to the database.

Overview

You can add new items and items similar to existing ones. If an item already exists, you are asked whether to replace the item or not. You must specify the file location before the file name, if the file is not located where you run the command. You

can use the **unlock** option, if the item is locked by any other user. The add command checks for loop dependencies in a job stream. There is a loop dependency in a job stream, if `JOB1` follows `JOB2` and `JOB2` follows `JOB1`.

You must have **add** access to add an item to the database. If the item already exists and locked by any other user, then you must have **modify** and **unlock** access to the item.

Syntax and command line options

You must provide at least one scheduling item with the command to perform the action. You need to use the **;unlock** parameter, if the item already exists and it is locked by the same user and session.

```
ocli model add|a filename [;unlock]
```

filename: You must specify the name of the file that contains the item definition.

;unlock

You must add the parameter, if the item you want to edit is locked by the same user and session.

Important: Definitions created in YAML or JSON format using older HCL Workload Automation versions cannot be imported. To add such definitions, create them as new.

Examples

1. Run the following command to add jobs from `jobspend`:

   ```
   ocli model add jobspend
   ```

2. Run the following command to add job streams from `skedon`:

   ```
   ocli model add skedon
   ```

Related information

Managing multiple config.yaml files on page 887

delete

You can use the delete command to remove scheduling items from the database.

Overview

You can use the command option to delete items located in specific folders or multiple items that are located in different folders. You must have the *delete* access to remove scheduling items.

Important: You must make sure that the item you want to delete is not locked by any other user.

Syntax and command line options

Before you run the command, you must specify at least one scheduling item you want to delete. The **noask** parameter is optional. You can run the command as follows:

```
ocli model delete | de | del Scheduling item [;noask]
```

You can use the following scheduling parameters to view specific information.

Scheduling items

Table 87. Scheduling items

Scheduling items	Description
`calendars | calendar | cal [folder/]calname`	You must specify the *[folder/]calname* that you want to delete. Wildcard characters are permitted.
`domain | dom domainname`	You must specify the *domainname* that you want to delete. Wildcard characters are permitted.
`folder | fol foldername`	If you do not provide the *foldername* or if the folder contains scheduling items, an error message is displayed. Wildcard characters are permitted.
`jobs | jobdefinition | jd [[folder/]workstationname#][folder/]jobname`	You can specify the *jobname* that you want to delete. You can provide *[folder/]workstationname#]* and the folder in which the job stream is defined to make the selection more specific. Wildcard characters are permitted.
`sched | jobstream | js [[folder/]workstationname#][folder/]jstreamname[valid from date|valid to date |valid in date date] [;full] [;force]`	You must specify the *jstreamname* that you want to delete. You can provide *[folder/]workstationname#]* and the folder name to make the selection more specific. Wildcard characters are permitted. You can also add the options below to provide more details of the job stream. *Valid from date*: You can specify a date which will restrict the selection of job streams, that have a *valid from* date equal to the value specified. The format is *mm/dd/yyyy*. *Valid to date*: You can specify a date which will restrict the selection of job streams, that have a *valid to* date equal to the value specified. The format is *mm/dd/yyyy*.

Table 87. Scheduling items (continued)

Scheduling items	Description
	valid in date date: Specify the time frame at which the job is run. You can specify one of the dates as `@`. The format is *mm/dd/yyyy-mm/dd/yyyy*. **full**: If you add the option, all the job definitions in the job stream are displayed. **Important:** To delete a job stream that has been referenced in other items, use the command **./ocli model delete js jobstreamSelector ;force**. When you use **force**, not only the job streams are deleted, but also all the references associated with it.
`prompt | pr [folder/]promptname`	You must specify the *[folder/]promptname* that you want to delete. Wildcard characters are permitted.
`resource | re | reso [folder/]resourcename`	You must specify the *[folder/]resourcename* that you want to delete. You can provide the *[folder/]workstationame#]* and the folder in which the resource is defined to make the selection more specific. Wildcard characters are permitted.
`runcyclegroup | rcg [folder/]rcgname]`	You must specify the *[folder/]rcgname]* that you want to delete. Wildcard characters are permitted.
`users | user [[folder/]workstationame#]username [;password]`	You must specify the *[[folder/]workstationame#]username [;password]* that you want to delete. Wildcard characters are permitted.
`parms | parm | variable | vb [[folder/]tablename.]variablename`	You must specify the *[[folder/]tablename.]variablename* that you want to delete. Wildcard characters are permitted.
`vartable | vt [folder/]tablename`	You must specify the *[folder/]tablename* that you want to delete. Wildcard characters are permitted.
`workstation | ws | cpu {[folder/]workstationame`	You can specify the name of the workstation that you want to delete. Optionally, you can provide the folder in which the items are located. Wildcard characters are permitted.
`workstationclass | wsc [folder/]workstationclassname]`	You must specify the *[folder/]workstationclassname]* that you want to delete. Wildcard characters are permitted.

;noask

When you add the option as an argument the agent will not ask for confirmation before taking action on each qualifying job.

Examples

1. Run the following command to delete `sac5` stored in the `oldfolder` folder that is launched on workstation `stat1` stored in the `test` folder:

   ```
   ocli model delete jobs=test/stat1#oldfolder/sac5
   ```

2. Run the following command to delete all workstations with names starting with `xl`:

   ```
   ocli model de cpu=xl@
   ```

3. Run the following command to delete all job streams stored in the folder path `test/oldfolder` on all workstations:

   ```
   ocli model de sched=@#test/oldfolder/@
   ```

Related information

Managing multiple config.yaml files on page 887

display

You can use the command option to view all the available information about item definitions of the same type.

Overview

You can configure the command line syntax with different parameters to view specific results. In this way, you can view the information about one or all item definitions of the same type available in the database. The details for each item are displayed as described in the table.

Table 88. Output format for displaying items

items	Details
calendar	• **Calendar Name** • **Updated On** • **Locked By**
domain	• **Domain Name** • **Parent Domain** • **Master** • **Updated On** • **Locked By**

Table 88. Output format for displaying items (continued)

items	Details
folder	• **Name** • **Updated On** • **Locked By**
job definition	• **Workstation** • **Job Definition Name** • **Updated On** • **Locked By** • **Job Definition**
job stream	• **Workstation** • **Job Stream Name** • **Valid From** • **Updated On** • **Locked By** • **Job Stream Definition**
prompt	• **Prompt Name** • **Updated On** • **Locked By** • **Prompt Definition**
resource	• **Workstation** • **Resource Name** • **Quantity** • **Updated On** • **Locked By**
run cycle group	• **Run Cycle Group Name** • **Updated On** • **Locked By**

Table 88. Output format for displaying items (continued)

items	Details
user	• **Workstation** • **Username** • **Updated On** • **Locked By**
variable	• **Variable Table Name** • **Variable Name** • **Updated On** • **Locked By**
variable table	• **Variable Table Name** • **Default** • **Updated On** • **Locked By** • **Members**
workstation	• **Workstation Name** • **Type** • **Ignored** • **Updated On** • **Locked By** • **Workstation Definition**
workstation class	• **Workstation Name** • **Type** • **Domain** • **Ignored** • **Updated On** • **Locked By**

Syntax and command line options

You must provide at least one scheduling with the command to perform the action.

```
ocli model display | di Scheduling item
```

For the list of scheduling parameters that you can use and the format that needs to be followed, see the tables below.

Scheduling items

Table 89. Scheduling items

Scheduling items	Description
`calendars | calendar | cal [folder/]calname`	You can specify the *[folder/]calname* and optionally, the folder in which the calendar is defined. If you do not specify the *[folder/]calname*, all the calendar definitions in the database are displayed in order.
`domain | dom domainname`	You can specify the *domainname*. If you do not provide the *domainname*, all the domain definitions in the database are displayed. Wildcard characters are permitted.
`folder | fol foldername`	You can specify the name of the folder. If you do not provide the *foldername*, all the folder definitions are displayed. Wildcard characters are permitted.
`jobs | jobdefinition | jd [[folder/]workstationame#][folder/]jobname`	If you provide a specific *jobname* only that particular job definition is displayed and if not, all the job definitions are displayed. You can provide *[folder/]workstationame#]* and the folder in which the job is defined to make the selection more specific. Wildcard characters are permitted.
`sched | jobstream | js [[folder/]workstationame#][folder/]jstreamname[valid from date|valid to date |valid in date date][;full]`	If you provide a specific *jstreamname* only that particular job stream definition is displayed and if not, all the job stream definitions are displayed. You can provide *[folder/]workstationame#]* and the folder in which the job stream is defined to make the selection more specific. Wildcard characters are permitted. You can also add the options below to provide more details of the job stream. *Valid from date*: You can specify a date which will restrict the selection of job streams, that have a *valid from* date equal to the value specified. The format is *mm/dd/yyyy*. *Valid to date*: You can specify a date which will restrict the selection of job streams, that have a *valid to* date equal to the value specified. The format is *mm/dd/yyyy*. *valid in date date*: Specify the time frame at which the job is run. You can specify one of the dates as `@`. The format is *mm/dd/yyyy-mm/dd/yyyy*.

Table 89. Scheduling items (continued)

Scheduling items	Description
	full: If you add the option, all the job definitions in the job stream are displayed.
`parms | parm | variable | vb [[folder/]tablename.]variablename`	You must provide the *[[folder/]tablename.]variablename* to display a specific variable within a specific table. If you specify the *variablename* only, the variable definition in the default variable is displayed. If you do not provide the *[[folder/]tablename.]variablename*, all the global variable definitions in the default variable table are displayed. Wildcard characters are permitted on both *[folder/]tablename* and *variablename*.
`prompt | pr [folder/]promptname`	You can specify the *[folder/]promptname*. If you do not specify a *[folder/]promptname*, all the prompt definitions in the database are displayed. Wildcard characters are permitted.
`resource | re | reso [folder/]resourcename`	You can specify the *[folder/]resourcename*. You can provide the *[folder/]workstationname#]* and the folder in which the resource is defined to make the selection more specific. If you do not provide the *[folder/]resourcename*, all the resource definitions in the database are displayed. Wildcard characters are permitted.
`runcyclegroup | rcg [folder/]rcgname]`	You can specify the *[folder/]rcgname]*. If you do not specify a *[folder/]rcgname]*, all the run cycle group definitions in the database are displayed. Wildcard characters are permitted.
`vartable | vt [folder/]tablename`	You can specify the name of the variable table and optionally, you can provide the folder in which the variable table is defined. If you do not specify the *tablename* and add wildcard characters, all the variable table definitions are displayed. Wildcard characters are permitted.
`users | user [[folder/]workstationname#]username [;password]`	If you do not provide any specific *username*, all the usernames in the database are displayed. You can provide *[folder/]workstationname#]* and the folder in which the user is defined, to make the selection more specific. Wildcard characters are permitted.
`workstation | ws | cpu {[folder/]workstationname}`	You can specify the name of the workstation and optionally, you can provide the folder in which the items are located. Wildcard characters are permitted.

Table 89. Scheduling items (continued)

Scheduling items	Description
`workstationclass | wsc [folder/]workstationclassname]`	You can specify the *[folder/]workstationclassname]*. If you do not specify a *[folder/]workstationclassname]*, all the workstation class definitions in the database are displayed. Wildcard characters are permitted.

Note:

When you run the display command, the template also displays the unique ID assigned to each item. If you want to view the template without these IDs, you can use the **noid** parameter along with the command.

For example,

```
ocli model -format json display jd @#@ noid
```

Example

1. Run the following command to display all calendars:

   ```
   ocli model display ws=@
   ```

   The sample output is as follows:

   ```
    HCL Orchestration Command Line Interface
   Licensed Materials - Property of HCL
   (c) Copyright HCL Technologies Ltd. 2019, 2024.
   Workstation Name Type               Domain           Ignored Updated On Locked By
   ---------------- ------------------ ---------------- ------- ---------- ---------------
   LONDON_1   AGENT               -                      02/27/2024 -

   CPUNAME /LONDON_1
     DESCRIPTION "This workstation was automatically created."
     OS UNIX
     NODE LONDON
     SECUREADDR 35114
     TIMEZONE Europe/Rome
     FOR MAESTRO HOST EU-HWS-LNX12_DWB
       AGENTID "C55040CCCA6EM6783A7F930722A5C"
       TYPE AGENT
       SECURITYLEVEL HTTPS
       PROTOCOL https
   END
   ```

Related information

Managing multiple config.yaml files on page 887

draft/undraft

The model command to set one or more job streams in the model in draft mode. You can temporarily disable one or more job streams, and re-enable them when necessary.

Overview

The command to set one or more job streams in the model in draft mode. A draft job stream is not added to the production plan.

Authorization

You must have DRAFT or MODIFY access to the job stream. If you have DRAFT access, this means you can only set the object as draft or non-draft. If you have MODIFY access, you can modify all the properties of the object. For more information about authorization for Orchestration CLI commands, see Access required to run Orchestration CLI commands on page 766.

Syntax and command line options

```
ocli model [draft/undraft] jobstreamselect [;noask]
```

You can enter the command as follows:

jobstreamselect

Specify the job stream name. The job stream must exist in the model; otherwise, an error occurs. You can operate on multiple job streams by using wildcards.

noask

If specified, the draft/undraft option in the job stream is updated without a confirmation prompt.

Examples

Run the following command to temporarily set in draft mode the specific job stream (`JOB_STREAM` on workstation `WKS_AGT`) in the model:

```
OCLI model draft WKS_AGT#JOBSTREAM
```

Run the following command to restore a job stream previously set as draft (`JOB_STREAM` on workstation `WKS_AGT`) to a non-draft state:

```
OCLI model undraft WKS_AGT#JOBSTREAM
```

Run the following command to apply the draft mode to **all job streams** on **all workstations** in the model:

```
ocli model draft @#@.@
```

extract

You can use the command option to create a text file by copying item definitions from the database.

Overview

You can configure the command line syntax with different parameters to create specific item definitions. You can either use `create` or `extract` to copy item definitions. You must have **display** access to the items being copied and **modify** access, if you want to use the **;lock** keyword.

Syntax and command line options

You must provide at least one scheduling item and the *filename* with the command to perform the action. The **;lock** parameter is optional.

```
ocli model create | cr | extract | ext filename from Scheduling item [;lock]
```

For the list of scheduling parameters that you can use and the format that needs to be followed, see the tables below.

Scheduling items

Table 90. Scheduling items

Scheduling items	Description
`domain | dom domainname`	If you provide a specific *domainname*, only that particular domain definition is copied and if not, all the domain definitions in the database are copied into the file. Wildcard characters are permitted.
`folder | fol foldername`	You can specify the name of the folder. If you do not provide the *foldername*, all the folder definitions are copied into the file. Wildcard characters are permitted.
`jobs | jobdefinition | jd [[folder/]workstationame#][folder/]jobname`	If you provide a specific *jobname* only that particular job definition is copied and if not, all the job definitions are copied into the file. You can provide *[folder/]workstationame#]* and the folder in which the job is defined to make the selection more specific. Wildcard characters are permitted.
`sched | jobstream | js [[folder/]workstationame#][folder/]jstreamname[valid from date|valid to date |valid in date date][;full]]`	If you provide a specific *jstreamname* only that particular job stream definition is copied and if not, all the job stream definitions are copied into the file. You can provide *[folder/]workstationame#]* and the folder in which the job stream is defined to make the selection more specific. Wildcard characters are permitted. You can also add the options below to provide more details of the job stream. *Valid from date*: You can specify a date which will restrict the selection of job streams, that have a *valid from* date equal to the value specified. The format is *mm/dd/yyyy*.

Table 90. Scheduling items (continued)

Scheduling items	Description
	Valid to date: You can specify a date which will restrict the selection of job streams, that have a *valid to* date equal to the value specified. The format is *mm/dd/yyyy*. *valid in date date*: Specify the time frame at which the job is run. You can specify one of the dates as `@`. The format is *mm/dd/yyyy-mm/dd/yyyy*. **full**: If you add the option, all the job definitions in the job stream are displayed.
`prompt | pr` *[folder/]promptname*	If you provide a specific *[folder/]promptname* only that particular prompt definition is copied and if not, all the prompt definitions in the database are copied into the file. Wildcard characters are permitted.
`resource | re | reso` *[folder/]resourcename*	If you provide a specific *[folder/]resourcename* only that particular resource definition is copied and if not, all the resource definitions in the database are copied into the file. Wildcard characters are permitted.
`runcyclegroup | rcg` *[folder/]rcgname]*	If you provide a specific *[folder/]rcgname]* only that particular run cycle group definition is copied and if not, all the run cycle group definitions in the database are copied into the file. Wildcard characters are permitted.
`workstation | ws | cpu` *[folder/]workstationame]*	You can specify the name of the workstation or workstation class or domain and optionally, you can provide the folder in which these items are located. Wildcard characters are permitted.
`workstationclass | wsc` *[folder/]workstationclassname]*	If you provide a specific *[folder/]workstationclassname]* only that particular workstation class definition is copied and if not, all the workstation class definitions in the database are copied into the file. Wildcard characters are permitted.

;lock

You can use the option to lock the selected items. If the selected items are locked by any other user, you cannot copy the definitions into the file.

Example

1. Run the following command to create the file `store` with all job streams defined in the database:

```
ocli model cr store from jobstream=@
```

Related information

Managing multiple config.yaml files on page 887

list

You can use the list command to view a list of items of the specified type.

Overview

You can configure the command line syntax with different parameters to view a list of items of the same type. You cannot run the command to view a list of different item types at the same time.

You must have the *list* access for the item to perform the action.

Table 91. Output format for displaying items

items	Details
calendar	• **Calendar Name** • **Updated On** • **Locked By**
domain	• **Domain Name** • **Parent Domain** • **Master** • **Updated On** • **Locked By**
folder	• **Name** • **Updated On** • **Locked By**
job definition	• **Workstation** • **Job Definition Name**

Table 91. Output format for displaying items (continued)

items	Details
	• **Updated On** • **Locked By**
job stream	• **Workstation** • **Job Stream Name** • **Valid From** • **Updated On** • **Locked By**
prompt	• **Name** • **Updated On** • **Locked By**
resource	• **Workstation** • **Resource Name** • **Quantity** • **Updated On** • **Locked By**
run cycle group	• **Run Cycle Group Name** • **Updated On** • **Locked By**
user	• **Workstation** • **Username** • **Updated On** • **Locked By**
variable	• **Variable Table Name** • **Variable Name** • **Updated On** • **Locked By**

Table 91. Output format for displaying items (continued)

items	Details
variable table	• **Variable Table Name** • **Default** • **Updated On** • **Locked By**
workstation	• **Workstation Name** • **Type** • **Domain** • **Ignored** • **Updated On** • **Locked By**
workstation class	• **Workstation Name** • **Type** • **Domain** • **Ignored** • **Updated On** • **Locked By**

Syntax and command line options

You must provide at least one scheduling item with the command to perform the action. The **;showid** parameter is optional.

```
ocli model list | l | li Scheduling item [;showid]
```

You can use the following scheduling parameters to view specific information.

Table 92. Scheduling items

Scheduling items	Description
`calendars | calendar | cal [folder/]calname`	You can specify the name of the calendar and optionally, the folder in which the calendar is defined. If you do not specify the *[folder/]calname*, all the calendar definitions in the database are listed in order.
`domain | dom domainname`	You can specify the *domainname*. If you do not provide the *domainname*, all the domain definitions in the database are listed. Wildcard characters are permitted.

Table 92. Scheduling items (continued)

Scheduling items	Description
`folder | fol` *foldername*	You can specify the name of the folder. If you do not provide the *foldername*, all the folder definitions in the database are listed. Wildcard characters are permitted.
`parms | parm | variable | vb` *[[folder/]tablename.]variablename*	You must provide the *[[folder/]tablename.]variablename* to view a specific variable within a specific table. If you specify the *variablename* only, the variable definition in the default variable is listed. If you do not provide the *[[folder/]tablename.]variablename*, all the global variable definitions in the default variable table are listed. Wildcard characters are permitted on both *[folder/]tablename* and *variablename*.
`prompt | pr` *[folder/]promptname*	You can specify the *[folder/]promptname*. If you do not provide the *[folder/]promptname*, all the prompt definitions in the database are listed. Wildcard characters are permitted.
`resource | re | reso` *[folder/]resourcename*	You can specify the *[folder/]resourcename*. You can provide the *[folder/]workstationname#]* and the folder in which the resource is defined to make the selection more specific. If you do not provide the *[folder/]resourcename*, all the resource definitions in the database are listed. Wildcard characters are permitted.
`runcyclegroup | rcg` *[folder/]rcgname]*	You can specify the *[folder/]rcgname]*. If you do not provide the *[folder/]rcgname]*, all the run cycle group definitions in the database are listed. Wildcard characters are permitted.
`vartable | vt` *[folder/]tablename*	You can specify the name of the variable table and optionally, you can provide the folder in which the variable table is defined. If you do not specify the *tablename* and add wildcard characters, all the variable table definitions are listed. Wildcard characters are permitted.
`jobs | jobdefinition | jd` *[[folder/]workstationame#][folder/]jobname*	If you provide a specific *jobname* only that particular job definition is listed and if not, all the job definitions are listed. You can provide *[folder/]workstationame#]* and the folder in which the job is defined to make the selection more specific. Wildcard characters are permitted.
`sched | jobstream | js` *[[folder/]workstationame#][folder/]jstreamname[valid from date|valid to date |valid in date date][;full]*	If you provide a specific *jstreamname* only that particular job stream definition is listed and if not, all the job stream definitions are listed. You can provide

Table 92. Scheduling items (continued)

Scheduling items	Description
	[folder/]workstationame#] and the folder in which the job stream is defined to make the selection more specific. Wildcard characters are permitted. You can also add the options below to provide more details of the job stream. *Valid from date*: You can specify a date which will restrict the selection of job streams, that have a *valid from* date equal to the value specified. The format is *mm/dd/yyyy*. *Valid to date*: You can specify a date which will restrict the selection of job streams, that have a *valid to* date equal to the value specified. The format is *mm/dd/yyyy*. *valid in date date*: Specify the time frame at which the job is run. You can specify one of the dates as `@`. The format is *mm/dd/yyyy-mm/dd/yyyy*. **full**: If you add the option, all the job definitions in the job stream are listed.
`users | user [[folder/]workstationame#]username [;password]`	If you do not provide any specific *username*, all the usernames in the database are listed. You can provide *[folder/]workstationame#]* and the folder in which the user is defined, to make the selection more specific. Wildcard characters are permitted.
`workstation | ws | cpu [folder/]workstationame]`	You can specify the name of the workstation that you want to view. Optionally, you can provide the folder in which the items are located. Wildcard characters are permitted.
`workstationclass | wsc [folder/]workstationclassname]`	You can specify the *[folder/]workstationclassname]*. If you do not provide the *[folder/]workstationclassname]*, all the workstation class definitions in the database are listed. Wildcard characters are permitted.

;showid

You can specify the parameter with the item workstation and the unique identifier for the item is displayed in the results.

Examples

1. Run the following command to list all calendars:

   ```
   ocli model list calendars @
   ```

2. Run the following command to list workstation `kottayam`, stored in folder `kerala`:

```
ocli model list ws kerala/kottayam
```

Related information

Managing multiple config.yaml files on page 887

listfolder

You can use the command option to view the list of folders available in the database.

Overview

You can use the option to view the list of folders in the database or the folders under a specific folder in the database.

Syntax and command line options

You can run the command as described below.

```
ocli model listfolder | lf foldername
```

foldername: If you want to view the list of folders under a specific folder, you need to specify the name of that folder. If you want to view the list of folders in the root, then specify a forward slash ("/"). If you want to view the list of all folders and sub-folders in the root, then specify a forward slash followed by an ampersand character ("@").

Examples

1. Run the following command to view the list of all folders and sub-folders in the database:

```
ocli model listfolder /@
```

2. Run the following command to view the list of folders under the folder `RITS` in the root:

```
ocli model listfolder /RITS
```

3. Run the following command to view the list of folders and sub-folders under the folder `RITS`:

```
ocli model listfolder /RITS/@
```

Related information

Managing multiple config.yaml files on page 887

lock

You can use the command option to lock access to a scheduling item definition.

Overview

Syntax and command line options

You must provide at least one scheduling item with the command to perform the action.

```
ocli model lock|lo Scheduling item
```

For the list of scheduling parameters that you can use and the format that needs to be followed, see the table below.

Table 93. Scheduling items

Scheduling items	Description
`calendars | calendar | cal [folder/]calname`	You must specify the *[folder/]calname* that you want to lock. Wildcard characters are permitted.
`domain | dom domainname`	You must specify the *domainname* that you want to lock. Wildcard characters are permitted.
`folder | fol foldername`	You must specify the name of the folder that you want to lock. Wildcard characters are permitted.
`jobs | jobdefinition | jd [[folder/]workstationame#][folder/]jobname`	If you provide a specific *jobname* only that particular job definition is locked and if not, all the job definitions are locked. You can provide *[folder/]workstationame#]* and the folder in which the job is defined to make the selection more specific. Wildcard characters are permitted.
`sched | jobstream | js [[folder/]workstationame#][folder/]jstreamname[valid from date|valid to date |valid in date date][;full]]`	If you provide a specific *jstreamname* only that particular job stream definition is locked and if not, all the job stream definitions are locked. You can provide *[folder/]workstationame#]* and the folder in which the job stream is defined to make the selection more specific. Wildcard characters are permitted. You can also add the options below to provide more details of the job stream. *Valid from date*: You can specify a date which will restrict the selection of job streams, that have a *valid from* date equal to the value specified. The format is *mm/dd/yyyy*. *Valid to date*: You can specify a date which will restrict the selection of job streams, that have a *valid to* date equal to the value specified. The format is *mm/dd/yyyy*. *valid in date date*: Specify the time frame at which the job is run. You can specify one of the dates as `@`. The format is *mm/dd/yyyy-mm/dd/yyyy*.

Table 93. Scheduling items (continued)

Scheduling items	Description
	full: If you add the option, all the job definitions in the job stream are displayed for editing.
`prompt | pr [folder/]promptname`	You must specify the *[folder/]promptname* that you want to lock. Wildcard characters are permitted.
`resource | re | reso [folder/]resourcename`	You must specify the *[folder/]resourcename* that you want to lock. You can provide the *[folder/]workstationname#]* and the folder in which the resource is defined to make the selection more specific. Wildcard characters are permitted.
`runcyclegroup | rcg [folder/]rcgname]`	You must specify the *[folder/]rcgname]* that you want to lock. Wildcard characters are permitted.
`users | user [[folder/]workstationname#]username [;password]`	You must specify the *[[folder/]workstationname#]username [;password]* that you want to lock. Wildcard characters are permitted.
`parms | parm | variable | vb [[folder/]tablename.]variablename`	You must specify the *[folder/]tablename* that you want to lock. Wildcard characters are permitted.
`vartable | vt [folder/]tablename`	You must specify the *[[folder/]tablename.]variablename* that you want to lock. Wildcard characters are permitted.
`workstation | ws | cpu [folder/]workstationname]`	You can specify the name of the workstation that you want to lock. Optionally, you can provide the folder in which the items are located. Wildcard characters are permitted.
`workstationclass | wsc [folder/]workstationclassname]`	You must specify the *[folder/]workstationclassname]* that you want to lock. Wildcard characters are permitted.

Examples

1. Run the following command to lock the jobs `my_JOBS`:

   ```
   ocli model lock JOB=my_JOBS
   ```

2. Run the following command to lock the folder `delhi`:

   ```
   ocli model lock folder /delhi
   ```

Related information

Managing multiple config.yaml files on page 887

mkfolder

You can use the command option to create new folders.

Overview

You can create new folders by using the new command also. However, you can create new folders with the mkfolder command and if needed, as sub-folders under existing folders also. You must have **add** access to perform the action.

Syntax and command line options

You must provide a valid name for the folder, before you run the command. You can create the command as described below:

```
ocli model mkfolder|mf foldername
```

or

```
ocli model mkfolder|mf /existing folder/foldername
```

foldername

You must specify a valid folder name.

Examples

1. Run the following command to create a folder `test2`:

   ```
   ocli model mkfolder test2
   ```

2. Run the following command to create a folder `test2` under the existing folder `parent`:

   ```
   ocli model mkfolder /parent/test2
   ```

Related information

Managing multiple config.yaml files on page 887

modify

You can use the command option to modify existing scheduling items in the database.

Overview

The command option performs similar to the new command, when you want to modify an item that is not available in the database. When you run the command, the selected item definition is copied and opened in a temporary file. You can then edit the file and replace the existing item.

Syntax and command line options

You must provide at least one scheduling item with the command to perform the action.

```
ocli model modify|m Scheduling item
```

Important: If you modify an item by changing its key, a new item is created without modifying the existing one. You can use the rename command to modify the item.

For the list of scheduling parameters that you can use and the format that needs to be followed, see the tables below.

Table 94. Scheduling items

Scheduling items	Description
`calendars | calendar | cal [folder/]calname`	You can specify the *[folder/]calname* that you want to modify. If you do not specify a *[folder/]calname*, all the calendar definitions in the database are displayed in a file for editing. Wildcard characters are permitted.
`domain | dom domainname`	You can specify the *domainname* that you want to modify. If you do not specify a *domainname*, all the domain definitions in the database are displayed in a file for editing. Wildcard characters are permitted.
`folder | fol foldername`	You can specify the name of the folder. If you do not provide the *foldername*, all the folder definitions in the database are displayed in a file for editing. Wildcard characters are permitted.
`jobs | jobdefinition | jd [[folder/]workstationame#][folder/]jobname`	If you provide a specific *jobname* only that particular job definition is displayed for editing and if not, all the job definitions are displayed. You can provide *[folder/]workstationame#]* and the folder in which the job is defined to make the selection more specific. Wildcard characters are permitted.
`sched | jobstream | js [[folder/]workstationame#][folder/]jstreamname[valid from date|valid to date |valid in date date][;full]]`	If you provide a specific *jstreamname* only that particular job stream definition is displayed for editing and if not, all the job stream definitions are displayed. You can provide *[folder/]workstationame#]* and the folder in which the job stream is defined to make the selection more specific. Wildcard characters are permitted. You can also add the options below to provide more details of the job stream. *Valid from date*: You can specify a date which will restrict the selection of job streams, that have a *valid from* date equal to the value specified. The format is *mm/dd/yyyy*. *Valid to date*: You can specify a date which will restrict the selection of job streams, that have a *valid to* date equal to the value specified. The format is *mm/dd/yyyy*.

Table 94. Scheduling items (continued)

Scheduling items	Description
	valid in date date: Specify the time frame at which the job is run. You can specify one of the dates as `@`. The format is *mm/dd/yyyy-mm/dd/yyyy*. **full**: If you add the option, all the job definitions in the job stream are displayed for editing.
`prompt | pr [folder/]promptname`	You can specify the *[folder/]promptname* that you want to modify. If you do not specify a *[folder/]promptname*, all the prompt definitions in the database are displayed in a file for editing. Wildcard characters are permitted.
`resource | re | reso [folder/]resourcename`	You can specify the *[folder/]resourcename* that you want to modify. You can provide the *[folder/]workstationame#]* and the folder in which the resource is defined to make the selection more specific. If you do not specify a *[folder/]resourcename*, all the resource definitions in the database are displayed in a file for editing. Wildcard characters are permitted.
`runcyclegroup | rcg [folder/]rcgname]`	You can specify the *[folder/]rcgname]* that you want to modify. If you do not specify a *[folder/]rcgname]*, all the run cycle group definitions in the database are displayed in a file for editing. Wildcard characters are permitted.
`users | user [[folder/]workstationame#]username [;password]`	You can specify the *[[folder/]workstationame#]username [;password]* that you want to modify. Wildcard characters are permitted.
`parms | parm | variable | vb [[folder/]tablename.]variablename`	You can specify the *[[folder/]tablename.]variablename* that you want to modify. Wildcard characters are permitted.
`vartable | vt [folder/]tablename`	You can specify the *[folder/]tablename* that you want to modify. Wildcard characters are permitted.
`workstation | ws | cpu [folder/]workstationame]`	You can specify the name of the workstation that you want to update. Optionally, you can provide the folder in which the items are located. If you do not specify the *workstationame*, all the workstation definitions in the database are displayed in a file for editing. Wildcard characters are permitted.
`workstationclass | wsc [folder/]workstationclassname]`	You can specify the *[folder/]workstationclassname]* that you want to modify. If you do not specify a *[folder/]workstationclassname]*, all the workstation class

Table 94. Scheduling items (continued)

Scheduling items	Description
	definitions in the database are displayed in a file for editing. Wildcard characters are permitted.

Examples

1. Run the following command to edit all the job definitions:

   ```
   ocli model modify job @
   ```

2. Run the following command to modify the job stream `FINAL` located at folder `TEAMS` that is launched on workstation `WS1`:

   ```
   ocli model modify sched WS1#TEAMS/FINAL
   ```

Related information

Managing multiple config.yaml files on page 887

new

You can use the new command to create a new scheduling item definition.

Overview

The new command opens a predefined template in an editor based on the operating system that you run the command on.

On Windows operating systems, the predefined template opens in notepad.

In UNIX operating systems, the predefined template opens in Visual Editor.

You can save the template after you customize the definitions. If you do not want to use the predefined template, run the command without adding an item, which opens an empty file.

Note: When you create a variable in an existing variable table, the variable table is locked and no other user can modify it.

Syntax and command line options

Before you run the command, you must add at least one scheduling item. You can run the command as follows:

```
ocli model new scheduling item
```

You can create the following items and use either the spelled out form or an abbreviation as described below:

- calendar | cal
- domain | dom

- folder | fol
- job | jobdefinition | jd
- jobstream | js
- prompt | pr
- resource | re | reso
- runcyclegroup | rcg
- user
- variable | parms | vb
- vartable | vt
- workstation | ws
- workstationclass | wsc

Examples

1. Run the following command to add a new folder definition:

```
ocli model new folder
```

2. Run the following command to add a new calendar definition:

```
ocli model new job -format yaml
```

Related information

Managing multiple config.yaml files on page 887

nop/unnop

The model command to set one or more jobs or one or more job streams in the model operational or non-operational. It temporarily disables one or more jobs in one or more job streams without changing their dependencies.

Overview

The command updates one or more jobs in one or more job streams without deleting them or changing their dependencies. If the job contains standard dependencies, they are released and the successors are launched. If the dependency is conditional, the condition must be verified as true before the successors run. After the command runs, the specified job changes to **CANCEL PEND** status.

If you apply the command at the job stream level by specifying the job stream name in the Orchestration CLI, the operational or non-operational setting applies to all jobs contained within that job stream. Alternatively, you can specify the name of an individual job to apply the setting only to that specific job within the job stream.

For example, if you need to temporarily skip a notification job due to a service outage, you can nop that job in the job stream until the service is restored.

Authorization

You must have NOOPDB or MODIFY access to the job. If you have NOOPDB access, this means you can only set the object as operational or non-operational. If you have MODIFY access, you can modify all the properties of the object. For more information about authorization for Orchestration CLI commands, see Access required to run Orchestration CLI commands on page 766.

Syntax and command line options

```
ocli model [nop|unnop] [jobselect|jobstreamselect] [;noask]
```

You can enter the command as follows:

jobselect*|*jobstreamselect

Specify the job or job stream name. The job or job stream must exist in the model; otherwise, an error occurs. You can operate on multiple jobs and job streams by using wildcards.

noask

If specified, the nop/unnop option in the job or job stream is updated without a confirmation prompt.

Examples

Run the following command to temporarily disable the specific job (`JOB` inside `JOB_STREAM` on workstation `WKS_AGT`) in the model without deleting it or modifying its dependencies:

```
OCLI model nop WKS_AGT#JOB_STREAM.JOB
```

Run the following command to restore a job previously set as non-operational (`JOB` inside `JOB_STREAM` on workstation `WKS_AGT`) to an operational state:

```
OCLI model unnop WKS_AGT#JOB_STREAM.JOB
```

Run the following command to apply the "no operation" action to **all jobs** across **all job streams** on **all workstations** in the model:

```
ocli model nop /@/@#/@/@.@
```

Run the following command to apply the "no operation" action to `job1`:

```
ocli model nop wks#js.job1
```

Run the following command to apply the "no operation" action to **all jobs** in the JS job stream:

```
ocli model nop wks#js.@
```

Run the following command to apply the "no operation" action to all jobs defined on a specific workstation:

```
ocli model nop wks#@
```

Run the following command to apply the "no operation" action to all jobs in the `JSfolder`:

```
ocli model nop wks#/JSfolder/@
```

rename

You can use the rename command to rename a scheduling item in the database.

Overview

Before you rename an item, make sure no other item with the same name exists in the database. You can use the rename command to move the item to a specified folder at the same time. Specify the folder path before the *new_item_identifier* (example: /folder path/*new_item_identifier*). You must have **delete** and **add** access to the item to perform the action. When you rename a variable, the variable table that contains the variable is locked and no other user can run commands to lock the variable or the table.

Syntax and command line options

Before you run the command, you must specify the name of the item that you want to modify and a new name. The **;preview** parameter is optional.

```
ocli model rename | rn Scheduling item =/old_item_identifier//new_item_identifier/ [;preview]
```

where,

- *old_item_identifier* is the name of the scheduling item.
- *new_item_identifier* is the name for the selected item to be updated.

For the list of scheduling items that you can use and the required format, see the following table.

Table 95. Scheduling items

Scheduling items	Description
`calendars | calendar | cal [folder/]calname`	You can specify the *[folder/]calname* that you want to rename. Wildcard characters are permitted.
`domain | dom domainname`	You can specify the *domainname* that you want to rename. Wildcard characters are permitted.
`folder | fol foldername`	You can specify the *foldername* to rename.
`jobs | jobdefinition | jd [[folder/]workstationame#][folder/]jobname`	You can specify the *jobname* to rename. You can provide the *[folder/]workstationame#]* path and the folder that contains the job definition to make the selection more specific.
`sched | jobstream | js [[folder/]workstationame#][folder/]jstreamname`	You can specify the *jstreamname* job stream to update. You can provide the *[folder/]workstationame#]* path and the folder that contains job stream definition to make the selection more specific.
`prompt | pr [folder/]promptname`	You can specify the *[folder/]promptname* to rename. Wildcard characters are permitted.

Table 95. Scheduling items (continued)

Scheduling items	Description
`resource | re | reso [folder/]resourcename`	You must specify the *[folder/]resourcename* that you want to rename. You can provide the *[folder/]workstationame#]* and the folder in which the resource is defined to make the selection more specific. Wildcard characters are permitted.
`runcyclegroup | rcg [folder/]rcgname]`	You must specify the *[folder/]rcgname]* that you want to rename. Wildcard characters are permitted.
`users | user [[folder/]workstationame#]username [;password]`	You can specify the *[[folder/]workstationame#]username [;password]* that you want to rename. Wildcard characters are permitted.
`parms | parm | variable | vb [[folder/]tablename.]variablename`	You can specify the *[[folder/]tablename.]variablename* that you want to rename. Wildcard characters are permitted.
`vartable | vt [folder/]tablename`	You can specify the *[folder/]tablename* that you want to rename. Wildcard characters are permitted.
`workstation | ws | cpu [folder/]workstationame`	You can specify the name of the workstation to rename. Optionally, you can specify the path of the folder that contains the required items. Wildcard characters are permitted.
`workstationclass | wsc [folder/]workstationclassname]`	You must specify the *[folder/]workstationclassname]* that you want to rename. Wildcard characters are permitted.

;preview

If you specify this optional parameter, the results are displayed without the name changes. You can use this option to check on the impact of commands on items when you plan to rename multiple items.

Example

1. Run the following command to rename the **London** job defined on workstation **UK_WKS_1** in folder **UK** to job **Berlin** defined on workstation **Germany_WKS_1** in folder **Germany**:

```
ocli model rn jobs /UK/UK_WKS_1#London /Germany/Germany_WKS_1#Berlin
```

Related information

Managing multiple config.yaml files on page 887

renamefolder

You can use the renamecommand to rename folder definitions.

Overview

The maximum length for the full folder path which is the name including the path to the parent folder is 1000 characters.

Syntax and command line options

You can run the command as follows:

```
ocli model renamefolder | rnf previousname newname
```

where,

- *previous name* specifies the target folder name.
- *newname*: Specifies the new folder name.

Example

1. Run the following command to rename the RTF1 folder in the root to RTF2:

   ```
   ocli model rename /RTF1 /RTF2
   ```

Related information

Managing multiple config.yaml files on page 887

replace

You can use the replace command to replace the existing scheduling item definitions in the database.

Overview

The replace command is similar to add command in some scenarios. When you run the command, if the item definition is in the database, then the definition is replaced by the new one and if not, a new item definition is created in the database. The replace command checks for loop dependencies. For example, if `JOB1` follows `JOB2` and `JOB2` follows `JOB1`, there is a loop dependency in the job stream.

To replace an existing item, you must have *modify* access. To add a new item, you must have the *add* access.

Syntax and command line options

Before you run the command, you must specify the name of the file that contains the item definition. You can run the command as follows:

```
ocli model replace|rep filename [;unlock]
```

where, *filename* is the name of the file that contains the item definition.

;unlock

You must use the option, if the existing item that you want to update is locked by other users. To perform this action, you must have the *modify* and *unlock* access to the item you want to update.

Important: If you replace an item by changing its key, a new item is created without modifying the existing one.

Important: Definitions created in YAML or JSON format using older HCL Workload Automation versions cannot be imported. To add such definitions, create them as new.

Example

1. Run the following command to replace the job definitions from the file `goodjob` file:

```
ocli model replace goodjob
```

Related information

Managing multiple config.yaml files on page 887

rmfolder

You can use the rmfolder command to delete folders in the database.

Overview

You can use the command to delete folders and sub-folders that does not contain scheduling items. If not, the command is returned with an error. To remove a folder, you must unlock it.

Syntax and command line options

You can run the rmfolder command as follows:

```
ocli model rmfolder | rf foldername
```

foldername: If you want to delete the folders under a specific folder, you must specify the name of that folder. You can use the forward slash before the *foldername* variable, to delete a folder in the root. To delete all folders and sub-folders in the database, specify a forward slash followed by an ampersand "/@".

Examples

1. Run the following command to delete the `RITS` folder:

```
ocli model rmfolder RITS
```

2. Run the following command to delete the `RITS` folder located at the root:

```
ocli model rmfolder /RITS
```

3. Run the following command to delete all folders and sub-folders located at the root:

```
ocli model rmfolder /@
```

Related information

Managing multiple config.yaml files on page 887

unlock

You can use the command option to unlock scheduling items that are locked.

Overview

You can unlock items that you have previously locked. You must have the unlock access to the scheduling items.

Syntax and command line options

You must provide at least one scheduling item with the command to perform the action. You can create the command as described below.

```
ocli model unlock|un Scheduling item [;forced]
```

For the list of scheduling parameters that you can use and the format that needs to be followed, see the table below.

Table 96. Scheduling items

Scheduling items	Description
`calendars | calendar | cal [folder/]calname`	You must specify the *[folder/]calname* that you want to unlock. Wildcard characters are permitted.
`domain | dom domainname`	You must specify the *domainname* that you want to unlock. Wildcard characters are permitted.
`folder | fol foldername`	You must specify the name of the folder that you want to unlock. Wildcard characters are permitted.
`jobs | jobdefinition | jd [[folder/]workstationame#][folder/]jobname`	If you provide a specific *jobname* only that particular job definition is locked and if not, all the job definitions are unlocked. You can provide *[folder/]workstationame#]* and the folder in which the job is defined to make the selection more specific. Wildcard characters are permitted.
`sched | jobstream | js [[folder/]workstationame#][folder/]jstreamname[valid from date|valid to date |valid in date date][;full]]`	If you provide a specific *jstreamname* only that particular job stream definition is unlocked and if not, all the job stream definitions are unlocked. You can provide *[folder/]workstationame#]* and the folder in which the job stream is defined to make the selection more specific. Wildcard characters are permitted. You can also add the options below to provide more details of the job stream.

Table 96. Scheduling items (continued)

Scheduling items	Description
	Valid from date: You can specify a date which will restrict the selection of job streams, that have a *valid from* date equal to the value specified. The format is *mm/dd/yyyy*. *Valid to date*: You can specify a date which will restrict the selection of job streams, that have a *valid to* date equal to the value specified. The format is *mm/dd/yyyy*. *valid in date date*: Specify the time frame at which the job is run. You can specify one of the dates as `@`. The format is *mm/dd/yyyy-mm/dd/yyyy*. **full**: If you add the option, all the job definitions in the job stream are displayed for editing.
`prompt | pr` *`[folder/]promptname`*	You must specify the *[folder/]promptname* that you want to unlock. Wildcard characters are permitted.
`resource | re | reso` *`[folder/]resourcename`*	You must specify the *[folder/]resourcename* that you want to unlock. You can provide the *[folder/]workstationname#]* and the folder in which the resource is defined to make the selection more specific. Wildcard characters are permitted.
`runcyclegroup | rcg` *`[folder/]rcgname]`*	You must specify the *[folder/]rcgname]* that you want to unlock. Wildcard characters are permitted.
`users | user` *`[[folder/]workstationname#]username [;password]`*	You must specify the *[[folder/]workstationname#]username [;password]* that you want to unlock. Wildcard characters are permitted.
`parms | parm | variable | vb` *`[[folder/]tablename.]variablename`*	You must specify the *[[folder/]tablename.]variablename* that you want to unlock. Wildcard characters are permitted.
`vartable | vt` *`[folder/]tablename`*	You must specify the *[folder/]tablename* that you want to unlock. Wildcard characters are permitted.
`workstation | ws | cpu` *`[folder/]workstationname]`*	You can specify the name of the workstation that you want to unlock. Optionally, you can provide the folder in which the items are located. Wildcard characters are permitted.
`workstationclass | wsc` *`[folder/]workstationclassname]`*	You must specify the *[folder/]workstationclassname]* that you want to unlock. Wildcard characters are permitted.

forced

You can use this option to unlock item definitions that are locked by your user id, regardless of the session.

Examples

1. Run the following command to unlock the job definition `way2`:

   ```
   ocli model unlock jd=@#way2
   ```

2. Run the following command to unlock the folder `delhi`:

   ```
   ocli model unlock folder /delhi
   ```

Related information

Managing multiple config.yaml files on page 887

Plan commands

You can use the various command options in Orchestration CLI to achieve specific results. The command options are not case sensitive and you can use either uppercase or lowercase letters.

The commands that you can use in Orchestration CLI have abbreviated forms. You can either use the abbreviation or the spelled-out form to complete the task. You can use the following list of commands in Orchestration CLI:

- ADDDEP JOB on page 809
- ADDDEP SCHED on page 810
- ALTJOB on page 812
- ALTPRI on page 814
- CANCEL JOB on page 815
- CANCEL SCHED on page 817
- change resource on page 822
- CONFIRM on page 819
- DELDEP JOB on page 823
- DELDEP SCHED on page 824
- FENCE on page 825
- KILL on page 826
- LIMIT CPU on page 827
- LIMIT SCHED on page 828
- LISTFOLDER on page 828
- nop/unnop on page 829
- RELEASE JOB on page 829
- RELEASE SCHED on page 830
- reply on page 831
- rerun on page 833
- showcpu on page 835

- showfile on page 840
- showjobs on page 846
- showprompt on page 860
- show resource on page 866
- showschedules on page 871
- SUBMIT DOCOMMAND on page 876
- SUBMIT JOB on page 878
- SUBMIT SCHED on page 879

Related information

Configuring Orchestration CLI on page 757

Running commands from Orchestration CLI on page 762

ADDDEP JOB

You can use the adddep command to add dependencies to a job.

Overview

To add the dependencies to a job, you must have *adddep* access to the job. For more information about security access roles, see the section about security role definition in Administration Guide.

Syntax and command line options

Before you run the command, you must specify options such as job name, dependencies, and required permissions. you can enter the command as follows:

```
ocli plan adddep job | adj = <jobselect> <dependency> <noask>
```

You can add the dependency according to the specific results you want to accomplish. The details that are required with each parameters are as follows:

Jobselect

Specify the job to update, by means of attributes and qualifiers.

For example, `WA_AGT_0#JOB_STREAM.JOB`.

Dependency

Specify the type of dependency and add at least one to complete the command. The type of dependencies that can be used to create the command line are listed in the topic about the adddep job command in *User's Guide and Reference*.

noask

When you add the option as an argument the agent will not ask for confirmation before taking action on each qualifying job.

Notes:

1. If you add twice a dependency on a job stream to a job, both dependencies are treated.
2. When using the **deadline** keyword, ensure the **bm check deadline** option is set to a value higher than 0 in the `localopts` configuration file on the workstation you are working on. You can define the **bm check deadline** option on each workstation on which you want to be aware of the deadline expiration, or, if you want to obtain up-to-date information about the whole environment, define the option on the master domain manager. Deadlines for critical jobs are evaluated automatically, independently of the **bm check deadline** option.
3. If you add a dependency to a job after it has completed, it is not evaluated. However, any subsequent reruns of the job will process the dependency correctly.

Restictions:

1. Wildcard characters are not permitted.
2. If you do not specify a value for `priority`, the job reverts to its original scheduled priority.
3. If you do not specify a workstation in `follows`, `needs`, or `opens`, the default is the workstation on which the job runs.

Examples

1. To add an external **follows** dependency on job `JOB022` in job stream `MLN#/FOLDER1/SCHED_02(0600 02/24/23)` from `JOBA` in job stream `MLN#/FOLDER2/NEW_TEST(0900 02/19/23)`, run the following command:

   ```
   ocli plan adj=MLN#/FOLDER2/NEW_TEST(0900 02/19/23).JOBA ;
   follows MLN#/FOLDER1/SCHED_02(0600 02/24/23).JOB022
   ```

2. To stop the job `PAYROLL_JOB` in job stream `ABSENCES_JS` when it is completed more than 9 hours and 1 minute, run the following command:

   ```
   ocli plan adj DUBAI#ABSENCES_JS.PAYROLL_JOB ;
   maxdur=901 ;onmaxdur kill
   ```

ADDDEP SCHED

You can use the **adddep** command to add dependencies to a job stream.

Overview

To add the command option to make changes in a job stream, you must have *adddep* access to the job stream. For more information about security access roles, see the section about security role definition in Administration Guide.

Syntax and command line options

Before you run the command, you must specify the options such as job stream name, dependencies, and required permissions. you can create the command as described below.

```
ocli plan adddep sched | ads = <jstreamselect> ><dependency><noask>
```

You can add the dependency according to the specific results you want to accomplish. The details that are required with each parameters are as follows:

jstreamselect

Specify the job stream to update, by means of attributes and qualifiers. For more information, see to the section about selecting job streams in commands in *User's Guide and Reference*.

Dependency

Specify the type of dependency and add at least one to complete the command. The type of dependencies that you can use to create the command line are listed in the topic about the adddep sched command in *User's Guide and Reference*.

noask

When you add the option as an argument the agent will not to ask for confirmation before taking action on each qualifying job.

Notes:

1. If you add a dependency on a job stream to another job stream twice, only one dependency is considered.
2. When using the **deadline** keyword, ensure the **bm check deadline** option is set to a value higher than 0 in the `localopts` configuration file on the workstation you are working on. You can define the **bm check deadline** option on each workstation on which you want to be aware of the deadline expiration, or, if you want to obtain up-to-date information about the whole environment, define the option on the master domain manager. Deadlines for critical jobs are evaluated automatically, independently of the **bm check deadline** option.
3. If you add a dependency to a job after it has completed, it is not evaluated. However, any subsequent reruns of the job will process the dependency correctly.

Restictions:

1. Wildcard characters are not permitted.
2. If you do not specify a workstation in `follows`, the default is the workstation on which the job runs.

Examples

1. To add an external **follows** dependency to job `JOBB` in job stream `CPUA#SCHED_02(0600 02/24/06)` to job stream `CPUA#TEST(0900 02/19/06)`, run the following command:

```
ocli plan ads=CPUA#TEST(0900 02/19/06) ; follows
```

ALTJOB

You can use the command option to modify a job in plan before it starts running.

Overview

Use the command to make changes to the job definition in plan and start the run. The updated job definition start separately and you can keep the original definition in the database. You must have the *submit* access to the job. For more information about security access roles, see the section about security role definition in Administration Guide.

Restriction: When the command is submitted, the agent will not ask for your confirmation before taking action on each job.

Syntax and command line options

Before you run the command, specify options such as job stream name, dependencies, and required permissions. You can enter the command as follows:

```
ocli plan altjob | aj = <jobselect><streamlogon|logon=new_logon><docommand="new_command">
<script="new_script"><noask>
```

The details that are required with each parameter are as follows:

Important: The **docommand** and **script** parameters are mutually exclusive with each other and cannot not be used together.

Note: If you edit the definition of a job in plan that contains variables, the job completes the run, but the variables are not resolved with their value.

jobselect

Specify the job to update, by means of attributes and qualifiers.

For example, `WA_AGT_0#JOB_STREAM.JOB`.

streamlogon|logon=new_logon

You can specify a new user name to run the selected job under.

docommand="new_command"

You can specify a new command that the job must run in place of the original command.

script="new_script"

You can specify a new script that the job must run in place of the original script.

noask

When you add the option as an argument the agent will not ask for confirmation before taking action on each qualifying job.

altpass

You can use the altpass command to change the password of a user.

Overview

You can use the altpass command to change the password of a user specified in the credential definition. The altpass command is equivalent to the following command that you can use to update the password.

```
ocli model modify [credential | user] username
```

Syntax and command line options

You can enter the command as follows:

```
ocli plan altpass [[folder/]workstation#] username [;password]
```

The details that are required with each parameters are as follows:



Note: The **[folder/]workstation#** parameter is optional and the **username** and **password** parameters are mandatory.

[folder/]workstation#

Specify the workstation where the user is defined. Optionally, you can provide the folder on your workstation. You must use uppercase letters when you add the workstation name, even if the credential definition includes mixed case letters.

username

Add the username specified in the credential definition. The username is case-sensitive.

password

You can specify the new password.

Examples

1. Run the following command to change the password of a credential, whose ID is `fen` to `myoldpw`. The ID is defined on the workstation `mrt5` workstation, in the `atfoder` folder.

   ```
   ocli plan altpass atfolder/MRT5#fen;"myoldpw"
   ```

2. Run the following command to change the password of a credential whose ID is `fen` to `myoldpw` on the `mrt5` workstation without displaying the password.

   ```
   ocli plan altpass MRT5#fen
   password: xxxxxxxx
   confirm: xxxxxxxx
   ```

3. Run the following command to change the password of a credential whose ID is `fen` to `myoldpw`. The ID is defined in an active directory that is managed by Windows domain named `trtson`.

   ```
   ocli plan altpass TRTSON\FEN;"myoldpw"
   ```

Related information

Managing multiple config.yaml files on page 887

ALTPRI

You can use the altpri command to change the priority for a job or job stream.

Overview

The default priority value of a job in plan or a job stream in plan is 10. You can use the altpri command to manage the priority of a job or job stream. This gives you an additional control over the process and order of job or job stream run. You must have *altpri* access on the job or job stream to run the command.

Syntax and command line options

You can enter the command as follows:

```
ocli plan (altpri|ap) (jobSelect/jobstreamSelect) priority noask
```

The *jobSelect/jobstreamSelect* and *priority* parameters are mandatory. The details required with each parameter are as follows:

jobSelect/jobstreamSelect

Specify the name of the job or job stream to set the priority, by means of attributes and qualifiers.

Priority

Specify a value to set as priority for the job or job stream. You can use the following values:

- Numerals 0 to 101
- `hi` (Equivalent to 100)
- `go` (Equivalent to 101)

Important: When the priority is set to `hi` or `go`, the job or job stream runs on the workstation even if the workstation limit is exceeded.

noask

When you add the option as an argument the agent will not ask for confirmation before taking action on each qualifying job.

Example

1. To set the priority of `OCTWON` job in `MONTHCRIC` job stream as `go`, run the following command:

```
ocli plan altpri MONTHCRIC.OCTWON go
```

CANCEL JOB

You can use the cancel command to cancel a job.

Overview

As the name suggests the cancel command is used to cancel a job. If you cancel a job before it is launched, the job is not launched. When you cancel a job after it is launched, it continues to run. If you cancel a job that is running and it completes in the ABEND state, no automatic job recovery steps are attempted to relaunch the job.

Restriction: When the command is submitted, the agent will not ask for your confirmation before taking action on each job.

To run the cancel command, you must have the *cancel* access to the job. For more information about security access roles, see the section about security role definition in Administration Guide.

Syntax and command line options

Before you run the command, you can specify the different options to customize the result. The command line parameters **pend** and **noask** are optional. This allows you to complete a command using the single parameter **jobselect**. You can create the command as described below.

```
ocli plan cancel job | cj = <jobselect><pend><noask>
```

You can add the option `pend` to resolve the dependencies associated with the job, before the job is cancelled. The details that are required with each parameters are as follows:

jobselect

Specify the job to update, by means of attributes and qualifiers.

For example, `WA_AGT_0#JOB_STREAM.JOB`.

pend

You can use the **pend** option to remove the dependencies that are associated with the job that you want to cancel. If you do not use the **pend** option, jobs and job streams that are dependent on the cancelled job are released immediately from the dependency. The different scenarios that are related to the **pend** option are as follows:

Action	Job status	Result
pend added	Not started	The job is not cancelled until all the associated dependencies are resolved. After the job is cancelled, the jobs and job streams that are related to the cancelled job are removed from the dependency. **Note:** While the cancel action is postponed, the notation `Cancel Pend` is listed in the **Dependencies** column of the job in a **showjobs** display.
pend added	started	The option is ignored and any job or job streams that are related to the cancelled job are removed from the dependency.

Notes:

1. You can use the **rerun** command to rerun jobs that are cancelled, or that are marked `Cancel Pend`. You can also add and delete dependencies on jobs that are marked `Cancel Pend`.
2. You can immediately cancel a job that is marked `Cancel Pend`, either enter a **release** command for the job or enter another cancel command without the **pend** option.
3. For the jobs with expired **until** parameter, the notation `Until` is listed in the **Dependencies** column in a **showjobs** display, and their dependencies are no longer evaluated. If such a job is also marked `Cancel Pend`, it

is not cancelled until you release or delete the **until** time, or enter another cancel command without the **pend** option. To stop evaluating the options added along with the job, set the priority of a job to zero with the **altpri** command. To resume dependency evaluation, set the priority to a value greater than zero.

noask

When you add the option as an argument the agent will not ask for confirmation before taking action on each qualifying job.

Note:

In the case of internetwork dependencies, cancelling a job in the `EXTERNAL` job stream releases all local jobs and job streams from the dependency. Jobs in the `EXTERNAL` job stream represent jobs and job streams that have been specified as internetwork dependencies. The status of an internetwork dependency is not checked after a cancel is performed.

Examples

1. To cancel job report in job stream `apwkly(0900 02/19/06)` on workstation `site3`, run the following command:

   ```
   ocli plan cj=site3#apwkly(0900 02/19/06).report
   ```

2. To cancel job setup in job stream `mis5(1100 02/10/06)`, if it is not in the `ABEND` state, run the following command:

   ```
   ocli plan cj mis5(1100 02/10/06).setup~state=abend
   ```

3. To cancel job `job3` in job stream `sked3(0900 02/19/03)` only after its dependencies are resolved, run the following command:

   ```
   ocli plan cj sked3(0900 02/19/06).job3;pend
   ```

CANCEL SCHED

You can use the cancel command to cancel a job stream.

Overview

If you cancel a job stream before it starts, the job stream does not run. When you cancel a job stream after it starts, the jobs in the job stream that are started will complete. The remaining jobs from the job stream are cancelled.

To run the cancel command, you must have the *cancel* access to the job. For more information about security access roles, see the section about security role definition in Administration Guide.

Restriction: When the command is submitted, the agent will not ask for your confirmation before taking action on each job.

Syntax and command line options

Before you run the command, you can specify the different options to customize the result. The **pend** and **noask** parameters are optional. This allows you to complete a command using only the **jstreamselect** parameter. You can create the command as follows:

```
ocli plan cancel sched | cs = <jstreamselect><pend><noask>
```

You can add the option `pend` to resolve the dependencies associated with the job, before the job is cancelled. The details that are required with each parameters are as follows:

jstreamselect

Specify the job stream to update, by means of attributes and qualifiers. For more information, see to the section about selecting job streams in commands in *User's Guide and Reference*.

pend

You can use the **pend** option to remove the dependencies that are associated with the job stream which you want to cancel. If you do not use the **pend** option, jobs and job streams that are dependent on the cancelled job are released immediately from the dependency. The different scenarios that are related to the **pend** option are described below.

Action	Job stream status	Result
pend added	Not launched	The job stream is not cancelled until all the associated dependencies are resolved. After the job stream is cancelled, the jobs and job streams that are related to the cancelled job stream are removed from the dependency. **Note:** While the cancel action is postponed, the notation `Cancel Pend` is listed in the **Dependencies** column of the job in a **showschedules** display.
pend added	Launched	The jobs that are not started is cancelled and any job or job streams that are related are removed from the dependency.

Notes:

1. You can immediately cancel a job stream that is marked `Cancel Pend`, either enter a **release** command for the job stream or enter another cancel command without the **pend** option.
2. If you want to stop evaluating the dependencies that are added along with the job stream, set the priority of the job stream to zero with the **altpri** command. To resume dependency evaluation, set the priority to a value greater than zero.
3. If the cancelled job stream contains jobs defined with the **every** option, only the last instance of such jobs is listed as canceled in a **showjobs** display.

noask

When you add the option as an argument the agent will not ask for confirmation before taking action on each qualifying job.

Note:

In the case of internetwork dependencies, cancelling a job in the `EXTERNAL` job stream releases all local jobs and job streams from the dependency. Jobs in the `EXTERNAL` job stream represent jobs and job streams that have been specified as internetwork dependencies. The status of an internetwork dependency is not checked after a cancel is performed. For more information, refer to Managing internetwork dependencies in the plan.

Examples

1. To cancel job stream `sked1(1200 02/17/23)` on workstation `site2`, run the following command:

   ```
   ocli plan cs=site2#sked1(1200 02/17)
   ```

2. To cancel job stream `mis2(0900 02/19/23)` if it is in the STUCK state, run the following command:

   ```
   ocli plan cs mis2(0900 02/19)+state=stuck
   ```

CONFIRM

You can use the confirm command to validate the status of a job after completion.

Overview

The command is used to change the status of a job after it completes the run. When a job is finished with the status ABEND, you can use the confirm command to change the status to SUCC, so that any other depended jobs can start the run. It is also possible for a job to remain incomplete, either successfully or unsuccessfully. These jobs are finished as DONE and not SUCC or ABEND. The confirm command can be used to change the status of these jobs to either SUCC or ABEND, so that the workflow is not interrupted. The command option can also be used when there is a delay for the job to finish.

When you use the command option while the job is running, tthe confirmation overrides the evaluation status when the job is finished. Based on the (SUCC or ABEND) status that you provide with the command, the jobs finish with either SUCC or ABEND.

You can also override the evaluation of the output conditions. For example, if you set one or more output conditions to true (by using confirm SUCC), the specified output conditions are set to true and any other conditions in the job are set to false. You must have *confirm* access to the job. For more information about security access roles, see the section about security role definition in Administration Guide.

Restriction: When the command is submitted, the agent will not ask for your confirmation before taking action on each job.

Syntax and command line options

Before you run the command, you must specify options such as job name, arguments, and required permissions. You can enter the command as follows:

```
ocli plan confirm | conf = <jobselect><succ | abend><IF 'output_condition_name[, output_condition_name]
     [, ...]']><noask>
```

The details that are required with each parameters are described below.

Arguments

jobselect

Specify the job to update, by means of attributes and qualifiers.

For example, `WA_AGT_0#JOB_STREAM.JOB`.

`SUCC`

To confirm that the job ended successfully.

`ABEND`

To confirm that the job ended unsuccessfully.

output_condition_name

To confirm the SUCC or ABEND status for one or more specified output conditions. The conditions which are not specified are set to false. This setting overrides any other evaluation.

noask

When you add the option as an argument the agent will not ask for confirmation before taking action on each qualifying job.

Change in status after confirm command

Table 97. Change in status after confirm command

Initial job status	Status after `confirm ;succ`	Status after `confirm ;abend`
READY	No effect, with or without output conditions	No effect, with or without output conditions.

Table 97. Change in status after confirm command (continued)

Initial job status	Status after `confirm ;succ`	Status after `confirm ;abend`
HOLD	No effect, with or without output conditions	No effect, with or without output conditions.
EXEC	SUCC_P, with or without output conditions	ABEND_P, with or without output conditions
ABENP	SUCCP, with or without output conditions	**without output conditions** No effect. **with output conditions** ABENP and selected output conditions are set to satisfied.
SUCCP	No effect, with or without output conditions	No effect, with or without output conditions.
PEND	**without output conditions** SUCC **with output conditions** SUCC and selected output conditions are set to satisfied.	**without output conditions** ABEND **with output conditions** ABEND and selected output conditions are set to satisfied.
DONE	SUCC, with or without output conditions.	ABEND, with or without output conditions.
SUCC	**without output conditions** The operation is not supported. **with output conditions** SUCC and selected output conditions are set to satisfied.	**without output conditions** The operation is not supported. **with output conditions** The operation is not supported.
ABEND	SUCC, with or without output conditions.	**without output conditions** No effect. **with output conditions** ABEND and selected output conditions are set to satisfied.

Table 97. Change in status after confirm command (continued)

Initial job status	Status after `confirm ;succ`	Status after `confirm ;abend`
SUPPR	**without output conditions** SUCC **with output conditions** SUCC and selected output conditions are set to satisfied.	The operation is not supported, with or without output conditions.
FAIL	The operation is not supported, with or without output conditions.	The operation is not supported, with or without output conditions.
SCHED	No effect, with or without output conditions	No effect, with or without output conditions
ERROR (for shadow jobs only)	SUCC, with or without output conditions	ABEND, with or without output conditions
any job in the **EXTERNAL** job stream	SUCC, with or without output conditions	ABEND, with or without output conditions

Examples

1. To issue a **succ** confirmation for job `job3` in job stream `misdly(1200 02/17/23)`, run the following command:

   ```
   ocli plan confirm misdly(1200 02/17/23).job3;succ
   ```

2. To issue an **abend** confirmation for job number **234**, run the following command:

   ```
   ocli plan confirm 234;abend
   ```

3. To issue a **succ** confirmation for job `job4` and set `MYOUTPUTCOND` to true in the `daily(1130 02/17/2023) job stream`, run the following command:

   ```
   ocli plan confirm daily(1130 02/17/2023).job4;succ if MYOUTPUTCOND
   ```

change resource

You can use the changeresource command to update the quantity of a resource.

Overview

You can either increase or decrease the quantity of a single resource or else all the resources at the same time. The maximum number of units that you can add to a single resource is 2147483647.

Syntax and command line options

You must provide a quantity to run the command. You can run the command as follows:

```
ocli plan [change resource | cr | resource] [[/workstationfolder/workstationname#/foldername/resourcename] |
 [workstationname#/foldername/resourcename] | [/foldername/resourcename] | resourcename] quantity ;noask
```

resourcename

Enter the resource key to display a specific definition or use wildcard characters to display all the resource definitions at the same time. If the resource is defined in the root folder, specify only the name of the resource. Otherwise add the resource key that contains the workstation name and folder name as follows:

```
[[/workstationfolder/workstationname#/foldername/resourcename] | [workstationname#/foldername/resourcename] |
 [/foldername/resourcename] | resourcename]
```

Quantity

Specify any positive integer as quantity. You can enter any value up to 2147483647 units.

noask

When you add the **noask** option as an attribute, the agent does not ask for confirmation before it acts on each qualifying item.

Examples

1. To update the quantity of `SRC1` to `100`, run the following command.

   ```
   ocli plan cr SRC1 100
   ```

2. To update the quantity of all resources to `100`, run the following command.

   ```
   ocli plan cr @#@ 100
   ```

DELDEP JOB

You can use the **deldep job**command to delete the dependencies from a job.

Overview

To remove the dependencies from a job, you must have the *deldep* access to the job. For more information about security access roles, see the section about security role definition in Administration Guide.

Syntax and command line options

Before you run the command, you must specify options such as job name, dependencies, and required permissions. You can enter the command as follows:

```
ocli plan deldep job | ddj = <jobselect><dependency><noask>
```

You can add the dependency according to the specific results you want to accomplish. The details that are required with each parameters are as follows:.

jobselect

Specify the job to update, by means of attributes and qualifiers.

For example, `WA_AGT_0#JOB_STREAM.JOB`.

dependency

Specify the type of dependency and you must add at least one to complete the command. You can use wildcard characters in `workstation`, `jstream`, `job`, `resource`, `filename`, and `promptname`. When you **rerun** the job, the deleted dependencies are not considered.

Restriction: Do not use a wildcard character when you want to delete all the `follow` dependencies from the jobs contained in a specific job stream.

```
follows=job_stream_name.@
```

The command will be rejected.

The type of dependencies that can be used to create the command line are listed in the topic about the deldep job in *User's Guide and Reference*.

noask

When you add the option as an argument the agent will not ask for confirmation before taking action on each qualifying job.

Examples

1. To delete all external **follows** dependency from job stream `CPUA#TEST(0900 02/19/06)`, run the following command:

```
ocli plan ddj=CPUA#TEST(0900 02/19/06).JOBA ; follows
```

DELDEP SCHED

You can use the deldep schedcommand to delete the dependencies from a job stream.

Overview

To remove the dependencies from a job stream, you must have the *deldep* access to the job stream. For more information about security access roles, see the section about security role definition in Administration Guide.

Syntax and command line options

Before you run the command, you must specify options such as job stream name, dependencies, and required permissions. You can enter the command as follows:

```
ocli plan deldep sched  | dds = <jstreamselect><dependency><noask>
```

You can add the dependency according to the specific results you want to accomplish. The details that are required with each parameters are as follows:

jstreamselect

Specify the job stream to update, by means of attributes and qualifiers. For more information, see to the section about selecting job streams in commands in *User's Guide and Reference*.

dependency

Specify the type of dependency and you must add at least one to complete the command. You can use wildcard characters in `workstation`, `jstreamname`, `jobname`, `resource`, `filename`, and `promptname`. When you **rerun** the job stream, the deleted dependencies are not considered.

Notes:

1. When you delete the dependency `priority`, the job reverts to its original scheduled priority.
2. When you delete an `opens` dependency, you can only include the base file name and the Orchestration CLI performs a case-insensitive search for matching files, ignoring the directory names. The dependencies on all the matching files are deleted.

Restriction: Do not use wildcard characters when you want to delete the `follows` dependency. The command will be rejected.

The type of dependencies that can be used to create the command line are listed in the topic about the deldep sched in *User's Guide and Reference*.

noask

When you add the option as an argument the agent will not ask for confirmation before taking action on each qualifying job.

Examples

1. To delete all **follows** dependencies from job stream `sked3(1000 04/19/06)`, run the following command:

```
ocli plan dds sked3(1000 04/19/06);follows
```

FENCE

You can use the fence command to set the priority for a workstation to filter jobs.

Overview

The command sets a priority for the workstation. The jobs with a higher priority run on the workstation. The feature helps you to run only the high priority jobs in low priority job streams and suppress the low priority jobs in high priority job streams. You must have *fence* access on the workstation. The default fence value for a workstation is zero. You can use the show cpu command to view the current fence value of a workstation.

Syntax and command line options

You can enter the command as follows:

```
ocli plan (fence|f) [folder/]workstation priority noask
```

The *[folder/]workstation* and *priority* parameters are mandatory. The details required with each parameter are as follows:

[folder/]workstation

You can specify the name of the specific workstation to set the limit, by means of attributes and qualifiers. You can also specify the folder that contains the workstation definition. Specifying the folder is optional.

Priority

Specify a value to set as priority for the workstation. You can use the following values:

- Numerals 0 to 101
- `hi` (Equivalent to 100)
- `go` (Equivalent to 101)
- `system` (Equivalent to zero)

noask

When you add the option as an argument the agent will not ask for confirmation before taking action on each qualifying job.

Example

1. To change fence on `WA_AGT_0` workstation as 10, run the following command:

```
ocli plan fence WA_AGT_0 20
```

KILL

You can use the kill command to stop a job in progress.

Overview

When you use the kill command, the action is performed by HCL Workload Automation production process, so there might be a short delay.

The jobs that are stopped by the kill command are finished in the ABEND state. Any jobs or job streams that are dependent on these jobs are not released and you can rerun them.

Restriction: When the command is submitted, the agent will not ask for your confirmation before taking action on each job.

Syntax and command line options

Before you run the command, you must specify options such as job stream name, dependencies, and required permissions. you can enter the command as follows:

```
ocli plan kill | k = <jobselect><noask>
```

The details that are required with each parameters are as follows:

jobselect

Specify the job to update, by means of attributes and qualifiers.

For example, `WA_AGT_0#JOB_STREAM.JOB`.

noask

When you add the option as an argument the agent will not ask for confirmation before taking action on each qualifying job.

Examples

1. To stop the job `report` in job stream `apwkly(0600 03/05/06)` on workstation `site3`, run the following command:

   ```
   ocli plan kill site3#apwkly(0600 03/05/06).report
   ```

2. To stop the job number `124` running on workstation `geneva`, run the following command:

   ```
   ocli plan kill geneva#124
   ```

LIMIT CPU

You can use the limit cpu command to set the number of jobs that can run simultaneously on a workstation.

Overview

The default job limit depends on the type of workstation. For cloud task launcher, the default value is 1000 and for all other types of workstations the value is 100. You can use the limit cpu command to update the limit, and you must have limit access to run the command. Use the show cpu command to view the current limit of a workstation.

Syntax and command line options

You can enter the command as follows:

```
ocli plan (limit cpu|lc) [folder/]workstation limit noask
```

The *[folder/]workstation* and *limit* parameters are mandatory. The details required with each parameter are as follows:

[folder/]workstation

You can specify the name of the specific workstation to set the limit, by means of attributes and qualifiers. You can also specify the folder that contains the workstation definition. Specifying the folder is optional.

Limit

Specify the number of jobs that you want to run simultaneously on a workstation. There is no maximum limit, and you can specify any positive numerical value. If you do not want to set any specific limits, use the term `system`.

Important: If you set the limit to `zero`, only the jobs with `hi` or `go` priority run on the workstation.

noask

When you add the option as an argument the agent will not ask for confirmation before taking action on each qualifying job.

Example

1. To set 10 jobs as a limit on `WA_AGT_0` as a workstation limit, run the following command:

```
ocli plan limit cpu WA_AGT_0 10
```

LIMIT SCHED

You can use the limit sched command to update the limit set in the definition of a job stream.

Syntax and command line options

Before you run the command, you must specify options such as job name, dependencies, and required permissions. You can enter the command as follows:

```
ocli plan limit sched | ls = <jstreamselect><limit><noask>
```

The details that are required with each parameters are as follows:

jstreamselect

Specify the job stream to update, by means of attributes and qualifiers. For more information, see to the section about selecting job streams in commands in *User's Guide and Reference*.

limit

Specify the limit you want to apply to the selected job stream.

noask

When you add the option as an argument the agent will not to ask for confirmation before taking action on each qualifying job.

Examples

1. To change the job limit on all job streams that include `sales` in their name, run the following command:

```
ocli plan ls sales@;4
```

2. To change the job limit on job stream `CPUA#Job1`, run the following command:

```
ocli plan ls=CPUA#apwkly;6
```

LISTFOLDER

You can use the listfolder command to view the folders in a directory.

Syntax and command line options

Before you run the command, you must spceify details of the folder that you want to view. You must have the *list* or *list* and *display* access to the folders. For more information about security access roles, see the section about security role definition in Administration Guide. You can enter the command as follows:

```
ocli plan listfolder | lf = <foldername>
```

The details that are required with each parameters are as follows:

foldername

Use the forward slash ("/") to see all the folders in the directory. If you specify a folder name after the forward slash, all the folders saved under the specified folder are listed. To see all the folders in a tree structure, specify the at sign ("@").

Examples

1. To list all folders in the root, run the command:

   ```
   ocli plan listfolder /
   ```

2. To list all folders contained in the folder named "Test", run the command:

   ```
   ocli plan listfolder /Test/
   ```

3. To list all folders and sub-folders contained in the folder named "Test", run the command:

   ```
   ocli plan listfolder /Test/@
   ```

nop/unnop

The plan command to set a job in plan operational or non-operational using Orchestration CLI. It temporarily disables a job in a job stream without changing its dependencies.

Overview

The command updates jobs in a job stream without deleting them or changing their dependencies. If the job contains standard dependencies, they are released and the successors are launched. If the dependency is conditional, the condition must be verified as true before the successors run. After the command runs, the specified job changes to **CANCEL PEND** status.

For example, if you need to temporarily skip a notification job due to a service outage, you can nop that job in the job stream until the service is restored.

Syntax and command line options

```
ocli plan [nop|unnop] jobselect [noask]
```

You can enter the command as follows:

jobselect

Specify the job name. The job must exist in the plan; otherwise, an error occurs. You can operate on multiple jobs by using wildcards.

noask

If specified, the agent updates the nop/unnop option in the job without a confirmation prompt.

RELEASE JOB

You can use the release job command to release normal and time dependencies from a job.

Overview

The command can be applied to jobs that are waiting for the resolution of a dependency. The status of such jobs are marked as HOLD. When you use the command, the dependency is removed only for the job that are in progress. When you rerun the same job, the dependency persist. To remove the dependency permanently from a job stream, refer to DELDEP JOB on page 823. You must have the *release* access to the job to release the dependencies. For more information about security access roles, see the section about security role definition in Administration Guide.

Syntax and command line options

Before you run the command, you must specify options such as job name, dependencies, and required permissions. You can enter the command as follows:

```
ocli plan release job | rj = <jobselect><dependency><noask>
```

You can add the dependency according to the specific results you want to accomplish. The details that are required with each parameters are as follows:

jobselect

Specify the job to update, by means of attributes and qualifiers.

For example, `WA_AGT_0#JOB_STREAM.JOB`.

dependency

Specify the type of dependency and add at least one to complete the command. The type of dependencies that can be used to create the command line are listed in the topic about the release job command in *User's Guide and Reference*.

noask

When you add the option as an argument the agent will not ask for confirmation before taking action on each qualifying job.

Examples

1. To release job `job3` in job stream `ap(1000 03/05/18)`, stored in folder `myfolder`, from all of its dependencies, run the following command:

   ```
   ocli plan rj myfolder/ap(1000 03/05/18).job3
   ```

2. To release all jobs on workstation `site4` from their dependency on a job stream named `pred`, run the following command:

   ```
   ocli plan rj=site4#@.@;follows pred
   ```

RELEASE SCHED

You can use the release command to release the normal and time dependencies from a job stream.

Overview

You can apply the release sched command to job streams that are waiting for the resolution of a dependency. These job streams are marked with the HOLD status. When you run the command, the release sched dependency is removed only for the job that are in progress. When you rerun the same job, the dependency persist. You must have the *release* access to the job to release the dependencies. For more information about security access roles, see the section about security role definition in Administration Guide.

Syntax and command line options

Before you run the command, you must specify options such as job name, dependencies, and required permissions. you can enter the command as follows:

```
ocli plan release sched | rs = <jstreamselect><dependency><noask>
```

You can add the dependency according to the specific results you want to accomplish. The details that are required with each parameters are as follows:

jstreamselect

Specify the job stream to update, by means of attributes and qualifiers. For more information, see to the section about selecting job streams in commands in *User's Guide and Reference*.

dependency

Specify the type of dependency and add at least one to complete the command. The type of dependencies that you can use to create the command line are listed in the topic about the release sched command in *User's Guide and Reference*.

noask

When you add the option as an argument the agent will not ask for confirmation before taking action on each qualifying job.

Examples

1. To release job `job3` in job stream `ap(1000 03/05/18)`, stored in folder `myfolder`, from all of its dependencies, run the following command:

   ```
   ocli plan rj myfolder/ap(1000 03/05/18).job3
   ```

2. To release all jobs on workstation `site4` from their dependency on a job stream named `pred`, run the following command:

   ```
   ocli plan rj=site4#@.@;follows pred
   ```

reply

Job streams or jobs that contain prompts are not completed until the assigned user provides confirmation. You can use the reply command to confirm a specific prompt or multiple prompts in one step.

Overview

Prompts are textual messages that pause a job or job stream from processing. These prompts act as an interface for user interaction offering predefined choices you can use to confirm or reply.

There are three types of prompts. For more information, see Prompt.

To run this command, you must have *reply* access to the following items:

- The named or global prompts.
- The prompts and their associated jobs or job streams (for unnamed prompts).

To confirm a message, type `yes` or `no`. When you confirm a prompt using the reply command, it resumes all associated jobs and job streams.

Syntax and command line options

You can enter the command as follows:

```
ocli plan reply #{message_number | prompt_name} [reply_value] [noask]
```

Specify the **message_number** or **prompt_name**. At least one of these attributes is mandatory to run the command. The details required for each attribute are as follows:

message_number

Specify the unique message number assigned to the prompt.

prompt_name

Specify the prompt name. This attribute is applicable only if the prompt type is named or global. You can also use wildcard characters.

reply_value

Type `yes` to confirm the prompt or `no` to reject it. This attribute is mandatory.

noask

Confirm multiple instances of prompts at once, rather than providing permissions for each individual instance. This attribute is optional.

You can use wildcard characters to confirm multiple instances of prompts with a single step. The scenarios are explained in the following examples.

Examples

1. The job stream **GOLD** contains the prompt **#4DOLLAR**. The unique message number for the prompt is **4**. Run the command as follows to confirm the prompt:

   ```
   ocli plan reply DOLLAR yes
   ```

   or

```
ocli plan reply #4 yes
```

2. The job streams **SALARY_JUNE** and **SALARY_JULY** contain the prompts **SALARY_ADVANCE** and **SALARY_PEND**, respectively. To confirm these multiple prompts in a single action, run the command as follows:

   ```
   ocli plan reply SAL@ yes
   ```

rerun

You can use the rerun command to rerun a job.

Overview

You can run the command when a job is in `SUCC` or `FAIL` or `ABEND` state. When you submit the command, the rerun of the job is placed in the same job stream as the original job and all the associated dependencies of the original job are added. If you rerun a job that is scheduled to run multiple times, the rerun is scheduled to launch at the same rate as original job.

Note: When you rerun a job that contains variables using docommand or with script arguments, the job is launched and completed, but the variables are not resolved with their value.

Restriction: When the command is submitted, the agent will not ask for your confirmation before taking action on each job.

Syntax and command line options

Before you run the command, you must specify options such as job name and required permissions. you can create the command as described below.

```
ocli plan <rerun | rr>
 jobselect<[;from=[[folder/]wkstat#]job[at=time][pri=pri]|[streamlogon|logon=new_logon]]<docommand="new_comman
d"|script="new_script"><step=step><sameworkstation=><noask>
```

You must provide a job name which is a mandatory parameter to run the command. All the other parameters are optional. The details required with each parameters are as follows:

jobselect

Specify the job to update, by means of attributes and qualifiers.

from=[[folder/]wkstat#]job

You can specify the job name which you want to run instead of job specified in **jobselect**. You can also run the jobs in SUPPR state, if they are from job streams that are not in suppressed or cancelled state.

- **wkstat#**

  You can specify the name of the workstation on which you want to run the job.

- **job**

  You can specify the name of the job that you want to run.

Restriction: The following types are not permitted:

- The jobs submitted using the submit docommand.
- The alias names of jobs submitted using the submit job command.

The table below describes the recovery options of the initial job and the job mentioned in **from** option.

Table 98.

recovery option	from intial job	Retrieved from "from" job
stop	No	Yes

at=time

You can specify the start time for the rerun as follows:

```
hhmm [timezone|tz tzname] [+n days | date]
```

hhmm: Specify the hour and minute.

+n days: Specify the next occurrence of rerun in number of days.

date: Specify the date for the next occurrence of rerun in *mm/dd/yy*.

timezone|tz name: Specify the timezone. For valid time zones, see Managing time zones on page 1064.

streamlogo|logon=new_logon

You can use the option to run the job under a new user name. The option is only applicable for completed jobs. The option is mutually exclusive with **from**, **script** and **docommand**.

docommand="new_command"

You can specify new command to rerun the jobs instead of original command. The option is only applicable for completed jobs. The option is mutually exclusive with **script** and **from**.

script="new_script"

You can specify new script to rerun the jobs instead of original script. The option is only applicable for completed jobs. The option is mutually exclusive with **docommand** and **from**.

noask

When you add the option as an argument the agent will not ask for confirmation before taking action on each qualifying job.

Examples

1. To rerun job `job4` in job stream `sked1` on workstation `main`, run the following command:

   ```
   ocli plan rr main#sked1.job4
   ```

2. To rerun job `job5` in job stream `sked2` using the job definition for job `jobx` where the job's **at** time is set to 6:30 p.m. , run the following command:

```
ocli plan rr sked2.job5;from=jobx;at=1830;
```

showcpu

You can use the showcpu command to display the information related to workstations or cpus.

Overview

The cpus or workstations are the building blocks of HCL Workload Automation on which the jobs or job streams are run. The network contains at least one domain which is the master domain. Multiple domains can be added to expand and divide the network to smaller groups. You can use the showcpu command to display the details of these workstations such as, type and properties of workstations, the OS version and hardware model, the version of HCL Workload Automation agent installed and so on. For more details, see Syntax and command line options on page 835 section.

Restriction: When the results are displayed:

- The workstations are not listed in order.
- Incorrect status for the workstation links and processes.

Syntax and command line options

You can view the information in two different formats, **standard** and **info**. If you do not specify the **info** option the results are displayed in **Standard** format. You can use specific options to filter the results. You can enter the command as follows:

```
ocli plan <showcpus | sc><[domain!][folder|workstation]> <info> <offline> <showid>
```

The details that are required with each parameters are as follows:

domain

You can specify the name of the domain under which all the workstations or cpus that you want display are grouped. The default value is the workstation on which you run the command.

folder|workstation

You can specify the name of the specific workstation or cpus that you want display. You can also specify the folder in which the workstation is defined, which is optional.

Standard

If you do not specify any option, the information is displayed in standard format. The result includes the following informations:

- **CPUID**

  The name of the workstation of which the information is displayed.

- **RUN**

  The value in `symphony` file that indicates the number of jobs that have run on the workstation.

- **NODE**

  Specifies the node type and workstation type.

  The node type are as follows:
  - UNIX®
  - WNT
  - OTHER
  - ZOS
  - IBM i

  The workstation types are as follows:
  - MASTER
  - MANAGER
  - FTA
  - S-AGENT
  - X-AGENT
  - POOL
  - D-POOL
  - REM-ENG

- **LIMIT**

  The number of jobs that can run simultaneously on the workstation.

- **FENCE**

  Specifies the priority limit set on the workstation for filtering the jobs to be launched.

- **DATE TIME**

  Specifies the date and time when HCL Workload Automation started the run for the current production plan. The date is displayed in *mm/dd/yy* format.

- **State**

  Specifies the state of the following:

  1. Workstation links and process. The status is displayed in up to five characters.

     ```
     [L|F] [T|H|X|B] [I] [J] [E] [D] [A|R]
     ```

     Each character indicates specific informations as follows:

- `L`

  The workstation is connected to the domain or upper manager. If the workstation is an agent or remote engine, the character indicates the workstation is connected to the workload broker server. If the workstation is part of a pool or dynamic pool, the character indicates the workload broker workstation of the pool or dynamic pool is connected to the domain or upper manager.

- `F`

  The workstation is connected to all the primary and secondary connections, such as the domain manager and all the backup domain managers.

- `T`

  The fault tolerant agent is directly linked to the domain manger.

- `H`

  The workstation is connected through the host.

- `X`

  The workstation is connected as an extended agent.

- `B`

  The workstation communicates through the workload broker server.

- `I`

  If the workstation is a MASTER, MANAGER, FTA, S-AGENT or X-AGENT, the flag indicates the jobmon program completed the startup initialization.

  If the workstation is part of a pool or dynamic pool, the flag indicates the agent is correctly initialized.

  If the workstation is a remote engine, the flag indicates that the communications initialized between the remote engine workstation and the remote engine.

- `J`

  If the workstation is of type agent MASTER, MANAGER, FTA, S-AGENT, X-AGENT, this flag indicates that jobman program is running.

  If the workstation is of type agent, this flag indicates that JobManager is running.

  If the workstation is of type pool, this flag indicates that the JobManager process is running on at least one agent registered to the pool.

  If the workstation is of type remote engine, this flag indicates that the ping command to the remote engine is successful.

2. Monitoring agent and the status will displayed as characters up to two.

   ```
   [E] [D]
   ```

   Each character indicates specific informations as described below.

   - `E`

     The event processing server is installed and running on the workstation.

   - `D`

     Indicates the monitoring configuration package installed on the workstation is up-to-date.

3. The status of WebSphere Application Server Liberty and if the application server is installed, one character is displayed. No character is displayed if the application is down or not installed.

   ```
   [A|R]
   ```

   Each character indicates specific informations as described below.

   - `A`

     WebSphere Application Server Liberty has started.

   - `R`

     WebSphere Application Server Liberty is restarting.

- **METHOD**

  The option is only for extended agents and specifies the access method specified in the definition.

- **DOMAIN**

  Specifies the name of the domain for the workstation.

-

info

If you add the option with the command the results are displayed in **info** format. The result includes the following informations.

- **CPUID**

  The name of the workstation.

- **VERSION**

  The version of the HCL Workload Automation agent installed on the workstation.

- **TIMEZONE**

  Specifies the time zone of the workstation which is the same value of TZ environment variable. For an extended agent and remote engine workstation, this will be the time zone of host and remote engine respectively.

- **INFO**

  An informational filed which displays the operating system version and hardware model. For extended agents and broker workstations, no information is displayed. The remote engine is displayed for remote engine workstation.

showid

When you add this option, he results display a unique workstation identifier. In plan, the workstations are not only identified by the name but also the folder in which it is defined. The name of the workstation and the location of the folder where it is defined is linked to a unique identifier.

Examples

1. Run the following command to display all the workstations in all the folders.

   ```
   ocli plan sc /@/@
   ```

2. Run the following command to display all the workstations in all the folders with the unique identifier.

   ```
   ocli plan sc @;showid
   ```

   A sample output for this command is as follows:

   ```
   CPUID              RUN NODE        LIMIT FENCE DATE     TIME      STATE       METHOD  DOMAIN
   AP-MERGERS-WIN      35 *UNIX MASTER   10     0 10/01/23 23:59    I J  M EA             MASTERDM
    {AP-MERGERS-WIN    }
   AP-MERGERS-LNX86    35  UNIX AGENT    10     0 10/01/23 23:59    LBI J  M             MASTERDM
    {AP-MERGERS-LNX86}
   AP-MERGERS-LNX36    35  OTHR BROKER   10     0 10/01/23 23:59    LTI JW               MASTERDM
    {AP-MERGERS-LNX36}
   MASTERAGENTS        35  OTHR POOL     10     0 10/01/23 23:59    LBI J                MASTERDM
    {MASTERAGENTS    }
   >>/MERGERS/AP/
   WINFTA              35  UNIX FTA      10     0 10/01/23 23:59    LTI JW M             MASTERDM
    {0AAA5D7ZC7BY24A6}
   ```

3. Run the following command to display the results in **info** format.

   ```
   ocli plan showcpus ;info
   ```

   A sample output for this command is as follows:

   ```
   CPUID         VERSION   TIME ZONE              INFO
   MASTER         10.2.8  US/Pacific             Linux 2.6.5-7.191-s390 #1 SM
   FTA1           10.2.8                         Linux 2.4.9-e.24 #1 Wed May
   FTA2           10.2.8                         HP-UX B.11.11 U 9000/785
   ```

4. Run the following command to display the information in **link** format.

```
ocli plan sc @!@;link
```

A sample output for this command is as follows:

```
CPUID       HOST      FLAGS  ADDR  NODE
MASTER     MASTER     AF T  51099  9.132.239.65
FTA1       FTA1       AF T  51000  CPU235019
FTA2       FTA2       AF T  51000  9.132.235.42
BROKER1    MASTER     A  T  51111  9.132.237.17
```

showfile

The showfile command lists text files marked as dependencies for jobs or job streams. The associated jobs or job streams complete only after the orchestrator verifies and confirms the existence of these files.

Syntax and command line options

The showfile command checks for the existence of specific files. Use attributes to filter results or view data in various formats, including the job stream and job information associated with the file.

To run the command, provide a file name with wildcard characters or use the wildcard characters alone. The command requires at least one wildcard character to run. It then lists all files in the database, regardless of their current status.

```
ocli plan [showfile | sf] [[folder/workstation#][file_name]] [state] [keys]
```

or

```
ocli plan [showfile | sf] [[folder/workstation#][file_name]] [state] [deps:[info|key|logon]]
```

You can filter the results by adding the following attributes with the command:

- ***[folder/]workstation***

  Specify the name of the workstation where the file resides. Include the folder path to make the results more specific. You can also use wildcard characters to view details of similar files or multiple files within the same folder or workstation.

- **file_name**

  Specify the name of the file. You can also use wildcard characters to view details of similar files or multiple files within the database.

- **state**

  Specify the status of the file. You can also add any of the following:

  - **Not Checked**

    The existence of the file is not yet checked.

  - **Checked**

    The existence of the file is checked but not yet confirmed.

- **Exists**

  The existence of the file is confirmed.

- **Not Exists**

  The non-existence of the file is confirmed.

Add any one of the following attributes to the command to view the results in specific formats.

1. **standard format**

   If you do not add any attributes to the command, the results are displayed in the standard format. This format shows the status and name of each file in the database.

2. **keys**

   Lists the unique ID for each file.

3. **deps**

   Lists the files alongside their dependent job streams or jobs. The command returns the following details:

   - **deps**
     - **Workstation**

       The workstation on which the job or job stream runs.

     - **Job Stream**

       The name of the job stream.

     - **SchedTime**

       The date and time on which the job is scheduled to run. The date is displayed in *mm/dd* format.

     - **Job**

       The job name. The following notations may precede the job name.

       - **rerun as**

         The job that was rerun either by the rerun command or as a part of automatic recovery.

       - **rerun rerun_number of rerun_total**

         The job that was run as part of a rerun sequence and its position in that sequence.

       - **rerun step**

         The job was rerun using rerun;step command.

- **every run**

  The second and the subsequent runs of a job defined with the **every** attribute.

- **recovery**

  The job was run by a recovery job.

- **State**

  Specifies the status of the job or job stream.

- **Pr**

  The priority assigned to the job stream.

- **Start**

  The estimated start time of the job or job stream.

- **Elapse**

  The estimated run time of the job or job stream.

- **ReturnCode**

  Specifies the return code of the job.

You can further customize these results using the **keys**, **info**, or **logon** options.

- **deps;keys**

  Lists the unique ID of dependent job streams and jobs on each line.

- **deps;info**

  Lists the following details of the dependent jobs and job streams.

  - **workstation**

    The workstation on which the job or job stream runs.

  - **Job stream**

    The name of the job stream.

  - **SchedTime**

    The date and time on which the job is scheduled to run. The date is displayed in *mm/dd* format.

  - **Job**

    The job name. The following notations may precede the job name.

- **rerun as**

  The job that was rerun either by the rerun command or as a part of automatic recovery.

- **rerun rerun_number of rerun_total**

  The job that was run as part of a rerun sequence and its position in that sequence.

- **rerun step**

  The job was rerun using rerun;step command.

- **every run**

  The second and the subsequent runs of a job defined with the **every** attribute.

- **recovery**

  The job was run by a recovery job.

- **JobFile**

  Specifies the name of the script or the executable file of the job.

- **Opt**

  Specifies the available recovery options. The recovery options are **RE** for run, **CO** for continue, and **ST** for stop.

- **Job**

  Specifies the available recovery options.

- **Prompt**

  Specifies the number of the recovery prompt.

◦ **deps;logon**

Lists the following details of the dependent jobs and job streams.

- **workstation**

  The workstation on which the job or job stream runs.

- **Job stream**

  The name of the job stream.

- **SchedTime**

  The date and time on which the job is scheduled to run. The date is displayed in *mm/dd* format.

- **Job**

  The job name. The following notations may precede the job name.

  - **rerun as**

    The job that was rerun either by the rerun command or as a part of automatic recovery.

  - **rerun rerun_number of rerun_total**

    The job that was run as part of a rerun and the position of job in the sequence.

  - **repeated as**

    Specifies the subsequent run of every job.

- **State**

  Specifies the status of the job or job stream.

- **Job#**

  Specifies the job number.

- **Logon**

  Specifies the details of the user logged in.

- **ReturnCode**

  Specifies the return code of the job.

Examples

1. Run the command as follows to view all the files in the database:

   ```
   ocli plan showfile "@"
   ```

   The command produces an output in standard format:

   ```
   Status                 File Name
   FILE_STATUS_NOT_EXISTS /WA_AGT#/folder/test/new/testin1/unit(-w %p)
   FILE_STATUS_NOT_EXISTS /TEST/TESTWRKSTATION#/folder/test/new/testin2/unit(-f %p)
   FILE_STATUS_NOT_EXISTS /TEST/TESTWRKSTATION#/folder/test/new/testing/unit/one/two(-r %p)
   FILE_STATUS_NOT_EXISTS /TEST/EU-HWS-LNX12#/folder/test/new/testing/unit/one/two/three/four(-e %p)
   ```

2. Run the command as follows to view the results in **keys** format:

   ```
   ocli plan showfile "@" keys
   ```

The command produces an output similar to the following:

```
68626f15-6481-30ed-8072-a5d0e353c537

c8cb285f-6b78-33e1-bc40-9b99f80bd8af

e5375660-2a67-3612-be1c-f11c08c81113

7a62d51d-12e5-3bc1-8ef4-fe75fdb809c4
```

3. Run the command as follows to view the results in **deps** format:

```
ocli plan showfile "@" deps
```

The command produces an output similar to the following:

```
                                                                                                (Est)
 (Est)
Workstation         Job Stream  SchedTime       Job                                State  Pr   Start
 Elapse  ReturnCode  Dependencies

/TEST/EU-HWS-LNX12#/folder/test/new(-r %p) represents a dependency for:

  WA_AGT             #FILEJS    1534 03/02       ******************************** HOLD   10
ew(-r %p)
```

4. Run the command as follows to view the results in **deps logon** format:

```
ocli plan showfile "@" deps logon
```

The command produces an output similar to the following:

```
  Workstation  Job Stream      SchedTime  Job                              State  Job#  Logon
 ReturnCode

/WA_AGT#/folder/test/new/testin1/unit(-w %p) represents a dependency for:

WA_AGT         #TESTINGNEWJS1  1900 02/27 ******************************** HOLD

/TEST/TESTWRKSTATION#/folder/test/new/testin2/unit(-f %p) represents a dependency for:

/TEST/
TESTWRKSTATION #NEWONEJ1       1901 02/27 ******************************** HOLD

/TEST/TESTWRKSTATION#/folder/test/new/testing/unit/one/two(-r %p) represents a dependency for:

WA_AGT         #NEWTEST32       1914 02/27 ******************************** HOLD

/TEST/EU-HWS-LNX12#/folder/test/new/testing/unit/one/two/three/four(-e %p) represents a dependency for:

/TEST/
EU-HWS-LNX12   #NEW22           1916 02/27 ******************************** HOLD
```

5. Run the command as follows to view the results in **deps info** format:

```
ocli plan showfile "@" deps info
```

The command produces an output similar to the following:

```
  Workstation      Job Stream       SchedTime  Job                                  JobFile  Opt  Job
 Prompt
   Dependencies

/WA_AGT#/folder/test/new/testin1/unit(-w %p) represents a dependency for:

  WA_AGT           #TESTINGNEWJS1   1900 02/27

     /WA_AGT#/folder/test/new/testin1/unit(-w %p)

/TEST/TESTWRKSTATION#/folder/test/new/testin2/unit(-f %p) represents a dependency for:

  /TEST/
  TESTWRKSTATION   #NEWONEJ1        1901 02/27

      /TEST/TESTWRKSTATION#/folder/test/new/testin2/unit(-f %p)

/TEST/TESTWRKSTATION#/folder/test/new/testing/unit/one/two(-r %p) represents a dependency for:

  WA_AGT            #NEWTEST32      1914 02/27

     /TEST/TESTWRKSTATION#/folder/test/new/testing/unit/one/two(-r %p)

/TEST/EU-HWS-LNX12#/folder/test/new/testing/unit/one/two/three/four(-e %p) represents a dependency for:

  /TEST/
  EU-HWS-LNX12      #NEW22           1916 02/27

     /TEST/EU-HWS-LNX12#/folder/test/new/testing/unit/one/two/three/four(-e %p)
```

6. Run the command as follows to view the results in **deps keys** format:

   ```
   ocli plan showfile "@" deps keys
   ```

   The command produces an output similar to the following:

   ```
   17b81c03-87ee-3030-a161-4af4dafaea2a.JOB1

   17b81c03-87ee-3030-a161-4af4dafaea2a.JOB1
   ```

showjobs

You can use the showjobs command to view the informations related to jobs.

Overview

You can run the command alone or with different parameters to filter the results. If you run the command without adding any parameters the results are displayed in the **standard** format. For more information, see Standard format on page 851.The following parameters are used to view the results in eight different formats:

- **keys**
- **info**
- **logon**
- **keys retcod**

- **deps**
- **stdlist**.

  You can further filter the **deps** format by using the following options:

- **keys**
- **info**
- **logon**

You can also use the wildcards to filter the jobs and the folders within which they are defined. For more information, see Wildcards on page 553.

Restriction: A delay is observed for the results displayed if the output data is more than 10KB.

Syntax and command line options

You can run the command as follows:

```
ocli plan <showjobs | sj><jobselect><deps;[keys | info | logon]><short | single><showid><props>
```

or

```
ocli plan <showjobs | sj><jobselect><[;keys | ;info | ;logon | ;keys retcod]><[short | single><showid>
```

or

```
ocli plan <showjobs | sj><jobselect |[[folder/]workstation#]jobnumber.hhmm]><stdlist[;keys]><short |
 single><showid><props>
```

The details that are required with each parameters are as follows:

jobselect

Specify the job you want to display.

jobnumber

You can specify the job number for the job.

hhmm

The start time of the job.

deps

The jobs used in `follows` dependencies are displayed followed by the dependent jobs and job streams. The jobs and job streams are listed in the standard showjobs and standard showschedules format. You can use the options **keys**, **info** and **logon** to further filter the results.

- **deps;keys**

  If you use the **keys** option with **deps** parameter, the jobs used in the `follows` dependencies are displayed with only the job id per line.

- **deps;info**

  If you use the **info** option with **deps** parameter, the jobs are displayed in showjobs;info on page 856 format. The job streams are displayed in standard showschedules on page 873 format.

- **deps;logon**

  If you use the **logon** option with **deps** parameter, the jobs are listed in the showjobs;logon on page 858 format. The job streams are listed in the standard showschedules on page 873 format.

keys

When you add the option, the job names are displayed one on each line. It is displayed as follows.

```
Workstation    Job stream SchedTime  Job     State Pr Start Elapse ReturnCode Dependencies

MYCPU+#SCHED_F+ 0600 03/04 ******* HOLD  55(03/04)              [03/04/18]; #33
                (M235062+#)JOBMDM  HOLD  30(03/04)              #1(PRMT3);-16 JOBSLOTS-

MYCPU+#SCHED_F+ 1010 03/04 ******* HOLD  55(03/04)              [03/04/18]; #34
                (M235062+#)JOBMDM  HOLD  30(03/04)              #1(PRMT3);-16 JOBSLOTS-
```

Info

If you add this option, the job details are displayed in **Info** format. For more information, see info format on page 856.

step

If you add this option, the job details are displayed in **step** format. For more information, see step format on page 859.

logon

If you add this option, the job details are displayed in **logon** format. For more information, see Logon format on page 858.

retcod

You can use this option with **keys** parameter. The command will display the return code for the job.

Example:

```
ocli plan %sj @; keys retcod
```

stdlist

If you add this option the details are displayed in **stdlist**, You can use the **keys** option to further modify the results. The following informations are displayed:

- Header and trailer banners: The header contains job name, username, the workstation on which the jobs are run, job number, date (*mm/dd/yy* or *mm/dd* format), time and the command performed. The trailer part displays informations such as, **Exit Status**, **Elapsed Time**, **Job CPU usage**, **Job Memory usage**, date and time.
- Displays the output of the job or error output, if the job is failed.

Note: Information related to archived jobs are not recoverable using the **stdlist** option.

Stdlist;keys format

On each line, the names of the standard files for the selected jobs are listed.

workstation

The name of the workstation on which the job runs. You can filter the jobs with wildcard characters.

short

The display is reduced for **every** and **rerun** jobs to include the details described below:

- The first iteration
- Jobs in different states
- Exactly matched jobs.

Note: The field displays specific properties, if the job is a shadow job or a job defined in JSDL.

single

You can add this option to view the specific instance of a job. You must identify the jobs by job number in **jobselect**.

showid

When you add the option, the result displays the unique identifier for the job and job streams.

props

If you have the display access to the props of the specified job and use this option with the command, then the following informations are displayed.

General information

- job
- Workstation Folder
- Workstation
- Task
- Task Type
- Job Stream
- Job Stream Workstation Folder
- Job Stream Workstation
- Scheduled Time
- Priority
- Login
- Monitored

- Requires Confirmation
- Interactive
- Critical

Runtime Information

- Actual Workstation
- Status
- Internal Status
- Not Satisfied Dependencies
- Job Number
- Rerun Options
- Information
- Promoted
- Return Code
- Return Code Mapping Expression
- Successful output conditions related to a SUCC job status
- Other output conditions

Time Information

- Actual Start
- Earliest Start
- Latest Start
- Latest Start Action
- Maximum Duration
- Maximum Duration Action
- Minimum Duration
- Minimum Duration Action
- Critical Latest Start
- Deadline
- Repeat Range
- Actual Duration
- Estimated Duration
- Confidence Interval

Recovery information

- Action
- Message
- Job Definition
- Workstation Folder
- Workstation

- Retry after
- Number of attempts
- Current attempt
- Run on same workstation

Extra information

The additional properties specific for the shadow jobs and jobs defined by JSDL are displayed in this section. The section contains the following informations for shadow jobs:

For distributed shadow jobs:

- Remote Job Scheduled Time
- Remote Job
- Remote Job Stream
- Remote Job Stream Workstation

For z/OS shadow jobs:

- Remote Job Scheduled Time
- Remote Job
- Remote Job Workstation
- Remote Job Error Code

Restriction:

- You cannot recover the information on archived jobs using **props** option.
- When the output of a job is displayed, the job definitions and the output conditions are not displayed.

Standard format

If you do not specify any options with showjob command, the results are displayed in standard format.

The following details are displayed in the standard format.

- **Workstation**

  The workstation on which the job runs.

- **Job stream**

  The name of the job stream.

- **SchedTime**

  The date and time on which the job is scheduled to run. The date is displayed in *mm/dd* format.

- **job**

  The job name. The notation described below may precede the job name.

  - **rerun as**

    The job that was rerun either by the rerun command or as a part of automatic recovery.

  - **rerun rerun_number of rerun_total**

    The job that was run as part of a rerun and the position of job in the sequence.

  - **rerun step**

    Specifies the job was rerun using `rerun;step` command.

  - **every run**

    The second and the subsequent runs of the jobs.

  - **recovery**

    Specifies the run of a recovery job.

- **State**

  Specifies the state of a job or job stream. The different types are described below:

  - **ABEND**

    The job ended abnormally.

  - **ABENP**

    Indicates the **ABEND** confirmation is received but the job is not completed.

  - **ADD**

    The job was added by the user.

  - **CANCL**

    Specifies the job or job stream is cancelled and applicable for internetwork dependencies only.

  - **DONE**

    The job is completed with an unknown status.

  - **ERROR**

    Indicates an error occurred when checking the remote status and applicable for internetwork dependencies only.

- **EXEC**

  The selected job is running.

- **EXTRN**

  The status is unknown and applicable for internetwork dependencies only. The status may be due to any of the following reasons:
  - An error occurred
  - A rerun action is performed on the job in the `EXTERNAL` job stream.

- **FAIL**

  The job is not launched.

- **FENCE**

  Indicate the priority of the job is below the **FENCE**.

- **HOLD**

  The job stream is waiting for the dependency to be resolved.

- **INTRO**

  Indicates the job will launch at any moment.

- **PEND**

  The job is completed and waiting for confirmation.

- **READY**

  All the dependencies are resolved and the job stream is ready to launch.

- **SCHED**

  Indicates that there is still time left before the job is scheduled to launch.

- **SUCC**

  The job is completed successfully.

- **SUCCP**

  The job is not completed, but the confirmation is received.

- **SUPPR**

  The job is suppressed as the related conditional dependencies of the predecessors are not resolved.

- **WAIT**

  Indicates the job is in wait.

The different states for job streams are described below:

- **ABEND**

  The job stream ended abnormally.

- **ADD**

  The job stream was added by the user.

- **CANCP**

  Specifies the job stream is pending cancellation. The cancellation is differed until all the associated dependencies are resolved.

- **ERROR**

  Indicates an error occurred when checking the remote status and applicable for internetwork dependencies only.

- **EXEC**

  The selected job stream is running.

- **EXTRN**

  The status is unknown and applicable for internetwork dependencies only. Specifies the state of `EXTERNAL` job streams.

- **HOLD**

  The job stream is waiting for the dependency to be resolved.

- **READY**

  All the dependencies are resolved and the job stream is ready to launch.

- **STUCK**

  The job stream run is interrupted and no jobs are launched without the user intervention.

- **SUCC**

  The job stream run is completed successfully.

- **SUPPR**

  The job stream is suppressed as the related conditional dependencies of the predecessors are not resolved.

- **Pr**

  Specifies the priority of the job stream.

- **(Est)Start**

  Specifies the start time of the job stream. If the start time is before or after 24 hours, instead of time the date (*mm/dd/yy* or *mm/dd* format) is displayed.

- **(Est)Elapse**

  Specifies the run time of the job stream.

Notes:

- The job stream or job name is displayed for `follows` dependency. If it is a pending predecessor, the name is followed by [P].
- For conditional dependencies the name of the predecessor job or job stream is followed by the output conditions in the format, `IF <condition name>`. The <*condition name*> can be the final status of the predecessor job, the job stream status or condition based on the predecessor job output. When you have multiple conditions, they are separated by the vertical bar (|) symbol.
- The file name is displayed for `opens` dependency. If the file is located in an extended agent, only the last 25 characters of the file name is displayed.
- The resource name enclosed in hyphens (-) is displayed for `needs` dependency. When there are more than one unit is requested, the number appears before the first hyphen.
- For an **until** time, the time is preceded by an angle bracket **(<)**.
- The prompt number is displayed as **#num** and the prompt name is followed by parentheses for global prompts.
- For a **deadline** time, the time displayed is preceded by an angle bracket**(<)**.
- The repetition rate for an every rate is preceded by an ampersand **(&)** character.
- You can identify the cancelled job streams as they are labelled **[Cancelled]**.
- The job streams that are cancelled with the **;pend** option are labeled **[Cancel Pend]**.
- When the results are displayed for showschedules command, the time dependencies are shown in the **start** column as described below:
  - Only the time *hh:mm*, if the date on which the command is run matches with the date set on the job stream.
  - Only the date *dd/mm*, if the date on which the command is run does not matches with the date set on the job stream.
- The Process Identification Number (PID) for running jobs are displayed in *#Jnnnnn* format.
- The jobs are labelled with **Until** option, when it is expired **until** times. This is applicable to jobs cancelled with **;pend** option.
- The different statuses that are displayed when specific options set are met:
  - **[MaxDurationExceeded]** and **maxdur=hhh:mm**: When the specified maximum duration time is exceeded.
  - **[KillSubmitted]**: When the specified maximum duration time is exceeded and the **onmaxdur** option is set to `kill`.

- **[Continue]**: When the specified maximum duration time is exceeded and the **onmaxdur** option is set to `continue`.
- **[MinDurationNotReached]** and **mindur=hhh:mm**: When the specified minimum time is not reached and the job is completed successfully.
- **[Continue]**: When the specified minimum duration time is not reached and the **onmindur** option is set to `continue`.
- **[Abended]**: When the specified minimum duration time is not reached and the **onmindur** option is set to `Abend`.
- **[ConfirmSubmitted]**: When the specified minimum duration time is not reached and the **onmindur** option is set to `Confirm`.
- **[Recovery]**: Indicates the user intervention is required.
- **[Confirmed]**: Indicates confirmation is required as the job is scheduled using the **confirm** keyword.

info format

You can add the **info** option with showjob command to view the information related jobs in info format.

The details described below are displayed in info format.

- **Workstation**

  The workstation on which the job runs.

- **Job stream**

  The name of the job stream.

- **SchedTime**

  The date and time on which the job is scheduled to run. The date is displayed in *mm/dd* format.

- **job**

  The job name. The notation described below may precede the job name.

  - **rerun as**

    The job that was rerun either by the rerun command or as a part of automatic recovery.

  - **rerun rerun_number of rerun_total**

    The job that was run as part of a rerun and the position of job in the sequence.

  - **rerun step**

    Specifies the job was rerun using `rerun;step` command.

- **every run**

  The second and the subsequent runs of the jobs.

- **recovery**

  Specifies the run of a recovery job.

- **Job File**

  Specifies the name of the script or the executable file of the job.

  Note: File names that are too long may wrap, resulting in incorrect paging. You can use the option **more** to have a clear view for the results displayed.

- **Opt**

  Specifies the available recovery options. The recovery options are **RE** for run, **CO** for continue, and **ST** for stop. The details described below are also displayed.

- **Job**

  Specifies the available recovery options.

- **Prompt**

  Specifies the number of recovery prompt.

Examples

1. When you run a command as described below,

   ```
   ocli plan sj;info | more
   ```

   the result is displayed as follows:

   ```
   --------Restart---------
   CPU   Schedule SchedTime  Job      JobFile               Opt  Job Prompt
   M235062+#SCHED_22 1010 03/06
                       JOBMDM   /usr/acct/scripts/gl1
                     (B236153+#)JOB_FTA  echo job12
   M235062+#SCHED_22 0600 03/07
                       JOBMDM   /usr/acct/scripts/gl1
                     (B236153+#)JOB_FTA  echo job12
   M235062+#FINAL    2359 02/13
                       STARTAPPSERVER  /opt/HCL/TWA/TWS/../wastools/startWas.sh
                                                                 CO
                       MAKEPLAN  /opt/HCL/TWA/TWS/MakePlan TWSRCMAP:(RC=0) OR (RC=4)
                       SWITCHPLAN  /opt/HCL/TWA/TWS/SwitchPlan
   M235062+#FINALPOSTREPORTS    2359 02/13
                       CHECKSYNC /opt/HCL/TWA/TWS/CheckSync
                       CREATEPOSTREPORTS  /opt/HCL/TWA/TWS/CreatePostReports
                                                                 CO
                       UPDATESTATS  /opt/HCL/TWA/TWS/UpdateStats
                                                                 CO
   ```

```
M235062+#SCHED12  1010 03/06
                    JOBMDM   /usr/acct/scripts/gl1
                  (B236153+#)JOB_FTA  echo job12
```

Logon format

You can add the **Logon** option with showjob command to view the information related jobs in logon format.

The following details are displayed in logon format.

- **Workstation**

  The workstation on which the job runs.

- **Job stream**

  The name of the job stream.

- **SchedTime**

  The date and time on which the job is scheduled to run. The date is displayed in *mm/dd* format.

- **Job**

  The job name. The following notations may precede the job name.

  - **rerun as**

    The job that was rerun either by the rerun command or as a part of automatic recovery.

  - **rerun rerun_number of rerun_total**

    The job that was run as part of a rerun.

  - **repeated as**

    Specifies the subsequent run of every job.

- **State**

  Specifies the state of job or job stream. For more information, see .

- **Return code**

  Specifies the return code of the job.

- **job#**

  You can specify the job number for the job.

- **Logon**

  Specifies the details related to the user.

Note: You can view any one of the following information for windows operating system.

- username
- domain\username
- username@internet_domain

step format

You can add the **step** option with showjob command to view the information related jobs in step format.

The following details are displayed in step format.

- **CPU**

  The workstation on which the job runs.

- **Schedule**

  The name of the job stream.

- **SchedTime**

  The date and time on which the job is scheduled to run. The date is displayed in *mm/dd* format.

- **job**

  The job name. The notation described below may precede the job name.

  - **rerun as**

    The job that was rerun either by the rerun command or as a part of automatic recovery.

  - **rerun rerun_number of rerun_total**

    The job that was run as part of a rerun.

  - **repeated as**

    Specifies the subsequent run of every job.

- **State**

  Specifies the state of job or job stream. For more information, see .

- **Return code**

  Specifies the return code of the job.

- **job#**

  You can specify the job number of the job.

- **Step**

  The processes related to the jobs listed in order. Only host processes are listed for extended jobs.

showprompt

The command lists all prompts associated with jobs and job streams. Prompts are textual messages that are displayed to the user when a job or job stream runs. These jobs or job streams resume only after the user answers the prompts.

Syntax and command line options

You can filter results based on the prompt name, message number, or prompt status. The command also checks for the existence of specific prompts.

You can use attributes to view data in various formats, including the job stream and job information associated with the prompt. Use wildcard characters to broaden your search. The command requires at least one wildcard character to run. It then lists all prompts in the database, regardless of their current status.

```
ocli plan [showprompt | showprompts | sp] [[folder/]prompt_name] [msgnum] [state] [keys] [showid]
```

or

```
ocli plan [showprompt | showprompts | sp] [[folder/]prompt_name] [msgnum] [state] [deps;[keys | info | logon]]
```

You can filter the results by adding the following attributes with the command:

- ***[folder/]prompt_name]***

  Specify the name of the workstation where the prompt is defined. Include the folder path to make the results more specific. You can also use wildcard characters to view details of similar prompts or multiple prompts within the database.

- **msgnum**

  Specify the unique number assigned to the prompt.

- **state**

  Specify the state of the prompt to be displayed. You can add any one of the following:

  - **NOT_ASKED**

    The prompts are not yet issued.

  - **ASKED**

    The prompts are issued and awaiting reply.

- **ANSWERED_YES**

  The prompts are issued and confirmed with a YES reply.

- **ANSWERED_NO**

  The prompts are issued and confirmed with a NO reply.

Add any one of the following attributes with the command to view the results in specific formats.

1. **Standard format**

   If you do not add attributes to the command, the results are displayed in the standard format. This format shows the following details of the prompt:

   - **Prompt**

     The name of the prompt.

   - **Status**

     The current status of the prompt.

   - **Prompt Type**

     The prompt type.

   - **Number**

     The unique number assigned to the prompt.

   - **Message or Prompt**

     Specifies the textual content within the prompt.

2. **Keys**

   Lists the unique ID for each prompt.

3. **deps**

   Lists the prompts alongside their dependent job streams or jobs. The following details are displayed:

   - deps

     - **Workstation**

       The workstation on which the task or job stream runs.

     - **Job Stream**

       The name of the job stream.

     - **SchedTime**

       The date and time on which the task is scheduled to run. The date is displayed in *mm/dd* format.

- **Job**

  The job name. The following notations may precede the job name.

  - **rerun as**

    The job that was rerun either by the rerun command or as a part of automatic recovery.

  - **rerun rerun_number of rerun_total**

    The job that was run as part of a rerun sequence and its position in that sequence.

  - **rerun step**

    The job was rerun using the rerun;step command.

  - **every run**

    The second and the subsequent runs of a job defined with the **every** attribute.

  - **recovery**

    The job was run by a recovery job.

- **State**

  Specifies the status of the job or job stream.

- **Pr**

  The priority assigned to the job stream.

- **Start**

  The estimated start time of the job or job stream.

- **Elapse**

  The estimated run time of the job or job stream.

- **ReturnCode**

  Specifies the return code of the job.

- **Dependencies**

  Specifies the prompt associated with the job or job stream.

You can further customize these results using the **keys**, **info**, or **logon** options.

- **deps;keys**

  Lists the unique ID of the dependent job streams and jobs on each line.

- **deps;info**

  Lists the following details of the dependent jobs and job streams.

- **workstation**

  The workstation on which the job or job stream runs.

- **Job stream**

  The name of the jobstream.

- **SchedTime**

  The date and time on which the job is scheduled to run. The date is displayed in *mm/dd* format.

- **Job**

  The job name. The following notations may precede the job name.

  - **rerun as**

    The job that was rerun either by the rerun command or as a part of automatic recovery.

  - **rerun rerun_number of rerun_total**

    The job that was run as part of a rerun sequence and its position in that sequence.

  - **rerun step**

    The job was rerun using the rerun;step command.

  - **every run**

    The second and the subsequent runs of a job defined with the **every** attribute.

  - **recovery**

    The task was run by a recovery job.

- **JobFile**

  Specifies the name of the script or the executable file of the job.

- **Opt**

  Specifies the available recovery options. The recovery options are **RE** for run, **CO** for continue, and **ST** for stop.

- **Job**

  Specifies the available recovery options.

- **Prompt**

  Specifies the number of the recovery prompt.

- **Dependencies**

  The prompt name is displayed. For predefined or global prompts, the name is preceded by the message number. For adhoc prompts only the number is displayed.

- **deps;logon**

  Lists the following details of the dependent jobs and job streams.

  - **workstation**

    The workstation on which the job or job stream runs.

  - **Job stream**

    The name of the job stream.

  - **SchedTime**

    The date and time on which the job is scheduled to run. The date is displayed in *mm/dd* format.

  - **Job**

    The job name. The following notations may precede the job name.

    - **rerun as**

      The job that was rerun either by the rerun command or as a part of automatic recovery.

    - **rerun rerun_number of rerun_total**

      The job that was run as part of a rerun and the position of job in the sequence.

    - **repeated as**

      Specifies the subsequent run of every job.

  - **State**

    Specifies the status of the job or job stream.

  - **Job#**

    Specifies the job number.

- **Logon**

  Specifies the details of the user logged in.

- **ReturnCode**

  Specifies the return code of the job.

4. **showid**

   Lists the following details of the prompts.

   - **Prompt ID**

     The unique ID of the prompt.

   - **Prompt**

     The name of the prompt.

   - **Status**

     The current status of the prompt.

   - **Prompt Type**

     The prompt type.

   - **Number**

     The unique number assigned to the prompt.

   - **Message or Prompt**

     Specifies the textual content within the prompt.

Examples

1. Run the command as follows to view all the prompts in the database:

```
ocli plan sp @
```

   The command produces an output in standard format:

```
Prompt                          Status           Prompt Type          Number     Message or Prompt
1C/MASTERAGENTS#/JS2 [Tue Mar ...ASKED           Ad Hoc Prompt        1          Adhoc Prompt
/PRMT3                          ASKED            Predefined Prompt    2          ocli prompt
3C/MASTERAGENTS#/JS2 [Wed Mar ...NOT_ASKED       Ad Hoc Prompt        3          Adhoc Prompt
/PRMT1                          ASKED            Predefined Prompt    4          ready for job4?
 (y/n)
```

2. Run the command as follows to view the prompts in **keys** format:

```
ocli plan sp @ keys
```

   The command produces an output similar to the following:

```
56aab039-30a9-36ef-9435-68d95580f76b

0c7cec83-19c8-336d-b453-28576d0e8755
```

```
38c49d4b-b1f0-3f8a-7854-e545a9b1d2a6

920fe5a8-fd20-320f-dfh1-c435969cf87a

da3b216c-17da-3fec-7458-c96badf86fc7

381d17e2-7ae2-38d4-7gt1-d20db6b1d344

587da0f7-e415-330d-1452-b9bc524341ba
```

3. Run the command as follows to view the prompts in **deps** format:

```
ocli plan sp @ deps
```

The command produces an output similar to the following:

```
  Workstation      Job Stream         SchedTime   Job                    State  Pr  Start  Elapse
 ReturnCode  Dependencies

1(/PRMT1) represents a dependency for:

  MASTERAGENTS     #JOBS              0702 02/16  ***********************READY  10
       {02/16/26}
                                                 JOB00                  HOLD   10
       #1(PRMT1)

       -1 /MASTERAGENTS#/RES1-
2(/PRMT2) represents a dependency for:

  MASTERAGENTS     #JOBS              0702 02/16  ***********************READY  10
       {02/16/26}
                                                 JOB01                  READY  10

7(/PRMT6) represents a dependency for:

  MASTERAGENTS     #JOBS              0702 02/16  ***********************READY  10
       {02/16/26}
                                                     JOB12      HOLD   10
       #7(PRMT6)
                                                     JOB07      HOLD   10
       #7(PRMT6)
```

show resource

You can run the showresource command to view information about resources.

Overview

Resources are a type of dependency that you can add to a job or job stream to create more complex and customized configurations. You can use the showresource command to display details about these resources such as type and properties of resources, quantity, associated workstations and so on. You can view the information about resources in multiple formats. If you do not add any specific option in the command, the information is displayed in standard format. You can use the following attributes to view the results in three formats:

- showid
- keys
- deps

  You can further filter the deps format by using the following options:
  - keys
  - logon
  - info

Syntax and command line options

You can run the command as follows:

- `ocli plan showresource | sr [[/workstationfolder/workstationname#/foldername/resourcename] | [workstationname#/foldername/resourcename] | [/foldername/resourcename] | resourcename]`
- `ocli plan showresource | sr [[/workstationfolder/workstationname#/foldername/resourcename] | [workstationname#/foldername/resourcename] | [/foldername/resourcename] | resourcename] ; showid`
- `ocli plan showresource | sr [[/workstationfolder/workstationname#/foldername/resourcename] | [workstationname#/foldername/resourcename] | [/foldername/resourcename] | resourcename] ; keys`
- `ocli plan showresource | sr [[/workstationfolder/workstationname#/foldername/resourcename] | [workstationname#/foldername/resourcename] | [/foldername/resourcename] | resourcename] ; deps [keys | info | logon]`

The detail required with each parameter is as follows:

resourcename

Enter the resource key to display a specific definition or use wildcard characters to display all the resource definitions at the same time. If the resource is defined in the root folder, specify only the name of the resource. Otherwise add the resource key that contains the workstation name and folder name as follows:

```
[[/workstationfolder/workstationname#/foldername/resourcename] | [workstationname#/foldername/resourcename] | [/foldername/resourcename] | resourcename]
```

Customizable display formats

- **Standard format**

  If you do not specify any option with the command, the details are displayed in the standard format. The following details are displayed:

  Workstation

  The workstation associated with the resource.

  Resource

  The name of the resource.

  Total

  The total quantity of the resource in units.

Available

The number of units available in the resource for allocation to jobs or job streams.

Quantity

The number of units allotted to a particular job or job stream.

Shared

If the resource is shared between multiple jobs or jobstreams, then `Y` is displayed; otherwise `N` is displayed.

Used by

All the job streams using the resource units are displayed.

Example:

```
ocli plan sr @#@

Workstation      Resource         Total  Available   Qty   Shared Used By
/WS_AGT_0        RES1             25     25          No holders of this resource
/WS_AGT_0        RES2             25     25          No holders of this resource
/WS_AGT_0        RES3             25     25          No holders of this resource
```

- **showid**

  When you add this option, the result shows all the fields that are displayed for standard format with one difference. The **Used by** field shows a unique identifier that identifies the job or job stream. In plan, the jobs or job streams are not only identified by the name but also the folder in which it is defined. The name of the job or job stream and the location of the folder where it is defined are linked to a unique identifier.

  Example:

  ```
  ocli plan sr RES9090646687 showid

  Workstation      Resource         Total  Available   Qty   Shared Used By
  /WS_AGT_0        RES9090646687    20     17          3     Y
   24709d66-0717-48d6-9302-507b3725bffd
  ```

- **keys**

  Only the unique ID associated with each resource is displayed.

  Example:

  ```
  ocli plan sr RES9090646687 keys
  ```

```
b31fe964-bba7-4520-89d5-0cde3377fb00.JRESPLAN01085954927
```

- **deps**

  The result shows details of the jobs and job streams dependent on the selected resource. The following details are displayed:

  Workstation

  The workstation associated with the resource.

  Job stream

  The ID of the job stream associated with the resource.

  SchedTime

  The date and time on which the job is scheduled to run.

  Job

  The ID of the job associated with the resource.

  State

  The current state of the job stream associated with the resource.

  Pr

  The priority of the job stream.

  (Est)Start

  The start time of the job or job stream. The (Est) element indicates the expected start time. If the start time is before or after 24 hours, the date instead of time is displayed, for example, `mm/dd/yy` or `mm/dd`.

  (Est)Elapse

  The run time of the job or job stream. The (Est) element indicates the expected elapsed time.

  Return code

  The return code of the job.

  Example:

```
ocli plan sr @#@ deps

                          (Est)  (Est)

  Workstation      Job Stream        SchedTime  Job
                 State Pr Start  Elapse ReturnCode Dependencies

/WS_AGT_0#/RES1085954927 represents a dependency for:
```

```
 WS_AGT_0        #JSRESPLAN01085954…0900 01/24
**************************************** SUCC  10 09:00  00:00    0
```

You can further filter the results with the following options:

- **deps;keys**

  Only the unique ID of the job or job stream dependent on the selected resource is displayed.

  Example:

  ```
  ocli plan sr @#@ deps keys

  b31fe964-bba7-4520-89d5-0cde3377fb00.JRESPLAN01085954927
  ```

- **deps;logon**

  If you add this option, in addition to the Workstation on page 869, Job stream on page 869, SchedTime on page 869, Job on page 869, and State on page 869 fields, the result also shows the following fields:

  Job#

  The unique number of the job associated with the agent.

  Logon

  Details of the user.

  Note: You can view the following information for the Windows operating system:

  - username
  - domain\username
  - username@internet_domain

  Example:

  ```
  ocli plan sr @#@ deps logon

    Workstation      Job Stream       SchedTime  Job
                                      State   Job#
   Logon       ReturnCode

  /WS_AGT_0#/RES1085954927 represents a dependency for:

    WS_AGT_0        #JSRESPLAN0108595…0900 01/24
  ```

```
JRESPLAN01085954927                    SUCC
#J755150wauser      0
```

- **deps;info**

  If you add this option, in addition to the Workstation on page 869, Job stream on page 869, SchedTime on page 869, Job on page 869 fields, the result also shows the following fields:

  Job file

  The name of the script or the executable file of the job.

  Note: File names that are too long might wrap, resulting in incorrect paging. You can use the **more** option to have a clear view of the results that are displayed.

  opt

  The recovery option specified for the job, if any.

  Job

  The job alias, if the job is submitted with an alias.

  Example:

```
ocli plan sr @#@ deps info

  -------- Restart ---------
  Workstation      Job Stream       SchedTime  Job
                                   JobFile
     Opt  Job            Prompt

/WS_AGT_0#/RES1085954927 represents a dependency for:

  WS_AGT_0        #JSRESPLAN010859540900 01/24

 JRESPLAN01085954927


    ---
    executable:
      script: ls
      credential:
        userName: wauser
        password: "${password:wauser}"
```

showschedules

You can use the showschedulescommand to display the informations related to job streams.

Overview

The informations related to job stream is displayed in three formats standard, keys, and deps. You can use **keys**, **info**, and **logon** options to further modify the results for **deps** display. You can use wildcards to filter job streams and the folders which they are defined. For more information, see Wildcards on page 553.

Restriction: A delay is observed for the results displayed if the output data is more than 10KB.

Syntax and command line options

The options that you can use with the command are as follows:

```
ocli plan <showscheds | ss><jstreamselect><keys><showid>
```

or

```
ocli plan <showscheds | ss><jstreamselect><deps[;keys | ;info | ;logon]<;showid>
```

If you do not specify any options the results are displayed in standard format. The details that are required with each parameters are as follows:

jstreamselect

You can specify the job stream that contains the jobs which you want to view.

keys

If you use the option, the results are displayed with only the job stream id per line.

deps

If you use the option, the results are displayed with job streams used in **follows** dependencies. You can use the **keys**, **info**, and **logon** options to further filter the results. The jobs and job streams are listed in the Standard format on page 851 and standard showschedules on page 873 format respectively.

- **deps;keys**

  If you use the **keys** parameter with **deps** option, the job streams used in the follows dependencies are displayed with only the job stream id per line.

- **deps;info**

  The jobs in the results are displayed in showjobs;info on page 856 format and the job streams are displayed in the standard showschedules on page 873 format.

- **deps;logon**

  The jobs in the results are displayed in showjobs;logon on page 858 format and the job streams are displayed in the standard showschedules on page 873 format.

showid

When you add this option, the results will display a unique identifier that identifies the job stream. In plan, the job streams are not only identified by the name but also the folder in which it is defined. The name of the job stream and the location of the folder where it is defined is linked to a unique identifier.

standard format

If you do not specify any options with the **showsched**, the results are displayed in standard format. The following informations are displayed:

- **Workstation**

  The name of the workstation on which the job stream runs.

- **Job stream**

  The job stream name.

- **SchedTime**

  The scheduled time and date of the job stream to start the run. The date is displayed in *mm/dd* format.

- **State**

  Specifies the current status of the job stream. The different states are described below.

  - **ADD**

    The job stream was added by the user.

  - **ABEND**

    A malfunction in the system has ended the job stream.

  - **CANCP**

    The job stream is pending cancellation. The job stream is cancelled when all the dependencies are resolved.

  - **ERROR**

    The check for internetwork dependencies is returned with an error.

  - **EXEC**

    The job stream is in progress.

  - **EXTRN**

    Applicable for internetwork dependencies. Specifies the state of external job streams that contains jobs referencing to job or job streams in the remote network.

- **HOLD**

  The job stream is waiting for the dependencies to resolve.

- **READY**

  All the dependencies of the job stream are resolves and ready to launch.

- **STUCK**

  The job stream run is interrupted. The jobs in the job stream are not launched without the user intervention.

- **SUCC**

  The job stream finished successfully.

- **SUPPR**

  The job stream is suppressed as the related conditional dependencies are not satisfied.

- **Pr**

  Specifies the priority of the job stream.

- **(Est)Start**

  Specifies the start time of the job stream. If the start time is before or after 24 hours, instead of time the date is displayed in *mm/dd* format.

- **(Est)Elapse**

  Specifies the run time of the job stream.

- **Jobs #**

  Specifies the number of jobs in the job stream.

- **Jobs OK**

  Specifies the number of jobs that have completed successfully.

- **Sch Lim**

  Specifies the limit of job stream.

- **Notes:**
  - The job stream or job name is displayed for `follows` dependency. The name is followed by [P], if it is a pending predecessor.
  - For conditional dependencies the name of the predecessor job or job stream is followed by the output conditions in the format, `IF <condition name>`. The <*condition name*> can be the final status of the

predecessor job, the job stream status or condition based on the predecessor job output. When you have multiple conditions, they are separated by the vertical bar (|) symbol.
- The file name is displayed for `opens` dependency. If the file is located in an extended agent, only the last 25 characters of the file name is displayed.
- The resource name enclosed in hyphens (-) is displayed for `needs` dependency. When there are more than one unit is requested, the number appears before the first hyphen.
- For an until time, the time preceded by an angle bracket **(<)**.
- The prompt number is displayed as *#num* and the prompt name is followed by parentheses for global prompts.
- You can identify the cancelled job streams as they are labelled **[Cancelled]**.
- The Job streams that are cancelled with the **;pend** option are labeled **[Cancel Pend]**.
- For a **deadline** time, the time displayed is preceded by an angle bracket**(<)**.
- The job streams with the **carryforward** keyword are labelled **[Carry]**.
- The initial name and date of job streams are displayed in brackets that are carried forward form the previous production plan.
- When the results are displayed for showschedules command, the time dependencies are shown in the **start** column as described below:
  - Only the time *hh:mm*, if the date on which the command is run matches with the date set on the job stream.
  - Only the date *dd/mm*, if the date on which the command is run does not matches with the date set on the job stream.

Examples

1. To display the status of job stream `CLEM_DOCOM` defined in the folder `PROD`, on workstation `SITE3` defined in the folder `EU`, and ask for the job stream identifier, run the following command:

   ```
   ocli plan %ss /EU/SITE3#/PROD/JS_DOCOM ;showid
   ```

   The sample output is given below:

   ```
    (Est) (Est)  Jobs  Sch
   Workstation Job Stream SchedTime State Pr Start Elapse #  OK Lim
   >>          /PROD/
   site3      #JS_DOCOM  0600 11/26 SUCC  10 11/26 00:01  1   1      {0AAAAAAAAAAAACRZ}
   ```

2. To display the status of all job streams defined in all folders in the HOLD state on the workstation on which you are running Orchestration CLI, run the following command:

   ```
   ocli plan showschedules /@/@+state=hold
   ```

   The sample output is given below:

   ```
   (Est)  (Est)  Jobs   Sch
   Workstation Job Stream SchedTime  State Pr  Start  Elapse #  OK  Lim
   >>          /PROD/
   site3       #FILE_JS1  0600 11/26 HOLD  10 (11/26)        1   0       parms FILE_JS1`
   ```

3. To display the status of all job streams with name beginning with `sched` on workstation `CPUA` in the `deps;info` format, run the following command:

   ```
   ocli plan ss CPUA#sched@;deps;info
   ```

   The sample output is given below:

   ```
   --------Restart---------
     CPU   Schedule SchedTime Job    JobFile              Opt   Job       Prompt

   CPUA  #JS_FIRST1[(0600 03/10/18),(0AAAAAAAAAAAABVY)] Dependencies are:
     CPUA#MOD 0212 03/10
                              JOBMDM /usr/scripts/gl1(B236153+#)JOB_FTA1  echo Start gl1?
     CPUA#MOD 0251 03/10
                              JOBMDM /usr/scripts/gl2(B236153+#)JOB_FTA2  echo Start gl2?
   ```

4. To display the status of all job streams defined in all folder and on all workstations defined in all folders, run the following command:

   ```
   ocli plan %ss /@/@#/@/@
   ```

   The sample output is given below:

   ```
                                           (Est)  (Est)    Jobs  Sch
   Workstation Job Stream  SchedTime  State Pr  Start  Elapse  #  OK  Lim
   >>          /PROD/
   site3       #JS_DOCOM   0600 09/20 SUCC  10  09/20  00:01   1   1
   site3       #JS_SCRIPT  0600 09/20 SUCC  10  09/20  00:03   1   1
   site2       #JS_PRED1   1000 09/20 SUCC  10  09/20  00:01   1   1
   site3       #JS_SCRIPT1 0600 09/20 ABEND 10  09/20  00:01   1   0
   site3       #LFILEJOB   0600 09/20 READY 10                  1   0
   site1       #RES_100    0600 09/20 SUCC  10  09/20  00:09   1   1
   site3       #FILE_JS1   0600 09/20 HOLD  10 (09/20)         1   0        parms FILE_JS1`
   site3       #FILE_JOB   0600 09/20 SUCC  10  09/20  00:01   1   1
   site3       #JS_CALC    0000 09/20 HOLD  10 (09/20)         1   0
   JS_REPORT (0000 09/20/18).JOB12P IF STATUS_OK
   JS_TAX (0000 09/20/18).@ IF ABEND | SUPPR
   ```

SUBMIT DOCOMMAND

You can use the submit docommand command to schedule the start of a command as a job.

Overview

You can use the command to schedule the start of a command as a job and if needed, with pre-defined conditions. You must have *submit* access for the job to run the command. You also need to specify the user running the command, either by defining the `logon` option in the command line or by defining the `user` option in the `config.yaml` file, as described in Configuring Orchestration CLI on page 757. If you want to use the **alias** keyword for the job, make sure the job name in the database is specified along with the keyword.

You can also perform this action from any workstation other than the master domain manager. In such cases you must meet the following requirements:

- Valid credentials defined in the `useropts` file to connect to the master domain manager through WebSphere Application Server Liberty Base.
- Authorized to perform **submit** commands in the security file stored on the master domain manager.

Syntax and command line options

Before you run the command as a job, you must specify options such as the command and required permissions. There are mandatory and optional parameters which you can use to schedule the start of a command. You can enter the command as follows:

```
ocli plan submit docommand | sbd = <[folder/]workstation#]><"cmd"><alias[=name]>
<[into=[[folder/]workstation#]{jobstream_id;schedid |[folder/]jobstreamname([hhmm[date]])}]><[joboption[;...]>
```

Note: On Windows operating system, when you launch a submit docommand, you must mask the equal (=) sign as `'\='\`.

For example, to set the local variable *var1* to `hello` you must issue the following command:

```
ocli plan %sbd "set var1\"=\"hello"
```

You can add the parameters according to the specific results you want to achieve. The only mandatory parameter is **cmd**. The details required with each parameters are as follows:

workstation

Specify the name of the workstation on which you want to run the job. You can use wildcard characters and in such cases, the job is started on all the matching workstations. You cannot specify a domain or workstation class.

cmd

You must specify a valid command and can use up to 255 characters. The command is treated as a job and all the job rules are applicable. You must enclose the entire command in double quotation marks (").

into=[folder/]jobstream_instance

Specify the jobstream instance to start a job. You can also provide the folder that contains the job stream definition. If you do not specify any jobstream, the jobs are added to the default job stream JOBS. The associated files of JOBS job stream are always located at the root (/).

Select the job stream instance as follows:

```
[[folder/]workstation#][folder/]jobstreamname([hhmm[ date]])
```

or

```
[[folder/]workstation#]jobstream_id ;schedid
```

joboption

The optional parameters to customize the launch of a job. For a list of supported options, see the topic about the submit docommand in *User's Guide and Reference*.

Important: You must use the semicolon (;) before adding any of the below mentioned job options to filter the results.

Examples

1. To submit a rm command into the job stream `JOBS` with a `follows` dependency, run the following command:

   ```
   ocli plan submit docommand = "rm apfile";follows sked3
   ```

2. To submit a sort command with the alias `sortit` and place the job in the job stream `reports` with an **at** time of `5:30 p.m.`, run the following command:

   ```
   ocli plan sbd "sort < file1 > file2";alias=sortit;into=reports;at=1730
   ```

3. To submit chmod commands on all workstations with names beginning with `site`, run the following command:

   ```
   ocli plan sbd site@#"chmod 444 file2";alias
   ```

SUBMIT JOB

You can use the submit job command to launch a job.

Overview

You can use the submit job command to start a job and if needed, with pre-defined conditions. You must have *submitdb* access for the job to run the command. If you want to use the **alias** keyword for the job, make sure the job name in the database is specified along with the keyword.

You can also perform this action from any workstation other than the master domain manager. In such cases you must meet the following requirements:

- Valid credentials defined in the `useropts` file to connect to the master domain manager through WebSphere Application Server Liberty Base.
- Authorized to perform **submit** commands in the security file stored on the master domain manager.

Syntax and command line options

Before you run the command, you must specify options such as job name and required permissions. There are mandatory and optional parameters which you can use to schedule the start of a job. You can enter the command as follows:

```
ocli plan submit job|sbj = <workstation#><[folder/><jobname><alias[=name]>
<into=[[folder/]workstation#]{jobstream_id;schedid |
 [folder/]jobstreamname([hhmm[date]])}><joboption[;...]><vartable=tablename><noask>
```

You can add the parameters according to the specific results you want to achieve. The only mandatory parameter is **jobname**. The details required with each parameters are as follows:

workstation

Specify the workstation to run the job on. You can use wildcard characters. If you use wild cards, the job starts on all matching workstations. If you do not specify a workstation, the job runs on the default workstation.

jobname

You must specify the name of the job to start. You can use wildcard characters. I you use wildcard characters all matching jobs are submitted. If the job is already in plan, and submitted into the same job stream, you must use the **alias** keyword to assign a unique name.

alias=name

You can specify a unique name for the job which can be used in place of **jobname** parameter. If you add the **alias** keyword without specifying a name, a name is created using the first 5 alphanumeric characters followed by a random ten digit number.

into=[folder/]jobstream_instance

Specify the jobstream instance to launch the job. You can also provide the folder that contains the job stream definition. If you do not specify any jobstream, the jobs are added to the default job stream JOBS. The associated files of JOBS job stream are always located at the root (/).

Select the job stream instance as follows:

```
[[folder/]workstation#][folder/]jobstreamname([hhmm[ date]])
```

or

```
[[folder/]workstation#]jobstream_id ;schedid
```

joboption

The optional parameters to customize the launch of a job. For a list of supported options, see the topic about submit job command in *User's Guide and Reference*.

When you add the option as an argument the agent will not ask for confirmation before taking action on each qualifying job.

Examples

1. To submit the `test` job into the job stream `JOBS`, run the following command:

   ```
   ocli plan sbj = test
   ```

2. To submit a job with an alias of `rptx4` and place the job in the job stream `reports` with `at` time of 5:30 p.m., run the following command:

   ```
   ocli plan sbj = rjob4;alias=rptx4;into=reports;at=1730
   ```

3. To submit the `test` job in the job stream `ORDERS` that is found in the folder named `ONLINE`, run the following command:

   ```
   ocli plan sbj = test;into=/ONLINE/orders
   ```

4. To submit job `txjob3` on all workstations whose names begin with `site`, run the following command:

   ```
   ocli plan sbj = site@#txjob3;alias;noask
   ```

SUBMIT SCHED

You can use the submit sched command to start a job stream.

Overview

You can use the command option to start a job stream and if needed, with pre-defined conditions. You must have *submit* access for the job stream to run the command. If you want to use the **alias** keyword for the job, make sure the job name in the database is specified along with the keyword.

You can also perform this action from any workstation other than the master domain manager. In such cases you must meet the following requirements:

- Valid credentials defined in the `useropts` file to connect to the master domain manager through WebSphere Application Server Liberty Base.
- Authorized to perform **submit** commands in the security file stored on the master domain manager.

Syntax and command line options

Before you run the command, you must specify options such as job stream name and required permissions. There are mandatory and optional parameters which you can use to schedule the start of a job stream. You can create the command as described below.

```
ocli plan submit sched = | sbs = <[[folder/]workstation#][folder/]jstreamname>
<[alias[=name]]><[jstreamoption[;...]]><[vartable=tablename]><[noask]>
```

You can add the parameters according to the specific results you want to achieve. The only mandatory parameter is **jstreamname**. The details required with each parameters are described below.

workstation

Specify the workstation to run the job on. You can use wildcard characters. If you use wild cards, the job starts on all matching workstations. If you do not specify a workstation, the job runs on the default workstation.

[folder/]

You can specify the name of the job stream folder in which the job stream is defined.

jstreamname

You must specify the name of the job stream which you want to launch. You can use wildcard characters and in such cases all the matching job streams are submitted. If the job stream is already in the production plan, you must use the **alias** parameter to assign a unique name.

alias=name

You can specify a unique name for the job stream which can be used in place of **jstreamname**. If yo add the **alias** keyword without specifying a name, a name is created using the first 5 alphanumeric characters followed by a random ten digit number.

jstreamoption

The optional parameters to customize the launch of a job. For a list of supported options, see the topic about submit sched command in *User's Guide and Reference*.

Examples

1. To submit the `adhoc` job stream included in the `payroll` folder on workstation `site1` stored in folder `myfolder`, run the following command:

   ```
   ocli plan submit sched = myfolder/site1#PAYROLL/adhoc
   ```

2. To submit job stream `fox4` with an **until** time of midnight, run the following command:

   ```
   ocli plan sbs = fox4;until=0000
   ```

3. To submit job stream `sched3` on all workstations with names that start with `site`, run the following command:

   ```
   ocli plan sbs = site@#sched3
   ```

help

You can use the help command to view the online help or details about the commands that Orchestration CLI supports.

Overview

The help command provides information on how to use the Orchestration CLI and its various commands. You can use the help command alone or in combination with parameters to achieve specific results. The help command provides quick access to information that you need, to use Orchestration CLI. You can use it to access the online help, view a list of available commands, or get details about a specific command.

Syntax and command line options

You can enter the command as follows:

```
ocli [context|model|plan|plugin] [help|h] [command]
```

To view the details about a specific command, add the **command** parameter.

Command

You can specify the command name to view details about it. You must enter the spelled-out form of the command. No abbreviations are supported.

Examples

1. Run the following command to display the list of Orchestration CLI commands:

   ```
   ocli [plan|model] help commands
   ```

2. Run the following command to view the details of install command:

   ```
   ocli plugin install
   ```

3. Run the following command to view the details of set command:

   ```
   ocli context set
   ```

version

You can use the version command to view the version of Orchestration CLI that is installed in your system.

Syntax and command line options

You can create the command as follows:

```
ocli [context|model|plan|plugin] version|v
```

Example

1. Run the following command to view the version of Orchestration CLI that is installed on your system.

   ```
   ocli [context|model|plan|plugin] version
   ```

   An example of the output is displayed as follows:

   ```
   HCL Maestro(windows)/OCLI stable_F3-SNAPSHOT
   Licensed Materials - Property of HCL
   (c) Copyright HCL Technologies LTD. 2019, 2024
   ```

Special characters as wildcards and delimiters

When you want to use filters to select scheduling items in the database, you can use a list of selected special characters as wildcards and delimiters.

Wildcards

You can use the following special characters as wildcards in HCL Workload Automation.

- **@**: To indicate a null pattern or single character or more than one character (alphabets, numbers and/or alphanumeric).

  Example: `T@S` : The search result is displayed with all items starting with T and ending with S, with multiple characters between them, single characters between them, items named TS, and T@S in the database.

  If you want the results to be more specific, you can use **?** which can replace a single character (alphabet, number and/or alphanumeric).

  Example: You can specify `T?ST` to find out items, such as TEST or T3ST.

  Important: If you want specific results with a particular special character, use a backslash (\) as prefix. For example, use T\?S, to find an item named T?S.

- You can use **/@/** in the syntax to indicate multiple folders. When you want to filter the scheduling items defined in folders, use the below syntax:

  ```
  [folder/]workstationname#/[folder/]jobstreamname.job
  ```

  Examples:

1. `/@/@` : The search result is displayed with all the job streams defined in all the folders. In this example, `@` represents the job stream and `/@/` represents the folder in which the job stream is defined.
2. `@#/@` : The search result is displayed with all items in all the job streams that are defined in the root folder (/). In this example, `@#` represents the workstation, `/` represents the root folder, and `@` represents the job stream.
3. `@#@.@` : The search result is displayed with all items in all the jobs in all the job streams that are defined in the default folder. In this example, `@#` represents the workstation, `@` represents the job stream, and `.@` represents the job.
4. `/@/@#/@/@` : The search result is displayed with all items in all the job streams defined in all the folders in all the workstations that are defined in all the folders. In this example, the first instance of `/@/` represents the folder in which the workstation is defined, `@#` represents the workstation, the second instance of `/@/` represents the folder in which the job stream is defined, and `@` represents the job stream.
5. `@#/@/@.@` : The search result is displayed with all items in all the jobs in all the job streams that are defined in all folders. In this example, `@#` represents the workstation, `/@/` represents the folder in which the job stream is defined, `@` represents the job stream, and `.@` represents the job.

Delimiters

You can use the below list of special characters as delimiters.

Table 99. Special characters as delimiters

Characters	Description
+	You can use the character to include a specific argument. For example, `sj payroll_job + at 0600`
~	You can use the character to exclude a specific argument. For example, `sj payroll_job ~ at 0600`

;	You can use the character as a delimiter between arguments in a command. For example,
	`adj HCL#ABSENCES_JS.PAYROLL_JOB ;`
	`maxdur=80 ;onmaxdur kill`

,	You can use the character to add multiple values for an argument. For example, `state = done, succ, succp`
=	You can use the character to give the specific value for an argument. For example,

Table 99. Special characters as delimiters (continued)

Characters	Description
	`state = done`

Orchestration CLI return codes

Orchestration CLI return codes

Orchestration CLI return-code management

When you run a Orchestration CLI command, the command line can show an outcome return code. To find the return code, perform the following action:

On Windows Operating systems:

```
echo %ERRORLEVEL%
```

On UNIX Operating systems:

```
echo $?
```

The Orchestration CLI command line provides the following return codes for the Orchestration CLI plan and model commands:

Table 100.

Return code	Description

0	The model command completed successfully.
	Run the add command for a simple job that will finish successfully:
	• Run this command: ocli model new job
	• Define the job as follows and save it:
	`$JOBS`
	`WS_AGT_0#JOBDEF1`
	`DOCOMMAND "dir"`
	`STREAMLOGON Administrator`
	`TASKTYPE WINDOWS`
	`RECOVERY STOP`

Table 100. (continued)

Return code	Description
	• The system responds with this message: `> AWSMRP001I You have successfully run the command "add" on the item "WS_AGT_0#JOBDEF1".` • To see the return code, run this command: echo %ERRORLEVEL% This returns code result `0`
4	The model command generated a warning. For example, try to add an existing item such as `simple_job`: • Run this command: `ocli model add <file_path>\fileName`. where <*file_path*> is the path of the file that contains the definition of the `simple_job`. The following error message is displayed: • The system responds with this message: `> AWSMRP004W The item "WS_AGT_0#JS_RS_2115749800" already exists.` `Do you want to replace the item (enter "y" for yes, "n" for no)?` • To see the return code, run this command: `echo %ERRORLEVEL%` This returns code result `4`
8	The model command generated an error. For example, define an item with an invalid syntax. For the purposes of this example, attempt to define a job without including `"$JOBS"` at the beginning og the definition. • The system responds with this message: `> AWSMCG005E The definition does not specify the type of the item.`

Table 100. (continued)

Return code	Description
	• Run this command: `echo %ERRORLEVEL%`. This returns code result `8`
0	The plan command completed successfully. For example, run a sj command on a valid Job Stream as follows : • Run te command: ocli plan sj WS_AGT_0#JS_ADJ_115749800 Workstation Job Stream SchedTime Job State Pr Start Elapse ReturnCode Dependencies WS_AGT_0 #JS_ADJ_115749800 1139 02/17 ********** HOLD 10(02/18) JS_RS_2115749800 HOLD 10 • Run the command: echo %ERRORLEVEL% This returns code result `0`
8	The plan command generates an error. For example, run an sbj command on a job that doesn't exist. • Run this command: `ocli plan sbj WS_AGT_0#invalidJob` • The system reponds with this message: `> AWSMSL009E The criteria specified in the command does not match any item.` • To see the return code, run this command: `echo %ERRORLEVEL%` This returns code result `8`

Managing multiple `config.yaml` files

You can create multiple `config.yaml` files in the database to run commands on different contexts without changing the default context. The feature permits you to use multiple `config.yaml` files at the same time.

When working with different contexts, such as development, staging, and production, this functionality is particularly useful. Each `config.yaml` file includes specific configuration settings for the corresponding context, allowing seamless switching between contexts. To create a new `config.yaml` file, simply duplicate the existing one and assign it a unique name that reflects the context it represents. Then, you can update the configuration settings within the new file to match the requirements of the specific context. After you create multiple `config.yaml` files in place, you can easily switch between them by specifying the desired file name when running commands. This ensures that the appropriate configuration settings are applied for each context, preventing accidental changes to the default context. By managing multiple `config.yaml` files, you can streamline your workflow and reduce the risk of errors caused by manual configuration changes. This capability provides a flexible and efficient way to handle different contexts.

You can run the command as follows, either using the [-file=<*config.yaml path*>] format or the [file <*config.yaml path*>] format:

[-file=<*config.yaml path*>] format

```
ocli [model|plan|context] [-file=<path_to_config.yaml>] command
```

or

```
ocli [model|plan|context] command [-file=<path_to_config.yaml>]
```

or

```
ocli [-file=<path_to_config.yaml>] [model|plan|context] command
```

[-file <*config.yaml path*>] format

```
ocli [model|plan|context] [-file <path_to_config.yaml>] command
```

or

```
ocli [model|plan|context] command [-file <path_to_config.yaml>]
```

or

```
ocli [-file <path_to_config.yaml>] [model|plan|context] command
```

Where:

[-file=<*path_to_config.yaml*>] or [-file <*path_to_config.yaml*>]

Specify the location of `config.yaml` file in your computer.

Example

Examples:

```
ocli plan ss @ -file=\Users\john.morris\.OCLI\config_LOCAL.yaml
```

```
ocli plan -file \Users\john.morris\.OCLI\config_LOCAL.yaml ss @
```

Managing multiple formats

You can set the format by configuring the `config.yaml` file or by adding the parameter in model commands.

Overview

The multiple format options in HCL Workload Automation ensures that you have various data serialization formats to create/modify an item or to display an item. When you run a command, the default value specified for format in the `config.yaml` file or at gateway server is considered. You can change this value either by updating the value in `config.yaml` file or by providing the format option in Orchestration CLI commands.

Syntax and command line options

You can specify the required format as follows:

```
ocli model [-format [JSON | YAML | schedlang]]
[model command] [filename | scheduling items]
```

-format

You can specify any one of the following formats:

- JSON
- schedlang
- YAML

The values are case-insensitive.

Model commands

You can specify the command to create/modify an item or to display results in required format.

Specify any one of the following command to create or modify an existing item in the format of your choice:

- add on page 775
- new on page 799
- modify on page 796

If you run the new command, make sure you specify the item name also to view the template in the required format.

Specify any one of the following command to display or extract data about an existing item in the format of your choice:

- display on page 779
- extract on page 785

filename | scheduling items

You need to specify the *filename* or *scheduling items* according to the model command that you want to run.

Example:

```
ocli model -format json display JS @#@
```

If you do not specify the format option in the command, the format specified in the `config.yaml` is taken. By default, the value for format in the `config.yaml` is YAML, and you can change it to JSON or schedlang. You can change the value manually by editing the `config.yaml`.

If you do not specify any value in model command or `config.yaml` file, the default value (YAML) in the gateway server is considered. In such cases, you must make all the updates in YAML format or else an error message is displayed. The value in the gateway server cannot be modified.

Chapter 15. Temporarily modifying the parameters in the `config.yaml` file

You can temporarily modify the parameters in the default `config.yaml` file when you run a plan, model or plug-in command. This feature permits you to customize the behavior of a specific command without modifying the entire `config.yaml` file.

Overview

This temporary modification of parameters is especially useful in cases where you have multiple `config.yaml` files created for different stages of a workflow. To modify a parameter temporarily, you need to include the parameter and its new value in the command. The parameter value is temporarily updated to run the specified command. This allows you to override the default value in the `config.yaml` file to run a single command, without affecting the overall configuration. It provides flexibility in customizing the behavior of commands based on specific requirements. By specifying the parameter and its new value in this way, the command will use the modified value for that specific instance, while the default value of the parameter remains unchanged in the `config.yaml` file. In conclusion, the ability to modify context parameters temporarily provides a convenient way to customize the behavior of commands on a per-instance basis, allowing for more flexibility and adaptability in your workflows. You can modify the following parameters in the `config.yaml` file:

- Host
- Port
- ContextRoot
- Protocol
- Insecure
- Proxy
- ProxyPort
- ProxyUser
- ProxyPassword
- Jwt
- Timeout
- CurrentFolder
- ExternalConfigurationFile
- ShowWarnings
- CurrentEnvironment

You can enter the command as follows:

```
ocli [model|plan] -parameter=value  command
```

or

```
ocli [model|plan] command  -parameter=value
```

or

```
ocli -parameter=value [model|plan] command
```

Chapter 16. Monitoring HCL Workload Automation

Using internal and third-party tools to monitor HCL Workload Automation

You have a number of options when it comes to monitoring and managing your environment:

Event management

Using an out-of-the -box solution, which monitors your environment and launches a predefined set of actions in response to events that occur on the nodes where HCL Workload Automation runs. For more information about event management, see the section about event management in *Dynamic Workload Console User's Guide*.

External tools

Interacting with and monitoring HCL Workload Automation, using third-party tools, for example Prometheus, Splunk, Fluentd, and so on. You can also use AI Data Advisor (AIDA) to detect anomalies and anticipate failure or degradations. For more information about accessing metrics to monitor the state and health of your environment, see Exposing metrics to monitor your workload on page 891. For more information about using the `BmEvents` configuration file to configure the integration with external monitoring tools, see The BmEvents configuration file on page 896.

Exposing metrics to monitor your workload

To control and monitor your workload, you can have HCL Workload Automation expose a number of metrics that provide insight into the state, health, and performance of your workload environment and infrastructure. By further analyzing these values through a data analytics tool, such as AI Data Advisor (AIDA), you detect anomalies and anticipate failure or degradations.

For more information about AIDA and how to use it, see *AI Data Advisor (AIDA) User's Guide*.

Collecting these metrics can be useful for many reasons:

- Generating alerts and addressing problems before they actually occur.
- Monitoring and analyzing trends
- Comparing historical data
- Detecting anomalies

Table 101: Workload Automation exposed metrics on page 891 shows a list of the metrics retrieved, along with their description.

See Accessing and visualizing the metrics on page 893 to find out where and how to find the metrics.

Table 101. Workload Automation exposed metrics

Metric Display Name	Metric name	Description
Monitoring	application_wa_licence_uncountedJobs	The number of jobs that ran but were not counted.

Table 101. Workload Automation exposed metrics (continued)

Metric Display Name	Metric name	Description
Workload	application_wa_JobsInPlanCount_job	Workload by job status: WAITING, READY, HELD, BLOCKED, CANCELED, ERROR, RUNNING, SUCCESSFUL, SUPPRESS, UNDECIDED
	application_wa_JobsByWorkstation	Job status by workstation
	application_wa_JobsByFolder_jobs	Job status by folder
Critical Jobs	application_wa_criticalJob_incompletePredecessor	Incomplete predecessors
	application_wa_criticalJob_potentialRisk_boolean	Risk level: potential risk
	application_wa_criticalJob_highRisk_boolean	Risk level: high risk
	application_wa_criticalJob_estimateEnd_seconds	Estimated end
	application_wa_criticalJob_confidence_factor	Confidence factor
WA Server - Internal Message Queues	application_wa_msgFileFill_percent	Internal message queue usage for Appserverbox.msg, Courier.msg, mirrorbox.msg, Mailbox.msg, Monbox.msgn, Moncmd.msg, auditbox.msg, clbox.msg, planbox.msg, Intercom.msg, pobox messages, and server.msg
Workstation Status	application_wa_workstation_running_boolean	Workstations running
	application_wa_workstation_linked_boolean	Workstations linked
Database Connection Status	application_wa_DB_connected_boolean	1 - connected, 0 - not connected
WA Server and Console - Liberty	memory_usedHeap_bytes	Heap usage percentage
	session_activeSessions	Active sessions
	session_liveSessions	Live sessions
	threadpool_activeThreads	Active threads

Table 101. Workload Automation exposed metrics (continued)

Metric Display Name	Metric name	Description
	threadpool_size	Threadpool size
	gc_time_seconds	Time per garbage collection cycle moving average
WA Sever and Console - Connection Pools (Liberty)	connectionpool_inUseTime_total_seconds	Average time usage per connection in milliseconds
	connectionpool_managedConnections	Managed connections
	connectionpool_freeConnections	Free connections
	connectionpool_connectionHandles	Connection handles
	connectionpool_destroy_total	Created and destroyed connections

Accessing and visualizing the metrics

If you use AIDA, you can use the metrics exposed by HCL Workload Automation to detect anomalies in your workload and prevent problems.

For more information about AIDA and how to use it, see *AI Data Advisor (AIDA) User's Guide*.

You can also use other monitoring tools which support the OpenMetrics standard, for example Grafana, Prometheus, Splunk, and so on.

If you use Grafana, you have access to an out-of-the-box preconfigured dashboard. You can access the preconfigured dashboard named, **Grafana Dashboard: Distributed Environments**, from Automation Hub to use in your on-premises deployments including Docker.

A separate preconfigured dashboard named, **Grafana Dashboard: Kubernetes Environments**, is available for cluster monitoring, including monitoring pods. Automation Hub gives you access to the downloadable JSON file on the Grafana web site. The dashboard visualizes the metrics for observability.

The metrics are exposed so that any monitoring tool supporting the OpenMetrics standard can display them. To access the metrics:

From the master domain manager (or server in a cloud environment):

You can view the metrics from any browser by accessing the `/metrics` endpoint. The product REST APIs retrieve and expose the metrics data through the following address:

```
https://MDM_HOST:MDM_PORT_HTTP/metrics
```

where,

MDM_HOST

Represents the hostname or IP address of the master domain manager.

MDM_PORT_HTTP

Represents the HTTP port number of the master domain manager.

From the Dynamic Workload Console (or console in a cloud environment):

You can view the metrics from any browser by accessing the `/metrics` endpoint with the credentials of the user defined in the `authentication_config.xml` file. The product REST APIs retrieve and expose the metrics data through the following address:

```
https://DWC_HOST:DWC_PORT_HTTP/metrics
```

where,

DWC_HOST

Represents the hostname or IP address of the console.

DWC_PORT_HTTP

Represents the HTTP port number of the console.

You can also filter the metrics by scope, for example:

```
https://DWC_HOST:DWC_PORT_HTTP/metrics?scope=SCOPE
```

where

SCOPE

Represents the scope of the metric: **vendor**, **base**, or **application**.

Prometheus is an open-source monitoring and alerting solution. It is particularly useful for collecting time series data that can be easily queried. Prometheus pulls data from targets and then exposes it as metrics through a host address. Prometheus can be configured to retrieve metrics at regular intervals.

Prometheus integrates with monitoring tools like Grafana to visualize the metrics collected. Grafana uses the Prometheus system as a datasource and all of the HCL Workload Automation metrics can be accessed and added to dashboards.

Dashboards display information such as:

- Middleware metrics (WebSphere Application Server Liberty)
- HCL Workload Automation infrastructure (message files)
- Workload statistics (jobs per status, total count or grouped by folder or by workstation)
- Critical job information (risk level, confidence factor, incomplete predecessors, estimated end)
- Workstation status (running, linked)

See also Frequently asked questions about metrics and monitoring on page 894 for a series of questions and answers.

Frequently asked questions about metrics and monitoring

A list of questions and answers about metrics and monitoring.

- Q: **What is OpenMetrics in HCL Workload Automation?**

  A: It is an HTTP endpoint exposed by HCL Workload Automation that returns system and business metrics in a standard format compatible with monitoring and analytics tools.

- Q: **How are the metrics generated and collected?**

  A: Metrics are not read directly from the database upon every request. They are generated internally within the HCL Workload Automation code, stored in an in-memory cache, and returned quickly when the collector makes a request.

- Q: **Where is the metrics endpoint located?**

  A: The endpoint is exposed on the HCL Workload Automation component IP address and port. For the master domain manager, this is https://<*MDM_IP*>:<*PORT*>/metrics.

- Q: **Is authentication required to access the metrics?**

  A: Yes, access is protected for enterprise environments. A robust authentication mechanism, for example, token, client certificate, or HTTP Basic Auth, is typically required and must be configured in your monitoring and analytics tools scrape job.

- Q: **How often is the data updated (Data Freshness)?**

  A: Data freshness is determined by two factors:
  - The frequency with which the HCL Workload Automation backend collector generates new metrics (usually every few seconds).
  - The scraping interval configured in your monitoring and analytics tools (typically every 5-15 seconds).

- Q: **What is the maximum end-to-end latency?**

  A: The maximum perceived latency, that is the longest time between an event and its display in the metrics tool, is equal to the metrics tool scrape frequency. If your scrape interval is 15 seconds, the maximum latency is 15 seconds.

- Q: **Does accessing metrics impact HCL Workload Automation server performance?**

  A: The impact is minimal. The use of an in-memory cache ensures HCL Workload Automation spends minimal resources responding to a metrics request, avoiding intensive, time-consuming queries on the HCL Workload Automation database.

- Q: **What are the crucial labels (metadata) for HCL Workload Automation metrics?**

  A: Critical labels for data correlation include: instance (the master domain manager or agent hostname), job_stream_name, workstation, and status (success, failure, abend).

- Q: **What should I check if a metric is missing or incorrect?**

  A:

- Check the scrape status: verify in your monitoring and analytics tools that the HCL Workload Automation target is up and running.
- Review HCL Workload Automation logs: check the HCL Workload Automation logs for errors related to metric generation.
- Verify Authentication: confirm that the monitoring and analytics tools service account or configuration has the necessary permissions.

- Q: **Is there a difference between master domain manager and agent metrics?**

  A: Yes. The master domain manager exposes control and planning metrics, for example, plan state, number of dependencies. The dynamic agents expose execution and health metrics, for example local Agent CPU and memory usage, and local job run count.

The BmEvents configuration file

About this task

The BmEvents configuration file is named *TWA_DATA_DIR*`/BmEvents.conf`. Use it to configure HCL Workload Automation production processes on each workstation that has an agent installed. Its contents are described below.

 # comment

A comment line.

OPTIONS=MASTER|OFF

If the value is set to MASTER then all job scheduling events gathered by that workstation are reported. If that workstation is the master domain manager or the backup master domain manager with full status on, then all scheduling events from the scheduling environment are reported. If the value is set to OFF, no job scheduling events are reported from that workstation. If commented, it defaults to MASTER on the master domain manager workstation, while it allows to report all job scheduling events regarding that workstation only on a workstation different from the master domain manager.

LOGGING=ALL|KEY

Disables or enables the key flag filter mechanism. Possible values are:

ALL

If set to this value, all events from all jobs and job streams are logged.

KEY

If set to this value, the event logging is enabled only for those jobs and job streams that are marked as key. The key flag is used to identify the most critical jobs or job streams. To set it in the job or job stream properties use:

- The keywords `KEYSCHED` (for job streams) and `KEYJOB` (for jobs) from the HCL Workload Automation command-line interface.
- The job `Monitored` check box and job stream `Monitored` check box from the Dynamic Workload Console.

The **TEPConfig** script sets the value of this parameter to `KEY`.

SYMEVNTS=YES|NO

It determines whether events concerning jobs and job streams are to be reported immediately after a plan creation.

YES

If set to this value, it tells the production process, **batchman**, to report the jobs and job streams status events immediately after having generated the new production day plan. For the integration with Tivoli Enterprise Portal, this value will cause a bulk discovery to be automatically performed after each new plan creation.

NO

Set to `NO` if report is not required.

The default value is `NO`.

CHSCHED=HIGH|LOW

When set to HIGH, batchman sends an event for any schedule status transaction. When set to LOW, batchman only tracks the initial schedule status transactions. For the lifetime of schedule jobs no change of status is reported until the final state is reached. When a job has more than one final state, an event is sent for each. For example, a schedule completes with an ABEND state and event 151 is sent (schedule abended). The job is then rerun and completes successfully. The schedule is completed with a SUCC state and event 154 is sent (schedule completed). The default is HIGH. Table Table 102: Events filtered by CHSCHED on page 897 lists the events that are filtered by CHSCHED when it is set to LOW.

Table 102. Events filtered by CHSCHED

Event	Description	Filtered on LOW
151	Schedule abended	NO
152	Schedule is stuck	NO
153	Schedule started	YES
	Note: This event is not logged for Workstation Master.	
154	Schedule ended	NO
155	Until time expired onuntil = suppr	NO
156	Schedule submitted	YES

Table 102. Events filtered by CHSCHED (continued)

Event	Description	Filtered on LOW
157	Schedule cancelled	NO
158	Schedule ready	YES
159	Schedule hold	YES
160	Schedule extrn	YES
161	Schedule is cancel pending	NO
162	Schedule properties changed	YES
163	Schedule is late	NO
164	Until time expired onuntil = continue	NO
165	Until time expired onuntil = cancel	NO

EVENT= *n* [*n* ...]

The list of events to be reported. Event numbers must be separated by at least one space. If omitted, the events reported by default are:

```
51 101 102 105 151 152 155 201 202 203 204 251 252
```

Event 51 causes mailman and batchman to report the fact that they were restarted. Events 1, 52, and 53 are not valid in this file .

If the EVENT parameter is included, it completely overrides the defaults. To remove only event 102 from the list, for example, you must enter the following:

```
EVENT=51 101 105 151 152 155 201 202 203 204 251 252
```

PIPE=*filename*

If set, job scheduling events are written to a FIFO file. To have events sent to the IBM Workload Scheduler/NetView agent, the setting must be:

```
PIPE=MAGENT.P
```

PIPE=*filename*

Used for communicating with a Unison Fifo file in UTF-8 format. The format of this file is a 4-byte string followed by the messages.

PIPE_NO_UTF8=*filename*

Used for communicating with a Unison Fifo file in local language. The format of this file is a 4-byte string followed by the messages.

FILE=*filename*

Path and file name of an ASCII log file where job scheduling events are written in UTF-8 format. This file is truncated whenever the **batchman** and **mailman** processes are restarted, for example at the end of each production day.

FILE_NO_UTF8=*filename*

Path and file name of an ASCII log file where job scheduling events are written in local language. This file is truncated whenever the **batchman** and **mailman** processes are restarted, for example at the end of each production day.

JSON=*filename*

Used for appending the log at the end of a file in .json format in UTF-8 format. Files in json format can be processed by third-party monitoring tools, such as Splunk and Fluentd. This file is truncated whenever the **batchman** and **mailman** processes are restarted, for example at the end of each production day.

JSON_NO_UTF8=*filename*

Used for appending the log at the end of a file in .json format in local language. Files in json format can be processed by third-party monitoring tools, such as Splunk and Fluentd. This file is truncated whenever the **batchman** and **mailman** processes are restarted, for example at the end of each production day.

MSG=*filename*

Used for communicating with a Unison Message file in local language.

MSG_NO_UTF8=*filename*

Used for communicating with a Unison Message file in UTF-8 format.

Logged events

About this task

Table 103: Logged events on page 899 lists the events logged by HCL Workload Automation.

Table 103. Logged events

Event	Number	EventName	Description
mstReset	1	Reset	Job reset
mstProcessReset	51	ProcessReset	Process reset
mstProcessGone	52	ProcessGone	Process gone
mstProcessAbend	53	ProcessAbend	Process abended
mstXagentConnLost	54	XagentConnLost	Connection to x-agent lost
mstJobAbend	101	JobAbend	Job abended (see note on page 903)
mstJobFailed	102	JobFailed	External job failed

Table 103. Logged events (continued)

Event	Number	EventName	Description
mstJobLaunch	103	JobLaunch	Job launched
mstJobDone	104	JobDone	Job done
mstJobUntil	105	JobUntil	Job suspended until expired
mstJobSubmit	106	JobSubmit	Job submitted
mstJobCancel	107	JobCancel	Job cancelled
mstJobReady	108	JobReady	Job in READY status
mstJobHold	109	JobHold	Job in HOLD status
mstJobRestart	110	JobRestart	Job restarted
mstJobCant	111	JobCant	Job Failed (see note on page 903)
mstJobSuccp	112	JobSuccp	Job Successful pending
mstJobExtrn	113	JobExtrn	Job extern
mstJobIntro	114	JobIntro	Job in INTRO status
mstJobStuck	115	JobStuck	Job stuck
mstJobWait	116	JobWait	Job in WAIT status
mstJobWaitd	117	JobWaitd	Job in wait deferred status
mstJobSched	118	JobSched	Job scheduled
mstJobModify	119	JobModify	Job modified
mstJobLate	120	JobLate	Job is late
mstJobUntilCont	121	JobUntilCont	Job UNTIL time expired with Continue option
mstJobUntilCanc	122	JobUntilCanc	Job UNTIL time expired with Cancel option
mstJobMaxDurationExceeded Continue	123	MaxDurationExceededContinue	Job maximum duration exceeded, job continues to run
mstJobMaxDurationExceeded Kill	124	MaxDurationExceededKill	Job maximum duration exceeded, Kill action triggered
mstJobMinDurationNotReache dContinue	125	MinDurationNotReachedConti nue	Job minimum duration not reached, job continues to run
mstJobMinDurationNotReache dAbend	126	MinDurationNotReachedAbend	Job minimum duration not reached, Abend action triggered

Table 103. Logged events (continued)

Event	Number	EventName	Description
mstJobMinDurationNotReachedConfirm	127	MinDurationNotReachedConfirm	Job minimum duration not reached, Confirm action triggered
mstJobRisklevelHigh	128	JobRiskLevelHigh	Critical job with risk level set to high
mstJobRisklevelPotential	129	JobRiskLevelPotential	Critical job with risk level set to potential
mstJobRisklevelNone	130	JobRiskLevelNone	Critical job with risk level set to either high or potential that is then removed from the plan
mstJobPromoted	131	JobPromoted	Job in a critical network, that has not yet started, approaches the critical start time and gets promoted so that additional operating system resources are assigned and the submission of the job is prioritized
mstJobSuppress	132	JobSuppress	The job is suppressed when the conditional dependencies associated to the job's predecessors are not satisfied.
mstSchedAbend	151	SchedAbend	Job stream abended
mstSchedStuck	152	SchedStuck	Job stream is stuck
mstSchedStart **Note:** This event is not logged for Workstation Master.	153	SchedStart	Job stream started
mstSchedDone	154	SchedDone	Job stream done
mstSchedUntil	155	SchedUntil	Job Stream suspended, until time expired
mstSchedSubmit	156	SchedSubmit	Job stream submitted
mstSchedCancel	157	SchedCancel	Job Stream cancelled
mstSchedReady	158	SchedReady	Job stream in READY status
mstSchedHold	159	SchedHold	Job stream in HOLD status
mstSchedExtrn	160	SchedExtrn	Job stream extern
mstSchedCnpend	161	SchedCnpend	Job Stream in CANCEL Pending status
mstSchedModifyName	162	SchedModify	Name changed for a scheduled job

Table 103. Logged events (continued)

Event	Number	EventName	Description
mstSchedLate	163	SchedLate	Job Stream is late
mstSchedUntilCont	164	SchedUntilCont	Job Stream Until time expired with continue option
mstSchedUntilCanc	165	SchedUntilCanc	Job Stream until time expired with cancel option
mstSchedSuppress	166	SchedSuppress	The job stream is suppressed when the conditional dependencies associated to the job stream's predecessors are not satisfied.
mstGlobalPrompt	201	GlobalPrompt	Global prompt issued
mstSchedPrompt	202	SchedPrompt	Prompt scheduled
mstJobPrompt	203	JobPrompt	Job prompt issued
mstJobRecovPrompt	204	JobRecovPrompt	Job Recovery prompt issued
mstLinkDropped	251	LinkDropped	Link dropped
mstLinkBroken	252	LinkBroken	Link broken
mstLinkSet	253	LinkSet	Link set
mstMonitorStop	261	MonitorStop	Monitor stop
mstMonitorStart	262	MonitorStart	Monitor start
mstDomainMgrSwitch	301	DomainMgrSwitch	Domain manager switch
mstReplyJobLocalPrompt	302	ReplyJobLocalPrompt	Job prompt replied to
mstReplyGlobalPrompt	303	ReplyGlobalPrompt	GLobal prompt replied to
mstReplySchedLocalPromp	304	ReplySchedLocalPromp	Job stream local prompt replied to
mstLinkToCpu	305	LinkToCpu	CPU linked
mstProcessStopped	306	ProcessStopped	Process stopped
mstSchedComplete	307	SchedComplete	Job stream completed
mstJobBound	308	JobBound	For shadow jobs: the shadow job matched a remote job instance in the remote plan. For HCL Workload Automation for Z agents: the job is on the JES queue.
mstProductAlert	309	ProductAlert	Symphony corruption

Note: In a Windows environment, a failed job generates **event ID 101** in the event log. In contrast, on a Linux environment, a failed job generates **event ID 111**. This behavior is expected and results from the different mechanisms used by Windows and Linux to locate the script file.

Enabling observability with OpenTelemetry

Gain a deeper insight into your environment with OpenTelemetry.

About this task

OpenTelemetry is available by default on each master domain manager and Dynamic Workload Console installed with a fresh installation. You can also enable it after upgrading to the current version.

After the installation or upgrade has completed, perform the following steps to enable OpenTelemetry:

1. Install and configure a tracing tool of your choice, for example Jaeger, Prometheus, or Splunk.
2. Stop WebSphere Application Server Liberty, as described in the topic about application server - starting and stopping in*Administration Guide*.
3. Browse to the following paths:

   master domain manager

   On UNIX operating systems

   TWA_home`/usr/servers/engineServer`

   On Windows operating systems

   TWA_home`\usr\servers\engineServer`

   Dynamic Workload Console

   On UNIX operating systems

   DWC_home`/usr/servers/dwcServer`

   On Windows operating systems

   DWC_home`\usr\servers\dwcServer`

4. Edit the following properties in the `server.env` configuration file, based on the specifics of your environment:

   ```
   OTEL_EXPORTER_OTLP_ENDPOINT=http://{OPENTELEMETRY_HOSTNAME}:{OPENTELEMETRY_PORT}
   OTEL_EXPORTER_OTLP_TRACES_ENDPOINT={OPENTELEMETRY_HOSTNAME}:{OPENTELEMETRY_PORT}
   OTEL_SDK_DISABLED=false
   OTEL_TRACES_EXPORTER
   OTEL_EXPORTER_OTLP_PROTOCOL
   ```

   where

   OTEL_EXPORTER_OTLP_ENDPOINT

   A base endpoint URL for any signal type, with an optionally-specified port number

OTEL_EXPORTER_OTLP_TRACES_ENDPOINT

Endpoint URL for trace data only, with an optionally-specified port number

OTEL_SDK_DISABLED

Disable the SDK for all signals

OTEL_TRACES_EXPORTER

Trace exporter to be used

OTEL_EXPORTER_OTLP_PROTOCOL

OTLP transport protocol. Supported values are as follows:

`grpc`

for protobuf-encoded data using gRPC wire format over HTTP/2 connection

`http/protobuf`

for protobuf-encoded data over HTTP connection

`http/json`

for JSON-encoded data over HTTP connection

For more information about the properties in the `server.env` file, see OpenTelemetry documentation.

5. Set the following properties in the `jvm.option` file as described below:

```
-Dotel.resource.attributes = service.name=<service_name>
-Dotel.metrics.exporter = none
-javaagent:<TWA_DIR>/usr/servers/engineServer/opentelemetry-javaagent.jar
```

6. Start WebSphere Application Server Liberty, as described in the topic about application server - starting and stopping in*Administration Guide*.

Results: You have now configured OpenTelemetry to work with HCL Workload Automation. The resulting telemetry data are displayed on the workstation you specified in the `server.env` file.

When enabling OpenTelemetry, it is important to be aware that it generates a substantial amount of data, which may impact system performance, especially on AIX operating systems.

Configuration file example

About this task

Consider the following example in which tracing is configured with Jaeger:

```
OTEL_EXPORTER_OTLP_ENDPOINT=http://localhost:4317
OTEL_EXPORTER_OTLP_TRACES_ENDPOINT=http://localhost:4317
OTEL_SDK_DISABLED=false
OTEL_TRACES_EXPORTER=otlp
```

```
OTEL_EXPORTER_OTLP_PROTOCOL=grpc
OTEL_EXPORTER_OTLP_TRACES_PROTOCOL=grpc
```

Leveraging Instana for advanced observability

You can integrate HCL Workload Automation with Instana, a platform that provides deep observability into microservices and containerized applications, giving you a complete view of your system's health and performance.

Its core capabilities are fully automated and include:

- Application Performance Monitoring (APM) to track service health.
- Root cause analysis to quickly identify the source of failures.
- Anomaly detection to proactively find and address unusual behaviour.

Integrating HCL Workload Automation with Instana enhances your monitoring capabilities by helping you track and manage business process objectives. Through detailed analytics and historical trends, you can easily identify patterns, troubleshoot issues, and optimize performance over time. Key benefits of this integration include the ability to:

- Gain end-to-end visibility into job processing across all systems and applications.
- Receive real-time alerts for job delays, failures, or performance degradations.
- Automatically correlate workload jobs with underlying infrastructure and application metrics.
- Accelerate root cause analysis with contextual data and intelligent alerts, reducing investigation time.
- Create custom dashboards and define KPIs to track what matters most to your business.

Integrating Instana

The integration of HCL Workload Automation with Instana includes the installation of the Instana agent and, optionally, the OpenTelemetry configuration:

- **Instana agent installation**

  This step is required to start integrating Instana with HCL Workload Automation. After having installed the Instana agent, you can perform the monitoring of the metrics.

  You can directly deploy the Instana agent using standard methods like Docker, Kubernetes, or other system packages. The configuration process is streamlined, requiring only minimal details such as your API key and environment information. After the deployment, the agent automatically discovers all services, containers, and infrastructure components, enabling immediate tracing and metrics collection.

  To ensure your HCL Workload Automation environment is detected, navigate to the **Infrastructure** page on the Instana user interface and verify that the virtual machine containing the master domain manager is listed. If the virtual machine is not listed, you must install an agent on that machine.

  For the full installation procedure of the Instana agent, see Installing host agents.

  To download the Instana dashboard, see Instana Dashboard.

- **OpenTelemetry configuration**

  You can leverage OpenTelemetry to analyze the traces. The configuration requires three steps:

  1. **Enable the OpenTelemetry SDK**

     Follow the steps described in Enabling observability with OpenTelemetry on page 903 to enable the OpenTelemetry SDK.

  2. **Install the OpenTelemetry Collector**

     Install the OpenTelemetry Collector.

     OpenTelemetry is available by default on each master domain manager and Dynamic Workload Console installed with a fresh installation. You can also enable it after upgrading to the current version.

     After the installation or upgrade has completed, perform the following steps to enable OpenTelemetry:

     a. Install and configure a tracing tool of your choice, for example Jaeger, Prometheus, or Splunk.
     b. Stop WebSphere Application Server Liberty, as described in the topic about application server - starting and stopping in*Administration Guide*.
     c. Browse to the following paths:

        master domain manager

        On UNIX operating systems

        TWA_home`/usr/servers/engineServer`

        On Windows operating systems

        TWA_home`\usr\servers\engineServer`

        Dynamic Workload Console

        On UNIX operating systems

        DWC_home`/usr/servers/dwcServer`

        On Windows operating systems

        DWC_home`\usr\servers\dwcServer`

     d. Edit the following properties in the `server.env` configuration file, based on the specifics of your environment:

```
OTEL_EXPORTER_OTLP_ENDPOINT=http://
{OPENTELEMETRY_HOSTNAME}:{OPENTELEMETRY_PORT}
OTEL_EXPORTER_OTLP_TRACES_ENDPOINT={OPENTELEMETRY_HOSTNAME}:{OPENT
ELEMETRY_PORT}
OTEL_SDK_DISABLED=false
OTEL_TRACES_EXPORTER
OTEL_EXPORTER_OTLP_PROTOCOL
```

where

OTEL_EXPORTER_OTLP_ENDPOINT

A base endpoint URL for any signal type, with an optionally-specified port number

OTEL_EXPORTER_OTLP_TRACES_ENDPOINT

Endpoint URL for trace data only, with an optionally-specified port number

OTEL_SDK_DISABLED

Disable the SDK for all signals

OTEL_TRACES_EXPORTER

Trace exporter to be used

OTEL_EXPORTER_OTLP_PROTOCOL

OTLP transport protocol. Supported values are as follows:

`grpc`

for protobuf-encoded data using gRPC wire format over HTTP/2 connection

`http/protobuf`

for protobuf-encoded data over HTTP connection

`http/json`

for JSON-encoded data over HTTP connection

For more information about the properties in the `server.env` file, see OpenTelemetry documentation.

e. Set the following properties in the `jvm.option` file as described below:

```
-Dotel.resource.attributes = service.name=<service_name>
-Dotel.metrics.exporter = none
-
javaagent:<TWA_DIR>/usr/servers/engineServer/opentelemetry-javaage
nt.jar
```

f. Start WebSphere Application Server Liberty, as described in the topic about application server - starting and stopping in*Administration Guide*.

Results: You have now configured OpenTelemetry to work with HCL Workload Automation. The resulting telemetry data are displayed on the workstation you specified in the `server.env` file.

When enabling OpenTelemetry, it is important to be aware that it generates a substantial amount of data, which may impact system performance, especially on AIX operating systems.

3. **Configure the OpenTelemetry Collector**

   A default configuration is available after the installation of the OpenTelemetry Collector, but you can customize the parameters according to your needs by editing the `config.yaml` file.

Visualizing HCL Workload Automation health and performance with Instana

The Instana dashboard download file on Automation Hub contains a default dashboard configuration that you can import on Instana. This dashboard configuration offers a functional layout, and it is designed to highlight the most relevant HCL Workload Automation performance metrics and environment status indicators, strategically arranged to support your monitoring activities.

Dashboard widgets help you quickly identify issues and areas of concern. With their easily readable graphs, you can instantly monitor your environment and configure alerts for critical business events. The clarity of the data visualizations helps any team member, regardless of their technical background, to gain valuable insights at a glance without needing specific Instana training.

This base configuration can be further extended: you can design custom widgets to display additional data, such as metrics exposed directly by the server, or information gathered via OpenTelemetry, ensuring total control over your monitoring activities. For the complete list of all the available metrics, see Exposing metrics to monitor your workload on page 891.

The default HCL Workload Automation dashboard configuration on Instana contains the following widgets:

- **Database connection**

  The **database connection** metric monitors the stability of the connection between the HCL Workload Automation server and the database. If the value displayed is 0, the database is not connected; if the value displayed is 1, the database is connected.

- **Job status overtime**

  The **job status overtime** metric monitors how many jobs, over a specific period of time, have been in `Canceled`, `Error`, `Ready`, `Successful`, and `Waiting` status.

- **Job status pie chart**

  The **job status pie chart** aggregates the jobs according to their status, which can be `Canceled`, `Error`, `Ready`,`Running`, `Successful`, or `Waiting`.

- **Message queue usage**

  The **message queue usage** widget displays the percentage of message queues in use. If the percentage is 0 or near to 0, the environment is healthy.

- **CPU utilization**

  The **CPU utilization** widget displays the percentage of the CPU that is in use during the selected period of time.

- **HEAP utilization**

  The **HEAP utilization** widget displays the percentage of the HEAP memory that is in use during the selected period of time.

- **Workstation link status**

  The **Workstation link status** widget monitors if the workstations, during a specific period of time, are linked or not.

- **Workstation running status**

  The **Workstation running status** widget monitors if the workstations, during a specific period of time, are running or not.

- **Critical jobs information**

  Five widgets are dedicated to the monitoring of critical jobs:

  Incomplete predecessors

  Monitors the number of incomplete predecessor for each critical job.

  High risk

  Indicates whether a critical job must be considered to be at high risk or not. If the value is 0, the job is not at high risk; if the value is 1, the job is at high risk.

  Potential risk

  Indicates whether a critical job must be considered to be potentially at risk or not. If the value is 0, the job is not potentially at risk; if the value is 1, the job is potentially at risk.

  Confidence factor

  Indicates in percentage the confidence that the critical job will meet its deadline.

  Estimated end

  Displays a calculated end time for each job, based on an analysis of all available performance metrics.

Chapter 17. Extending HCL Workload Automation capabilities

You can extend HCL Workload Automation capabilities by integrating with third-party products. This integration allows you to easily start HCL Workload Automation jobs on external products, while using HCL Workload Automation scheduling capabilities. HCL Workload Automation also provides jobs that perform everyday operations, such as file transfer and web services, and utility jobs that automate and simplify operations, such as the job stream submission.

The integration consists of a number of job types with advanced options available with the Dynamic Workload Console and with the **composer** command.

You can also create custom plug-ins to implement your own job types with advanced options for applications that are not supported by HCL Workload Automation. Before you create a new plug-in, check if the plug-in that you are looking for already exists on Automation Hub. If you do not find what you need, you can access the Workload Automation, Lutist Development Kit through the Automation Hub.

Standard HCL Workload Automation jobs are generic executable files, programs, or commands. You can define jobs to perform specific tasks, such as performing file transfers, and running commands on remote systems where no HCL Workload Automation component is installed, using the job types with advanced options. You can easily define these jobs without having specific skills on the applications on which the job runs.

For more information about defining standard HCL Workload Automation jobs, see Job definition on page 228.

Once job definitions have been submitted into the production plan, you still have the opportunity to make one-off changes to the definitions before they run, or after they have run. You can update the definition of a job that has already run and then rerun it. The job definition in the database remains unchanged.

Note: Many of the old plug-ins previously provided with the product, are now out-of-the-box integrations available on Automation Hub. The related documentation has been removed from the product library and has been made available on Automation Hub.

In addition to these job plug-ins, you can find new integrations on Automation Hub that extend your automation processes. New integrations are constantly added to Automation Hub, so make sure to visit it for an up-to-date list of all available integrations.

You can run job types with advanced options only on workstations with dynamic capabilities that is dynamic agents, pools, and dynamic pools. These workstation types use the dynamic functions built into HCL Workload Automation and provide the possibility at run time to dynamically associate your submitted workload (or part of it) to the best available resources. For more information about dynamic scheduling, see Workstation on page 31.

Remote command jobs

A remote command job runs on remote computers that are not required to have the HCL Workload Automation agent installed.

A description of the job properties and valid values are detailed in the context-sensitive help in the Dynamic Workload Console by clicking the question mark (?) icon in the top-right corner of the properties pane.

Note: On Windows systems, the RemoteCommand plug-in has a hardcoded timeout set to 5 minutes (300 seconds). It might happen that this timeout is reached when a job is still running, causing its abnormal ending. To prevent this, a new property file, `RemoteCommandExecutor.properties`, has been added to the plug-in, having the attribute **timeout** that can be set to a different amount of seconds to give more time to run to the job. The attribute format is as follows: **timeout**=*sec*, where *sec* is the amount of time in seconds. Restart the agent to make this change effective.

Note: A Remote Command job that runs on a Windows workstation that is configured to use the samba protocol version 2 or 3, without an active SSH server, fails.

This section describes the required and optional attributes for remote command jobs. Each job definition has the following format and arguments:

Table 104. Required and optional attributes for the definition of a remote command job

Attribute	Description/value	Required
application name	remotecommand	✓

userName	The user name authorized to start a connection on the remote computer using the defined protocol. As an alternative to hard-coding actual values, you can parametrize in one of the following ways:
	• Enter a *username* specified in the database with the user definition (it is applicable to all operating systems on this job type) and key the statement:
	`<jsdl:password>${password:username}</jsdl:password>`
	The password is retrieved from the *username* user definition in the database and resolved at runtime. See Using user definitions on job types with advanced options on page 285 for further details.
	You can also specify the user of a different workstation and use the following syntax for the password:
	`<jsdl:password>${password:workstation#username}`
	`</jsdl:password>`
	• Enter a user and password defined with the `param` utility command locally on the dynamic agent that will run the job (if the job is to be submitted to a pool or to a dynamic pool, the definition must be present on all the agents of the pool). Provided you defined the user name with the variable *user* and a password, the corresponding credential statements would be:
	`<jsdl:userName>${agent:user}</jsdl:userName>`
	`<jsdl:password>${agent:password.user}</jsdl:password>`
		✓

password	The password of the authorized user. The password is encrypted when the job is created. See description for userName for more details.	
server name	The host name of the computer where the remote command instance is running.	✓
port	The port number of the remote computer where the command runs.	✓
protocol	Possible values:	

Table 104. Required and optional attributes for the definition of a remote command job (continued)

Attribute	Description/value	Required
	AUTO The protocol is selected automatically from the existing protocols: SSH, Windows, RSH and REXEC. The product tries using the SSH protocol first. If this protocol fails, the Windows protocol is used. When using SSH, the path has to be in the SSH format. In this case the Cygwin ssh server is mounted on /home/Administrator. **SSH** A network protocol that provides file access, file transfer, and file management functions over any data stream. **WINDOWS** The Microsoft™ file sharing protocol. The default port used is 445. At least one samba share must exist on the server regardless of the command to be executed. **RSH** Remote Shell Protocol (rsh) is a protocol that allows a user to execute commands on a remote system without having to log in to the system. **REXEC** The Remote Execution (REXEC) server is a Transmission Control Protocol/Internet Protocol (TCP/IP) application that allows a client user to submit system commands to a remote system. The Remote Execution Protocol (REXEC) allows processing of these commands or programs on any host in the network. The local host then receives the results of the command processing.	
keystore file path	The fully qualified path of the keystore file containing the private key used to make the connection. A keystore is a database of keys. Private keys in a keystore have a certificate chain associated with them which authenticates the corresponding public key on the remote server. A keystore also contains certificates from trusted entities. Applicable to SSH protocol only.	
keystore password	The password that protects the private key and is required to make the connection. This attribute is required only if you specify a keystore file path. If the keystore file path and keystore password combination fail to make a connection, then an attempt is made using the userName and password that correspond to the user authorized to start a connection on the remote computer.	✓
command	Type the command to be submitted on the remote computer.	✓
environment	The standard output and standard error files for the remote command. These files are located on the agent, not locally on the workstations where the remote command runs. Ensure you have write rights on the specified directories, otherwise no file will be created. **Standard Output** Specify the path and file name where the standard output for the command is to be saved. Specify either an absolute path name or a path name relative to the working directory. The file is overwritten each time the command produces a new output.	

Table 104. Required and optional attributes for the definition of a remote command job (continued)

Attribute	Description/value	Required
	Standard Error Specify the path and file name where the standard error for the command is to be saved. Specify either an absolute path name or a path name relative to the working directory. The file is overwritten each time the command produces a new error.	

The following example shows the JSDL “application” section of a sample job definition for a remote command job:

```
$JOBS
NC112024#REMCMD
 TASK
 <?xml version="1.0" encoding="UTF-8"?>
<jsdl:jobDefinition xmlns:jsdl="http://www.ibm.com/xmlns/prod/scheduling/1.0/jsdl"
   xmlns:jsdlremotecommand="http://www.ibm.com/xmlns/prod/scheduling/1.0/
    <jsdlremotecommand" name="REMOTECOMMAND">
  <jsdl:application name="remotecommand">
    <jsdlremotecommand:remotecommand>
   <jsdlremotecommand:RemoteCommandParameters>
    <jsdlremotecommand:taskPanel>
     <jsdlremotecommand:command>ping -c 10 localhost </jsdlremotecommand:command>
    </jsdlremotecommand:taskPanel>
    <jsdlremotecommand:environmentPanel>
     <jsdlremotecommand:standardOutput>stdout</jsdlremotecommand:standardOutput>
     <jsdlremotecommand:standardError>stderr</jsdlremotecommand:standardError>
    </jsdlremotecommand:environmentPanel>
    <jsdlremotecommand:serverPanel>
     <jsdlremotecommand:serverInfo>
      <jsdlremotecommand:serverName>9.168.112.16</jsdlremotecommand:serverName>
      <jsdlremotecommand:port>23</jsdlremotecommand:port>
      <jsdlremotecommand:protocol>ssh</jsdlremotecommand:protocol>
     </jsdlremotecommand:serverInfo>
     <jsdlremotecommand:credentials>
      <jsdl:userName>userName</jsdl:userName>
      <jsdl:password>{aes}mv0GJqOHWo8lbuhcpFaluL9RkGQKrYvTiAUpKTMgp90=
             </jsdl:password>
     </jsdlremotecommand:credentials>
     <jsdlremotecommand:certificates>
      <jsdlremotecommand:keystoreFilePath>/var/keyStoreFile</jsdlremotecommand:
                keystoreFilePath>
      <jsdlremotecommand:keystorePassword>pwd</jsdlremotecommand:keystorePassword>
     </jsdlremotecommand:certificates>
    </jsdlremotecommand:serverPanel>
   </jsdlremotecommand:RemoteCommandParameters>
  </jsdlremotecommand:remotecommand>
  </jsdl:application>
</jsdl:jobDefinition>
 RECOVERY STOP
```

See also

From the Dynamic Workload Console you can perform the same task as described in

the *Dynamic Workload Console User's Guide*, section about Creating job definitions.

For more information about how to create and edit scheduling objects, see

the *Dynamic Workload Console User's Guide*, section about Designing your Workload.

Creating custom plug-ins

This topic provides an overview and introduction to how you can extend HCL Workload Automation.

In the previous versions of the product, you could create a custom plug-in using the Integration Workbench to generate a plug-in project. The Integration Workbench is now deprecated, but you can create custom plug-ins using the Workload Automation, Lutist Development Kit.

Before you create a new plug-in, check if the plug-in that you are looking for already exists on Automation Hub.

Automation Hub is a new automation command center that empowers your HCL Workload Automation by continuously delivering new plug-ins.

If you do not find what you need, you can access the Workload Automation, Lutist Development Kit through at Automation Hub.

The Workload Automation, Lutist Development Kit is a maven-based project that enables you to easily and quickly create new plug-ins.

Once your custom plug-in has been created, you can publish it on Automation Hub and share it with the Workload Automation community.

Chapter 18. Using utility commands

This chapter describes HCL Workload Automation utility commands. These commands, with the exceptions listed below, are installed in the *TWA_home*`/bin` directory. You run utility commands from the operating system command prompt.

The **StartUp** is installed in the *TWA_home* directory and **version** is installed in the *TWA_home*`/version` directory.

Command descriptions

Table 105: List of utility commands on page 915 contains the list of the utility commands, and for each command, its description and the operating systems it supports.

Table 105. List of utility commands

Command	Description	Operating system
at	Submits a job to be run at a specific time.	UNIX®
batch	Submits a job to be run as soon as possible.	UNIX®
cpuinfo	Returns information from a workstation definition.	UNIX®, Windows®
dataexport	Exports all scheduling object definitions and global options from the source environment	UNIX®, Windows®
dataimport	Imports all scheduling object definitions and global options to the target environment database	UNIX®, Windows®
datecalc	Converts date and time to a required format.	UNIX®, Windows®
datestconn	Checks connection to a dynamic agent or dynamic workload broker	UNIX®, Windows®
delete	Removes script files and standard list files by name.	UNIX®, Windows®
evtdef	Imports/exports custom events definitions.	UNIX®, Windows®
evtsize	Defines the maximum size of event message files.	UNIX®, Windows®
filemonitor	Checks for changes in files (files that were either created or modified)	UNIX®, Windows®

Table 105. List of utility commands (continued)

Command	Description	Operating system
exportservedata	Downloads the list of dynamic workload broker instances from the HCL Workload Automation database and changes a port number or a host name.	UNIX®, Windows®
importservedata	Uploads the list of dynamic workload broker instances to the HCL Workload Automation database after editing the temporary file to change a port number or a host name.	UNIX®, Windows®
jobinfo	Returns information about a job.	UNIX®, Windows®
jobstdl	Returns the pathnames of standard list files.	UNIX®, Windows®
listproc	Lists processes. This command is not supported.	Windows®
killproc	Kills processes. This command is not supported.	Windows®
maestro	Returns the HCL Workload Automation home directory.	UNIX®, Windows®
makecal	Creates custom calendars.	UNIX®, Windows®
metronome.pl	Is replaced by **wa_pull_info**.	UNIX®, Windows®
morestdl	Displays the contents of standard list files.	UNIX®, Windows®
movehistorydata	Moves the data present in the HCL Workload Automation database to the archive tables.	UNIX®, Windows®
param	Creates, displays, and deletes variables and user passwords on dynamic agents.	UNIX®, Windows®
parms	Displays, changes, and adds parameters.	UNIX®, Windows®
release	Releases units of a resource.	UNIX®, Windows®
rmstdlist	Removes standard list files based on age.	UNIX®, Windows®

Table 105. List of utility commands (continued)

Command	Description	Operating system
sendevent	Sends generic events to the currently active event processor server.	UNIX®, Windows®
showexec	Displays information about executing jobs.	UNIX®
shutdown	Stops the **netman** process and, optionally, WebSphere Application Server Liberty.	UNIX®, Windows®
ShutDownLwa	Stops the agent locally.	UNIX®, Windows® **Note:** On UNIX systems, it can be run by *TWS_user* or root user only.
StartUp	Starts the **netman** process and, optionally, WebSphere Application Server Liberty.	UNIX®, Windows®
StartUpLwa	Starts the agent locally.	UNIX®, Windows® **Note:** On UNIX systems, it can be run by *TWS_user* or root

Table 105. List of utility commands (continued)

Command	Description	Operating system
		user only.
wa_pull_info	Collects data on the local HCL Workload Automation instance and workstation, WebSphere Application Server Liberty, and DB2 for diagnostic purposes. It is documented in wa_pull_info on page 974.	UNIX®, Windows®
version	Displays version information.	UNIX®
wappman	Creates, replaces, deletes, displays, and lists a workload application.	UNIX®, Windows®

at and batch

Submit ad hoc commands and jobs to be launched by HCL Workload Automation.

These command runs on UNIX® only.

See `at.allow` and `at.deny` below for information about the availability to users.

Syntax

at -V | **-U**

at {-s *jstream* | **-q** *queue***}** *time-spec*

batch -V | **-U**

batch [**-s** *jstream*]

Arguments

-V

Displays the command version and exits.

-U

Displays command usage information and exits.

-s jstream

Specifies the *jobstream_id* of the job stream instance into which the job is submitted. If a job stream instance with that *jobstream_id* does not exist, it is created a new job stream having *jstream* both as alias and as

jobstream_id. The name must start with a letter, and can contain alphanumeric characters and dashes. It can contain up to 16 characters.

If the **-s** and **-q** arguments are omitted, a job stream name is selected based on the value of the environment variable ***ATSCRIPT***. If ***ATSCRIPT*** contains the word **maestro**, the job stream alias will be the first eight characters of the user's group name. If ***ATSCRIPT*** is not set, or is set to a value other than **maestro**, the job stream alias will be `at` (for jobs submitted with **at**), or `batch` (for jobs submitted with **batch**).

The following keywords apply only to **at** jobs:

-qqueue

Specifies to submit the job into a job stream with the name *queue*, which can be a single letter (a through z). See Other considerations on page 921 for more information about job streams.

time-spec

Specifies the time at which the job will be launched. The syntax is the same as that used with the UNIX® **at** command.

Comments

After entering **at** or **batch**, enter the commands that constitute the job. End each line of input by pressing the Return key. The entire sequence is ended with end-of-file (usually **Control+d**), or by entering a line with a period (.). Alternatively, use an angle bracket (<) to read commands from a file. See Examples on page 919.

Information about **at** and **batch** jobs is sent to the master domain manager, where the jobs are added to job streams in the production plan, `Symphony` file. The jobs are launched based on the dependencies included in the job streams.

The UNIX® shell used for jobs submitted with the **at** and **batch** commands is determined by the ***SHELL_TYPE*** variable in the `jobmanrc` configuration script. Do not use the C shell. For more information, see Customizing job processing on a UNIX workstation - jobmanrc on page 83.

Once submitted, jobs are launched in the same way as other scheduled jobs. Each job runs in the submitting user's environment. To ensure that the environment is complete, **set** commands are inserted into the script to match the variable settings in the user's environment.

Example

Examples

To submit a job into job stream with *jobstream_id* `sched8` to be launched as soon as possible, run the following command:

```
batch -s sched8
command <Return>
...
<Control d>
```

To submit a job to be launched two hours from the time when the command was entered, run the following command:

```
at now + 2 hours
 command <Return>
...
<Control d>
```

If the variable ***ATSCRIPT*** is null, the job is submitted into a job stream having the same name as the user's group. Otherwise, it is submitted into a job stream named `at`.

To submit a job into a job stream instance with *jobstream_id* `sked-mis` to be launched at 5:30 p.m., run the following command:

```
at -s sked-mis 17h30
 command <Return>
...
<Control d>
```

The following command is the same as the previous command, except that the job's commands are read from a file:

```
at -s sked-mis 17h30 < ./myjob
```

The fact that the commands are read from a file does not change the way they are processed. That is, the commands are copied from the `./myjob` file into a script file.

Replacing the UNIX® commands

The standard UNIX® **at** and **batch** commands can be replaced by HCL Workload Automation commands. The following commands show how to replace the UNIX® **at** and **batch** commands:

```
$ mv /usr/bin/at /usr/bin/uat
$ mv /usr/bin/batch /usr/bin/ubatch
$ ln -s TWShome/bin/at /usr/bin/at
$ ln -s TWShome/bin/batch /usr/bin/batch
```

The at.allow and at.deny files

The **at** and **batch** commands use the files `/usr/lib/cron/`**`at.allow`** and `/usr/lib/cron/`**`at.deny`** to restrict usage. If the `at.allow` file exists, only users listed in the file are allowed to use **at** and **batch**. If the file does not exist, `at.deny` is checked to see if the user is explicitly denied permission. If neither of the files exists, only the **root** user is permitted to use the commands.

Script files

The commands entered with **at** or **batch** are stored in script files. The file are created by HCL Workload Automation using the following naming convention:

TWS_home/atjobs/***epoch.sss***

where:

epoch

The number of seconds since 00:00, 1/1/70.

sss

The first three characters of the job stream name.

Note: HCL Workload Automation removes script files for jobs that are not carried forward. However, you should monitor the disk space in the `atjobs` directory and remove older files if necessary.

Job names

All **at** and **batch** jobs are given unique names by HCL Workload Automation when they are submitted. The names consist of the user's process ID (PID) preceded by the user's name truncated so as not to exceed eight characters. The resulting name is upshifted.

Other considerations

- The job streams into which **at** and **batch** jobs are submitted should be created beforehand with composer. The job streams can contain dependencies that determine when the jobs will be launched. At a minimum, the job streams should contain the **carryforward** keyword. This ensures that jobs that do not complete, or are not launched, while the current production plan is in process are carried forward to the next production plan.
- Include the expression **on everyday** to have the job streams selected every day.
- Use the **limit** keyword to limit the number of submitted jobs that can be run concurrently.
- Use the **priority** keyword to set the priority of submitted jobs relative to other jobs.

If the time value is less than the current time, the value is regarded as for the following day. If the time value is greater than the current time, the value is regarded as for the current day.

cpuinfo

Returns information from a workstation definition.

Syntax

cpuinfo -V | **-U**

cpuinfo *[folder/]workstation* [*infotype*] [...]

Arguments

-V

Displays the command version and exits.

-U

Displays command usage information and exits.

[folder/]workstation

The name of the workstation, optionally preceded by the folder name in which the worstation is defined.

infotype

The type of information to display. Specify one or more of the following:

os_type

Returns the value of the **os** field: **UNIX®**, **WNT**, **ZOS**, **OTHER**, and **IBM i**. The value **ZOS** applies only to remote engine workstations used to communicate to an HCL Workload Automation for Z controller.

node

Returns the value of the **node** field. For a workload broker server it is the host name or the TCP/IP address of the workstation where you installed the HCL Workload Automation Bridge. For a remote engine workstation it is the host name of workstation where the remote engine is installed. In any other case specify the host name or the TCP/IP address of the workstation.

port

Returns the value of the **tcpaddr** field. If you are defining a workload broker workstation, specify the value of the **TWS.Agent.Port** property of the TWSAgentConfig.properties file. For remote engine workstations the value of this field is the HTTP port number used by the remote engine. If HTTPS protocol is used the value of this field is 31111.

sslport

Returns the value of the **secureaddr** field. It is the port used to listen for incoming SSL connections. For remote engine workstations the value of this field is the HTTPS port number used by the remote engine. If HTTP protocol is used the value of this field is 31113.

protocol

Returns the value of the **protocol** field: HTTP or HTTPS. When the type of workstation is remote engine this value indicates the protocol used to communicate between the broker server and the remote engine.

sec_level

Returns the value of the **securitylevel** field: NONE, ENABLED, ON, or FORCE.

autolink

Returns the value of the **autolink** field: ON or OFF.

fullstatus

Returns the value of the **fullstatus** field: ON or OFF.

resolvedep

Returns ON or OFF. No longer used in version 8.6.

behindfirewall

Returns the value of the **behindfirewall** field: ON or OFF.

host

Returns the value of the **host** field. It is the name of the workstation hosting the agent.

domain

Returns the value of the **domain** field.

ID

Returns the agent identifier used by the workstation when connecting to the broker server. For workstation with type: AGENT, REM-ENG, POOL, D-POOL.

method

For extended and network agents only. Returns the value of the **access** field.

server

Returns the value of the **server** field.

type

Returns the value of **type** field. It shows the type of workstation: MASTER, MANAGER, FTA, S-AGENT, REM-ENG, AGENT, POOL, D-POOL and X-AGENT.

time_zone

Returns the value of **timezone** field. It shows the time zone of the workstation. For an extended agent, the field is blank. For a remote engine workstation, this is the time zone of the remote engine.

version

Returns the HCL Workload Automation version that is running on the workstation. For an extended agent, the field is blank.

info

Returns the operating system version and workstation model. For extended agents the field is blank. For remote engine workstations this field displays Remote Engine.

Comments

The values are returned, one on each line, in the same order that the arguments were entered on the command line. If no arguments are specified, all applicable information is returned with labels, one on each line.

Example

Examples

The examples below are based on the following workstation definition:

```
Workstation Name  Type     Domain            Updated On  Locked By
----------------  -------  ----------------  ----------  ------------------
RE-ZOS            REM-ENG  -                 09/06/2023  -

CPUNAME RE-ZOS
  OS ZOS
  NODE 9.168.119.189 TCPADDR 635
  FOR MAESTRO HOST NC123162_DWB
    TYPE REM-ENG
    PROTOCOL HTTP
END
```

To print the **type** and **protocol** for workstation `RE-ZOS`, run the following command:

```
>cpuinfo RE-ZOS type protocol
REM-ENG
HTTP
```

To print all information for workstation `RE-ZOS`, run the following command:

```
>cpuinfo RE-ZOS
 OS_TYPE: ZOS
 NODE: 9.168.119.189
 PORT: 635
 SSLPORT: 31113
 ENGINEADDR: 0
 PROTOCOL: HTTP
 AUTOLINK: OFF
 FULLSTATUS: OFF
 RESOLVEDEP: OFF
 BEHINDFIREWALL: OFF
 HOST: NC123162_DWB
 DOMAIN: MASTERDM
 ID: D795263CBCD2365CA7B5C5BC0C3DD363
 SERVER:
 TYPE: REM-ENG
 TIME_ZONE: Europe/Rome
 VERSION: 10.2.8
 INFO: Remote Engine
```

dataexport

Run the dataexport command or script on a master domain manager to export all scheduling object definitions and global options from a source environment.

Before running the command, specify the passphrase used for encrypting your data. You can use one of the following methods:

- set the **WA_EXPORT_PWD** environment variable with your password in the shell where you plan to run the command. When you run the `export WA_EXPORT_PWD` command, this makes the password accessible to any command run from that shell session, allowing subsequent commands to run without prompting for it again.
- specify the password at the command prompt

Syntax

dataexport *source_dir export_dir*

Arguments

where:

source_dir

is the *TWS_HOME* directory of the source environment instance of HCL Workload Automation, which corresponds to *installation_dir*/TWS.

Note: The dataexport utility only accepts *source_dir* values that end in *TWS*.

export_dir

is the directory where the export files are created. Ensure the user running the command has write rights on this directory.

Example

```
dataexport.cmd F:\IWS101\twsDB2user F:\IWS101\export
```

For further details about the dataexport command usage, see the section about cloning scheduling definition from one environment to another in *Administration Guide*.

dataimport

Run the **dataimport** command or script on a master domain manager to import all scheduling object definitions and global options to the target environment database.

Before running the command, specify the passphrase used for exporting your data. You can use one of the following methods:

- set the **WA_EXPORT_PWD** environment variable with your password in the shell where you plan to run the command. When you run the `export WA_EXPORT_PWD` command, this makes the password accessible to any command run from that shell session, allowing subsequent commands to run without prompting for it again.
- specify the password at the command prompt

Syntax

dataimport *source_dir export_dir*

Arguments

where:

source_dir

is the *TWS_HOME* directory of the source environment instance of HCL Workload Automation, which corresponds to *installation_dir*/TWS.

export_dir

is the directory where you copied the object definitions and the global options retrieved from the source environment.

Example

Example

```
dataimport.cmd /home/twsuser /home/twsuser/export
```

For some scenarios involving the dataimport command, see the sections about cloning scheduling definitions and connecting the master domain manager to a new database in *Administration Guide*.

datecalc

Run the datecalc utility on a master domain manager, a fault-tolerant agent workstation, or a dynamic agent to resolve date expressions and return dates in the format you choose.

On dynamic agents, ensure the Symphony file is available on the agent before using the **freedays** and **workdays** arguments.

Syntax

datecalc -V | **-U**

datecalc *base-date*
[*offset*]
[**pic** *format*]
[**freedays** *Calendar_Name* [**-sa**] [**-su**]]

datecalc -t *time*
[*base-date*]
[*offset*]
[**pic** *format*]

datecalc *yyyymmddhhtt*
[*offset*]
[**pic** *format*]

Arguments

-V

Displays the command version and exits.

-U

Displays command usage information and exits.

base-date

Specify one of the following:

day | *date* | **today** | **tomorrow** | **scheddate**

where:

day

Specifies a day of the week. Valid values are: **su**, **mo**, **tu**, **we**, **th**, **fr**, or **sa**.

date

Specifies a date, in the format *element*/*element*[/*element*], where *element* is: *d*[*d*], *m*[*m*], and *yy*[*yy*]. Any different format of date is not valid.

If two digits are used for the year (*yy*), a number greater than 70 is a 20th century date, and a number less than 70 is a 21st century date.

The parameter refers to the actual date, not to the UNIX *date* command. The following example shows an option to use the output of the UNIX *date* as input for HCL Workload Automation *date* parameter.

```
hdate=´date +"%m/%d/%y"´
echo $hdate
datecalc $hdate pic mm/dd/yyyy
```

Valid values for the month (*m*[*m*]) are **jan**, **feb**, **mar**, **apr**, **may**, **jun**, **jul**, **aug**, **sep**, **oct**, **nov**, or **dec**.

The slashes (/) can be replaced by dashes (-), periods (.), commas (,), or spaces. For example, any of the following can be entered for March 28, 2021:

03/28/21
3-28-2021
28.mar.21
21,28,3
mar 28 2021
28 3 21

If numbers are used, it is possible to enter an ambiguous date, for example, 2,7,04. In this case, **datecalc** uses the date format defined in the HCL Workload Automation message catalog to interpret the date. If the date does not match the format, **datecalc** generates an error message.

today

Specifies the current system date.

tomorrow

Specifies the current system date plus one day, or, in the case of time calculations, plus 24 hours.

scheddate

Specifies the date of the production plan. This might not be the same as the system date. When used inside jobs within a job stream that is not a carried forward job stream, it returns the date when the job should run, which could be different from the production date of the job stream if the job has an `at` dependency specified.

When used inside jobs within a carried forward job stream, it returns the date when the job should have run, which could be different from the production date of the carried forward job stream if the job has an `at` dependency specified. If the `at` dependency is used with the following syntax: **at**=*hhmm* + *n* **days**, the *n* **days** are not added to the variable ***TIVOLI_JOB_DATE*** and therefore, the **datecalc** command does not report these days.

For example, consider a plan for the day 15/01/2021 with a start of day set at 0700, and this schedule

```
SCHEDULE NET92A#JS0200CF
ON RUNCYCLE RULE1 "FREQ=DAILY;"
AT 0200
:
NET92A#DATECALC
END
```

If the job runs at 0200, **datecalc** returns the time 0200 of the day 16/01/2021. If the schedule is carried forward, and the job runs at 1000, the reported result for **datecalc** is the time 1000 of the day 15/01/2021.

-t *time* [*base-date*]

Specify *time* in one of the following formats:

now | **noon** | **midnight** | [*h*[*h*][[:]*mm*] [**am** | **pm**] [**zulu**]

where:

now

Specifies the current system date and time.

noon

Specifies 12:00 p.m. (or 1200).

midnight

Specifies 12:00 a.m. (or 0000).

***h*[*h*][[:]*mm*]**

Specifies the hour and minute in 12-hour time (if **am** or **pm** are used), or 24-hour time. The optional colon (:) delimiter can be replaced by a period (.), a comma (,), an apostrophe ('), the letter **h**, or a space. For example, any of the following can be entered for 8:00 p.m.:

8:00pm
20:00
0800pm
2000
8pm
20
8,00pm
20.00
8\'00pm
20 00

zulu

Specifies that the time you entered is Greenwich Mean Time (Universal Coordinated Time). **datecalc** converts it to the local time.

yyyymmddhhtt

Specifies the year, month, day, hour, and minute expressed in exactly twelve digits. For example, for 2021, May 7, 9:15 a.m., enter the following: **202105070915**

offset

Specifies an offset from *base-date* in the following format:

{**[+** | **]** | **-** | *number* | **nearest]** | **next**} **day[s]** | **weekday[s]** | **workday[s]** | **week[s]** | **month[s]** | **year[s]** | **hour[s]** | **minute[s]** | *day* | *calendar*

where:

+ | **>**

Specifies an offset to a later date or time. Use + (Plus) in Windows®; use > (greater than) in UNIX®. Be sure to add a backslash (\) before the angle bracket (>).

- |

Specifies an offset to an earlier date or time. Use - (Minus) in Windows®; use < (less than) in UNIX®. Be sure to add a backslash (\) before the angle bracket (>).

number

The number of units of the specified type.

nearest

Specifies an offset to the nearest occurrence of the unit type (earlier or later).

next

Specifies the next occurrence of the unit type.

day[s]

Specifies every day.

weekday[s]

Specifies every day except Saturday and Sunday.

workday[s]

Same as **weekday[s]**, but also excludes the dates on the **holidays** calendar.

week[s]

Specifies seven days.

month[s]

Specifies calendar months.

year[s]

Specifies calendar years.

hour[s]

Specifies clock hours.

minute[s]

Specifies clock minutes.

day

Specifies a day of the week. Valid values are: **su**, **mo**, **tu**, **we**, **th**, **fr**, or **sa**.

calendar

Specifies the entries in a calendar with this name.

pic *format*

Specifies the format in which the date and time are returned. The *format* characters are as follows:

m

Month number.

d

Day number.

y

Year number.

j

Julian day number.

h

Hour number.

t

Minute number.

^|/

One space. Use / (slash) in Windows®; use ^ (carat) in UNIX® (add a backslash (\) before the carat (^) if you are in the Bourne shell).

You can also include punctuation characters. These are the same as the delimiters used in *date* and *time*.

If a format is not defined, **datecalc** returns the date and time in the format defined by the Native Language Support (NLS) environment variables. If the NLS variables are not defined, the native language defaults to C.

freedays

Specifies the name of a non-working days calendar *Calendar_Name* that is to replace **holidays** in the evaluation of *workdays*.

In this case, *workdays* is evaluated as *everyday* excluding *saturday*, *sunday*, and all the dates listed in *Calendar_Name*.

By default, *saturday* and *sunday* are not regarded as *workdays*, unless you explicitly state the opposite by adding **-sa** and **-su** after *Calendar_Name*.

You can also specify **holidays** as the name of the non-working days calendar.

Example

Examples

To return the next date, from today, on the `monthend` calendar, run the following command:

```
datecalc today next monthend
```

In the following examples, the current system date is Friday, April 16, 2021.

```
datecalc today +2 days pic mm/dd/yyyy
04/16/2021
```

```
datecalc today next tu pic yyyy\^mm\^dd
2021 04 16
```

```
LANG=american;export LANG
>datecalc -t 14:30 tomorrow
Sat, Apr 17, 2021 02:30:00 PM
```

```
LANG=french;datecalc -t 14:30 tomorrow
Samedi 17 avril 2021 14:30:00
```

In the following example, the current system time is 10:24.

```
datecalc -t now \> 4 hours pic hh:tt
14:24
```

da_test_connection

Run the da_test_connection utility on a dynamic agent to check the connection to a dynamic workload broker or on a dynamic workload broker to check the connection to a dynamic agent.

Syntax

On UNIX operating systems

da_test_connection.sh*hostname* [*port_number* **[IS_DA** *TRUE/FALSE*]]

On Windows operating systems

da_test_connection.bat*hostname*[*port_number* **[IS_DA** *TRUE/FALSE*]]

Arguments

hostname

Specify the hostname of the agent or dynamic workload broker to be checked.

port

Optionally specify the port number on which the connection is to be checked. The default value varies depending on value you specify for the **IS_DA** parameter:

IS_DA=TRUE

In this case the check is performed on the dynamic agent with the specified hostname and the default value of the **port** parameter is **31114**.

IS_DA=FALSE

In this case the check is performed on the dynamic workload broker with the specified hostname and the default value of the **port** parameter is **31116**.

IS_DA

Specify whether the workstation whose connection you want to check is a dynamic agent or dynamic workload broker. Use one of the following values:

TRUE

Specify **TRUE** if you want to check the connection of the dynamic agent with the specified hostname. This is the default value.

FALSE

Specify **FALSE** if you want to check the connection of the dynamic workload broker with the specified hostname.

Example

To check the connection for the dynamic agent with IP address *my_IP_address*, run the following command:

```
da_test_connection.sh my_IP_address 31114 TRUE
```

To check the connection for the dynamic workload broker with hostname *my_IP_address*, run the following command:

```
da_test_connection.sh my_IP_address 31116 FALSE
```

delete

Removes files. Even though this command is intended to remove standard list files you are suggested to use the `rmstdlist` command instead. The users **maestro** and **root** in UNIX®, and **Administrator** in Windows® can remove any file. Other users can remove only files associated with their own jobs.

Syntax

delete -V | -U

delete *filename*

Arguments

-V

Displays the command version and exits.

-U

Displays command usage information and exits.

filename

Specifies the name of the file or group of files to be removed. The name must be enclosed in quotes (") if it contains characters other than the following: alphanumerics, dashes (-), slashes (/), backslashes (\), and underscores (_). Wildcard characters are permitted.

Note: Use this command carefully. Improper use of wildcard characters can result in removing files accidentally.

Example

Examples

To remove all the standard list files for 4/11/24, run the following command:

```
delete d:\win32app\maestro\stdlist\2024.4.11\@
```

The following script, included in a scheduled job in UNIX®, removes the job's standard list file if there are no errors:

```
...
#Remove the stdlist for this job:
if grep -i error $UNISON_STDLIST
then
exit 1
else
`maestro`/bin/delete $UNISON_STDLIST
fi
...
```

The standard configuration script, `jobmanrc`, sets the variable ***UNISON_STDLIST*** to the name of the job standard list file. For more information about `jobmanrc`, see the section about customizing job processing on a UNIX workstation in *User's Guide and Reference*.

evtdef

Imports/exports a generic event provider XML definition file where you can add and modify custom event types. You can then use the **sendevent** command to send these events to the event processing server. See also the section about defining custom events in *User's Guide and Reference*.

Syntax

evtdef -U | **-V**

evtdef [*connection parameters*] **dumpdef** *file-path*

evtdef [*connection parameters*] **loaddef** *file-path*

Arguments

-U

Displays command usage information and exits.

-V

Displays the command version and exits.

connection parameters

If you are using **evtdef** from the master domain manager, the connection parameters were configured at installation and do not need to be supplied, unless you do not want to use the default values.

If you are using **evtdef** from the command line client on another workstation, the connection parameters might be supplied by one or more of these methods:

- Stored in the `localopts` file
- Stored in the `useropts` file
- Supplied to the command in a parameter file
- Supplied to the command as part of the command string

For an overview of these options, see the section about setting up options for using the user interfaces in *User's Guide and Reference*.

For full details of the configuration parameters see the topic on configuring the command-line client access in the *Administration Guide*.

dumpdef *file-path*

Downloads the generic event provider XML file. The file is downloaded with the file name and path you provide in *file-path*. You can edit the file to add your own custom event types.

The name of the generic event provider supplied with the product is `GenericEventPlugIn`. You can change this name by acting on the `name` tag of the `eventPlugin` keyword.

Important: You must use this name as the value of:

- The `source` keyword of the sendevent on page 967 command
- The `eventProvider` keyword in the definition of the event rules triggered by these custom events.

loaddef *file-path*

Uploads the modified generic event provider XML file from the file and path you provide in *file-path*.

Comments

The following rule language schemas are used to validate your custom event definitions and, depending upon the XML editor you have, to provide syntactic help:

- eventDefinitions.xsd
- common.xsd

The files are located in the `schemas` subdirectory of the HCL Workload Automation installation directory.

When you download the generic event provider template file, it looks like this:

```
<?xml version="1.0" encoding="UTF-8"?>
 <eventDefinitions
xmlns="http://www.ibm.com/xmlns/prod/tws/1.0/event-management/plugins/events"
```

```
xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
xsi:schemaLocation="http://www.ibm.com/xmlns/prod/tws/1.0/event-management/
plugins/events/eventDefinitions.xsd">
eventPlugin>
<complexName displayName="Custom event" name="GenericEventPlugIn"/>
<scopes>
<scope name="Generic">
<scopedef text="{Param1} on {Workstation}"/>
</scope>
</scopes>

 <event baseAliasName="genericEvt" scope="Generic">
 <complexName displayName="Generic event" name="Event1"/>
 <displayDescription>The event is sent when the specified expression is
 matched.</displayDescription>
 <property type="string" required="true" wildcardAllowed="true"
multipleFilters="true" minlength="1">
<complexName displayName="Parameter 1" name="Param1"/>
<displayDescription>The value of parameter 1</displayDescription>
 </property>
 <property type="string" required="true" wildcardAllowed="false"
multipleFilters="false" minlength="1>
<complexName displayName="Workstation" name="Workstation"/>
<displayDescription>The workstation for which the event is
generated.</displayDescription>
</property>
</property>
</event>
</eventPlugin>
</eventDefinitions>
```

You then edit this file to add the property types that you need to define a specific event. You can add the following property types:

Table 106. Additional properties that can be used for defininig custom events.

Property type	Add into XML event file as shown
boolean	`<property type="boolean" required="false" wildcardAllowed="false" multipleFilters="true" minlength="1"> <complexName displayName="Boolean field" name="Boolean"/> <displayDescription>Add a boolean field</displayDescription> </property>`
date	`<property type="date" required="false" wildcardAllowed="false" multipleFilters="true" minlength="1"> <complexName displayName="Date field" name="Date"/> <displayDescription>Add a date field</displayDescription> </property>`
datetime	`<property type="datetime" required="false" wildcardAllowed="false" multipleFilters="true" minlength="1"> <complexName displayName="Date Time field" name="Datetime"/> <displayDescription>Add a date time field</displayDescription> </property>`
datetimeutc	`<property type="datetimeutc" required="false" wildcardAllowed="false" multipleFilters="true" minlength="1"> <complexName displayName="Date Time UTC field" name="Datetimeutc"/> <displayDescription>Add a date time UTC field</displayDescription> </property>`

Table 106. Additional properties that can be used for defininig custom events.

(continued)

Property type	Add into XML event file as shown
duration	`<property type="duration" required="false" wildcardAllowed="false" multipleFilters="true" minlength="1"> <complexName displayName="Duration field" name="Duration"/> <displayDescription>Add a duration field</displayDescription> </property>`
fileSize	`<property type="fileSize" required="false" wildcardAllowed="false" multipleFilters="true" minlength="1"> <complexName displayName="File size field" name="filesize"/> <displayDescription>Add a file size field</displayDescription> </property>`
nonnegativeinteger	`<property type="nonnegativeinteger" required="false" wildcardAllowed="false" multipleFilters="true" minlength="1"> <complexName displayName="Non negativeinteger field" name="nonnegativeinteger"/> <displayDescription>Add a non negativeinteger field</displayDescription> </property>`
numeric	`<property type="numeric" required="false" wildcardAllowed="false" multipleFilters="true" minlength="1"> <property type="numeric" required="false" wildcardAllowed="false" multipleFilters="true" minlength="1"> <displayDescription>Add a numeric field</displayDescription> </property>`
percentage	`<property type="percentage" required="false" wildcardAllowed="false" multipleFilters="true" minlength="1"> <complexName displayName="Percentage field" name="percentage"/> <displayDescription>Add a percentage field</displayDescription> </property>`
string	`<property type="string" required="true" wildcardAllowed="true" multipleFilters="true" minlength="1"> <complexName displayName="String with wildcards" name="StringWithWildcards"/> <displayDescription>Add a string with wildcards</displayDescription> </property>`
	or
	`<property type="string" required="true" wildcardAllowed="false" multipleFilters="true" minlength="1"> <complexName displayName="String without wildcards" name="StringWithoutWildcards"/> <displayDescription>Add a string without wildcards</displayDescription> </property>`

You can change the values of all the property attributes, with the exception of `type`, to fit your requirements.

The properties so defined are converted into input fields after the event definition is uploaded and opened in the Dynamic Workload Console.

You can also define more than one event by repeating `<eventPlugin>...</eventPlugin>` sections. For example:

```
<?xml version="1.0" encoding="UTF-8"?>
  <eventDefinitions
xmlns="http://www.ibm.com/xmlns/prod/tws/1.0/event-management/plugins/events"
xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
xsi:schemaLocation="http://www.ibm.com/xmlns/prod/tws/1.0/event-management/
plugins/events/eventDefinitions.xsd">
  <eventPlugin>
  <complexName displayName="Custom event" name="GenericEventPlugIn"/>
  <scopes>
  <scope name="Art042StockQuantity">
```

```
  <scopedef text="{Art042pieces}"/>
  </scope>
  </scopes>

  <event baseAliasName="genericEvt" scope="Generic">
  <complexName displayName="Stock of article 042 reaches minimum level"
name="art042qty"/>
<displayDescription>The event is sent when the number of art.42
items on stock reaches
minimum level.</displayDescription>
<property type="numeric" required="true" wildcardAllowed="false"
multipleFilters="true" minlength="1">
<complexName displayName="Art042 items on stock" name="art042items"/>
<displayDescription>The number of art042 items left</displayDescription>
</property>
</event>

  <event baseAliasName="Hard drive saturation" scope="Generic">
  <complexName displayName="Hard drive saturation" name="HDSatEvent"/>
  <displayDescription>displayDescription>The event is sent when the percentage field
  reaches the warning level.</displayDescription>
  <property type="percentage" required="true" wildcardAllowed="false"
multipleFilters="true" minlength="1>
<complexName displayName="Percentage Full" name="PercentFull"/>
<displayDescription>The percentage of total disk space used</displayDescription>
</property>
<property type="string" required="true" wildcardAllowed="false"
multipleFilters="false" minlength="1>
<complexName displayName="Workstation" name="Workstation"/>
<displayDescription>The workstation where the hard drive is installed
</displayDescription>
</property>
</event>
  </eventPlugin>
  </eventDefinitions>
```

Example

Examples

In this example you:

1. Download the generic event provider XML file as file `c:\custom\myevents.xml`

   ```
   evtdef dumpdef c:\custom\myevents.xml
   ```

2. Edit the file to add your own event type definitions.
3. When finished, you upload the generic event provider XML file from file `c:\custom\myevents.xml`

   ```
   evtdef loaddef c:\custom\myevents.xml
   ```

evtsize

Defines the size of the HCL Workload Automation message files. This command is used by the HCL Workload Automation administrator either to increase the size of a message file after receiving the message, “End of file on events file.”, or to monitor the size of the queue of messages contained in the message file. You must be **maestro** or **root** in UNIX®, or **Administrator** in Windows® to run **evtsize**. Stop the HCL Workload Automation engine before running this command.

Syntax

evtsize -V | **-U**

evtsize *filename size*

evtsize -compact *filename* [*size*]

evtsize -info *filename*

evtsize -show *filename*

evtsize -info | **-show pobox**

Arguments

-V

Displays the command version and exits.

-U

Displays command usage information and exits.

-compact *filename* [*size*]

Reduces the size of the specified message file to the size occupied by the messages present at the time you run the command. You can optionally use this keyword to also specify a new file size.

-info *filename*

Displays the percentage use of the queue of messages contained in the message file.

-show *filename*

Displays the size of the queue of messages contained in the message file

filename

The name of the event file. Specify one of the following:

`Courier.msg`

`Intercom.msg`

`Mailbox.msg`

`Planbox.msg`

`pobox/workstation.msg`

`mirrorbox.msg`

`mirrorbox<n>.msg`

size

The maximum size of the event file in bytes. It must be no less than 1048576 bytes (1 MB).

When first built by HCL Workload Automation, the maximum size is set to 60 MB.

Note: The size of the message file is equal to or bigger than the real size of the queue of messages it contains and it progressively increases until the queue of messages becomes empty; as this occurs the message file is emptied.

-info | -show pobox

Displays the name of the message file, within the `pobox` directory, with the largest queue size calculated as a percentage of the total file size. Both the name of the file and the percentage used are returned. Either **-info** and **-show** return the same results.

Example

Examples

To set the maximum size of the `Intercom.msg` file to 20 MB, run the following command:

```
evtsize Intercom.msg 20000000
```

To set the maximum size of the `pobox` file for workstation `chicago` to 15 MB, run the following command:

```
evtsize pobox\chicago.msg 15000000
```

The following command:

```
evtsize -show Intercom.msg
```

returns the following output:

```
HCL Workload Automation (UNIX)/EVTSIZE 10.2.8 (20241118)
Licensed Materials - Property of IBM* and HCL**
5698-WSH
(C) Copyright IBM Corp. 1998, 2016 All rights reserved.
(C) Copyright HCL Technologies Ltd. 2016, 2026 All rights reserved.
* Trademark of International Business Machines
** Trademark HCL Technologies Limited
AWSDEK703I Queue size current 880, maximum 10000000 bytes (read 48, write 928)
```

where:

880

Is the size of the current queue of the `Intercom.msg` file

10000000

Is the maximum size of the `Intercom.msg` file

read 48

Is the pointer position to read records

write 928

Is the pointer position to write records

If the following command:

```
evtsize -info Mailbox.msg
```

returns:

```
25
```

it means that 25 percent of the file has been used.

Filemonitor

Use the **filemonitor** utility to check for file changes (files that were either created or modified). This could be useful when, for example, you want to make sure that a file exists before running a job that processes that file. By defining a job that runs the filemonitor utility, you can implement file dependency, that is, a relationship between a file and an operation in which specific activity on the file determines the start of the operation.

You can use the **filemonitor** utility as a stand-alone command, or you can set the **filemonitor** keywords as additional parameters for the start condition of a job stream, either in the Workload Designer or from the **composer** command line. For more information about the start condition, see Condition-based workload automation on page 169.

Run the **filemonitor** utility to monitor whether the specified files have been modified within a time interval. All product components must be at least at the Version 9.4, Fix Pack 1 level.

Syntax

filemonitor -V | -U

filemonitor {**-path** *path_to_monitor* | **-epath** *path_to_monitor*}
[-**exitOnPathToMonitorNotFound**]
-**event** {**fileCreated** | **fileModified**} [-**modificationCompletedTime** *seconds*]
[-**repositoryName** *repository_name*]
[-**repositoryPath** *repository_path*]
[-**recursive**]
[-**outputFile** *output_filename*]
[-**scanInterval** *scan_interval*]
[-**maxEventsThreshold** *max_events*]
[-**minFileSize** *min_file_size*]

[-**timeout** *seconds* | *time_of_the_day*]
[-**preserveEventsOnDelete**]

filemonitor -reset
[-**repositoryName** *repository_name*]
[-**repositoryPath** *repository_path* - **generateEventsOnFirstScan**]

Arguments

Note: If you set the same argument more than once, the last value specified is applied and no error message is reported.

-V

Displays the command version and exits.

-U

Displays command usage information and exits.

-path *path_to_monitor*

The path where the files to be processed are located. You can specify blank or special characters within double quotation marks. Wildcard characters are supported. To include more files in the monitoring process, store the files in a specific directory and specify the directory with the -**path** option.

The following syntax rules apply:

- Paths containing blank or special characters must be specified within double quotation marks.
- Wildcard characters question mark (?) and asterisk (*) are supported.

Universal Naming Convention (UNC) paths are also supported with the following syntax types:

- `\\server_name\share_name\directory_name\...`
- `\\?\UNC\server_name\share_name\directory_name`
- `\\?\path_name`

where the question mark (**?**) indicates extended-length paths.

-epath *path_to_monitor*

The path where the files to be processed are located, specified with slashes (/) as separators. Backslashes (\) are not allowed as separators, even if you are indicating a Windows™ path. To include more files in the monitoring process, store all the files in the directory set with the -**epath** argument.

The following syntax rules apply:

- Paths containing blank or special characters must be specified within double quotation marks.
- Supported wildcard characters: question mark (?) and asterisk (*).
- Any character other than backslash (\), question mark (?), asterisk (*), square brackets ([ ]) or a backslash at the end of the value (\) is intended exactly as it is written. For example, `MYpath` is not equivalent to `mypath`.
- Use the syntax `[class description]` to indicate a single character as follows:

  `[range_of_characters]`

  A range of characters separated by a minus sign (-). For example, `A-B` or `1-9`.

  `[list_of_characters]`

  A string of characters. For example, `ABC` or `1aX`.

- The characters exclamation mark (!) and caret (^) are used to reverse the sense. For example, `[!A-Z]` matches a single character that is not equivalent to any letter from A to Z. `[!F]` matches any character that is not F.

For example:

- `-epath /mypath/myp?th/e[!1].txt`
- `-epath /mypath/my[1-9]path/e[A-Z].txt`
- `-epath c:/mypath/p?th/e[!1].txt`

[-exitOnPathToMonitorNotFound]

Optionally, specify this argument to have the command exit if the specified path is not found.

-event {fileCreated | fileModified | fileAccessed} [-modificationCompletedTime *seconds* | emodificationCompletedTime*seconds*]

The event type to be monitored. Supported types are **fileCreated**, **fileModified** and **fileAccessed**. This argument is required when you specify -**path**. If you specify the **fileCreated**, **fileModified** or **fileAccessed** argument, you can optionally specify the **-modificationCompletedTime** or **-emodificationCompletedTime** option which is a time interval, in seconds that is used to determine when the event is sent.

-event fileCreated

As soon as the file is created , the event, `FileCreated`, is sent.

-event fileModified

As soon as the file is modified, the event, `FileModified`, is sent.

-event fileAccessed

As soon as the file is accessed, the event, `FileAccessed`, is sent.

-event fileCreated -modificationCompletedTime <*seconds*>

When a file is created, the event is not sent until the interval specified in -**modificationCompletedTime** <*seconds*> elapses. During this interval, no subsequent changes were made to the file, which includes the file being deleted and recreated with the same name.

-event fileModified -modificationCompletedTime <*seconds*>

When a file is modified, the event is not sent until the interval specified in -**modificationCompletedTime** <*seconds*> elapses. During this interval, no additional changes were made to the file.

-event fileAccessed -modificationCompletedTime <*seconds*>

When a file is accessed, the event is not sent until the interval specified in -**modificationCompletedTime** <*seconds*> elapses. During this interval, no additional changes were made to the file.

-event fileCreated -emodificationCompletedTime <*seconds*> (available for Windows® operating system only)

When a file is created, the event is not sent until the interval specified in -**emodificationCompletedTime** <*seconds*> elapses. During this interval—on Windows® systems only—the system verifies that the file is available for shared read access.

-event fileModified -emodificationCompletedTime <*seconds*> (available for Windows® operating system only)

When a file is modified, the event is not sent until the interval specified in -**emodificationCompletedTime** <*seconds*> elapses. During this interval—on Windows® systems only—the system verifies that the file is available for shared read access.

-event fileAccessed -emodificationCompletedTime <*seconds*> (available for Windows® operating system only)

When a file is accessed, the event is not sent until the interval specified in -**emodificationCompletedTime** <*seconds*> elapses. During this interval—on Windows® systems only—the system verifies that the file is available for shared read access.

-repositoryName *repository_name*

Optionally, specify a database where to log the status of the retrieved files.. By default, the name `filemonitor.db` is used if you do not specify any names.

-repositoryPath *repository_path*

The path to the **filemonitor** database. By default, the following path is used, if you do not specify any paths:

On Windows operating systems

`TWA_Home\TWS\stdlist\JM\filemonitor`

On UNIX operating systems

TWA_DATA_DIR`/TWS/stdlist/JM/filemonitor`

Paths containing spaces must be enclosed in double quotes. Wildcards are not supported.

-generateEventsOnFirstScan

All files retrieved during the first scan performed by filemonitor are considered as created or modified and can generate events. This argument is available only if you specify the **repositoryPath** argument.

-recursive

Include sub-folders when monitoring files. This is an optional parameter.

-outputFile ***output_filename***

An output file where to store the retrieved events. Ensure that the directory where the output file is to be created is already existing. The command output is also printed to standard output and stored in the job properties, if you launch the **filemonitor** command from a job. Paths containing spaces must be enclosed in double quotes. Wildcards are not supported. This is an optional parameter.

-scanInterval ***scan_interval***

A period of time in seconds between two consecutive checks on the files being created or modified. The default value is 300 seconds. The supported range is 1-3600 seconds. This is an optional parameter.

-maxEventsThreshold ***max_events***

The maximum number of events to be returned. The default value is **1**. If you specify **all**, all events are returned. This is an optional parameter.

-minFileSize ***min_file_size***

The minimum size in bytes that files must reach to be included in the scan. The default value is **0**.

-timeout ***seconds*** **|** ***hh:mm:ss***

The maximum time, in seconds, that **filemonitor** runs or the time of the current day until which **filemonitor** runs. If you do not specify this parameter, **filemonitor** waits indefinitely. The time refers to the local time of the workstation where **filemonitor** is running. This is an optional parameter.

-preserveEventsOnDelete

Returns events on the specified file, also if the file was deleted in the meantime. If you do not specify this argument, when a file is deleted all events preceding the file deletion, if any, are discarded.

-reset

Resets the information collected. With this argument, you can optionally specify the **-repositoryPath** and **-repositoryName** arguments.

Configuring trace properties for filemonitor

To configure the trace properties, edit the [`FileMonitor.Logging`] section in the `FileMonitor.ini` file located in the following path:

On Windows operating systems

`<`*`TWAHome`*`>\TWS\ITA\cpa\config/`

On UNIX operating systems

TWA_DATA_DIR/TWS/ITA/cpa/config/

and restart the **filemonitor** utility.

The section containing the trace properties is named:

```
[FileMonitor.Logging.cclog]
```

FileMonitor.trhd.fileName

The name of the trace file.

FileMonitor.trhd.maxFileBytes

The maximum size that the trace file can reach. The default is 1024000 bytes.

FileMonitor.trhd.maxFiles

The maximum number of trace files that can be stored. The default is 3.

FileMonitor.trfl.level

Determines the type of trace messages that are logged. Change this value to trace more or fewer events, as appropriate, or on request from Software Support. Valid values are:

DEBUG_MAX

Maximum tracing. Every trace message in the code is written to the trace logs.

INFO

All *informational*, *warning*, *error* and *critical* trace messages are written to the trace. The default value.

WARNING

All *warning*, *error* and *critical* trace messages are written to the trace.

ERROR

All *error* and *critical* trace messages are written to the trace.

CRITICAL

Only messages which cause the agent to stop are written to the trace.

The output trace (by default, `FileMonitor_trace.log`) is provided in XML format, and is located in `<TWA_Home>/TWS/stdlist/JM`.

Return Codes

0

The operation completed successfully.

4

Filemonitor stopped running, because timeout expired. No results were returned.

8

Filemonitor cannot run because the timeout is set to a time that is already passed.

12

Filemonitor stopped running because the timeout expired without having ever successfully accessed the repository.

-1

An error occurred. Search the trace log (by default, FileMonitor_trace.log) for additional details.

Comments

Any parameters defined with this command override the default values internally set by HCL Workload Automation.

If one or more files have been created or modified in between subsequent invocations, the modifications are detected. However, files already detected in a previous run are not listed again in subsequent invocations. Wildcards are supported in both file names and directory names.

Example

Examples

In the following example, the **filemonitor** command checks every 2 minutes for all files created in the `C:\temp\logs` path and having a minimum size greater than 1024 bytes. The check is performed on all sub folders and the results are stored in `C:\backup\logs\reports.txt`:

```
filemonitor  -path "C:\temp\logs" -event fileCreated -recursive  -outputFile "C:\backup\logs\reports.txt"
 -scanInterval 120 -maxEventsThreshold all  -minFileSize 1024
```

jobinfo

Used in a job script to return information about the job. This command is not supported on dynamic agents, pools, dynamic pools, and job types with advanced options.

Syntax

jobinfo -V | **-U**

jobinfo *job-option* [...]

Arguments

-V

Displays the command version and exits.

-U

Displays command usage information and exits.

job-option

The job option. Specify one or more of the following:

confirm_job

Returns **YES** if the job requires confirmation.

is_command

Returns **YES** if the job was scheduled or submitted using the **docommand** construct.

job_name

Returns the job's name without the workstation and job stream names.

job_pri

Returns the job's priority level.

programmatic_job

Returns **YES** if the job was submitted with using the **at** or **batch** command. UNIX® only.

re_job

Returns **YES** if the job is being rerun as the result of a **conman rerun** command, or the rerun recovery option.

re_type

Returns the job's recovery option (**stop**, **continue**, or **rerun**).

rstrt_flag

Returns **YES** if the job is being run as the recovery job.

rstrt_retcode

If the current job is a recovery job, returns the return code of the parent job.

schedule

Returns the name of the job stream where the job is submitted.

schedule_ia

Returns the time and date the job stream is scheduled to start.

schedule_id

Returns the *jobstream_ID* of the job stream where the job is submitted.

time_started

Returns the time the job started running.

Comments

Job option values are returned, one on each line, in the same order they were requested.

Example

Examples

1. The script file `/jcl/backup` is referenced twice, giving it the job names **partback** and **fullback**. If the job runs as **partback**, it performs a partial backup. If it runs as **fullback**, it performs a full backup. Within the script, commands like the following are used to make the distinction:

   ```
   #Determine partial (1) or full (2):
   if [ "`\`maestro\`/bin/jobinfo job_name`" = "PARTBACK" ]
   then
   bkup=1
   else
   bkup=2
   fi
   ...
   ```

2. To display the return code of the parent job, if the current job is a recovery job, run the following command:

   ```
   $ jobinfo rstrt_retcode
   ```

   The first job (parent job) has been defined in the script `recovery.sh` while the second job (recovery job) gets enabled only if the first job abends.

   When combined with a return code condition, **jobinfo rstrt_retcode** can be used to direct the recovery job to take different actions depending on the parent job's return code. A recovery job is shown in the example below:

   ```
   $JOBS
   MASTER#DBSELOAD DOCOMMAND "/usr/local/tws/maestro/scripts/populate.sh"
   STREAMLOGON "^TWSUSER^"
   DESCRIPTION "populate database manual"
   RECOVERY RERUN AFTER MASTER#RECOVERY
   RCCONDSUCC "(RC = 0) OR ((RC > 4) AND (RC < 11))"
   ```

   Note: The job is defined with the recovery action RERUN. This enables the recovery job to take some corrective action, before the parent job attempts to run again.

   The recovery job itself is defined as shown in the example below:

   ```
   $ JOBS
   MASTER#RECOVERY DOCOMMAND "^TWSHOME^/scripts/recovery.sh"
   STREAMLOGON "^TWSUSER^"
   DESCRIPTION "populate database recovery manual"
   RECOVERY STOP
   ```

jobstdl

Returns the names of standard list files. This command must be run by the user for which HCL Workload Automation was installed. If you use this command without any parameters, ensure that you are logged on as an HCL Workload Automation user.

Syntax

jobstdl -V | **-U**

jobstdl
[**-day** *num*]
[{**-first** | **-last** | **-num** *n* | **-all**}]
[**-twslog**]
[{**-name** ["*[folder/]jobstreamname* [(*hhmm date*),(*jobstream_id*)].]*jobname*"
| *jobnum* | **-schedid** *jobstream_id*.*jobname*}]

Arguments

-V

Displays the command version and exits.

-U

Displays command usage information and exits.

-day *num*

Returns the names of standard list files that are the specified number of days old (1 for yesterday, 2 for the day before yesterday, and so on). The default is zero (today).

-first

Returns the name of the first qualifying standard list file.

-last

Returns the name of the last qualifying standard list file.

-num *n*

Returns the name of the standard list file for the specified run of a job.

-all

Returns the name of all qualifying standard list files.

-twslog

Returns the path of the current day *stdlist* file.

-name ["*[folder/]jobstreamname*[(*hhmm date*), (*jobstream_id*)].]*jobname*" | *jobnum*

Specifies the instance of the job stream and name of the job for which standard list file names are returned.

jobnum

Specifies the job number of the job for which standard list file names are returned.

-schedid *jobstream_id*.*jobname*

Specifies the job stream ID and name of the job for which standard list file names are returned.

Comments

File names are returned in a format suitable for input to other commands. Multiple names are returned separated by a space.

When you use the full syntax of the **-name** argument, the square brackets in the expression `[(hhmm date), (jobstream_id)]` are part of the command, not syntax indicators. Also, the whole job identification string must be enclosed in double quotation marks if the part identifying the job stream instance contains blanks. For example, because the *schedtime*, represented by `hhmm date`, has a space in it, you must enclose the whole job identification in double quotation marks.

You can also run abbreviated versions of the **-name** argument using a simpler syntax. If you want less specific outputs from the command, you can specify just the *schedtime* (the *date* is not required if it is for the same day) or the *jobstream_id* together with the *jobname*. As long as there are no blanks in the arguments, you can omit the double quotation marks. You can also omit the square brackets if you do not specify both the *schedtime* and the *jobstream_id*.

The following examples show the syntax you must use with the **-name** argument for the different types of information you expect in return, ranging from the more specific to the more general. In the example, `job_stream1` is the name of the job stream, `0600 04/05/06` is the scheduled time, `0AAAAAAAAAAAAAB5` is the job stream ID, and `job1` is the job name. The job number of `job1` is `310`. You can run **jobstdl** for `job1` as follows:

```
jobstdl -name "job_stream1[(0600 04/05/10),(0AAAAAAAAAAAAAB5)].job1"
```

Returns the standard list file name of `job1` for the specific instance of `job_stream1` with the specified *schedtime* and *jobstream_id*.

```
jobstdl -name job_stream1(0AAAAAAAAAAAAAB5).job1
```

Returns the standard list file name for `job1` for the instance of `job_stream1` with ID `0AAAAAAAAAAAAAB5`.

```
jobstdl -name "job_stream1(0600 04/05/10).job1"
```

Returns the standard list file names for `job1` for all possible instances of `job_stream1` scheduled to run at 0600 of 04/05/10.

```
jobstdl -name job_stream1(0600).job1
```

Returns the standard list file names for `job1` for all possible instances of `job_stream1` scheduled to run at 0600 of the current day.

```
jobstdl -name 310
```

Returns the standard list file names for `job1` for all the instances it had job number `310`.

Example

Examples

To return the path names of all standard list files for the current day, run the following command:

```
jobstdl
```

To return the path name of the standard list for the first run of job `MY_CPU#ELI[(1824 03/09/06),(0AAAAAAAAAAAAAEE)].DIR` on the current day, run the following command:

```
jobstdl -first -name "MY_CPU#ELI[(1824 03/09/06),(0AAAAAAAAAAAAAEE)].DIR"
```

To return the path name of the standard list for the first run of job `0AAAAAAAAAAAAAEE.DIR` on the current day, run the following command:

```
jobstdl -first -schedid 0AAAAAAAAAAAAAEE.DIR
```

To return the path name of the standard list for the second run of job `MY_CPU#ELI[(1824 03/09/06),(0AAAAAAAAAAAAAEE)].DIR` on the current day, run the following command:

```
jobstdl -num 2 -name "MY_CPU#ELI[(1824 03/09/06),(0AAAAAAAAAAAAAEE)].DIR"
```

To return the path names of the standard list files for all runs of job `MY_CPU#ELI[(1824 03/09/06),(0AAAAAAAAAAAAAEE)].DIR` from three days ago, run the following command:

```
jobstdl -day 3 -name "MY_CPU#ELI[(1824 03/09/06),(0AAAAAAAAAAAAAEE)].DIR"
```

To return the path name of the standard list for the last run of job `MY_CPU#ELI[(1824 03/09/06),(0AAAAAAAAAAAAAEE)].DIR` from four days ago, run the following command:

```
jobstdl -day 4 -last -name "MY_CPU#ELI[(1824 03/09/06),(0AAAAAAAAAAAAAEE)].DIR"
```

To return the path name of the standard list for job number **455**, run the following command:

```
jobstdl 455
```

To print the contents of the standard list file for job number **455**, run the following command:

```
cd `maestro`/bin
lp -p 6 `jobstdl 455`
```

maestro

Returns the path name of the HCL Workload Automation home directory, referred to as *TWS_home*.

Syntax

maestro [**-V** | **-U**]

Arguments

-V

Displays the command version and exits.

-U

Displays command usage information and exits.

Example

Examples

To display the HCL Workload Automation home directory, run the following command:

```
$ maestro
/usr/lib/maestro
```

To change the directory to the HCL Workload Automation home directory, run the following command:

```
$ cd `maestro`
```

makecal

Creates a custom calendar. In UNIX®, the Korn shell is required to run this command.

Syntax

makecal [**-V** | **-U**]

makecal
[**-c** *name*]
-d *n*
| **-e**
| {**-f 1** | **2** | **3 -s** *date*}
| **-l**
| **-m**
| **-p** *n*
| {**-r** *n* **-s** *date*}
| **-w** *n*
[**-i** *n*]
[**-x** | **-z**]
[**-freedays** *[folder/]Calendar_Name* [**-sa**] [**-su**]]

Arguments

-V

Displays the command version and exits.

-U

Displays command usage information and exits.

-c *name*

Specifies a name for the calendar. HCL Workload Automation keywords (such as *Freedays* or *Schedule*) cannot be used as calendar names. The name can contain up to eight alphanumeric characters and must start with a letter. Do not use the names of weekdays for the calendar names. The default name is: **C***hhmm*, where *hhmm* is the current hour and minute.

-d *n*

Specifies the *n*th day of every month.

-e

Specifies the last day of every month.

-f 1 | 2 | 3

Creates a fiscal month-end calendar containing the last day of the fiscal month. Specify one of the following formats:

1

4-4-5 week format

2

4-5-4 week format

3

5-4-4 week format

This argument requires the **-s** argument.

-i *n*

Specifies to insert *n* dates in the calendar.

-l

Specifies the last workday of every month. For this argument to work properly, the production plan (`Symphony` file) and the **holidays** calendar must already exist.



Note: Using this argument results in the new calendar also including the last workday of the month that precedes the date of creation of the calendar.

-m

Specifies the first and fifteenth days of every month.

-p *n*

Specifies the workday before the *n*th day of every month. For this argument to work properly, the production plan (`Symphony` file) and the **holidays** calendar must already exist

-r *n*

Specifies every *n*th day. This argument requires the **-s** argument.

-s *date*

Specifies the starting date for the **-f** and **-r** arguments. The date must be enclosed in quotation marks, and must be valid and unambiguous, for example, use **JAN 10 2024**, not **1/10/24**. See *base-date* for **datecalc** on page base-date on page 927 for more information about date formats.

-w *n*

Specifies the workday after the *n*th of the month. For this argument to work properly, the production plan (Symphony file) and the **holidays** calendar must already exist.

-x

Sends the calendar output to **stdout** instead of adding it to the database.

-z

Adds the calendar to the database and compiles the production plan (`Symphony` file).

Note: This argument re-submits jobs and job streams from the current day's production plan. It might be necessary to cancel job streams and jobs.

-freedays

Specifies the name of a non-working days calendar *Calendar_Name*, optionally preceded by the folder within which it is defined, that is to replace **holidays** in the evaluation of *workdays*.

In this case, *workdays* is evaluated as *everyday* excluding *saturday*, *sunday* and all the dates listed in *Calendar_Name*.

By default, *saturday* and *sunday* are not regarded as *workdays*, unless you explicitly state the opposite by adding **-sa** and/or **-su** after *Calendar_Name*.

You can also specify **holidays** as the name of the non-working days calendar.

This keyword affects the processing of **makecal** with options **-l**, **-p**, and **-w**.

Example

Examples

To make a two-year calendar with the last day of every month selected, run the following command:

```
makecal -e -i 24
```

To make a calendar with 30 days that starts on May 30, 2024, and has every third day selected, run the following command:

```
makecal -r 3 -s "30 MAY 2024" -i 30
```

metronome

Metronome has been replaced by the **wa_pull_info** script.

For more information about this command and the procedures to gather product data for troubleshooting purposes, contact Software Support.

morestdl

Displays the contents of standard list files. This command must be run by the user for which HCL Workload Automation was installed. If you use this command without any parameters, ensure that you are logged on as an HCL Workload Automation user. This command is supported for fault-tolerant agents and standard agents.

Syntax

morestdl -V | **-U**

morestdl
[**-day** *num*]
[**-first** | **-last** | **-num** *n* | **-all**]
[**-twslog**]
[{**-name** ["*[folder/]jobstreamname* [(*hhmm date*),(*jobstream_id*)].]*jobname*"
| *jobnum* | **-schedid** *jobstream_id.jobname*}]

Arguments

-V

Displays the command version and exits.

-U

Displays command usage information and exits.

-day *num*

Displays standard list files that are the specified number of days old (1 for yesterday, 2 for the day before yesterday, and so on). The default is zero (today).

-first

Displays the first qualifying standard list file.

-last

Displays the last qualifying standard list file.

-num *n*

Displays the standard list file for the specified run of a job.

-all

Displays all qualifying standard list files.

-twslog

Displays the content of the current day *stdlist* file.

-name ["*[folder/]jobstreamname* [(*hhmm date*),(*jobstream_id*)].]*jobname*"|*jobnum*

Specifies the instance of the job stream, optionally preceded by the folder within which the job stream is defined, and the name of the job for which the standard list file is displayed.

jobnum

Specifies the job number of the job for which the standard list file is displayed.

-schedid *jobstream_id.jobname*

Specifies the job stream ID and name of the job for which standard list file names are returned.

Comments

The square brackets in the expression `[(hhmm date), (jobstream_id)]` are part of the command, not syntax indicators. This means that you can supply either of the following for the **-name** argument:

```
morestdl -name ["[folder/]jobstreamname[(hhmm date),(jobstream_id)].jobname"
morestdl -name jobnum
```

The whole job identification string must be enclosed in double quotation marks if the part identifying the job stream instance contains blanks. For example, because the *schedtime*, represented by `hhmm date`, has a space in it you must enclose the whole job identification in double quotation marks.

If you just want to identify a job name, you do not need the double quotation marks.

The following is an example of the syntax to use when identifying a job both with and without its job stream. In the example, `job_stream1` is the name of the job stream, `0600 04/05/06` is the scheduled time, `0AAAAAAAAAAAAAB5` is the job stream ID, and `job1` is the job name. You can run the **morestdl** command against `job1` using either of these two formats:

```
morestdl -name "job_stream1[(0600 04/05/06),(0AAAAAAAAAAAAAB5)].job1"
morestdl -name job1
```

Example

Examples

To display the standard list file for the first run of job `MY_CPU#ELI[(1824 03/09/06),(0AAAAAAAAAAAAAEE)].DIR` on the current day, run the following command:

```
morestdl -first -name "MY_CPU#ELI[(1824 03/09/06),(0AAAAAAAAAAAAAEE)].DIR"
```

To display the standard list file for the first run of job `0AAAAAAAAAAAAAEE.DIR` on the current day, run the following command:

```
morestdl -first -schedid 0AAAAAAAAAAAAAEE.DIR
```

To display the standard list file for the second run of job `MY_CPU#ELI[(1824 03/09/06),(0AAAAAAAAAAAAAEE)].DIR` on the current day, run the following command:

```
morestdl -num 2 -name "MY_CPU#ELI[(1824 03/09/06),(0AAAAAAAAAAAAAEE)].DIR"
```

To display the standard list files for all runs of job `MY_CPU#ELI[(1824 03/09/06),(0AAAAAAAAAAAAAEE)].DIR` from three days ago, run the following command:

```
morestdl -day 3 -name "MY_CPU#ELI[(1824 03/09/06),(0AAAAAAAAAAAAAEE)].DIR"
```

To display the standard list file for the last run of job `MY_CPU#ELI[(1824 03/09/06),(0AAAAAAAAAAAAAEE)].DIR` from four days ago, run the following command:

```
morestdl -day 4 -last -name "MY_CPU#ELI[(1824 03/09/06),(0AAAAAAAAAAAAAEE)].DIR"
```

To print the standard list file for job number 455, run the following command:

```
morestdl 455 | lp -p 6
```

parms

Run the parms utility on a master domain manager or a fault-tolerant agent workstation (not supported on dynamic agents) to manage parameters defined locally on workstations. Parameters managed by **parms** can only be used in job or job stream definitions with the **scriptname** or **opens** keywords or in a job script file.

These parameters are resolved at submission time on the workstation where the job or job stream is submitted. If there is no match between the specified *parametername* and the name of the parameters defined in the local database on the workstation, then a *null* value is returned.

Authorization

You must have ***display*** access to the locally defined parameters database. In addition you must be authorized with the following access:

build on object file

If you use the **-b** option to create or rebuild the local parameters database.

delete

If you use the **-d** option to delete parameter definitions.

modify on object file

If you use the **-replace** option to add or modify parameter definitions.

Syntax

parms {[**-V** | **-u**] | **-build**}

parms {-replace | **-extract}** *filename*

parms [**-d**]*[folder/]parametername*

parms -c *parametername value*

Arguments

-V

Displays the command version and exits.

-u

Displays command usage information and exits.

-build

Creates the parameters database on the workstation if it does not exist. Rebuilds the parameters database, removing unused records and avoiding fragmentation from numerous additions and deletions, if it already exists.

-extract

Extracts all parameter definitions from the local database and stores them in the file with name *filename*.

Use this option if you want to export local parameter definitions to import them as global parameter definitions into the scheduling objects database using the **add** or the **replace** commands. For more information about these commands, see *User's Guide and Reference*.

-replace

Add in the local database new parameter definitions stored in a file named *filename* or substitute the already existing ones.

Use this option if you want to import, as local parameter definitions, the global parameter definitions contained in the file named *filename* and extracted from the scheduling objects database using the **extract** command. For more information about this command, see *User's Guide and Reference*.

-d

Deletes the parameters with name *[folder/]parametername* from the local database on the workstation.

[folder/]parametername

Specifies the name of the parameter, optionally preceded by the folder within which it is defined, whose value is displayed. When used with the argument **-d** it represents the name of the parameter to be deleted.

-c *name value*

Specifies the name and the value of a parameter. The name can contain up to 16 alphanumeric characters, including dashes (-) and underscores (_), and must start with a letter. The value can contain up to 72 characters. Enclose the value in double quotation marks if it contains special characters. If the parameter does not exist, it is added to the database. If the parameter already exists, its value is changed.

Comments

When **parms** is run on the command line without arguments, it prompts for parameter names and values.

The use of **parms** in either job definitions and job script files requires that the parameter already exists locally in the parameters database on the workstation.

This is a sample usage of a local parameter, `MYFILE`, in a file dependency clause:

```
schedule test_js
on everyday
opens "/usr/home/tws_99/'/usr/home/tws_99/bin/parms MYFILE'"
:
test_job
end
```

The following example explains how the variable ***var*** enclosed by carets (^) is replaced while the job is in process. If the job is submitted as an ad hoc job, the parameter ***var*** is expanded, that means replaced by the value assigned to ***var*** in the local database, at submission time and not when the job launches.

UNIX® job definition example:

```
DATA#UX_P_TEST DOCOMMAND "ls ^var^"
STREAMLOGON "mae82"
DESCRIPTION "Test parms in job definition on UNIX."
RECOVERY STOP
```

Windows® job definition example:

```
BORG#WIN_P_TEST DOCOMMAND "dir ^var^"
STREAMLOGON "mae82"
DESCRIPTION "Test parms in job definition on Windows."
RECOVERY STOP
```

When used in a job script file, the parameter is not expanded until the script launches. It is not expanded when the job stream containing the job is processed by **JnextPlan**. These are examples on how to use the ***var*** parameter in job script files.

UNIX® script example:

```
#!/bin/sh
TWS_HOME="/opt./tws/mae82/maestro"
export TWS_HOME
MDIR='$TWS_HOME/bin/parms var'
export MDIR
ls -l $MDIR
```

Windows® script example:

```
set TWS_HOME=d:\win32app\TWS\mae82\maestro
echo %TWS_HOME%
FOR /F "Tokens=*" %%a in (%TWS_HOME%\bin\parms var) do set MDIR=%%a
echo %MDIR%
dir %MDIR%
```

Example

Examples

To return the value of `myparm`, run the following command:

```
parms myparm
```

To change the value of `myparm` defined in the folder `myfolder`, run the following command:

```
parms -c myfolder/myparm "item 123"
```

To create a new parameter named `hisparm`, run the following command:

```
parms -c hisparm "item 789"
```

To change the value of `myparm` and add `herparm`, run the following command:

```
parms
 Name of parameter ? myparm < Return>
 Value of parameter? "item 456" < Return>
 Name of parameter ? herparm < Return>
 Value of parameter? "item 123" < Return>
 Name of parameter ? < Return>
```

For more information, see the section about customizing your workload using variable tables in *User's Guide and Reference*.

release

Run the release utility on a master domain manager or a fault-tolerant agent workstation (not supported on dynamic agents) to release jobs and job streams from **needs** dependencies on a resource. This command must be issued only from within the job script file. This command is effective only if the target jobs and job streams are running when the command is issued. If you rerun the job or job stream, the required resources are allocated again. All product components must be at least at the Version 9.4 level.

Syntax

release -V | **-U**

release
[**-s**| -**schedule**]
[*[folder/]workstation***#**]
[folder/]resourcename
[*count*]

Arguments

-V

Displays the command version and exits.

-U

Displays command usage information and exits.

-s | **-schedule**

Releases the **needs** dependency from the specified resource only at the job stream level.

If **-s** is not used, the **needs** dependency from the specified resource is released at the job level, or at the job stream level if the **needs** dependency from that resource is not found at the job level.

[folder/]workstation#

Specifies the name of the workstation or workstation class, optionally preceded by the folder in which the workstation or workstation class is defined, on which the resource is defined. The default is the local workstation.

[folder/]resourcename

Specifies the name of the resource, , optionally preceded by the folder in which it is defined, involved in the **needs** dependency.

count

Specifies the number of units of the resource to be released. If no number is specified, all resources are released.

Comments

Units of a resource are acquired by a job or job stream at the time it is launched and are released automatically when the job or job stream completes. The **release** command can be used in a job script to release resources before job or job stream completion or to release manually jobs and job streams from needs dependencies in emergency situations.

Example

Examples

In the following job stream, two units of the `dbase` resource are required by job stream `sked5`:

```
schedule ux1#sked5 on tu
needs 2 dbase :
job1
jobrel follows job1
job2 follows jobrel
end
```

To release the `dbase` resource before `job2` begins, the script file for `jobrel` contains the following command:

On UNIX™ operating systems:

```
`maestro`/bin/release -s dbase
```

On Windows™ operating systems:

```
<TWS_home>\bin\release -s dbase
```

Note: The **-s** argument can be omitted, because no resources were reserved at the job level.

Example

The following example demonstrates how to partially release resources at the job stream level.

In the following job stream, four units of the `dbase` resource are required by job stream `sked5`:

```
schedule ux1#sked5 on tu
needs 4 dbase :
job1
jobrel follows job1
job2 follows jobrel
end
```

To release the `dbase` resource before `job2` begins, the script file for `jobrel` contains the following command:

On UNIX™ operating systems:

```
`maestro`/bin/release -s dbase 3
```

On Windows™ operating systems:

```
<TWS_home>\bin\release -s dbase 3
```

In this case, while job `job1` is running, the number of resources required by the `ux1#sked5` job stream is 4. When job `jobrel` starts, launching the **release** command, the number of resources in use changes to one, because the **release** command has released three resources.

The following is the output of the **conman sr @#@** command launched while `job1` is running:

```
%sr @#@
CPU#Resource   Total Available Qty UsedBy
UX1#DBASE      10    6         4   UX1#SKED5[(1032 11/03/16),(0AAAAAAAAAAAAABM)]
```

The following is the output of the **conman sr @#@** command launched after the second job (`jobrel`) in the job stream has completed, and before the last job (`job2`) in the job stream completes:

```
%sr @#@
CPU#Resource   Total Available Qty UsedBy
UX1#DBASE      10    9         1   UX1#SKED5[(1032 11/03/16),(0AAAAAAAAAAAAABM)]
```

Example

The following example demonstrates how to completely release resources at the job stream level.

In the following job stream, four units of the `dbase` resource are required by job stream `sked5`:

```
schedule ux1#sked5 on tu
needs 4 ux1#dbase :
job1
jobrel follows job1
job2 follows jobrel
end
```

To release the `dbase` resource before `job2` begins, the script file for `jobrel` contains the following command:

On UNIX™ operating systems:

```
`maestro`/bin/release -s dbase 4
```

On Windows™ operating systems:

```
<TWS_home>\bin\release -s dbase 4
```

In this case, while job `job1` is running, the number of resources required by the `ux1#sked5` job stream is 4. When job `jobrel` starts, launching the **release** command, the number of resources in use changes to zero, because the **release** command has released all four resources. You can obtain the same result by specifying a higher number of resources than are actually required by the job stream or by specifying no number at all: in both cases, the command releases all resources required by the job stream.

The following is the output of the **conman sr @#@** command launched while `job1` is running:

```
%sr @#@
CPU#Resource   Total Available Qty UsedBy
UX1#DBASE      10    6         4    UX1#SKED5[(1040 11/03/16),(0AAAAAAAAAAAAABM)]
```

The following is the output of the **conman sr @#@** command launched after the second job (`jobrel`) in the job stream has completed, and before the last job (`job2`) in the job stream completes:

```
%sr @#@
CPU#Resource       Total Available  Qty UsedBy
UX1#DBASE          10    10         No holders of this resource
```

Example

The following example demonstrates how to partially release resources at the job level.

In the following job stream, four units of the `dbase` resource are required by job `jobrel`:

```
schedule ux1#sked5 on tu
:
job1
jobrel follows job1
needs 4 fta1#dbase
job2 follows jobrel
end
```

To release the `dbase` resource before `job2` begins, the script file for `jobrel` contains the following command:

On UNIX™ operating systems:

```
`maestro`/bin/release dbase 1
```

On Windows™ operating systems:

```
<TWS_home>\bin\release dbase 1
```

In this case, while job `job1` is running, the number of required resources is zero. As soon as job `jobrel` starts and before the **release** command it contains is launched, the number of resources in use changes to four. When the **release** command in job `jobrel` is launched, the number of resources in use changes to three because the **release** command has released one resource.

Example

The following example demonstrates how to partially release resources at the job stream level.

In the following job stream, 34 units of the `dbase` resource are required by job `jobrel`:

```
schedule ux1#sked5 on tu
needs 34 dbase
:
job1
jobrel follows job1
job2 follows jobrel
end
```

To release the `dbase` resource before `job2` begins, the script file for `jobrel` contains the following command:

On UNIX™ operating systems:

```
`maestro`/bin/release dbase
```

On Windows™ operating systems:

```
<TWS_home>\bin\release dbase
```

In this case, while job `job1` is running, the number of resources required by job stream `ux1#sked5` is 34. When job `jobrel` starts, the number of resources in use changes to two. This happens because the products divides the resources into blocks formed by 32 units. The dependency from 34 resources is evaluated by Workload Scheduler as a double dependency: the first dependency having 32 units, and the second one having two units. When the **release** command in job `jobrel` is launched, the number of resources in use changes to two because the **release** command (for which no quantity has been defined) has completed released the first dependency, containing 32 units.

The following is the output of the **conman sr @#@** command launched while `job1` is running:

```
%sr @#@
CPU#Resource  Total Available  Qty UsedBy
UX1#DBASE     34    0          34   UX1#SKED5[(1101 11/03/16),(0AAAAAAAAAAAAABR)]
```

The following is the output of the **conman sr @#@** command launched after the second job (`jobrel`) in the job stream has completed, and before the last job (`job2`) in the job stream completes:

```
%sr @#@
CPU#Resource  Total Available  Qty UsedBy
UX1#DBASE     34    32          2   UX1#SKED5[(1101 11/03/16),(0AAAAAAAAAAAAABR)]
```

Example

The following example demonstrates the internal working of the product and why no resource release occurs in this case

In the following job stream, four units of the `dbase` resource are required by job `job1`:

```
schedule ux1#sked5 on tu
:
job1
needs 4 dbase
jobrel
job2 follows jobrel
end
```

To release the `dbase` resource before `job2` begins, the script file for `jobrel` contains the following command:

On UNIX™ operating systems:

```
`maestro`/bin/release dbase 2
```

On Windows™ operating systems:

```
<TWS_home>\bin\release dbase 2
```

In this case, job `job1` requires four resources. When job `jobrel` starts, the release command it contains does not have any effect because no resource dependency is present for job `jobrel` . This happens because the **release** command releases resources only for the job instance which runs the command. In the case that other jobs or job streams, or other instances of the same job which launches the **release** command, are using units of a specific resource, such units are not released, even when the resource in use matches the resource name in the command.

rmstdlist

Removes or displays standard list files based on the age of the file. This utility should be used by the HCL Workload Automation administrator to maintain the scheduling environment.

Syntax

rmstdlist -V | -U

rmstdlist [**-p**] [*daysold*]

Arguments

-V

Displays the command version and exits.

-U

Displays command usage information and exits.

-p

Displays the names of qualifying standard list file directories. No directories or files are removed. If you do not specify **-p**, the qualifying standard list files are removed.

daysold

The minimum age, in days, of standard list file directories to be displayed or removed. The default is 10 days.

Note:

- Because the list of directories and files shown or deleted using **rmstdlist** is produced based on the last time they were accessed, the dates shown in the list of directories could differ from the dates displayed in the list of files.
- The rmstdlist command might give different results on different platforms for the same scenario. This is because on UNIX® platforms the command uses the *–mtime* option of the find command, which is interpreted differently on different UNIX® platforms.

Syntax

As a rule, you should regularly remove standard list files somewhere between every 10-20 days. Larger backlogs may be harder to manage and, if the number of files becomes exceedingly large, you might be required to erase some of them manually before you can use **rmstdlist** again.

This problem might occur on AIX systems, particularly, because of a currently unresolved limitation with the `rm -rf` command. When **rmstdlist** fails because of this limitation, it does not display any errors other than exit code 126. If you would rather have the `rm -rf` error displayed, you can edit the `rmstdlist` script in the following way:

1. Locate the script in the *`TWS_home`*`/bin` directory
2. Find the line:

```
rm -rf `cat /tmp/rm$$` 2> /dev/null
```

3. Remove the redirection to `/dev/null` so that the line becomes:

```
rm -rf `cat /tmp/rm$$`
```

Example

Examples

To display the names of standard list file directories that are more than 14 days old, run the following command:

```
rmstdlist -p 14
```

To remove all standard list files (and their directories) that are more than seven days old, run the following command:

```
rmstdlist 7
```

sendevent

The command sends the custom events defined with the evtdef on page 934 command to the event processor server currently active in the production plan. As the events are received by the event processor, they trigger the event rules in which they were specified.

Syntax

sendevent -V | ? | -help | -u | -usage

sendevent [**-hostname** *hostname*]
[{**-port** | **-sslport}** *port*]
eventType
source
[[*attribute*=*value*]...]

Arguments

-V

Displays the command version and exits.

? | -help | -u | -usage

Displays command usage information and exits.

-hostname *hostname*

Specifies the host name of an alternate event processor server other than the currently active one.

This parameter is required if the command is launched from a command-line client. For more information, see the section about the command-line client in *User's Guide and Reference*.

-port | -sslport} *port*

Specifies the port number of an alternate event processor server other than the currently active one. **-sslport** defines the port used to listen for incoming SSL connections.

This parameter is required if the command is launched from a command-line client.

eventType

One of the custom event types defined with the evtdef on page 934 command in the generic event provider and specified as the triggering event in an event rule definition.

source

The name of the event provider that you customized with evtdef on page 934. This is also the name you must specify as the argument for the `eventProvider` keyword in the definition of the event rules triggered by these custom events.

The default name is `GenericEventPlugIn`.

attribute=value

One or more of the attributes qualifying the custom event type that are specified as the triggering event attributes for the event rule.

Comments

This command can be run also on systems where only the HCL Workload Automation remote command line client is installed.

Example

Examples

In this example an application sends the `BusProcCompleted` custom event type to an alternate event processor server named `master3`. The event is that file `calcweek` finished processing.

```
sendevent -hostname master3 -port 4294 BusProcCompleted
GenericEventPlugIn TransacName=calcweek Workstation=ab5supp
```

The file name and the associated workstation are the two `BusProcCompleted` event attributes that were specified as triggering event attributes in an associated event rule.

showexec

Displays the status of running jobs. This command applies to UNIX® only. This command is for standard agents. On domain managers and fault-tolerant agents, use the **conman showjobs** command instead.

Syntax

showexec [**-V** | **-U** | **INFO**]

Arguments

-V

Displays the command version and exits.

-U

Displays command usage information and exits.

INFO

Displays the name of the job file instead of the user, date, and time.

Results

The output of the command is available in two formats: **standard** and **INFO**.

Example

Examples

To display running jobs in the **standard** format, run the following command:

```
showexec
```

To display running jobs in the **INFO** format, run the following command:

```
showexec INFO
```

Standard format

CPU

The workstation on which the job runs.

Schedule

The name of the job stream in which the job runs.

Job

The job name.

Job#

The job number.

User

The user name of the job.

Start Date

The date the job started running.

Start Time

The time the job started running.

(Est) Elapse

The estimated time, in minutes, that the job will run.

Info format

CPU

The workstation on which the job runs.

Schedule

The name of the job stream in which the job runs.

Job

The job name.

Job#

The job number.

JCL

The file name of the job.

shutdown

Stops the HCL Workload Automation processes, and optionally also stops the WebSphere Application Server Liberty. Applies to Windows® workstations only. You must have ***shutdown*** access to the workstation.

Syntax

shutdown [-V | -U] [**-appsrv**]

Arguments

-V

Displays the command version and exits.

-U

Displays command usage information and exits.

-appsrv

Stops also WebSphere Application Server Liberty.

Comments

Make sure the *TWS_user* you are using belongs to the Admnistrators group defined on the Windows® workstation.

Example

Examples

To display the command name and version, run the following command:

```
shutdown -V
```

To stop both the HCL Workload Automation processes and WebSphere Application Server Liberty, run the following command:

```
shutdown -appsrv
```

ShutDownLwa - Stop the agent

Stops the agent. On Windows systems, no specific access to the workstation is required. On UNIX systems, it can be run by *TWS_user* or root user only. Run this command locally on the agent you want to stop.

Syntax

ShutDownLwa

Arguments

No arguments are necessary.

Example

Examples

To stop the agent, run the following command:

```
ShutDownLwa
```

StartUp

Starts **netman**, the HCL Workload Automation network management process.

You must have ***start*** access to the workstation.

Syntax

StartUp [**-V** | **-U**]

Arguments

-V

Displays the command version and exits.

-U

Displays command usage information and exits.

Comments

In Windows®, the **netman** service is started automatically when a computer is restarted. **StartUp** can be used to restart the service if it is stopped for any reason.

In UNIX®, the **StartUp** command can be run automatically by invoking it from the `/etc/inittab` file, so that WebSphere Application Server Liberty infrastructure and **netman** is started each time a computer is rebooted. **StartUp** can be used to restart **netman** if it is stopped for any reason.

The remainder of the process tree can be restarted with the

```
conman start
conman startmon
```

commands.

See the section about **conman start** in *User's Guide and Reference* for more information.

Example

Examples

To display the command name and version, run the following command:

```
StartUp -V
```

To start the **netman** process, run the following command:

```
StartUp
```

StartUpLwa - Start the agent

Starts the agent . On Windows™ systems, no specific access to the workstation is required. On UNIX™ systems, it can be run by *TWS_user* or root user only. Run this command locally on the agent you want to start.

Syntax

StartUpLwa

Arguments

No arguments are necessary.

Example

Examples

To start the agent, run the following command:

```
StartUpLwa
```

version

Displays information about the current release of HCL Workload Automation installed on the system. This command applies to UNIX® only. The information is extracted from a version file.

Syntax

version -V | **-u** | **-h**

version [**-a**] [**-f** *vfile*] [*file* [...]]

Arguments

-V

Displays the version of the command and exits.

-u

Displays command usage information and exits.

-h

Displays command help information and exits.

-a

Displays information about all product files. The default is to display information only about the specified files.

-f *vfile*

Specifies the path and name of the version file if different from the default setting. The default is a file named `version.info` in the current working directory.

file

Specifies the names of product files, separated by spaces, for which version information is displayed. The default is to display no file information, or, if **-a** is used, all file information.

Results

The output header contains the product name, version, operating system, patch level, and installation date. The remaining display lists information about the file or files specified. The files are listed in the following format:

File

The name of the file.

Revision

The revision number of the file.

Patch

The patch level of the file, if any.

Size (bytes)

The size of the file in bytes.

Checksum

The checksum for the file. Checksum is calculated using the UNIX® **sum** command. On AIX®, **sum** is used with the **-o** argument.

Comments

HCL Workload Automation file information is contained in the `version.info` file. This file is placed in the *`TWS_home`*`/version` directory during installation. The `version.info` file is in a specific format and is not altered.

You can move the `version.info` file to another directory. However, you must then include the **-f** argument to locate the file.

Example

Examples

To display information about the release of HCL Workload Automation installed, run the following command:

```
./version
```

A sample output of this command is:

```
IBM Workload Scheduler/VERSION 9.21 (C) Copyright IBM Corp 1998, 2013

IBM Workload Scheduler 9.2 UNIX
```

To display information about all files, run the following command:

```
version/version -a -f version/version.info
```

To display information about the file `customize`, run the following command:

```
cd version
./version customize
```

To display information about the file `customize`, when `version.info` is in `/apps/maestro`, run the following command:

```
cd version
./version -f /apps/maestro/version.info customize
```

wa_pull_info

This is a script that produces information about your HCL Workload Automation environment and your local workstation, and can take a snapshot of DB2 and WebSphere Application Server Liberty data on the master domain manager, saving them as a dated package.

It can also generate a report containing not only the results of the snapshot, but also many configuration and environment parameters. The tool is useful when describing a problem to HCL Software Support. For best results, it must be run as soon as the problem is discovered.

The wa_pull_info script replaces the back-level tws_inst_pull_info script, which is no longer supported.

Syntax

```
wa_pull_info -help
```

```
wa_pull_info
-component DWC|TWS
[-date yyyy-mm-dd]
[-isroot=true|false]
[-output output_path]
-user userid
[-workdir working_directory]
```

Parameters

-?

Displays the usage of the command.

-component

The component whose data you want to capture. Supported values are DWC and TWS. The default value is TWS. This parameter is required.

-date

Used as the base date for collected data logs. If not specified, the script uses the current date by default and the day before the current date. Run the data capture utility as soon as a problem occurs, to collect the data specific to the date and time of the problem. Thus, if the problem occurs on the current date, this option is not required. If the problem occurred earlier, then the date on which the problem occurred must be specified in the yyyy-mm-dd format. Either the current date or the specified date is used to identify which files and logs are extracted. This parameter is optional.

-isroot

If you have performed a no-root installation, set this parameter to false: the tool uses the user specified with the -user on page 976 parameter. The default value is true, which indicates you have installed the component using the root user. This parameter is required if you perform a no-root installation.

-output

The base directory location where the collected data is stored. Ensure you have write access to the specified directory. This parameter is optional. The default value is as follows:

On Windows operating systems

```
C:\tmp\wadatagather\output
```

On UNIX operating systems

```
/tmp/wadatagather/output
```

Ensure you do not enter a path that contains the `stdlist` folder located in *TWA_DATA_DIR*.

`-user`

The product user that you specified when you installed the component. If you performed a no-root installation, set the -isroot on page 975 parameter to `false`. This user must exist in the registry file located on the master domain manager in the following path:

On Windows operating systems

```
C:\Windows
```

On UNIX operating systems, if you installed as root

```
/etc/TWA
```

On UNIX operating systems, if you installed as no-root

```
user home dir/.TWS/etc/TWA
```

or on the Dynamic Workload Console in the following path:

On Windows operating systems

```
C:\Windows\TWA
```

On UNIX operating systems, if you installed as root

```
/etc/TWA
```

On UNIX operating systems, if you installed as no-root

```
user home dir/.TWS/etc/TWA
```

or on the agent in the following path:

On Windows operating systems

```
C:\Windows
```

On UNIX operating systems, if you installed as root

```
/etc/TWS/TWSRegistry.dat
```

On UNIX operating systems, if you installed as no-root

```
user home dir/.TWS/etc/TWS/TWSRegistry.dat
```

This parameter is mandatory.

`-workdir`

The working directory used by the command for storing data while running. When the command stops running, the working directory is deleted. Ensure you have write access to the specified directory and enough space is available. This parameter is optional. The default value is as follows:

On Windows operating systems

```
C:\tmp\wadatagather\workdir
```

On UNIX operating systems

```
/tmp/wadatagather/workdir
```

Ensure you do not enter a path that contains the `stdlist` folder located in *TWA_DATA_DIR*.

Comments

For more information, contact software support.

Unsupported commands

The following unsupported utility commands provide functions in Windows® that are similar to UNIX® **ps** and **kill** commands. They can be used if similar Windows® utilities are not available.

Syntax

listproc

killproc *pid*

Comments

listproc

Displays a tabular listing of processes on the system.

killproc

Kills the process with the process ID *pid*.

Note: When run by the Administrator, **killproc** is capable of killing system processes.

Chapter 19. Using utility commands in the dynamic environment

This chapter describes HCL Workload Automation utility commands for the dynamic environment. While some commands run on the master domain manager or on a dynamic domain manager, others run on the agents. They are installed in path `*TWA_home*/TDWB/bin`, with the exceptions listed below, and run with the UNIX® and Windows® operating systems. You run utility commands from the operating system command prompt except the jobprop utility that you can use only in a job definition as described in Passing variables set by using jobprop in one job to another in the same job stream instance on page 280.

Table 107: List of utility commands for dynamic workstations on page 978 contains the list of the utility commands, and for each command, its description and the type of workstation where you can run it.

Table 107. List of utility commands for dynamic workstations

Command	Description	Workstation type
cleanuserjobs	Cleans up dynamic domain manager database tables in case of jobs in never-ending EXEC status in the USERJBOBS job stream	master domain manager or dynamic domain manager
exportserverdata	Downloads the list of dynamic workload broker instances from the HCL Workload Automation database and changes a port number or a host name.	master domain manager or dynamic domain manager
importserverdata	Uploads the list of dynamic workload broker instances to the HCL Workload Automation database after editing the temporary file to change a port number or a host name.	master domain manager or dynamic domain manager
jobprop	Sets variables values in a job that you can pass to the successive job in the same job stream instance. For more information about how to use this utility in a job definition, see Passing variables set by using jobprop in one job to another in the same job stream instance on page 280. It is installed in `<TWS_INSTALLATION_DIR>/TWS/bin` directory and runs on UNIX and Windows operating systems.	agents
movehistorydata	Moves the data present in the HCL Workload Automation database to the archive tables	master domain manager or dynamic domain manager
param	Creates, displays, and deletes variables and user passwords on dynamic agents.	agents

Table 107. List of utility commands for dynamic workstations (continued)

Command	Description	Workstation type
		This command is installed in *TWA_home*/TWS/CLI/bin.
resource	Creates, modifies, associates, queries, or sets resources online or offline.	master domain manager, dynamic domain manager, or agents
sendevent	Sends generic events to the currently active event processor server.	dynamic domain managers, and agents. This command is installed in *TWA_home*/TWS/CLI/bin.
twstrace	Modifies at runtime the settings for tracing on agents	agents This command is installed in *TWA_home*/TWS/CLI/bin.

Note: To remove the job logs files for dynamic agents workstations, set the value of the `MaxAge` property in the `JobManager.ini`. For more details, see *HCL Workload Automation manuals: Administration guide - Configuring the agent - Configuring common launchers properties [Launchers]*.

Command-line configuration file

The `CLIConfig.properties` file contains configuration information which is used when typing commands. By default, arguments required when typing commands are retrieved from this file, unless explicitly specified in the command syntax.

The `CLIConfig.properties` file is created at installation time and is located on the master domain manager in the following path:

on Windows

```
TWA_home\broker\config
```

on UNIX

```
TWA_DATA_DIR/broker/config
```

The `CLIConfig.properties` file contains the following set of parameters:

Dynamic workload broker default properties

ITDWBServerHost

Specifies the IP address of dynamic workload broker.

ITDWBServerPort

Specifies the number of the dynamic workload broker port. The default value is **9550**.

ITDWBServerSecurePort

Specifies the number of the dynamic workload broker port when security is enabled. The default value is **9551**.

use_secure_connection

Specifies whether secure connection must be used. The default value is **false**.

KeyStore and trustStore file name and path

keyStore

Specifies the name and path of the keyStore file containing private keys. A keyStore file contains both public keys and private keys. Public keys are stored as signer certificates while private keys are stored in the personal certificates. The default value is `TWSServerKeyFile.p12`.

trustStore

Specifies the name and path of the trustStore file containing public key certificates. A trustStore file is a key database file that should contain the master domain manager and the backup master domain manager public key certificates to enable the CLI to communicate with them. The public key is stored as a signer certificate. The keys are used for a variety of purposes, including authentication and data integrity. The default value is `TWSServerTrustFile.p12`.

Passwords for keyStore and trustStore files

keyStorepwd

Specifies the password for the keyStore file.

trustStorepwd

Specifies the password for the trustStore file.

File types for keyStore and trustStore files

keyStoreType

Specifies the file type for the keyStore file. The default value is PKCS12.

trustStoreType

Specifies the file type for the trustStore file. The default value is PKCS12.

Default user ID and password for dynamic workload broker

tdwb_user

Specifies the user name for a user authorized to perform operations on dynamic workload broker when security is enabled. The default value is **ibmschedcli**. This user must be previously defined on WebSphere Application Server Liberty Base.

tdwb_pwd

Specifies the password for a user authorized to perform operations on dynamic workload broker when security is enabled. This password must be previously defined on WebSphere Application Server Liberty Base.

Detail level for command-line log and trace information

logger.Level

Specifies the detail level for the command-line trace and log files. The command-line trace and log files are created in the following location:

log file

TWA_DATA_DIR/broker/logs/Msg_cli.log.log

trace file

TWA_DATA_DIR/broker/logs/Trace_cli.log

The default value is INFO.

logger.consoleLevel

Specifies the detail level for the log and trace information to be returned to standard output. The default value is FINE. Supported values for both the **consoleLevel** and **loggerLevel** parameters are as follows:

ALL

Indicates that all messages are logged.

SEVERE

Indicates that serious error messages are logged.

WARNING

Indicates that warning messages are logged.

INFO

Indicates that informational messages are logged.

CONFIG

Indicates that static configuration messages are logged.

FINE

Indicates that tracing information is logged.

FINER

Indicates that detailed tracing information is logged.

FINEST

Indicates that highly detailed tracing information is logged.

OFF

Indicates that logging is turned off.

logger.limit

Specifies the maximum size of a log file in bytes. The default value is 400000. When the maximum size is reached, a new file is created, until the maximum number of files is reached. When all files reach the maximum size and the maximum number of files is exceeded, the oldest file is re-written.

logger.count

Specifies the maximum number of log files. The default value is 6. When the maximum size is reached, a new file is created, until the maximum number of files is reached. When all files reach the maximum size and the maximum number of files is exceeded, the oldest file is re-written. When a new file is created the 0 suffix is appended after the file extension. The file with the 0 suffix is always the current file. Any older files are renumbered accordingly.

DAO common configuration

This section defines the RDBMS settings for the **exportserverdata**, **importserverdata**, and **movehistorydata** commands. These commands use the RDBMS installed on dynamic workload broker. These parameters are given values at installation time and should not be modified, except for `com.ibm.tdwb.dao.rdbms.useSSLConnections` as noted below.

com.ibm.tdwb.dao.rdbms.rdbmsName

Specifies the RDBMS name.

com.ibm.tdwb.dao.rdbms.useDataSource

Specifies the data source to be used.

com.ibm.tdwb.dao.rdbms.jdbcPath

Specifies the path to the JDBC driver.

com.ibm.tdwb.dao.rdbms.jdbcDriver

Specifies the JDBC driver.

com.ibm.tdwb.dao.rdbms.userName

Specifies the name of the RDBMS user.

com.ibm.tdwb.dao.rdbms.password

Specifies the password of the RDBMS user.

com.ibm.tdwb.dao.rdbms.useSSLConnections

Specifies that access to the HCL Workload Automation DB2 database by some of the CLI commands is over SSL. Is set to `FALSE` by default. You must set to `TRUE`, if the database is DB2 and you use FIPS security, for the following commands to work:

- exportserverdata
- importserverdata
- movehistorydata

cleanuserjobs

Use the **cleanuserjobs** command to clean up dynamic domain manager database tables in case of jobs in never-ending EXEC status in the USERJBOBS job stream.

Syntax

```
cleanuserjobs –dbUsr username –dbPwd password
–jobnum jobnumber | –file filename
```

Description

This command clears database tables on the dynamic domain manager by removing jobs stuck in a perpetual `EXEC` status within the `USERJOBS` job stream. You can provide the numbers of jobs to be deleted by using the **jobnumber** parameter, or provide a file containing a list of job identifiers by using the **file** parameter.

Options

-dbUsr *user_name*

The user name required to access the HCL Workload Automation database server.

-dbPwd *password*

The user password required to access the HCL Workload Automation database server.

-jobnumber

A specific job to delete. Supports the following formats:

- *#Jjobnumber*
- *Jjobnumber*
- *jobnumber*

-filename *filename*

A file containing one job identifier per line. Supports the following formats:

- *#Jjobnumber*
- *Jjobnumber*
- *jobnumber*

Example

Run the following command to remove job `45678` from the database:

```
./cleanuserjobs.sh -dbUsr twsadmin -dbPwd secret123 -jobnum J45678
```

Run the following command to remove the jobs listed in file `joblist.txt` from the database:

```
./cleanuserjobs.sh -dbUsr twsadmin -dbPwd secret123 -filename joblist.txt
```

The file contains the following entries:

```
#J938667870
J938667871
938667872
```

exportserverdata

Use the **exportserverdata** command to download the list of dynamic workload broker instances from the HCL Workload Automation database and change a port number or a host name.

Syntax

exportserverdata ?

exportserverdata -dbUsr *db_user_name* **-dbPwd** *db_user_password* **-exportFile** *filename*

Description

This command extracts a list of URIs (Uniform Resource Identifier) of all the dynamic workload broker instances from the HCL Workload Automation database and copies them to a temporary file so that, if either the hostname or the port number of any of the instances listed are changed, the administrator can record this information in the file and place it back in the database with the `importserverdata` command. By default, the list of URIs is saved to the server.properties file, located in the current directory.

This action is necessary because the list of dynamic workload broker instances must be kept up-to-date at all times, since the Resource Advisor agents periodically connect to the active instance to send their data about the resources discovered in each computer. They are able to automatically switch between the instances of this list and find the active one to copy these data in its Resource Repository. Since the master domain manager and every backup master are installed with a dynamic workload broker instance, the active dynamic workload broker instance runs in the master domain manager, while an idle instance resides in each backup master.

The URI pointing to each dynamic workload broker instance is the following:

```
https://hostname:port_number/JobManagerRESTWeb/JobScheduler
```

You can change only the hostname and the port number.



Important: The list is ordered. You can change the order of the instances as they appear in this list, and the agents will follow this order. If you have several backup masters and decide to follow a specific switching order when a

master fails, you can instruct the agents to switch to the right instance using this ordered list, speeding up the transition time.

If your HCL Workload Automation database is DB2 and you use FIPS security, to run this command successfully you need to have the **com.ibm.tdwb.dao.rdbms.useSSLConnections** option set to `TRUE` in the `CLIConfig.properties` file.

Options

?

Displays help information.

-dbUsr *db_user_name*

The user name required to access the HCL Workload Automation database server.

-dbPwd *db_user_password*

The user password required to access the HCL Workload Automation database server.

-exportFile *filename*

The name of the temporary file where the URIs extracted from the database are copied for editing. This text file is created when you run the command and you can open it with any editor to change the hostname or the port number. If you do not specify a path, the file is created in the same directory where the command is located, that is *TWA_home*`/TDWB/bin`. If you do specify a different path, make sure the path exists before you run this command.

Example

To download the current list of all (active and backup) dynamic workload broker instances and copy them in a file named `c:\myservers\uris160709`, run:

```
exportserverdata -dbUsr twsadm -dbPwd fprefect -exportFile c:\myservers\uris160709
```

The command returns file `uris160709`, that looks like this:

```
https://accrec015:42127/JobManagerRESTWeb/JobScheduler
https://prodop099:52529/JobManagerRESTWeb/JobScheduler
https://prodop111:31116/JobManagerRESTWeb/JobScheduler
```

`prodop099` is the active dynamic workload broker instance because is hosted by the currently active master domain manager, whereas `accrec015` and `prodop111` are idle because they are hosted by backup masters.

You can edit this file to apply your changes before using the `importserverdata` command to upload the URIs back to the database.

See Also

importserverdata on page 986

importserverdata

Use the **importserverdata** command to upload the list of dynamic workload broker instances to the HCL Workload Automation database after editing the temporary file to change a port number or a host name.

Syntax

importserverdata ?

importserverdata -dbUsr *db_user_name* **-dbPwd** *db_user_password* **-importFile** *filename*

Description

This command puts back the list of dynamic workload broker instances in the HCL Workload Automation database from the temporary file where they were previously downloaded with the `exportserverdata` command.

Use the `exportserverdata` and `importserverdata` commands if you have to record any hostname or port number changes in the URIs of the instances. This is necessary to keep the list of dynamic workload broker instances up-to-date at all times, since the Resource Advisor agents periodically connect to the active instance to send their data about the resources discovered in each computer. They are able to automatically switch between the instances of this list and find the active one to copy these data in its Resource Repository. Since the master domain manager and every backup master are installed with a dynamic workload broker instance, the active dynamic workload broker instance runs in the master domain manager, while an idle instance resides in each backup master.

Important: The list is ordered. You can change the order of the instances as they appear in this list, and the agents will follow this order. If you have several backup masters and decide to follow a specific switching order when a master fails, you can instruct the agents to switch to the right instance using this ordered list, speeding up the transition time.

If your HCL Workload Automation database is DB2 and you use FIPS security, to run this command successfully you need to have the **com.ibm.tdwb.dao.rdbms.useSSLConnections** option set to `TRUE` in the `CLIConfig.properties` file.

Options

?

Displays help information.

-dbUsr *db_user_name*

The user name required to access the HCL Workload Automation database server.

-dbPwd *db_user_password*

The user password required to access the HCL Workload Automation database server.

-importFile f*filename*

The name of the temporary file you specified with the `-exportFile` keyword in the `exportserverdata` command.

Example

To upload the edited list of dynamic workload broker instance URIs from file `c:\myservers\uris160709` to the HCL Workload Automation database, run:

```
importserverdata -dbUsr twsadm -dbPwd fprefect -importFile c:\myservers\uris160709
```

See Also

exportserverdata on page 984

jobprop

Use the **jobprop** command on a job definition to set variables locally for a job on dynamic agents.

You can use this command on a native or executable job to set variable value that you can pass in a successive job in the same job stream. The values are set at run time. To use this utility, you must be logged in as *TWS_user*.

Syntax

jobprop *variable value*

Arguments

variable

The variable name.

value

The value for *variable*.

Comments

Variable names are case-sensitive. If variable names or values contain spaces, they must be included in single quotes.

Example

Examples

On UNIX operating systems the jobprop utility set the following variables in the `NC125133#JOBA` executable job:

- *VAR1* variable set to `value1` value.
- *VAR2* variable set to `value2` value.
- *VAR3* variable set to `value3` value.
- *VAR4* variable set to `value4` value.

```
NC125133#JOBA
 TASK
    ?xml version="1.0" encoding="UTF-8"?>
<jsdl:jobDefinition xmlns:jsdl="http://www.ibm.com/xmlns/prod/
scheduling/1.0/jsdl" xmlns:
jsdle="http://www.ibm.com/xmlns/prod/scheduling/1.0/jsdle">
  <jsdl:application name="executable">
    <jsdle:executable interactive="false">
```

```
    <jsdle:script>#!/bin/sh
. /home/ITAuser/TWA/TWS/tws_env.sh
jobprop VAR1 value1
jobprop VAR2 value2
jobprop VAR3 value3
jobprop VAR4 value4
</jsdle:script>
    </jsdle:executable>
  </jsdl:application>
</jsdl:jobDefinition>
 DESCRIPTION "Sample Job Definition"
 RCCONDSUCC "RC>=0"
 RECOVERY STOP
```

On Windows operating systems, the jobprop utility set the following variables in the `WIN1#JOB1` executable job:

- *var1* variable set to `value1` value.
- *var2* variable set to `value2` value.
- *var3* variable set to `value3` value.
- *var4* variable set to `value4` value.

```
WIN1#JOB1
 TASK
    ?xml version="1.0" encoding="UTF-8"?>
<jsdl:jobDefinition xmlns:jsdl="http://www.ibm.com/xmlns/prod/
scheduling/1.0/jsdl" xmlns:
<jsdle="http://www.ibm.com/xmlns/prod/scheduling/1.0/jsdle">
  <jsdl:application name="executable">
    <jsdle:executable interactive="false">
    <jsdle:script>
call C:\Progra~1\HCL\TWA\TWS\tws_env.cmd
jobprop var1 value1
jobprop var2 value2
jobprop var3 value3
jobprop var4 value4
</jsdle:script>
    </jsdle:executable>
  </jsdl:application>
</jsdl:jobDefinition>
 DESCRIPTION "Sample Job Definition"
 RCCONDSUCC "RC>=0"
 RECOVERY STOP
```

Note: Before running the jobprop utility in the job definition, ensure you run the tws_env.cmd command with the correct syntax `call <TWS_INST_DIR>\TWS\tws_env.cmd` where <*TWS_INST_DIR*> is the HCL Workload Automation installation directory.

movehistorydata

Use the **movehistorydata** when access to the database becomes too slow. This command moves the data present in the Job Repository to the archive tables

Slow database access might be due to a huge number of records being present in the database, for example when bulk job submissions are performed.

When you run this command, the jobs are moved to the following tables in the database:

JOA_JOB_ARCHIVES

Contains archived job instances

JRA_JOB_RESOURCE_ARCHIVES

Contains resource information related to the jobs

MEA_METRIC_ARCHIVES

Contains metrics collected for the jobs

For more information on historical tables, refer to the *HCL Workload Automation: Administration Guide*, *SC23-9113*.

Note: Depending on the number of jobs and accesses to the database, a cleanup operation might cause some peaks in memory or CPU usage.

If your HCL Workload Automation database is DB2 and you use FIPS security, to run this command successfully you need to have the **com.ibm.tdwb.dao.rdbms.useSSLConnections** option set to `TRUE` in the `CLIConfig.properties` file.

Syntax

movehistorydata ?

movehistorydata -dbUsr *db_user_name*-**dbPwd** *db_user_password* [**-successfulJobsMaxAge** *successfulJobsMaxAge* [**-notSuccessfulJobsMaxAge** *notSuccessfulJobsMaxAge*]

Description

This command performs a cleanup operation on the Job Repository database. Based on the values you specify, information on submitted jobs is moved to the archive database and the information in the archive database is deleted.

Use this command to temporarily override the settings defined in the **JobDispatcherConfig.properties** file, when unexpected events require an immediate database cleanup. The settings in the **JobDispatcherConfig.properties** file remain unchanged. For more information on the **JobDispatcherConfig.properties** file, refer to the *HCL Workload Automation: Administration Guide*.

Options

?

Displays help information.

-dbUsr *db_user_name*

Specifies the username for a user authorized to perform operations on the database server.

-dbPwd ***db_user_password***

Specifies the password for a user authorized to perform operations on the database server.

-successfulJobsMaxAge ***successfulJobsMaxAge***

Specifies how many hours jobs completed successfully or canceled must be kept in the Job Repository database before being archived. The default value is 240 hours, that is ten days.

-notSuccessfulJobsMaxAge ***notSuccessfulJobsMaxAge***

Specifies how many hours jobs completed unsuccessfully or in unknown status must be kept in the Job Repository database before being archived. The default value is 720 hours, that is 30 days.

Return Values

The movehistorydata command returns one of the following values:

0

Indicates that movehistorydata completed successfully.

< > 0

Indicates that movehistorydata failed.

Example

Examples

1. To move to the archive database all successful jobs completed in the last 40 hours, type the following command:

   ```
   movehistorydata -dbUsr halmst -dbPwd dgordon -successfulJobsMaxAge 40
   ```

2. To move to the archive database all jobs in all supported statuses and remove from the archive database all jobs older than 700 hours, type the following command:

   ```
   movehistorydata -dbUsr halmst -dbPwd dgordon -successfulJobsMaxAge 0
                   -notSuccessfulJobsMaxAge 0
   ```

param

Use the **param** command to define and manage user passwords and variables locally on dynamic agents and HCL Workload Automation Agents.

You can use this command on job types with advanced options. The values are resolved at submission time on the agent where the job is submitted.

Authorization

To create, delete, or display variables or passwords, you must have Administrator or root user rights on the workstation that runs the agent or *TWS_user* rights on the agent.

Syntax

param -u | -V |

{-c | -ec} [*file.section.*|*file..*|*section.*] *variable* [*value*] |

[*file.section.*|*file..*|*section.*] *variable* |

{-d | -fd} [*file.section.*|*file..*|*section.*] *variable*

Arguments

-u

Displays command usage information and exits.

-V

Displays the command version and exits.

-c | -ec

Creates variable or password *variable* and defines its value *value*. The variable or password is placed in a namespace *file* that you can organize in one or more sections named *section*.

If you do not provide a file name *file*, the variable or password is placed in default file `jm_variables` in path *agent_installation_path*`\TWA\TWS\ITA\cpa\config\jm_variables_files` (`/TWA/TWS/ITA/cpa/config/jm_variables_files`) on the dynamic agent.

If you do not provide a section name *section*, the variable or password is placed in the main body of the file.



Important: If you are defining a password, you must specify a section named `password` for *variable*. This specifies that *variable* is a password.

If you are creating a variable, *variable* is the variable name and *value* is its value. If you are creating a password, *variable* is the user name and *value* is its password. If you do not enter *value* within the arguments, the command requests interactively to enter a value.

Argument **-c** creates the variable in clear form. Argument **-ec** creates the variable in encrypted form. Passwords are encrypted by default also if you use **-c**.

-d | -fd

Deletes (**-d**) or forces deletion (**-fd**) of a file, section, or variable (password). You can use the following wildcards:

*****

Replaces one or more alphanumeric characters.

?

Replaces one alphanumeric character.

With **-d** the command asks for confirmation before deleting. With **-fd** it deletes without asking confirmation.

When you delete all the variables in a section, the section is removed from the file. When you delete all the sections and all the variables from a file, the file is removed.

file

The name of the file used as a namespace for *variable*. If you do not specify *file*, the command uses the default file `jm_variables` in path *agent_installation_path*`\TWA\TWS\ITA\cpa\config\jm_variables_files` (`/TWA/TWS/ITA/cpa/config/jm_variables_files`).

All the variable namespaces go in path *agent_installation_path*`\TWA\TWS\ITA\cpa\config\jm_variables_files` (`/TWA/TWS/ITA/cpa/config/jm_variables_files`).

section

The name of the section within *file* where *variable* is defined. When *variable* is used for a password, it must be placed in a section named `password`. No section name is required to store variables.

value

The value for *variable*.

variable

Can be a variable name or a user identification. If it is used for identification, it must be placed in a section named `password` within the namespace file.

Comments

To display a variable or password, a namespace file, or a section, use the command as follows:

```
param [file.section.|file..|section.] variable
```

where you can use the * and ? wildcards described for the deletion command.

The namespace files, including default `jm_variables`, have no extension.

Variable names are case sensitive.

On IBM i systems, if you use the QP2TERM and the QSH shells, passwords are made visible during the creation process with `param` and are displayed clearly in the shell logs. To guarantee the obfuscation of a password, you need to use the AIXTERM or XTERM shells.

Example

Examples

The command:

```
param -c compassets.hardware.platform1 unix
```

defines variable `platform1` with value `unix` in section `hardware` of the new or existing file named `compassets`. The value is not encrypted.

The command:

```
param -c compassets..platform1 unix
```

defines variable `platform1` with value `unix` in the new or existing file named `compassets`. The value is not encrypted.

The command:

```
param -ec hardware.platform1 unix
```

defines variable `platform1` with value `unix` in section `hardware` in the default file *agent_installation_path*`\TWA\TWS\ITA\cpa\config\jm_variables_files\jm_variables`. The value is encrypted.

The command:

```
param -c compassets.password.jladams san07rew
```

defines variable `jladams` with value `san07rew` in section `password` of the new or existing file named `compassets`. Since `jladams` is defined in section `password`, it is interpreted as a username. The value `san07rew` is encrypted by default since it is interpreted as a password.

The command:

```
param *.*.platform1
```

lists variable `platform1` in all its defined locations. That is:

```
...\TWA\TWS\ITA\cpa\config\jm_variables_files\compassets.hardware.platform1=unix
...\TWA\TWS\ITA\cpa\config\jm_variables_files\compassets..platform1=unix
...\TWA\TWS\ITA\cpa\config\jm_variables_files\jm_variables.hardware.platform1=***
```

The command:

```
param password.*adam*
```

lists all variables including the string `adam` contained in the `password` section of all files. In this case:

```
...\TWA\TWS\ITA\cpa\config\jm_variables_files\compassets.password.jladams=********
```

The command:

```
param -d compassets.password.jladams
```

deletes variable `jladams`.

The command:

```
param -d compassets.password.*
```

deletes all the variables found in section `password` and therefore removes this section from file `compassets`.

The command:

```
param -d compassets.*.*
```

deletes all the contents (variables and sections containing variables) found in file `compassets` and therefore removes the file.

resource

Use the **resource** command to create, modify, associate, query, or set resources online or offline.

By correctly configuring the CLIConfig.properties file on the agent, you can run this command also from any connected HCL Workload Automation agent. See Using the resource command from an agent on page 1003 for details. For more information about the CLIConfig.properties file, see Command-line configuration file on page 979.

Syntax

resource ?

resource [**-usr** *user_name* **-pwd** *password*]
{
[**-create**{ **-logical** *name* **-type** *type*[**-quantity** *quantity*][**-offline**] |
-group *name*[**-offline**]}]
|
[**-delete**{**-logical** *name* |
-group *name* }]
|
[**-update**{**-computer** *name*{[**-setOnline** | **-setOffline**]} |
-logical *name*
[**-setName** *name*]
[**-setType** *type*]
[**-setQuantity** *quantity*]
[**-setOnline** | **-setOffline**]
[**-addComputer** *name* |
-addComputerByID *ID* |
-removeComputer *name* |
-removeComputerByID *ID*]
|
-group *name*
[**-setName** *name*]
[**-setOnline** | **-setOffline**]

[**-addComputer** *name* |
-addComputerByID *ID* |
-removeComputer *name* |
-removeComputerByID *ID* |
-addLogical *name* |
-removeLogical *name*]}]
|
[**-query**{**-computer** *name* [-v] |
-logical *name* [-v] |
-group *name* [-v]}
[**-configFile** *configuration_file*]
}

Description

Use this command to work with computers, logical resources, and resource groups. In particular it is possible to:

- Create, update, list, and delete logical resources or groups
- Create logical resources, associate them to computers, define groups of logical resources or computers, and set them online or offline
- Retrieve and update resource properties using the query and the update options
- Discover the list of computers associated to a logical resource performing a detailed query on the logical resource
- Change the association between computers and logical resources
- Set resources online or offline and query computer properties

Options

?

Displays help information.

-usr *user_name*

Specifies the user name for a user authorized to perform operations on the command line. This option is required when security is enabled and the user name is not defined in the `CLIConfig.properties` configuration file (with the `tdwb_user` keyword).

-pwd *password*

Specifies the password for a user authorized to perform operations on the command line. This option is required when security is enabled and the password is not defined in the `CLIConfig.properties` configuration file (with the `tdwb_pwd` keyword).

-create -logical *name* -type *type*

Creates the logical resource with the specified name and type. It is also possible to set a specific quantity or set the resource offline by using optional parameters in the following way:

-create -logical *name* **-type** *type***-quantity** *quantity* **-offline**

-create -group *name*

Creates the resource group with the specified name. It is also possible to set it offline by using the `-offline` optional parameter in the following way:

-create -group *name* **-offline**

-delete -logical *name*

Deletes the logical resource with the specified name.

-delete -group *name*

Deletes the resource group with the specified name.

-update -computer *name*

Updates the computer system with the specified name. You can set the computer online or offline as follows:

-update -computer *name* -setOnline

Sets the specified computer online.

-update -computer *name* -setOffline

Sets the specified computer offline.

-update -logical *name*

Updates the specified logical resource. You can update the properties and status of a resource in the following ways:

-update -logical *name* -setName *name*

Updates the name of the specified logical resource.

-update -logical *name* -setType *type*

Updates the type of the specified logical resource.

-update -logical *name* -setQuantity *quantity*

Updates the quantity of the specified logical resource.

-update -logical *name* -setOnline

Sets online the specified logical resource.

-update -logical *name* -setOffline

Sets offline the specified logical resource.

You can change the association between a logical resource and a computer in the following ways:

-update -logical *name* -addComputer *name*

Associates the specified logical resource to the computer with the specified name.

-update -logical *name* -addComputerByID *ID*

Associates the specified logical resource to the computer with the specified ID.

-update -logical *name* -removeComputer *name*

Removes the association between the specified logical resource and the computer with the specified name.

-update -logical *name* -removeComputerByID *ID*

Removes the association between the specified logical resource and the computer with the specified ID.

-update -group *name*

Updates the specified resource group. You can update the properties and status of a resource group in the following ways:

-update -group *name* -setName *name*

Updates the name of the specified resource group.

-update -group *name* -setOnline

Sets online the specified resource group.

-update -group *name* -setOffline

Sets offline the specified resource group.

You can add and remove logical resources or computers to and from a resource group in the following ways:

-update -group *name* -addLogical *name*

Adds the logical resource with the specified name to the resource group.

-update -group *name* -removeLogical *name*

Removes the logical resource with the specified name from the resource group.

-update -group *name* -addComputer *name*

Adds the computer with the specified name to the resource group.

-update -group *name* -addComputerByID *ID*

Adds the computer with the specified ID to the resource group.

-update -group *name* -removeComputer *name*

Removes the computer with the specified name from the resource group.

-update -group *name* -removeComputerByID *ID*

Removes the computer with the specified ID from the resource group.

-query -computer *name*

Retrieves the following properties of the specified computer:

- Name
- Computer ID
- Operating system name

- Operating system type
- Operating system version
- Status
- Availability status

Retrieves the following additional properties if you add the `-v` option:

- Physical memory
- Virtual memory
- CPU utilization
- Free physical memory
- Free virtual memory
- Free swap space
- Allocated physical memory
- Allocated virtual memory
- Allocated swap space
- Processors number
- Allocated processors number
- Processor type
- Processor speed
- Manufacturer
- Model
- Serial number
- Network interfaces
- File systems

You can use the asterisk (*) as a wildcard character in the following ways:

As a single parameter

You must enclose it between double quotation marks, for example:

```
C:\HCL\TWA\TDWB\bin>resource –query –computer "*"
```

This command returns a list of all existing computers.

To complete a computer name

You must enclose the entire name between double quotation marks, for example:

```
C:\HCL\TWA\TDWB\bin> resource –query –computer "lab123*"
```

This command returns a list of all existing computers with a name starting with `lab123`.

-query -logical *name*

Retrieves the name and the type of the specified logical resource. Retrieves the following additional properties if you add the `-v` option:

- Status
- Quantity
- Current allocation
- Computers list

You can use the asterisk (*) as a wildcard character in the following ways:

As a single parameter

You must enclose it between double quotation marks, for example:

```
C:\HCL\TWA\TDWB\bin>resource –query –logical "*"
```

This command returns a list of all existing logical resources.

To complete a resource name

You must enclose the entire name between double quotation marks, for example:

```
C:\HCL\TWA\TDWB\bin> resource –query –logical "myRes*"
```

This command returns a list of all existing logical resources with a name starting with `myRes`.

-query -group *name*

Retrieves the name and the status of the specified resource group. Retrieves the list of computers and of logical resources contained in the resource group if you use the `–v` option.

You can use the asterisk (*) as a wildcard character in the following ways:

As a single parameter

You must enclose it between double quotation marks, for example:

```
C:\HCL\TWA\TDWB\bin>resource –query –group "*"
```

This command returns a list of all existing resource groups.

To complete a resource group name

You must enclose the entire name between double quotation marks, for example:

```
C:\HCL\TWA\TDWB\bin> resource –query –group "myResGrou*"
```

This command returns a list of all existing resource groups with a name starting with `myResGrou`.

-configFile *configuration_file*

Specifies the name and the path of a custom configuration file. This keyword is optional. If you do not specify it, the default configuration file is assumed. For more information on the configuration file, see the section about the CLIConfig.properties.

Authorization

The user name and password for the command are defined in the `CLIConfig.properties` file. To override the settings defined in this file, you can enter the user name and the password when you type the command. For more information on the `CLIConfig.properties` file, see the section about the CLIConfig.properties.

Return Values

The resource command returns one of the following values:

0

Indicates that the command completed successfully.

< > 0

Indicates that the command failed.

Example

Examples

- To create a logical resource named `myApplication`, of type Applications, type the following command:

  ```
  resource.bat -usr john -pwd BXVFDCGS -create -logical myApplication -type Applications
  ```

  The following output is displayed:

  ```
  AWKCLI153I Logical resource "myApplication" created.
  ```

- To update the quantity of the logical resource named `myApplication`, type the following command:

  ```
  resource.bat -update -logical myApplication -setQuantity 5 -usr john -pwd BXVFDCGS
  ```

  The following output is displayed:

  ```
  AWKCLI165I Logical resource "myApplication" updated.
  ```

- To add the relationship between a logical resource and a computer, type the following command:

  ```
  resource.bat -update -logical myApplication -addComputer myComputer -usr john -pwd BXVFDCGS
  ```

  The following output is displayed:

  ```
  AWKCLI165I Logical resource "myApplication" updated.
  ```

- To retrieve details of a logical resource named `myApplication`, type the following command:

  ```
  resource.bat -usr john -pwd BXVFDCGS -query -logical myApplication -v
  ```

  The following output is displayed:

  ```
  AWKCLI171I Calling the resource repository to perform a query on resources.

  AWKCLI172I "1" logical resources were found for your query.
  Details are as follows:

  Resource Name:myApplication
  Resource Type:Applications
  Resource Status:Online
  Resource Quantity:5
  Resource Current Allocation:0
  Computers List:
          Computer Name:myComputer
          Computer ID:D656470E8D76409F9F4FDEB9D764FF59
  ```

```
        Computer Status:Online
        Computer Availability Status:Unavailable
```

- To set the logical resource named `myApplication` offline, type the following command:

```
resource.bat -usr john -pwd BXVFDCGS -update -logical myApplication -setOffline
```

The following output is displayed:

```
AWKCLI165I Logical resource "myApplication" updated.
```

- To set the computer named `myComputer` offline, type the following command:

```
resource.bat -usr john -pwd BXVFDCGS -update -computer myComputer -setOffline
```

The following output is displayed:

```
AWKCLI165I Computer "myComputer" updated.
```

- To retrieve basic properties of the computer named `myComputer`, type the following command:

```
resource.bat -usr john -pwd BXVFDCGS -query -computer myComputer
```

The following output is displayed:

```
AWKCLI171I Calling the resource repository to perform a query on resources.
AWKCLI174I "1" computers were found for your query.
Details are as follows:


Computer Name: myComputer
Computer ID:D656470E8D76409F9F4FDEB9D764FF59
Computer OS Name: Microsoft Windows XP Professional English (United States) version
Computer OS Type:Windows XP
Computer OS Version:5
Computer Status:Offline
Computer Availability Status:Unavailable
```

- To retrieve detailed properties of the computer named `myComputer`, type the following command:

```
resource.bat -usr john -pwd BXVFDCGS -query -computer myComputer -v
```

The following output is displayed:

```
AWKCLI171I Calling the resource repository to perform a query on resources.
AWKCLI174I "1" computers were found for your query.
Details are as follows:


Computer Name: myComputer
Computer ID:D656470E8D76409F9F4FDEB9D764FF59
Computer OS Name:Microsoft Windows XP Professional English (United States) version
Computer OS Type:Windows XP
Computer OS Version:5
Computer Status:Offline
Computer Availability Status:Unavailable
Computer details:
        Physic memory = 2095536.0
        Virtual memory = 3513788.0
```

```
        Cpu utilization = 16.0
        Free physic memory = 947972.0
        Free virtual memory = 2333484.0
        Free swap space = 52.0
        Allocated physic memory = 0.0
        Allocated virtual memory = 0.0
        Allocated swap space = 0.0
        Processors number = 1.0
        Allocated processors number = 0.0
        Processor type = x86
        Processor speed = 1995.00
        Manufacturer = HCL
        Model = 2668F8G
        Serial number = L3WZYNC
```

- To retrieve detailed properties of the logical resource named `geneva`, including the list of associated computers, type the following command:

```
resource.bat -usr john -pwd BXVFDCGS -query -logical geneva -v
```

The following output is displayed:

```
Setting CLI environment variables....
AWKCLI171I Calling the resource repository to perform a query on resources.
AWKCLI172I "1" logical resources were found for your query.
Details are as follows:


Resource Name:geneva
Resource Type:prod_wks
Resource Status:Online
Resource Quantity:1
Resource Current Allocation:0
Computers List:
    Computer Name:bd_ff139_1
    Computer ID:666AADE61CBA11E0ACBECD0E6F3527DE
    Computer Status:Online
    Computer Availability Status:Available
    AWKCLI171I Calling the resource repository to perform a query on resources.
```

- To create a resource group named `myGroup`, type the following command:

```
resource.bat -usr john -pwd BXVFDCGS -create -group myGroup
```

The following output is displayed:

```
AWKCLI153I Resource group  "myGroup" created.
```

- To retrieve basic properties of a resource group named `myGroup`, type the following command:

```
resource.bat -query -group myGroup
```

The following output is displayed:

```
Setting CLI environment variables....
  AWKCLI171I Calling the resource repository to perform a query on resources.
  AWKCLI173I "1" groups were found for your query.
```

```
  Details are as follows:


  Group Name:myGroup
  Group Status:Online
```

- To add the computer named `myComputer` to a resource group named `myGroup`, type the following command:

  ```
  resource.bat -update -group myGroup  -addComputer myComputer
  ```

  The following output is displayed:

  ```
  Setting CLI environment variables....
    AWKCLI165I Resource Group  "myGroup" updated.
  ```

- To retrieve details of a resource group named `myGroup`, type the following command:

  ```
  resource.bat -query -group myGroup -v
  ```

  The following output is displayed:

  ```
  Setting CLI environment variables....
    AWKCLI171I Calling the resource repository to perform a query on resources.
    AWKCLI173I "1" groups were found for your query.
    Details are as follows:


    Group Name:myGroup
    Group Status:Online
    Computers List:
     Computer Name:myComputer
             Computer ID:D656470E8D76409F9F4FDEB9D764FF59
             Computer Status:Online
     Computer Availability Status:Unavailable

    Resources List:
  ```

Using the resource command from an agent

You can create and manage resources and groups of resources and computers from HCL Workload Automation agents other than on the master domain manager.

Enabling the resource command

To enable this feature you must:

1. Add the runtime for Java jobs when installing the agent. See information on how to install the agent in the *Planning and Installation* manual.
2. Configure the CLIConfig.properties file. See Command-line configuration file on page 979.
3. Run the **resource** command. See Running the resource command on page 1004.

For this purpose an additional instance of the CLIConfig.properties file is installed on every agent. If you intend to run the resource command from an agent, you must configure the CLIConfig.properties locally.

Configuring the local CLIConfig.properties file

When you install the agent, a local copy of CLIConfig.properties is automatically installed and partially configured on your agent in the following path:

```
/home/ITAuser/datadir/TDWB_CLI/config
```

Running the resource command

Depending on your operating system, to run the command enter:

On Windows

```
resource.bat
```

On UNIX

```
resource.sh
```

Switching managers

You can define in the `CLIConfig.properties` file the backup broker servers to be contacted by the resource CLI if the current broker server does not respond. To configure the resource CLI to contact the backup servers in case of failure, you must

specify in the `CLIConfig.properties` file the connection properties for each backup broker server. List the same properties specified for the broker server running on the primary master domain manager.

Specify the following connection properties:

```
ITDWBServerHost
ITDWBServerPort
ITDWBServerSecurePort
use_secure_connection
tdwb_user
tdwb_pwd
```

For backup servers, the same ordinal number must be appended to each property name associated to the same backup server.

In the following example, in the `CLIConfig.properties` file is specified the broker server running on the primary master domain manager and two backup broker servers:

```
# Properties of the Broker Server running on the primary master domain manager
ITDWBServerHost = BrokerServer.mycompany.com
ITDWBServerPort = 51117
ITDWBServerSecurePort = 51118
use_secure_connection = true
tdwb_user = tdwbUser
tdwb_pwd = xxxx

# First (_1) Backup Broker Server Properties
ITDWBServerHost_1 = FirstBackupBrokerServer.mycompany.com
ITDWBServerPort_1 = 41117
ITDWBServerSecurePort_1 = 41118
use_secure_connection_1 = false
tdwb_user_1 = backup1TdwbUser
tdwb_pwd_1 = yyyy

# Second (_2) Backup Broker Server Properties
ITDWBServerHost_2 = SecondBackupBrokerServer.mycompany.com
ITDWBServerPort_2 = 61117
ITDWBServerSecurePort_2 = 61118
use_secure_connection_2 = false
tdwb_user_2 = backup2TdwbUser
tdwb_pwd_2 = zzzz
```

You can define a maximum of 10 broker servers.

To prevent the resource CLI from contacting unavailable servers, the name of the last successfully contacted broker server is saved in the `ITDWBLastGoodServerHost` property of the `CLIConfig.properties` file.

sendevent

The command sends from a dynamic agent or domain manager the custom events defined with the evtdef on page 934 command to the event processor server currently active in the production plan. As the events are received by the event processor, they trigger the event rules in which they were specified.

Syntax

sendevent -V | ? | -help | -u | -usage

sendevent [**-hostname** *hostname*]
[**-port** *port*]
eventType
source
[[*attribute*=*value*]...]

Arguments

-V

Displays the command version and exits.

? | -help | -u | -usage

Displays command usage information and exits.

-hostname *hostname*

Specifies the host name of an alternate event processor server other than the currently active one.

-port *port*

Specifies the port number of an alternate event processor server other than the currently active one.

eventType

One of the custom event types defined with the evtdef on page 934 command in the generic event provider and specified as the triggering event in an event rule definition.

source

The name of the event provider that you customized with evtdef on page 934. This is also the name you must specify as the argument for the `eventProvider` keyword in the definition of the event rules triggered by these custom events.

The default name is `GenericEventPlugIn`.

attribute=value

One or more of the attributes qualifying the custom event type that are specified as the triggering event attributes for the event rule.

Comments

The command in this form applies to the dynamic environment only. To send events from non-dynamic agents, see sendevent on page 967.

Example

Examples

In this example an application running on a dynamic agent sends the `BusProcCompleted` custom event type to the default event processor. The event is that file `calcweek` finished processing.

```
sendevent BusProcCompleted GenericEventPlugIn TransacName=calcweek
Workstation=acagn002
```

The file name and the associated workstation are the two `BusProcCompleted` event attributes that were specified as triggering event attributes in an associated event rule.

TWSTRACE

Changes the trace level on the dynamic agent and, at the same time, the same trace level on the job manager gateway, without having to stop and restart the agent.

Authorization

You must login with the credentials of the user which installed the dynamic agent. You can also use any authorization higher than the user which installed the dynamic agent.

Syntax

twstrace -u | **-V** | **-enable** | **-disable** [**-level** *value*] [**-maxFiles***files_number*] [**-maxFileBytes** *bytes_number*] [**-getLogs** [**-zipFile** *zip_file_name*] [**-host***hostname*] [**-protocol** *{http|https}*] [**-port** *port number*] [**-inifile** *ini_filename*]]

Arguments

enable

Enables tracing to the maximum level. The maximum level is **3000**. By default, traces are disabled.

disable

Disables tracing.

level *value*

The level of detail for the traces:

1000

Error, warning, and informational messages are traced.

2000

Error and warning messages are traced.

3000

Error messages are traced.

maxFiles ***files_number***

The maximum number of the trace files you want to create.

maxFileBytes ***bytes_number***

The maximum size in bytes that the trace file can reach. The default is **1024000** bytes.

getLogs

To collect the trace files, the message files and the configuration files in a compressed file.

zipfile ***zip_file_name***

The name of the compressed file that contains all the information, that is logs, traces, and configuration files (`ita.ini` and `jobManager.ini`) for the agent. The default is `logs.zip`.

host ***hostname***

The host name or the IP address of the agent for which you want to collect the traces. The default is **localhost**.

protocol ***http|https***

The protocol of the agent for which you are collecting the traces. The default is the protocol specified in the **.ini** file of the agent.

port ***port number***

The port of the agent. The default is the port number of the agent where you are running the command line.

inifile ***ini_filename***

The name of the **.ini** file that contains the SSL configuration of the agent for which you want to collect the traces. The default is the **.ini** file of the local agent. If you are collecting the tracing for a remote agent for which you customized the security certificates, you must import the certificate on the local agent and specify the name of the **.ini** file that contains the configuration. To do this, perform the following actions:

1. Extract the certificate from the keystore of the remote agent.
2. Import the certificate in a local agent keystore. You can create an ad hoc keystore whose name must be **TWSClientKeyStore.kdb**.
3. Create a **.ini** file in which you specify:

- **0** in the **tcp_port** property as follows:

  ```
  tcp_port=0
  ```

- The port of the remote agent in the **ssl_port** property as follows:

  ```
  ssl_port=<ssl_port>
  ```

- The path to the keystore you created in step 2. in the **key_repository_path** property as follows:

  ```
  key_repository_path=<local_agent_keystore_path>
  ```

u

Displays the command usage.

V

Displays the version of the product.

Example

Examples

To set the trace level to record error and warning messages, run the following command:

```
twstrace -enable twstrace -level 2000
```

To retrieve the information about the trace level, run the following command:

```
twstrace -level    twstrace -maxFilestwstrace -maxFileBytes
```

Output similar to the following is returned:

```
[root@ip-192.0.2.0:]# twstrace -level
JobManager:
[JobManager.Logging.cclog]
JobManager.trhd.maxFileBytes=10240000
JobManager.trhd.maxFiles=5
JobManager.trfl.level=3000

[root@ip-192.0.2.0:]# twstrace -maxFiles
JobManager:
[JobManager.Logging.cclog]
JobManager.trhd.maxFileBytes=10240000
JobManager.trhd.maxFiles=5
JobManager.trfl.level=3000

[root@ip-192.0.2.0:]# twstrace -maxFileBytes
JobManager:
[JobManager.Logging.cclog]
JobManager.trhd.maxFileBytes=10240000
JobManager.trhd.maxFiles=5
JobManager.trfl.level=3000
```

Chapter 20. Getting reports and statistics

This chapter describes the report commands that you use to get summary or detailed information about the previous or next production plan. These commands are run from the operating system command prompt on the master domain manager. The chapter is divided into the following sections:

- Setup for using report commands on page 1010
- Command descriptions on page 1011
- Sample report outputs on page 1021
- Report extract programs on page 1033
- Running Dynamic Workload Console reports and batch reports on page 1048
- Running batch reports from the command line interface on page 1057

Setup for using report commands

About this task

To configure the environment for using report commands set the *PATH* and *TWS_TISDIR* variables by running one of the following scripts:

- `. ./TWS_home/tws_env.sh` for Bourne and Korn shells in UNIX®
- `. ./TWS_home/tws_env.csh` for C shells in UNIX®
- `TWS_home\tws_env.cmd` in Windows®

The report commands must be run from the *TWS_home* directory.

The output of the report commands is controlled by the following environment variables:

MAESTROLP

Specifies the destination of the output of a command. The default is **stdout**. You can set it to any of the following:

filename

Writes the output to a file.

> filename

UNIX® only. Redirects output to a file, overwriting the contents of the file. If the file does not exist it is created.

>> filename

UNIX® only. Redirects output to a file, appending to the end of the file. If the file does not exist it is created.

| *command*

UNIX® only. Pipes output to a system command or process. The system command is always run.

|| ***command***

UNIX® only. Pipes output to a system command or process. The system command is not run if there is no output.

MAESTRO_OUTPUT_STYLE

Specifies the output style for long object names. With value **LONG**, full length (long) fields are used for object names.

If the variable is set to anything other than **LONG**, long names are truncated to eight characters and a plus sign. For example: **A1234567+**.

You should use a fixed font size to obtain the correct format of the reports outputs.

Changing the date format

About this task

In HCL Workload Automation, the date format affects all commands that accept a date as an input option (except the **datecalc** command), and the headers in all reports. The default date format is *mm/dd/yy*. To select a different format, edit the ***date format*** local option store in the `localopts` file. The values are:

Table 108. Date formats

date format **value**	**Corresponding date format output**
0	yy/mm/dd
1	mm/dd/yy
2	dd/mm/yy
3	Native language support variables.

See the *HCL Workload Automation Administration Guide* for details on modifying local variables in the `localopts` file.

Command descriptions

HCL Workload Automation report commands are listed in Table 109: List of report commands on page 1011:

Table 109. List of report commands

Command	Description
rep1	Report 01 - Job Details Listing
rep2	Report 02 - Prompt Listing
rep3	Report 03 - Calendar Listing
rep4a	Report 04A - Parameter Listing

Table 109. List of report commands (continued)

Command	Description
rep4b	Report 04B - Resource Listing
rep7	Report 07 - Job History Listing
rep8	Report 08 - Job Histogram
rep11	Report 11 - Planned Production Schedule
reptr	Report 09A - Planned Production Summary Report 09B - Planned Production Detail Report 09D - Planned Production Detail (Long Names) Report 10A - Actual Production Summary Report 10B - Actual Production Detail
xref	Report 12 - Cross Reference Report

rep1 - rep4b

These commands print the following reports:

Report 01

Job Details Listing

Report 02

Prompt Listing

Report 03

Calendar Listing

Report 04A

Parameters Listing

Report 04B

Resource Listing

Syntax

rep[*x*] [**-V**|**-U**]

Run the command from the *TWS_home* directory.

For `rep3`, run the command from a directory to which you have `write` access.

When printing reports for job types with advanced options, the JCL file field returns the application name.

Arguments

x

A number corresponding to the report. The numbers are: **1**, **2**, **3**, **4a**, or **4b**.

-U

Displays the command usage information and exits.

-V

Displays the command version and exits.

Comments

The Job Details Listing (report 01) cannot include jobs that were submitted using an alias name.

The elapsed time displayed for a shadow job is the elapsed time of the remote job to which it is bound.

Example

Examples

Print Report 03, User Calendar Listing:

```
rep3
```

Display usage information for the **rep2** command:

```
rep2 -U
```

In UNIX®, print two copies of report 04A, User Parameters Listing, on printer `lp2`:

```
MAESTROLP="| lp -dlp2 -n2"
export MAESTROLP
rep4a
```

This is a sample report for job WAGES2_1:

```
Job: WAGES2_1      #FTP                                Description:
JCL File         : filetransfer
Logon            :                                   Creator: tws86
Recovery Job     :
Recovery Type    : STOP
Recovery Prompt  :
Composer Autodoc : Yes
Total Runs       :     0 -     0     Successful,       0 Aborted

                  Elapsed(mins)         CPU(secs)
Total             0                     0
Normal            0
Last Run          0                     0 (On        0 at     0)
Maximum           0                     0 (On        0)
Minimum           0                     0 (On        0)>
```

rep7

This command prints Report 07-Job History Listing.

Syntax

rep7 -V|-U

rep7
[**-c** *wkstat*]
[**-s** *jstream_name*]
[**-j** *job*]
[**-f** *date*]
[**-t** *date*]
[**-l**]

Run the command from the *TWS_home* directory.

Arguments

-U

Displays the command usage information and exits.

-V

Displays the command version and exits.

-c *wkstat*

Specifies the name of the workstation on which the jobs run. The default is all workstations.

-s *jstream_name*

Specifies the name of the job stream in which the jobs run. The default is all job streams.

-j *job*

Specifies the name of the job. The default is all jobs.

-f *date*

Specifies to print job history from this date forward. Enter the date as *yyyymmdd*. The default is the earliest available date.

-t *date*

Specifies to print job history up to this date. Enter the date as *yyyymmdd*. The default is the most recent date.

-l

Limits the summary line information to the jobs which fall in the date range specified by the -f or -t options. Using this option causes the order of output to be reversed: the job summary line will be printed after the individual job run lines. This option is valid only if you also specify at least one of the -f or -t options.

Comments

The elapsed time displayed for a shadow job is the elapsed time of the remote job to which it is bound.

Any time you run **rep7** the output of the command contains the information stored until the last time you run **JnextPlan**, the information related to the run of the current production plan will be contained in the **rep7** output the next time you run **JnextPlan**. For this reason if you run **rep7** after having generated the production plan for the first time or after a **ResetPlan** command, the output of the command contains no statistic information.

Example

Examples

Print all job history for workstation `ux3`:

```
rep7 -c ux3
```

Print all job history for all jobs in job stream `sked25`:

```
rep7 -s sked25
```

Print job history for all jobs in job stream `mysked` on workstation `x15` between 1/21/2023 and 1/25/2023:

```
rep7 -c x15 -s mysked -f 20230121 -t 20230125
```

rep8

This command prints Report 08-Job Histogram.

Syntax

rep8 -V|-U

rep8
[**-f** *date* **-b** *time* **-t** *date* **-e** *time*]
[**-i** *file*]
[**-p**]

rep8
[**-b** *time* **-e** *time*]
[**-i** *file*]
[**-p**]

Run the command from the *TWS_home* directory.

Arguments

-U

Displays the command usage information and exits.

-V

Displays the command version and exits.

-f *date*

Specifies to print job history from this date forward. Enter the date as *yyyymmdd*. The default is today's date.

-b *time*

Specifies to print job history from this time forward. Enter the time as *hhmm*. The default is the HCL Workload Automation *startOfDay*.

-t *date*

Specifies to print job history up to this date. Enter the date as *yyyymmdd*. The default is the most recent date.

-e *time*

Specifies to print job history up to this time. Enter the time as *hhmm*. The default is the HCL Workload Automation start of day time.

-i *file*

Specifies the name of the log file from which job history is extracted. Note that log files are stored in the `schedlog` directory. The default is the current `Symphony` file.

Note: Ensure that the time range specified by the [**-f** *date* **-b** *time* **-t** *date* **-e** *time*] arguments is within the date and time range defined in the **-i** *file* log file name.

-p

Specifies to insert a page break after each run date.

Comments

Any time you run **rep8** the output of the command contains the information stored until the last time you run **JnextPlan**, the information related to the run of the current production plan will be contained in the **rep8** output the next time you run **JnextPlan**. For this reason if you run **rep8** after having generated the production plan for the first time or after a **ResetPlan** command, the output of the command contains no statistic information.

Example

Examples

Print a job histogram which includes all information in the current plan (`Symphony` file):

```
rep8
```

Print a job histogram beginning at 6:00 a.m. on 1/25/2023, and ending at 5:59 a.m. on 1/26/2023:

```
rep8 -f 20230125 -b 0600 -t 20230126 -e 0559 -i schedlog/M199801260601
```

Print a job histogram, from the current plan (`Symphony` file), beginning at 6:00 am, and ending at 10:00 pm:

```
rep8 -b 0600 -e 2200
```

rep11

This command prints Report 11-Planned Production Schedule.

Syntax

rep11 -V|-U

rep11
[**-m** *mm*[*yy*] [...]]
[**-c** *wkstat* [...]]
[**-s** *jstream_name*]
[**-o** *output*]

Run the command from the *TWS_home* directory.

Arguments

-U

Displays the command usage information and exits.

-V

Displays the command version and exits.

-m *mm*[*yy*]

Specifies the months to be reported. Enter the month number as *mm*. The default is the current month.

You can also enter a year as *yy*. The default is the current year or next year if you specify a month earlier than the current month.

-c *wkstat*

Specifies the workstations to be reported. The default is all workstations.

-s *jstream_name*

Specifies the name of the job stream in which the jobs run. The default is all job streams.

-o *output*

Specifies the output file. The default is the file defined by the ***MAESTROLP*** variable. If ***MAESTROLP*** is not set, the default is **stdout**.

Example

Examples

Report on June, July, and August of 2024 for workstations `main`, `site1` and `sagent1`:

```
rep11 -m 0624 0724 0824 -c main site1 sagent1
```

Report on June, July, and August of this year for all workstations, and direct output to the file `r11out`:

```
rep11 -m 06 07 08 -o r11out
```

Report on this month and year for workstation `site2`:

```
rep11 -c site2
```

reptr

This command prints the following reports:

Report 09A

Planned Production Summary

Report 09B

Planned Production Detail

Report 10A

Actual Production Summary

Report 10B

Actual Production Detail

Report 09A and Report 09B refer to future production processing while Report 10A and Report 10B show processing results and status of each single job of already processed production.

Syntax

reptr [**-V**|**-U**]

reptr -pre
[**-**{**summary** | **detail**}]
[*symfile*]

reptr -post
[**-**{**summary** | **detail**}]
[*logfile*]

Run the command from a directory to which you have `write` access.

Arguments

-U

Displays the command usage information and exits.

-V

Displays the command version and exits.

-pre

Specifies to print the preproduction reports (09A and 09B).

-post

Specifies to print the post-production reports (10A and 10B).

-summary

Specifies to print the summary reports (09A and 10A). If **-summary** and **-detail** are omitted, both sets of reports are printed.

-detail

Specifies to print the detail reports (09B and 10B). If **-summary** and **-detail** are omitted, both sets of reports are printed.

symfile

Specifies the name of the plan file from which reports will be printed. The default is `Symnew` in the current directory. If the file is not in the current working directory, you must add the absolute path to the file name.

logfile

Specifies the full name of the log file from which the reports will be printed. Note that plan log files are stored in the `schedlog` directory. The default is the current plan (`Symphony` file).

If the command is run with no options, the two **pre** reports (09A and 09B) are printed and the information is extracted from the `Symphony` file.

Example

Examples

Print the preproduction detail report from the `Symnew` file:

```
reptr -pre -detail
```

Print the preproduction summary report from the file `mysym`:

```
reptr -pre -summary mysym
```

Print the post-production summary report from the log file `M199903170935`:

```
reptr -post -summary schedlog/M199903170935
```

Print the preproduction reports reading from the Symphony file.

```
reptr
```

When the arguments are specified, the preproduction reports are based on information read from the `Symnew` file while the post-production reports are based on information read from the `Symphony` file.

xref

This command prints Report 12-Cross Reference Report.

Syntax

xref [**-V**|**-U**]

xref
[**-cpu** *wkstat*]
[**-depends**|**-files**|**-jobs**|**-prompts**|**-resource**|**-schedules**|**-when**[…]]

Run the command from the *TWS_home* directory.

Arguments

-U

Displays the command usage information and exits.

-V

Displays the command version and exits.

-cpu *wkstat*

Specifies to print the report for the named workstation. The @ wildcard is permitted, in which case, information from all qualified workstations is included. The default is all workstations.

-depends

Specifies to print a report showing the job streams and jobs that are successors of each job.

-files

Specifies to print a report showing the job streams and jobs that are dependent on each file.

-jobs

Specifies to print a report showing the job streams in which each job is run.

-prompts

Specifies to print a report showing the job streams and jobs that are dependent on each prompt.

-resource

Specifies to print a report showing the job streams and jobs that are dependent on each resource.

-schedules

Specifies to print a report showing the job streams and jobs that are successors of each job stream.

-when

Specifies to print a report showing job stream `Include` and `Exclude` dates.

If the command is run with no options, all workstations and all options are selected.

Example

Examples

Print a report for all workstations, showing all cross-reference information:

```
xref
```

Print a report for all workstations. Include cross-reference information about all successor dependencies:

```
xref -cpu @ -depends -schedules
```

Sample report outputs

Report 01 - Job Details Listing:

Example

```
TWS for UNIX (AIX)/REPORT1 10.2.8         hcl                           Page    1
Report 01                               Job Details Listing                 03/06/25

Job              : FTAWIN8+        #SCHEDDDD
Description      :
JCL File         : dir
Logon            : maestro_adm
Creator          : root
Recovery Job     :
Recovery Type    : STOP
Recovery Prompt  :
composer Autodoc : Yes
Total Runs       :     0 -     0     Successful,       0 Aborted

                  Elapsed(mins)      CPU(secs)
Total             00:00:00              0
Normal            00:00:00
Last Run          00:00:00              0 (On          at 00:00)
Maximum           00:00:00              0 (On         )
Minimum           00:00:00              0 (On         )


Job              : MASTER8+        #JnextPlan
Description      : ADDED BY composer FOR SCHEDULE MASTER821#FINAL.
JCL File         : /test/maestro_adm/tws/JnextPlan
Logon            : maestro_adm
Creator          : maestro_adm
Recovery Job     :
Recovery Type    : STOP
Recovery Prompt  :
composer Autodoc : Yes
Total Runs       :    11 -    11     Successful,       0 Aborted

                  Elapsed(mins)      CPU(secs)
Total             00:00:14             44
Normal            00:00:01
Last Run          00:00:01              4 (On 03/05/25 at 23:16)
```

```
Maximum           00:00:02              4 (On 03/04/25)
Minimum           00:00:01              4 (On 03/04/25)


Job              : MASTER8+        #JOB1
Description      : ADDED BY composer.
JCL File         : pwd
Logon            : ^ACCLOGIN^
Creator          : root
Recovery Job     :
Recovery Type    : STOP
Recovery Prompt  :
composer Autodoc : Yes
Total Runs       :     1 -     1     Successful,       0 Aborted

                  Elapsed(mins)      CPU(secs)
Total             00:00:01              0
Normal            00:00:01
Last Run          00:00:01              0 (On 03/05/25 at 22:22)
Maximum           00:00:01              0 (On 03/05/25)
Minimum           00:00:01              0 (On 03/05/25)

                       * * * *   E n d   o f   R e p o r t   * * * *
```

In the output you see the values set in the Job on page 1407 as follows::

composer Autodoc

Says if the job statement was described in the job stream definition using the command line interface.

CPU (secs)

Is the actual time, expressed in seconds, the job made use of the CPU to run.

Total

Is the sum of CPU time recorded for the 'Total Runs'.

Normal

Is the average value of CPU time recorded during the 'Total Runs'.

Last Run

Is the CPU time recorded during the last run of the job.

Maximum

Is the maximum among the values collected for CPU time during the 'Total Runs' (calculated only for jobs ended successfully).

Minimum

Is the minimum among the values collected for CPU time during the 'Total Runs' (calculated only for jobs ended successfully).

Creator

Is the name of the user who created the job definition.

Description

Is the textual description of the job set in the **description** field of the job definition statement.

Elapsed

Is the amount of time, expressed in minutes, that includes both the time during which the job made use of the CPU and the time the job had to wait for other processes to release the CPU.

Total

Is the sum of Elapsed time recorded for the 'Total Runs'.

Normal

Is the average value of Elapsed time recorded during the 'Total Runs'.

Last Run

Is the Elapsed time recorded during the last run of the job.

Maximum

Is the maximum among the values collected for Elapsed time during the 'Total Runs' (calculated only for jobs ended successfully).

Minimum

Is the minimum among the values collected for Elapsed time during the 'Total Runs' (calculated only for jobs ended successfully).

Note: The elapsed time displayed for a shadow job is the elapsed time of the remote job to which it is bound.

JCL File

Is the name of the file set in the **scriptname** field that contains the script to run, or the command specified in the **docommand** field to invoke when running the job.

Job

Is the identifier of the job, [*workstation***#**]*jobname*.

Logon

Is the user name, specified in the **streamlogon** field, under which the job runs.

Recovery Job

Is the job, specified as *after [workstation#]jobname*, that is run if the parent job abends.

Recovery Prompt

Is the text of the prompt, specified in the **abendprompt** field, that is displayed if this job abends.

Recovery Type

Is the recovery option set in the job definition. It can be set to **stop**, **continue**, or **rerun**.

Report 02 - Prompt Listing:

Example

```
TWS for UNIX (AIX)/REPORT2 10.2.8           hcl                          Page    1
Report 02                               Prompt Message Listing              03/06/25


Prompt      Message

PROMPT1     Reply YES when ready to run acc103 and acc104.
PROMPT2     Have all users logged out?
CALLNO      555-0911
CALLOPER    Call ^PERSON2CALL^ at ^CALLNO^ to ensure all time cards have been processed.
PERSON2CALL Lou Armstrong

Total number of prompts on file:      5

                         * * * *   E n d   o f   R e p o r t   * * * *
```

The Report 02 output lists the name and the text of the prompts defined in the environment.

Report 03 - Calendar Listing:

Example

```
TWS for UNIX (AIX)/REPORT3 10.2.8          hcl                          Page    1
Report 03                               User Calendar List                  03/06/22


Calendar Type: MONTHEND
Description: End of month until end of 2022.


          Jan 2022                      Feb 2022                      Mar 2022
Sun Mon Tue Wed Thu Fri Sat   Sun Mon Tue Wed Thu Fri Sat   Sun Mon Tue Wed Thu Fri Sat
                          .             .   .   .   .   .             .   .   .   .   .
 .   .   .   .   .   .   .     .   .   .   .   .   .   .     .   .   .   .   .   .   .
 .   .   .   .   .   .   .     .   .   .   .   .   .   .     .   .   .   .   .   .   .
 .   .   .   .   .   .   .     .   .   .   .   .   .   .     .   .   .   .   .   .   .
 .   .   .   .   .   .   .     .  28                         .   .   .   .  31
 .  31

          Apr 2022                      May 2022                      Jun 2022
Sun Mon Tue Wed Thu Fri Sat   Sun Mon Tue Wed Thu Fri Sat   Sun Mon Tue Wed Thu Fri Sat
                      .   .     .   .   .   .   .   .   .                 .   .   .   .
 .   .   .   .   .   .   .     .   .   .   .   .   .   .     .   .   .   .   .   .   .
 .   .   .   .   .   .   .     .   .   .   .   .   .   .     .   .   .   .   .   .   .
 .   .   .   .   .   .   .     .   .   .   .   .   .   .     .   .   .   .   .   .   .
 .   .   .   .   .   .  30     .   .  31                     .   .   .   .  30


          Jul 2022                      Aug 2022                      Sep 2022
Sun Mon Tue Wed Thu Fri Sat   Sun Mon Tue Wed Thu Fri Sat   Sun Mon Tue Wed Thu Fri Sat
                      .   .         .   .   .   .   .   .                     .   .   .
 .   .   .   .   .   .   .     .   .   .   .   .   .   .     .   .   .   .   .   .   .
```

```
 .   .   .   .   .   .   .    .   .   .   .   .   .   .    .   .   .   .   .   .   .
 .   .   .   .   .   .   .    .   .   .   .   .   .   .    .   .   .   .   .   .   .
 .   .   .   .   .   .   .    .   .   .  31               .   .   .   .   .  30
31

          Oct 2022                     Nov 2022                     Dec 2022
Sun Mon Tue Wed Thu Fri Sat   Sun Mon Tue Wed Thu Fri Sat   Sun Mon Tue Wed Thu Fri Sat
                        .             .   .   .   .   .                     .   .   .
 .   .   .   .   .   .   .    .   .   .   .   .   .   .    .   .   .   .   .   .   .
 .   .   .   .   .   .   .    .   .   .   .   .   .   .    .   .   .   .   .   .   .
 .   .   .   .   .   .   .    .   .   .   .   .   .   .    .   .   .   .   .   .   .
 .   .   .   .   .   .   .    .   .   .  30               .   .   .   .   .   .  31
 .  31


                        * * * *  E n d  o f  R e p o r t  * * * *
```

In the output you see highlighted the end of month days selected in calendar **MONTHEND**.

Report 04A - Parameter Listing:

Example

```
TWS for UNIX (AIX)/REPORT4A 10.2.8)         hcl                          Page    1
Report 4A                              User Parameter Listing               03/06/25


Parameter Name            Contents

ACCHOME                   /usr/local/Tivoli/maestro_adm
ACCLOGIN                  maestro_adm
BADEXIT                   99
GOODEXIT                  0
SCRPATH                   /usr/local/Tivoli/maestro_adm/scripts

Number of Parameters on file:      5

                          * * * *  E n d  o f  R e p o r t  * * * *
```

The Report 04A output lists the name and the content of the parameters defined in the environment.

Report 04B - Resource Listing:

Example

```
TWS for UNIX (AIX)/REPORT4B 10.2.8         hcl                          Page    1
Report 4B                              TWS Resources Listing                03/06/25



               Resource               Number
CPU              Name                  Avail          Description

FTAHP            #DATTAPES               1            DAT tape units
```

```
FTAWIN8+         #QUKTAPES               2           Quick tape units

MASTER8+         #TAPES                  2           Tape units

MASTER8+         #JOBSLOTS            1024           Job slots


Number of Resources on file:      4

                         * * * *  E n d  o f  R e p o r t  * * * *
```

The Report 04B output lists the name, the number of available resources defined in the environment and their description.

Report 07 - Job History Listing:

Example

```
TWS for UNIX (AIX)/REPORT7 10.2.8         hcl                        Page    1
Report 07                              Job History Listing                03/08/25

Date        Time      Job Stream Name     Elapsed       CPU     Status

Job:MASTER8+#MyJS Runs: Aborted 0  Successful 11  Elapsed Time: Normal 1 Min 1 Max 2

03/03/25    01:46     MASTER8+#JS1            1         4        SU
03/03/25    19:08     MASTER8+#JS2            1         4        SU
03/03/25    19:33     MASTER8+#JS3            1         4        SU
03/03/25    19:37     MASTER8+#JS4            1         4        SU
03/03/25    23:08     MASTER8+#JS5            2         4        SU
03/03/25    05:59     MASTER8+#JS_A           1         4        SU
03/05/25    05:59     MASTER8+#JS_G           1         4        SU
03/06/25    05:59     MASTER8+#JS_H           1         4        SU
03/06/25    21:57     MASTER8+#TIMEJ          2         4        SU
03/06/25    23:16     MASTER8+#SLEEPJ         1         4        SU

Job:MASTER8+#JOB1     Runs: Aborted 0   Successful 1  Elapsed Time: Normal 1 Min 1 Max 1

03/06/25    22:22     MASTER8+#JOBS           1         0        SU


                          * * * *  E n d  o f  R e p o r t  * * * *
```

The Report 7 reads the information about job run stored in the database and displays them. The possible states for a job are:

AB

for failed jobs

SU

for successfully completed jobs

DN

for submitted jobs whose state is unknown because neither a successful or a failure message has been received yet.

Report 08 - Job Histogram:

Example

```
TWS for UNIX (AIX)/REPORT8 10.2.8         hcl                           Page    1
Report 08            Job Histogram 03/03/26 14:05 - 03/04/26 14:04              03/04/26

Interval Per Column: 15 minutes

                        1   1   1   1   2   2   2   0   0   0   0   0   0   0   1   1   1
                        4   5   7   8   0   1   3   0   2   3   5   6   8   9   1   2   4
                        0   3   0   3   0   3   0   3   0   3   0   3   0   3   0   3   0
                        5   5   5   5   5   5   5   5   5   5   5   5   5   5   5   5   4

Job Name          Stat
03/04/26

CF05066+.JnextPlan   SU                                          .*.


                       * * * *  E n d  o f  R e p o r t  * * * *
```

The output of Report 8 shows the time slots during which the jobs run. The numbers at the top of the job histogram are times, written top-down, for example the first column 1405 means 2:05PM. The time slots when the job run are marked by asterisks when the position of the marker is aligned with a time written top-down, and dots.

Report 9B - Planned Production Detail:

Example

```
TWS for UNIX (AIX) REPORTER 10.2.8         hcl                                Page    1
Report 09B   Symnew       Planned Production Detail For 03/06/25                 03/06/25

                       Estimated
               Job Name Run Time Pri Start Time     Until           Every Limit Dependencies

Schedule NETAG #EXTERNAL           0
                E0000000           0
              Total        00:00
              Total        00:00

Schedule MYFTA #IWDSKE            10                                            NETAG#EXTERNAL.E0000000
                JOBIWD            10                 23:00(03/06/25) 01:00
              Total        00:00

Schedule MYMST #TESTSKE    00:29  10
                TESTCRO+   00:01  10
                NEWTEST    00:29  10 08:30(03/06/25)                            TESTCROME
              Total        00:29
```

```
Schedule MYMST #FINAL      00:00  10 05:59(03/07/25)
               JnextPlan   00:01  10
             Total        00:01
             Total        00:34

                          * * * *  E n d  o f  R e p o r t  * * * *
```

The output of Report 9B shows what is in plan to run on the selected date in the HCL Workload Automation environment. The information displayed is taken from the definitions stored in the HCL Workload Automation database. The output shows the job streams that are planned to run on the 6th of March 2022 with their description, the list of jobs they contain, the time dependencies, repetition rate, and job limit, if set, and the dependency on other jobs or job streams. For example, job stream named `iwdske` that is planned to run on `MYFTA` has a `follows` dependency on job `NETAG#EXTERNAL.E0000000` that is planned to run on the network agent named `NETAG`.

The **Start Time** field in the output of the reports generated by the **reptr** command shows:

A time restriction set in the job stream definition using the at keyword.

If the date is enclosed in parenthesis (), for example:

```
Start Time
06:00(03/20/25)
```

The time the job stream is planned to run set in the job stream definition using the schedtime keyword.

If the date is enclosed in braces {}, for example:

```
Start Time
06:00{03/20/25}
```

The time the job stream actually started to run.

If the date is not enclosed either in braces or in parenthesis, for example:

```
Start Time
06:00 03/20/25
```

Report 10B - Actual Production Detail:

Example

```
HCL Workload Automation for UNIX (AIX) REPORTER 10.2.8          hcl                              Page    1
Report 10B   Symphony            Actual Production Detail For 03/06/25          03/07/25

                        Estimated                          Actual    CPU    Job
               Job Name  Run Time Priority Start Time    Run Time Seconds Number  Status

Schedule NETAG #EXTERNAL              0                                          EXTRN
                E0000000              0                                          ERROR
                 Total      00:00                             00:00    0

Schedule MYMST #MONTHSKE    00:02    10     06:01(03/06/25)  00:03                SUCC
                GETLOGS     00:02    10     06:01(03/06/25)  00:03         #J11612   SUCC
                 Total      00:02                             00:03    0
```

```
Schedule MYFTA #IWSKE               10                                          HOLD
                JOBIWD              10                                          HOLD
                 Total      00:00                          00:00    0

Schedule MYMST #TESTSKE     00:29    10     06:01(03/06/25)  00:02               STUCK
                TESTCRO+    00:01    10     06:01(03/06/25)  00:02       #J11613  ABEND
                NEWTEST     00:29    10                                          HOLD
                 Total      00:30                          00:02    0

Schedule MYMST #FINAL       00:01    10     05:59(03/07/25)                      HOLD
               JnextPlan    00:01    10                                          HOLD
                 Total      00:01                          00:00    0


                 Total      01:38                          00:09    0

                              * * * *  E n d  o f  R e p o r t  * * * *
```

The output of Report 10B shows states of the scheduling activities currently running across the HCL Workload Automation network. The information displayed is taken from the copy of the `Symphony` file that is currently used and updated across the scheduling environment. This means that anytime this report command is run during the processing, the information displayed reflects the actual status of the planned activity.

If you compare this output with the output of Report 9B you see that job stream `MONTHSKE` has run during the current production day, the 6th of March, but is not in plan to run the next day, the 7th of March. The job stream `EXTERNAL` instead failed on the network agent `NETAG` and so the `IWSKE` job stream that has a `follows` dependency from `EXTERNAL` job stream remains in the HOLD state.

The job stream `TESTSKE`, instead, is in state STUCK, that means that operator intervention is needed, because within the job stream run time, job `TESTCROME`, after having started with job ID `J11613`, failed in ABEND state causing the depending job `NEWTEST` to turn into HOLD state.

The **Actual Run Time** field is displayed as `mm:ss`. However, if the job runs over 100 minutes, the number of seconds might not be correctly displayed.

Report 11 - Planned Production Schedule:

Example

```
TWS for UNIX (AIX)/REPORT11 10.2.8                                          Page    1
Report 11                Planned Production Schedule for FEB 2022              03/08/22

CPU: FTAWIN8+

         Num  Est Cpu  1  2  3  4  5  6  7  8  9 10 11 12 13 14 15 16 17 18 19 20 21 22 23 24 25 26 27 28
Schedule Jobs  Time   Tu We Th Fr Sa Su Mo Tu We Th Fr Sa Su Mo Tu We Th Fr Sa Su Mo Tu We Th Fr Sa Su Mo
SCHED1    1       1                                 *

An * between Schedule name and Num Jobs indicates that the schedule has jobs running on other cpus.
-------------------------------------------------------------------------------------------------------
Estimated Cpu Time Per Day in Seconds
```

```
     Mon      Tue      Wed      Thu      Fri      Sat      Sun

               1        2        3        4        5        6
               0        0        0        0        0        0

       7        8        9       10       11       12       13
       0        0        0        0        1        0        0

      14       15       16       17       18       19       20
       0        0        0        0        0        0        0

      21       22       23       24       25       26       27
       0        0        0        0        0        0        0

      28
       0


TWS for UNIX (AIX)/REPORT11 10.2.8 (1.7)                                      Page    2
Report 11                        Planned Production Schedule for FEB 2022       03/08/22

CPU: MASTER8+

         Num  Est Cpu  1  2  3  4  5  6  7  8  9 10 11 12 13 14 15 16 17 18 19 20 21 22 23 24 25 26 27 28
Schedule Jobs  Time   Tu We Th Fr Sa Su Mo Tu We Th Fr Sa Su Mo Tu We Th Fr Sa Su Mo Tu We Th Fr Sa Su Mo
FINAL     1       4    *  *  *  *  *  *  *  *  *  *  *  *  *  *  *  *  *  *  *  *  *  *  *  *  *  *  *  *

An * between Schedule name and Num Jobs indicates that the schedule has jobs running on other cpus.
-------------------------------------------------------------------------------------------------------
Estimated Cpu Time Per Day in Seconds
     Mon      Tue      Wed      Thu      Fri      Sat      Sun

               1        2        3        4        5        6
               4        4        4        4        4        4

       7        8        9       10       11       12       13
       4        4        4        4        4        4        4

      14       15       16       17       18       19       20
       4        4        4        4        4        4        4

      21       22       23       24       25       26       27
       4        4        4        4        4        4        4

      28
       4


                        * * * *  E n d  o f  R e p o r t  * * * *
```

The output of Report 11 shows when the job streams are planned to run during the selected month. In the first line it is displayed the number of jobs the job stream contains, the estimated CPU time used by the job stream to run, and when the job stream is planned to run. In the matrix it is displayed for each day of the selected month the estimated CPU time used by that job stream to run.

Report 12 - Cross Reference Report:

The output of Report 12 shows different information according to the flag used when issuing the `xref` command. In this section you find some samples of output. For each of these sample the corresponding flag used with the xref command is highlighted.

xref -when

Example

```
TWS for UNIX (AIX)/CROSSREF 10.2.8         hcl                              Page    1
   Report 12 Cross Reference Report for the ON, EXCEPT(*) and FREEDAYS(f) options. 03/08/25

   CPU: FTAHP

   WHEN                  Used by the following schedules:
   REQUEST               TRFINAL



   TWS for UNIX (AIX)/CROSSREF 10.2.8         hcl                              Page    2
   Report 12 Cross Reference Report for the ON, EXCEPT(*) and FREEDAYS(f) options. 03/08/25

   CPU: FTAWIN8+

   WHEN                  Used by the following schedules:
   MONTHEND              SCHED1
   REQUEST               SCHED1  , SCHEDDAA



   TWS for UNIX (AIX)/CROSSREF 10.2.8         hcl                              Page    3
   Report 12 Cross Reference Report for the ON, EXCEPT(*) and FREEDAYS(f) options. 03/08/25

   CPU: MASTER8+

   WHEN                  Used by the following schedules:
   EVERYDAY              FINAL
   REQUEST               TMP


                         * * * *  E n d  o f  R e p o r t  * * * *
```

xref -jobs

Example

```
TWS for UNIX (AIX)/CROSSREF 10.2.8         hcl                             Page    4
   Report 12                    Cross Reference Report for Job Names.           03/08/25

   CPU: FTAWIN8+

   Job Name              Exists in Schedules
```

```
   SCHEDDDD              SCHED1



   TWS for UNIX (AIX)/CROSSREF 10.2.8         hcl                             Page    5
   Report 12                     Cross Reference Report for Job Names.           03/08/25

   CPU: MASTER8+

   Job Name              Exists in Schedules
   JnextPlan              FINAL
   JOB1                  TMP


                         * * * *   E n d   o f   R e p o r t   * * * *
```

xref -resource

Example

```
TWS for UNIX (AIX)/CROSSREF 10.2.8         hcl                             Page    8
   Report 12                    Cross Reference Report for Resource Users.        03/08/25

   CPU: FTAWIN8+

   Resource              Used by the following:
   QUKTAPES(N/F)         SCHED1



   TWS for UNIX (AIX)/CROSSREF 10.2.8         hcl                             Page    9
   Report 12                    Cross Reference Report for Resource Users.        03/08/25

   CPU: MASTER8+

   Resource              Used by the following:
   TAPES(N/F)            TMP


                         * * * *   E n d   o f   R e p o r t   * * * *
```

xref -prompts

Example

```
TWS for UNIX (AIX)/CROSSREF 10.2.8         hcl                             Page    6
   Report 12                   Cross Reference Report for Prompt Dependencies.     03/08/25
   CPU: FTAWIN8+

   Prompt                Used by the following:

   User defined text     SCHED1
```

```
TWS for UNIX (AIX)/CROSSREF 10.2.8         hcl                              Page    7
Report 12                  Cross Reference Report for Prompt Dependencies.     03/08/25

CPU: MASTER8+

Prompt                Used by the following:
BADEXIT               FTAWIN8+#SCHED1
GOODEXIT              FTAWIN8+#SCHED1  , TMP

User defined text     TMP


                        * * * *  E n d  o f  R e p o r t  * * * *
```

xref -files

Example

```
TWS for UNIX (AIX)/CROSSREF 10.2.8         hcl                              Page   10
   Report 12                   Cross Reference Report for File Dependencies.      03/08/25

   CPU: MASTER8+

   File Name             Used by the following:
   /root/MY_FILE.sh      FTAWIN8+#SCHED1  , TMP


                           * * * *  E n d  o f  R e p o r t  * * * *
```

Report extract programs

Data extraction programs are used to generate several of the HCL Workload Automation reports. The programs are listed in Table 110: Report extract programs. on page 1033:

Table 110. Report extract programs.

Report extract program	Description
jbxtract	Used to generate Report 01 - Job Details Listing and for Report 07 - Job History Listing
prxtract	Used to generate Report 02 - Prompt Listing
caxtract	Used to generate Report 03 - Calendar Listing
paxtract	Used to generate Report 04A - Parameters Listing
rextract	Used to generate Report 04B - Resource Listing

Table 110. Report extract programs. (continued)

Report extract program	Description
r11xtr	Used to generate Report 11 - Planned Production Schedule
xrxtrct	Used to generate Report 12 - Cross Reference Report

The output of the extract programs is controlled by the ***MAESTRO_OUTPUT_STYLE*** variable, which defines how long object names are handled. For more information on the ***MAESTRO_OUTPUT_STYLE*** variable, refer to Command descriptions on page 1011.

jbxtract

Extracts information about jobs from the database.

Syntax

jbxtract [**-V** | **-U**]
[**-j** *job*]
[**-c** *wkstat*]
[**-o** *output*]

Arguments

-V

Displays the command version and exits.

-U

Displays command usage information and exits.

-j *job*

Specifies the job for which extraction is performed. The default is all jobs.

-c *wkstat*

Specifies the workstation of jobs for which extraction is performed. The default is all workstations.

-o *output*

Specifies the output file. The default is **stdout**.

Results

The ***MAESTRO_OUTPUT_STYLE*** variable specifies the output style for long object names. With value **LONG**, full length (long) fields are used for object names. If the variable is set to anything other than **LONG**, long names are truncated to eight characters and a plus sign. For example: `A1234567+`.

Each job record contains tab-delimited, variable length fields. The fields are described Table 111: Jbxtract output fields on page 1035.

Table 111. Jbxtract output fields

Field	Description	Max Length (bytes)
1	workstation name	16
2	job name	16
3	job script file name	4096
4	job description	65
5	recovery job name	16
6	recovery option (0=stop, 1=rerun, 2=continue)	5
7	recovery prompt text	64
8	auto-documentation flag (0=disabled, 1=enabled)	5
9	job login user name	36
10	job creator user name	36
11	number of successful runs	5
12	number of abended runs	5
13	total elapsed time of all job runs	8
14	total cpu time of all job runs	8
15	average elapsed time	8
16	last run date (yymmdd)	8
17	last run time (hhmm)	8
18	last cpu seconds	8
19	last elapsed time	8
20	maximum cpu seconds	8
21	maximum elapsed time	8
22	maximum run date (yymmdd)	8

Table 111. Jbxtract output fields (continued)

Field	Description	Max Length (bytes)
23	minimum cpu seconds	8
24	minimum elapsed time	8
25	minimum run date (yymmdd)	8

Note: The elapsed time displayed for a shadow job is the elapsed time of the remote job to which it is bound.

Example

Examples

To extract information about job `myjob` on workstation `main` and direct the output to the file `jinfo`, run the following command:

```
jbxtract -j myjob -c main -o jinfo
```

prxtract

Extracts information about prompts from the database.

Syntax

prxtract [**-V** | **-U**] [**-o** *output*]

Arguments

-V

Displays the command version and exits.

-U

Displays command usage information and exits.

-o *output*

Specifies the output file. The default is **stdout**.

Results

Each prompt record contains tab-delimited, variable length fields. The fields are described in Table 112: Prxtract output fields on page 1037.

Table 112. Prxtract output fields

Field	Description	Max Length (bytes)
1	prompt name	8
2	prompt value	200

Example

Examples

To extract information about all prompt definitions and direct the output to the file `prinfo`, run the following command:

```
prxtract -o prinfo
```

caxtract

Extracts information about calendars from the database.

Syntax

caxtract [**-V** | **-U**] [**-o** *output*]

Arguments

-V

Displays the command version and exits.

-U

Displays command usage information and exits.

-o *output*

Specifies the output file. The default is **stdout**.

Results

Each calendar record contains tab-delimited, variable length fields. The fields are described in Table 113: Caxtract output fields on page 1038.

Table 113. Caxtract output fields

Field	Description	Max Length (bytes)
1	calendar name	8
2	calendar description	64

Example

Examples

To extract information about all calendar definitions and direct the output to the file `cainfo`, run the following command:

```
caxtract -o cainfo
```

paxtract

Extracts information about global parameters (variables) from the database.

Syntax

paxtract [**-V** | **-U**] [**-o** *output*] [**-a**]

Arguments

-V

Displays the command version and exits.

-U

Displays command usage information and exits.

-o *output*

Specifies the output file. The default is **stdout**.

-a

Displays all the variables defined in all the variable tables. If not specified, only the variables defined in the default variable table are displayed.

Results

Each variable record contains tab-delimited, variable length fields. The fields are described in Table 114: Paxtract output fields on page 1039.

Table 114. Paxtract output fields

Field	Description	Max Length (bytes)
1	table name	80
2	variable name	16
3	variable value	72

Remember: If you do not specify the **-a** (all) option in the command, only fields 2 and 3 are displayed and the variables listed are the ones contained in the default variable table only.

Example

Examples

To extract information about all variable definitions and direct the output to the file `allvarinfo`, run the following command:

```
paxtract -a -o allvarinfo
```

rextract

Extracts information about resources from the database.

Syntax

rextract [**-V** | **-U**] [**-o** *output*]

Arguments

-V

Displays the command version and exits.

-U

Displays command usage information and exits.

-o output

Specifies the output file. The default is **stdout**.

Results

Each resource record contains tab-delimited, variable length fields. The fields are described in Table 115: Rextract output fields on page 1040.

Table 115. Rextract output fields

Field	Description	Max Length (bytes)
1	workstation name	8/16
2	resource name	8
3	total resource units	4
4	resource description	72

Example

Examples

To extract information about all resource definitions and direct the output to the file `reinfo`, run the following command:

```
rextract -o reinfo
```

r11xtr

Extracts information about job streams from the database.

Syntax

r11xtr [**-V** | **-U**]
[**-m** *mm*[*yy*]]
[**-c** *wkstat*]
[**-o** *output*]
[**-s** *jstream_name*]

Arguments

-V

Displays the program version and exits.

-U

Displays program usage information and exits.

-m *mm*[*yy*]

Specifies the month (*mm*) and, optionally, the year (*yy*) of the job streams. The default is the current month and year.

-c *wkstat*

Specifies the workstation to be reported. The default is all workstations.

-s *jstream_name*

Specifies the name of the job stream in which the jobs run. The default is all job streams.

-o *output*

Specifies the output file. The default is **stdout**.

Results

The ***MAESTRO_OUTPUT_STYLE*** variable specifies the output style for long object names. With value **LONG**, full length (long) fields are used for object names. If the variable is set to anything other than **LONG**, long names are truncated to eight characters and a plus sign. For example: `A1234567+`.

Each job stream record contains tab-delimited, variable length fields. The fields are described in Table 116: R11xtr output fields on page 1041.

Table 116. R11xtr output fields

Field	Description	Max Length (bytes)
1	workstation name	16
2	job stream name	16
3	job stream date (yymmdd)	6
4	estimated cpu seconds	6
5	multiple workstation flag (* means some jobs run on other workstations)	1
6	number of jobs	4
7	day of week (Su, Mo, Tu, We, Th, Fr, Sa)	2

Example

Examples

To extract information about job streams on June 2023 for workstation `main`, run the following command:

```
r11xtr -m 0623 -c main
```

To extract information about job streams on June of this year for all workstations, and direct the output to file `r11out`, run the following command:

```
r11xtr -m 06 -o r11out
```

xrxtrct

Extracts information about cross-references from the database.

Syntax

xrxtrct [**-V** | **-U**]

Arguments

-V

Displays the command version and exits.

-U

Displays command usage information and exits.

Results

The ***MAESTRO_OUTPUT_STYLE*** variable specifies the output style for long object names. With value **LONG**, full length (long) fields are used for object names. If the variable is set to anything other than **LONG**, long names are truncated to eight characters and a plus sign. For example: `A1234567+`.

The command output is written to eight files, **xdep_job**, **xdep_sched**, **xfile**, **xjob**, **xprompt**, **xresources**, **xsched**, and **xwhen**. These files are written in the current working directory. You must have write and execution rights on this directory to run the command.

Example

Examples

To extract information about all cross-references, run the following command:

```
xrxtrct
```

xdep_job file

The **xdep_job** file contains two record types. The first contains information about jobs and job streams that are dependent on a job. Each dependent job and job stream record contains the fixed length fields, with no delimiters. The fields are described in Table 117: Xdep_job output fields on page 1043.

Table 117. Xdep_job output fields

Field	Description	Length (bytes)
1	**03**	2
2	workstation name	16
3	job name	40
4	job stream name	16
5	not used	240
6	dependent job stream workstation name	16
7	dependent job stream name	16
8	dependent job workstation name	16
9	dependent job name	40
10	not used	6
11	not used	6
12	not used	8
13	end-of-record (null)	1

The second record type contains information about jobs and job streams that are dependent on an internetwork dependency. Each dependent job and job stream record contains fixed length fields, with no delimiters. The fields are described in Table 118: Xdep_job output fields (continued) on page 1043.

Table 118. Xdep_job output fields (continued)

Field	Description	Length (bytes)
1	**08**	2
2	workstation name	16
3	job name	120
4	not used	128
5	dependent job stream workstation name	16

Table 118. Xdep_job output fields (continued) (continued)

Field	Description	Length (bytes)
6	dependent job stream name	16
7	dependent job workstation name	16
8	dependent job name	40
9	not used	6
10	not used	6
11	not used	8
12	end-of-record (null)	1

xdep_sched file

The **xdep_sched** file contains information about jobs and job streams that are dependent on a job stream. Each dependent job or job stream record contains fixed length fields, with no delimiters. The fields are described in Table 119: Xdep_sched output fields on page 1044.

Table 119. Xdep_sched output fields

Field	Description	Length (bytes)
1	**02**	2
2	workstation name	16
3	job stream name	16
4	not used	248
5	dependent job stream workstation name	16
6	dependent job stream name	16
7	dependent job workstation name	16
8	dependent job name	40
9	not used	6
10	not used	6
11	not used	8
12	end-of-record (null)	1

xfile file

The **xfile** file contains information about jobs and job streams that are dependent on a file. Each record contains fixed length fields, with no delimiters. The fields are described in Table 120: Xfile output fields on page 1045.

Table 120. Xfile output fields

Field	Description	Length (bytes)
1	**07**	2
2	workstation name	16
3	file name	256
4	dependent job stream workstation name	16
5	dependent job stream name	16
6	dependent job workstation name	16
7	dependent job name	40
8	not used	6
9	not used	6
10	not used	8
11	end-of-record (null)	1

xjob file

The **xjob** file contains information about the job streams in which each job appears. Each job record contains fixed length fields, with no delimiters. The fields are described in Table 121: Xjob output fields on page 1045.

Table 121. Xjob output fields

Field	Description	Length (bytes)
1	**04**	2
2	workstation name	16
3	job name	40
4	not used	248
5	job stream workstation name	16
6	job stream name	16
7	not used	8
8	not used	8

Table 121. Xjob output fields (continued)

Field	Description	Length (bytes)
9	not used	6
10	not used	6
11	not used	8
12	end-of-record (null)	1

xprompt file

The **xprompt** file contains information about jobs and job streams that are dependent on a prompt. Each prompt record contains fixed length fields, with no delimiters. The fields are described in Table 122: Xprompts output fields on page 1046.

Table 122. Xprompts output fields

Field	Description	Length (bytes)
1	**05**	2
2	workstation name	16
3	prompt name or text	20
4	not used	236
5	dependent job stream workstation name	16
6	dependent job stream name	16
7	dependent job workstation name	16
8	dependent job name	40
9	not used	6
10	not used	6
11	not used	8
12	end-of-record (null)	1

xresource file

The **xresource** file contains information about jobs and job streams that are dependent on a resource. Each resource record contains fixed length fields, with no delimiters. The fields are described in Table 123: Xresource output fields on page 1046.

Table 123. Xresource output fields

Field	Description	Length (bytes)
1	**06**	2

Table 123. Xresource output fields (continued)

Field	Description	Length (bytes)
2	workstation name	16
3	resource name	8
4	not used	248
5	dependent job stream workstation name	16
6	dependent job stream name	16
7	dependent job workstation name	16
8	dependent job name	40
9	units allocated	6
10	not used	6
11	not used	8
12	end-of-record (null)	1

xsched file

The **xsched** file contains information about job streams. Each job stream record contains fixed length fields, with no delimiters. The fields are described in Table 124: Xsched output fields on page 1047.

Table 124. Xsched output fields

Field	Description	Length (bytes)
1	**00**	2
2	workstation name	16
3	job stream name	16
4	not used	248
5	workstation name (same as 2 above)	16
6	job stream name (same as 3 above)	16
7	not used	8
8	not used	8
9	not used	6
10	not used	6
11	not used	8

Table 124. Xsched output fields (continued)

Field	Description	Length (bytes)
12	end-of-record (null)	1

xwhen file

The **xwhen** file contains information about when job streams will run. Each job stream record contains the following fixed length fields, with no delimiters. The fields are described in Table 125: Xwhen output fields on page 1048.

Table 125. Xwhen output fields

Field	Description	Length (bytes)
1	**01**	2
2	workstation name	16
3	ON/EXCEPT name or date	8
4	except flag (*=EXCEPT)	1
5	not used	128
6	workstation name	16
7	job stream name	16
8	not used	8
9	not used	8
10	not used	6
11	offset num	6
12	offset unit	8
13	end-of-record (null)	1

Running Dynamic Workload Console reports and batch reports

You can run the following reports from the Dynamic Workload Console:

Job Run History Report

A report collecting the historical job run data during a specified time interval. It is useful to detect which jobs ended in error or were late, as well as critical and promoted jobs and the latest time within which the job can start without causing the critical job miss its deadline. It also shows which jobs missed their deadline, long duration jobs, and rerun indicators for reruns.

Job Run Statistics Report

A report collecting the job run statistics. It is useful to detect success, error rates; minimum, maximum, and average duration; late and long duration statistics.

Workstation Workload Summary Report

A report showing the workload on the specified workstations. The workload is expressed in terms of number of jobs that ran on them. It is useful for capacity planning adjustments (workload modelling and workstation tuning).

Workstation Workload Runtimes Report

A report showing job run times and duration on the specified workstations. It is useful for capacity planning adjustments (workload modelling and workstation tuning).

Planned Production Details Report

A report based on the information stored either in a trial or in a forecast plan. The information contained in these plans is retrieved from the HCL Workload Automation database. A Planned Production Details Report can be run on distributed engines (master domain manager and backup domain manager). A real production report extracted from a fault-tolerant agent might contain different information with respect to a plan extracted from a master domain manager. For example, the number of jobs and job streams is the same, but their status can change, because a job successful on the master can be in hold or ready on the agent. The update status rate is the same only on the full status agent that runs on the domain master.

Actual Production Details Report

A report based on the information stored either in the current or in an archived plan. The information contained in these plans is retrieved from the Symphony files. Actual Production Details Report can be run on distributed engines (master domain manager, backup domain manager, domain manager with connector, and fault-tolerant agent with connector).

Custom SQL Report

It enables you to create reports by running your own SQL queries. The reports will display a table with the column name as specified in the SELECT part of the SQL statement. The data for reporting is stored in a DB2 relational database and resides on the distributed side. HCL Workload Automation for Z connects to the database through the Java Database Connectivity (JDBC) interface. A JDBC driver type 4 is used to connect to the remote DB2 for LUW version 8.2, or later.

General audit report

The report provides information about objects that have been modified in the database. More specifically, it details who made the change, on which objects, and when.

For more information, see the information about keeping track of database changes using audit reports in the *Administration Guide*.

Details report

The report provides further details about the changes implemented. It specifies who made the change, on which objects, when, and what has been changed. More specifically it shows the object definition before and after the change.

For more information, see the information about keeping track of database changes using audit reports in the *Administration Guide*.

Personalized reports created with Business Intelligence and Reporting Tools (BIRT)

Use this function to upload your custom reports created with BIRT and import them as task.

For more information about creating reports from the Dynamic Workload Console, see the Dynamic Workload Console User's Guide, section about Reporting.

Some of these reports are also available as *batch reports* and can be run from a command line. For more information on how to run batch reports, see Running batch reports from the command line interface on page 1057.

Depending on the interface from where you run the report or the operating system of the engine the following output formats are available:

Table 126. Supported report output formats

Name of the report	Output formats supported by the Dynamic Workload Console	Output formats supported by batch reports
Job Run Statistics Report	HTML, CSV, PDF Table and chart formats	HTML, CSV, PDF Table and chart formats
Job Run History Report	HTML, CSV, PDF Only table format	HTML, CSV, PDF Only table format
Workstation Workload Summary Report	HTML, CSV, PDF Table and chart formats	HTML, CSV, PDF Table and chart formats
Workstation Workload Runtimes Report	HTML, CSV, PDF Table and chart formats	HTML, CSV, PDF Table and chart formats
Actual Production Details Report	XML, CSV Only table format	N/A
Planned Production Details Report	XML, CSV Only table format	N/A

Table 126. Supported report output formats (continued)

Name of the report	Output formats supported by the Dynamic Workload Console	Output formats supported by batch reports
Custom SQL Report	HTML, CSV, PDF Only table format	HTML, CSV, PDF Only table format
Auditing general report	N/A	HTML, CSV, PDF Only table format
Auditing details report	N/A	HTML, CSV, PDF Only table format

You must have the appropriate security file authorizations for report objects to run these reports (granted by default to the *tws_user* on fresh installations). See the *Administration Guide* for security file information.

See also the *Administration Guide* to learn how to configure the Dynamic Workload Console to view reports.

Historical reports

The following table summarizes the historical reports in terms of their:

- Functionality
- Selection criteria
- Output content options

Table 127. Summary of historical reports

Report name	Description	Selection criteria	Output content options
Job run history	Corresponds to **Report 07**. Collects historical job execution data during a time interval. Helps you find: • Jobs ended in error • Late jobs • Missed deadlines • Long duration	• Job name, job stream name, workstation name, and workstation name (job stream). Each field can be specified using a wildcard. • Status (Success, Error, Unknown) • Delay indicators	You can select from the following: • Actual start time • Estimated duration • Actual duration • Job number • Started late (delay) • Ended late (delay)

Table 127. Summary of historical reports (continued)

Report name	Description	Selection criteria	Output content options
	• Rerun indicators for reruns • Other historical information.	• Job execution interval • Include/Exclude rerun iterations	• Status • Critical latest start • Critical • Promoted • Long duration • Job definition name • CPU consumption (not available on Windows workstations) • Logon user • Rerun type • Iteration number • Return code The output is in table view.
Job run statistics	Corresponds to **Report 01**. Collects job execution statistics. Helps you find: • Success/error rates • Minimum and maximum elapsed and CPU times	• Job name, workstation name, and user login. Each field can be specified using a wildcard. • Percentage of jobs in Success, Error, Started	You can select from the following: • Job details: ◦ Logon user ◦ Job creator ◦ Description

Table 127. Summary of historical reports (continued)

Report name	Description	Selection criteria	Output content options
	• Average duration • Late and long duration statistics	late, Ended late, and Long duration	◦ Script

Table 127. Summary of historical reports (continued)

Report name	Description	Selection criteria	Output content options
	Note: The report does not include jobs that were submitted using an alias name.	• Total runs and total reruns	◦ Recovery information • Job statistics: ◦ Total runs (divided in Successful and Error) ◦ Total of runtime exceptions (Started late, Ended late, Long duration) ◦ Minimum, maximum and average duration and CPU times (for successful runs only) ◦ CPU consumption (not available on Windows workstations) • Report format: ◦ Charts view ◦ Table view ◦ Include table of contents by job or by workstation
Workstation work	Provides data on the workload in terms of the number of jobs that have run on each workstation. Helps making the necessary	• Workstation names. Each field can be specified using a wildcard.	You can select from the following:

Table 127. Summary of historical reports (continued)

Report name	Description	Selection criteria	Output content options
load summary	capacity planning adjustments (workload modeling, and workstation tuning).	• Date ranges or specific days for workload filtering. • Relative time intervals (allows to reuse the same report task for running each day and getting the report of the production of the day before)	• Workstation information granularity arranged by: ◦ Hour ◦ Day ◦ Production day • Information aggregation options: ◦ Provide cumulative and aggregated workstation summary information for all or a subset of workstations • Report format: ◦ Charts view ◦ Table view ◦ Include table of contents by date or by workstation
Workstation workload runtimes	Corresponds to **Report 08**. Provides data on the job runs (time and duration) on the workstations. Helps making the necessary capacity planning adjustments (workload modeling, and workstation tuning).	• Job and workstation names. Each field can be specified using a wildcard. • Workload execution period • Daily time intervals	You can select from the following: • Information grouped by: ◦ Workstation ◦ Run date

Table 127. Summary of historical reports (continued)

Report name	Description	Selection criteria	Output content options
			Can be ordered by rerun iteration • Production day • Job information: ◦ Actual duration ◦ Status ◦ Rerun iteration ◦ Job definition name • Report format: ◦ Charts view ◦ Table view
Custom SQL	A wizard helps you define your custom SQL query (only on the database views which you are authorized to access).	The criteria specified in the custom SQL query.	The resulting report has a table with the column name as specified in the SELECT part of the SQL statement.

Production reports

The following table summarizes the production reports in terms of their:

- Functionality
- Selection criteria
- Output content options

Table 128.

Report name	Description	Selection criteria	Output content options
Actual production details	Corresponds to **Report 10B**. Provides data on current and archived plans.	• Job name • Workstation name (job) • Job stream name • Workstation name (job stream)	You can select from the following: • Report format: ◦ Flat ◦ CSV ◦ Microsoft Project • Include: ◦ First level predecessor ◦ Job log
Planned production details	Corresponds to **Report 9B**. Provides data on trial and forecast plans.	• Job name • Workstation name (job) • Job stream name • Workstation name (job stream)	You can select from the following: • Report format: ◦ Flat ◦ CSV ◦ Microsoft Project • Include: ◦ First level predecessor ◦ Job log

Running batch reports from the command line interface

This section describes how you can run from the command line the reports listed in Historical reports on page 1051.

Using a command-line interface, you can schedule these reports to run on a timely basis.

A sample business scenario

To avoid unexpected slowing down in the workload processing, the analyst of a big company needs weekly reports collecting historical information about the processed workload to determine and analyze any workload peaks that might occur.

To satisfy this request, the TWSWEBUIDeveloper creates *Workload Workstation Summary Reports* (WWS) and *Workload Workstation Runtimes Reports* (WWR).

To accomplish his task, he runs the following steps:

1. He customizes the property files related to the Workload Workstation Summary and Workload Workstation Runtimes reports, specifying the format and content of the report output.
2. He schedules jobs to obtain WWS and WWR reports:
    - The first job generates a WWS report to be saved locally.
    - The second job runs a WWR report overnight on expected workload peaks time frames. The report output is sent using an mail to the analyst. The information collected are used to optimize the workload balance on the systems.
3. He adds the two jobs to a job stream scheduled to run weekly and generates the plan.

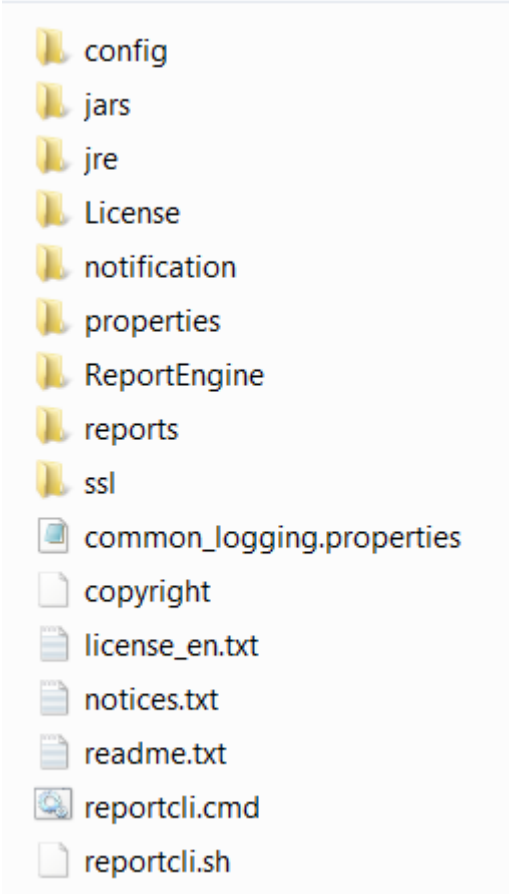
Setting up for command line batch reporting

About this task

Before running batch reports you must run a few setup steps:

1. The software needed to run batch reports is contained in a package named `TWSBatchReportCli`, in the `TWSBatchReportCli` directory. If you plan to run batch reports from within a scheduled job, extract the package file on one of the operating systems listed in the System Requirements Document at Dynamic Workload Console Detailed System Requirements.

   After you extract the package, you will have the following file structure:



Because the native UNIX™ tar utility does not support long file names, if you are extracting the files on AIX® systems, ensure that the latest GNU version of tar (gtar) is installed to extract the files successfully.

Note:

a. Make sure you run the following commands in the directory where you extracted the files:

On UNIX™

```
chmod -R +x *
chown -R username *
```

On Windows™

Ensure HCL Workload Automation is installed.

```
setown -u username *
```

where *username* is the HCL Workload Automation user that will run the reports.

b. If you plan to schedule jobs that run batch reports, the system where you extract the package must be accessible as network file system from a fault-tolerant agent defined in the local scheduling environment.

2. If you use an Oracle database, download the JDBC drivers required by your Oracle server version.

3. Copy the JDBC drivers in the *report_cli_installation_dir*\jars directory and in *report_cli_installation_dir*\ReportEngine\plugins\org.eclipse.birt.report.data.oda.jdbc_4.8.0.v201806261756\drivers directory. The report CLI automatically discovers the two jar files.
4. Configure the template file .\config\common.properties specifying the following information:
    a. If you use an Oracle database, connect to the database where the historical data are stored as follows:
        i. Retrieve the location of the Oracle JDBC drivers. This information is stored in the **com.ibm.tws.webui.oracleJdbcURL** property in the file *TWA_home*TWSDATA/usr/servers/engineServer/resources/properties/TWSConfig.properties on the workstation where the master domain manager is installed.

            For more information about this file, see the section about configuring for an Oracle database in *HCL Workload Automation: Administration Guide*.

        ii. Specify the location of the Oracle JDBC drivers in the **PARAM_DataSourceUrl** property in the common.properties file.

        No customization is required if you use DB2.
    b. Set the date and time format, including the time zone. The file .\config\timezone.txt contains a list of time zones supported by HCL Workload Automation and the information on how to set them. The time zone names are case sensitive.
    c. Make available the report output on the URL specified in **ContextRootUrl** field. This is an example of configuration settings:

```
########################################################################
 # HTTP Server information
 ########################################################################

 #Specify the context root where the report will be available
 #To leverage this possibility it needs to specify in the report output dir
 #the directory that is referred by your HTTP Server with this contect root

 ContextRootUrl=http://myserver/reportoutput
```

        In this case make sure that the *output_report_dir* specified when running the batch reports command points to the same directory specified in the **ContextRootUrl** field.
    d. Send the report output using a mail. This is an example of configuration settings:

```
########################################################################
 # Email Server configuration
 ########################################################################
 PARAM_SendReportByEmail=true

 #SMTP server
 mail.smtp.host=myhost.mydomain.com
 #IMAP provider
 mail.imap.socketFactory.fallback=false
 mail.imap.port=993
 mail.imap.socketFactory.port=993
 #POP3 provider
 mail.pop3.socketFactory.fallback=false
 mail.pop3.port=995
```

```
mail.pop3.socketFactory.port=995

######################################################################
# Email properties
######################################################################
PARAM_EmailFrom=user1@your_company.com
PARAM_EmailTo=user2@your_company.com,user3@your_company.com
PARAM_EmailCC=user4@your_company.com
PARAM_EmailBCC=user5@your_company.com
PARAM_EmailSubject=Test send report by email
PARAM_EmailBody=This is the report attached
```

An explanation of all the customizable fields is contained in the template file.

Note: If you plan to run Workstation Workload Runtime reports ensure that the file system where database is installed has enough free space. if a shortage of disk space occurs an SQL exception like the following is triggered:

```
DB2 SQL error: SQLCODE: -968, SQLSTATE: 57011
```

Running batch reports

The `\reports\templates` directory contains a sample template file for each type of report.

Before running any of these reports make sure you customize the corresponding template file.

In that file, named *`report_name`*`.properties`, you can specify:

- The information to display in the report header.
- How to filter the information to display the expected result.
- The format and content of the report output.

For more information about the specific settings see the explanation provided in the template file beside each field.

If you are using DBCS characters to specify the parameters in the template .properties files, ensure you save the file in UTF-8 encoding.

After you set up the environment as it is described in Setting up for command line batch reporting on page 1058, and you configured the report template file, use the following syntax to run the report:

reportcli -p *report_name.property*
[**-o** *output_report_dir*]
[**-r** *report_output_name*]
[**-k** key=*value*]
[**-k** key=*value*]
.....

where:

-p *report_name.property*

Specifies the path name to the report template file.

-o *output_report_dir*

Specifies the output directory for the report output.

-r *report_output_name*

Specifies the name of the report output.

-k key=*value*

Specifies the value of a settings. This value override the corresponding value, if defined, in the `common.properties` file or in the *`report_name`*`.properties` file.

Example

Examples

1. In this example the `reportcli.cmd` is run with the default parameter and produces `jrh1` report:

   ```
   reportcli.cmd -p D:\ReportCLI\TWSReportCli\reports\templates\jrh.properties
   -r jrh1
   ```

2. In this example the `reportcli.cmd` is run using the `-k` parameter to override the values set for **PARAM_DateFormat** in the `.\config\common.properties` file produces `jrh1` report:

   ```
   reportcli.cmd -p D:\ReportCLI\TWSReportCli\reports\templates\jrh.properties
   -r jrh2 -k PARAM_DateFormat=short
   ```

3. In this example the `reportcli.cmd` is run using the `-k` parameter to override the format specified for the report output in the PROPERTIES file produces `jrh1` report:

   ```
   ./reportcli.sh -p /TWSReportCli/REPCLI/reports/templates/wwr.properties
   -r wwr3 -k REPORT_OUTPUT_FORMAT=html -k OutputView=charts
   ```

4. Do the following if you want to run a Custom SQL report and make available the output of the report at the following URL as `http://myserver/reportoutput/report1.html`:
   a. Configure the `ContextRootUrl` parameter in the `common.properties` files as follows:

      ```
      ######################################################################
       # HTTP Server information
       ######################################################################

       #Specify the context root where the report will be available
       #To leverage this possibility it needs to specify in the report output dir
       #the directory that is referred by your HTTP Server with this contect root

       ContextRootUrl=http://myserver/reportoutput
      ```

   b. When you run a batch reports command specify as *output_report_dir* a directory that points to the same HTTP directory specified in the `ContextRootUrl`. For example, if you mapped locally the `http://myserver/` as `R:` driver, you can run the following command:

      ```
      reportclibat
         -p REPORT_CLI_DIR\reports\TWS\historical\templates\sql.properties
         -r report1
         -o R:\reportoutput
      ```

   c. As a confirmation for the successful run of the report, the following message is displayed:

```
AWSBRC0106I Report available on: http://myserver/reportoutput/report1.html
```

This URL shows where the report output is available.

Note: If the report is run through an HCL Workload Automation job, the output of the command is displayed in the job output.

Logs and traces for batch reports

The file `./common_logging.properties` contains the parameters you can use to configure tracing and logging.

The file contains the following settings:

```
logFileName=reportcli.log
traceFileName=trace.log
trace=off
birt_trace=off
```

where:

logFileName

Specifies the name of the file containing generic information, warning about potential problems, and information about errors. This file is store under `./log`.

traceFileName

Specifies the name of the file containing traces. If you set `trace=on` the trace file is store under `./log`.

trace

Specifies whether to enable or not traces. Enable the traces by setting `trace=on` if you want to investigate further about an error,

birt_trace

Specifies whether to enable or not traces to diagnose errors in BIRT engine. If you set `birt_trace=on` a file containing the trace and named `ReportEngine_aaaa_mm_dd_hh_mm_ss.log` is stored in the `/ReportEngine/logs` folder

Chapter 21. Managing time zones

HCL Workload Automation supports different time zones. If you enable time zones you can manage your workload across a multiple time zone environment.

Both the 3-character and the long time zone names are supported, but it is suggested that you use the long names if they exist. If you are scheduling in MST or EST time zones, you are required to use the long names, for example "America/New_York" for EST. This is because those time zones no longer observed daylight savings time (DST) rules and the product incorrectly schedules jobs or job streams during the DST time frame with one hour offset for time dependencies.

The 3-character notation is supported for compatibility with earlier versions of HCL Workload Automation.

The variable length notation format is area/city, for example Europe/Paris as equivalent to ECT (European Central Time).

The chapter is made up by the following sections:

- Enabling time zone management on page 1064
- How HCL Workload Automation manages time zones on page 1065
- Moving to daylight saving time on on page 1067
- Moving to daylight saving time off on page 1068
- General rules on page 1068

Enabling time zone management

About this task

You can enable or disable the management of time zones by modifying the setting assigned to the global option **enTimeZone** on the master domain manager using the **optman** command line. The setting takes effect after the next **JnextPlan** is run. These are the available settings:

no

Disable time zone management. This means that the values assigned to all **timezone** keywords in the definitions are ignored. All the `at`, `until`, and `deadline` time restrictions are managed individually by each fault-tolerant agent, including the master and the domain managers, thus ignoring the time zone of the agent scheduling the job or job stream. As a consequence, when different time zones are involved:

- For jobs, incorrect information is displayed about these time dependencies when looked at from an agent other than the job owner. This has no impact however on the scheduling process of the job.
- For job streams, the impact is that each agent processes the time dependencies by its own time zone, and therefore at different times, causing jobs of the same job stream, but defined on a different agent, to run at a different time.

yes

Enable time zone management. This means that the values assigned to the **timezone** settings are used to calculate the time when the jobs and job streams run on the target workstations.

The **enTimeZone** option is now deprecated and forced to `yes`.

For more details about using the optman command line to manage global options on the master domain manager, see the section about setting global options in *Administration Guide*.

How HCL Workload Automation manages time zones

When the time zone is enabled, you can use time zone settings in workstation, job, and job stream definitions.

While performing plan management activities, HCL Workload Automation converts the value set for the time zones into object definitions. The conversions are applied in this order:

1. When the job stream instances are added to the preproduction plan, the time zone set in the job stream definitions is converted into the GMT time zone and then the external follows dependencies are resolved.
2. When the production plan is created or extended, the job stream instances are assigned to workstations where the instance is scheduled to run and the time zone is converted from GMT into the time zone set in the target workstation definition.

This is why if you use the **conman showsched** or **conman showjobs** commands to see the information about scheduled jobs and job streams you see the time zone values expressed using the time zone set on the workstation where the job or job stream is planned to run.

Based on the setup of the *enLegacyStartOfDayEvaluation* global option, you can decide how the product manages time zones while processing, and precisely:

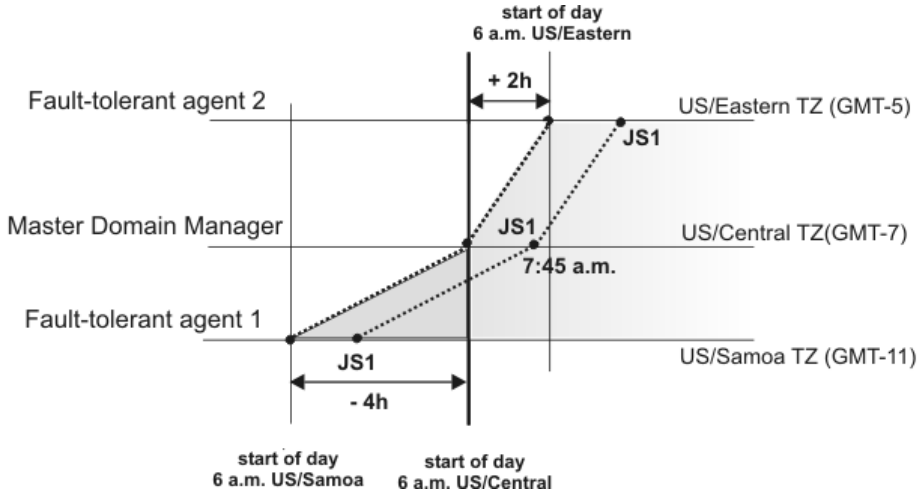
If you set the value of *enLegacyStartOfDayEvaluation* to *no*

The value assigned to the *startOfDay* option on the master domain manager is not converted into the local time zone set on each workstation across the network. This means that if the *startOfDay* option is set to 0600 on

the master domain manager, it is 0600 in the local time zone set on each workstation in the network. This also means that the processing day begins at the same hour, but not at the same moment, on all workstations.

Figure 25: Example when start of day conversion is not applied on page 1066 shows you how the start of day, set to 0600 on the master domain manager, is applied to the different time zones on the two fault-tolerant agents. The same time conversion is applied to the three instances of job stream **JS1** scheduled to run on the three machines and containing an **at** time dependency at `0745 US/Central` time zone. The time frame that identifies the new processing day is greyed out in Figure 25: Example when start of day conversion is not applied on page 1066.

Figure 25. Example when start of day conversion is not applied

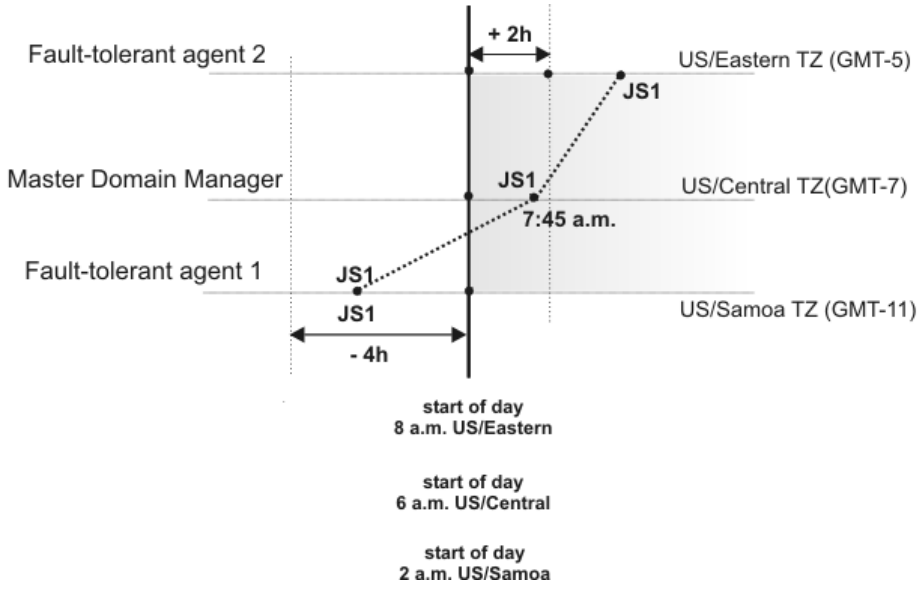


If you set the value of *enLegacyStartOfDayEvaluation* to *yes*

The value assigned to the *startOfDay* option on the master domain manager is converted into the local time zone set on each workstation across the network. This means that if the *startOfDay* option is set to 0600 on the master domain manager, it is converted on each workstation into the corresponding time according to the local time zone set on that workstation. This also means that the scheduling day begins at the same moment, but not necessarily at the same hour, on all workstations in the network.

Figure 26: Example when start of day conversion is applied on page 1067 shows you how the start of day, set to 0600 on the master domain manager, is applied to the different time zones on the two fault-tolerant agents. It also shows how the timing of the three instances of job stream **JS1** scheduled to run on the three machines and containing an **at** time dependency at `0745 US/Central` time zone is not modified because of the *startOfDay* conversion. The time frame that identifies the new processing day is greyed out in Figure 26: Example when start of day conversion is applied on page 1067.

Figure 26. Example when start of day conversion is applied



Note: Starting from version 8.3 there is no linking between the time set for the *startOfDay* and the moment when **JnextPlan** is run. **JnextPlan** can be run at any time and the *startOfDay* indicates only the moment when the new processing day starts.

By default the *enLegacyStartOfDayEvaluation* global option is set to **no**.

For more details on how to use the **optman** command line to manage global options on the master domain manager, refer to the *HCL Workload Automation Administration Guide*.

Moving to daylight saving time on

HCL Workload Automation manages the moving to daylight saving time (DST) when generating the production plan. This means that the date and time to run assigned to jobs and job streams in the plan is already converted into the corresponding date and time with DST on.

The following example explains how HCL Workload Automation applies the time conversion when **JnextPlan** is run to generate or extend the production plan while the time moves to DST.

If DST is turned on at 3:00 p.m., all job streams scheduled to start between 2:00 and 2:59 are set to start one hour later. For example a job defined to run AT 0230 in the morning, will be scheduled to run at 03:30 a.m.

On the day when DST starts, if the start of day happens in coincidence with the missing hour (for example from 00:00 to 00:59 for America/Sao_Paulo or from 02:00 to 02:59 for America/Chicago) both the day before DST starts and the day after DST starts, the plan is extended for 24 hours, postponing the production plan start time one hour later than the expected one. To prevent this problem, manually modify the first plan extension after the DST entering to extend for 23 hours instead of 24. This correction does not apply to forecast plan, trial plan, and plans generated using the JnextPlan script with the **-to** option.

Moving to daylight saving time off

Moving to daylight saving time (DST) off, the clock time is set to one hour earlier with respect to the DST time. To maintain consistency with production planning criteria, HCL Workload Automation ensures that the job stream instances planned to run during the hour before the time shift backward are run only one time. Because the time conversion is applied when generating or extending the production plan, the date and time to run assigned to jobs and job streams in the plan is already converted into the corresponding date and time with DST off.

If a job stream or a job run on a timezone where the DST time turns off, that is the clock is put one hour back, and if you define a time dependency for such job streams or jobs in relation to another timezone, it might happen that this time dependency occurs during the second, repeated time interval. In this case the time dependency would be resolved during the first time interval.

HCL Workload Automation recognizes that the time dependency occurs on the second, repeated time interval and resolves it accordingly.

General rules

When the time zone is enabled in the HCL Workload Automation environment, regardless of which value is set for the *enLegacyStartOfDayEvaluation* option, some general rules are applied. These rules are now described divided by topic:

Identifying default time zone settings for jobs and job streams:

Within a job stream definition you can set a time zone for the entire job stream and for the jobs contained in the job stream. These time zones can differ from each other. To manage all possible time zone settings the time zone conversion is made respecting the following criteria:

- If a time zone is not set for a job within a job stream, then that job inherits the time zone set on the workstation where the job is planned to run.
- If a time zone is not set for a job stream, then the time zone set is the one set on the workstation where the job stream is planned to run.
- If none of the mentioned time zones is set, then the time zone used is the one set on the master domain manager.

Choosing the correct time zone for the workstations:

To avoid inconsistencies, before enabling the time zone management feature across the HCL Workload Automation network, make sure that, if a time zone is set in the workstation definition, it is the same as the time zone set on the system where the workstation is installed.

Default time zone setting for the master domain manager:

If a time zone is not set in the master domain manager definition, it inherits the time zone set on the system where the master domain manager is installed. To see which time zone is set on the master domain manager you can run the following command:

```
conman showcpu;info
```

Using the time zone on extended agents:

Extended agents inherit the time zone of the master domain manager.

Displaying time zone setting in production for an AT time dependency:

If you use **conman** commands **sj** or **ss** to display a job or a job stream having an **at** time dependency with a time zone set, the time specified for the **at** dependency is displayed applying the time zone defined on the workstation where the job or job stream is defined to run.

Applying an offset to a time zone when scheduling a job stream:

If you submit in production a job stream specifying an **at** dependency with an offset of +n days, then HCL Workload Automation first adds the offset to the date and then converts the time zone set in the **at** dependency. This is important especially when referring to the time when daylight saving time moving occurs.

As a best practice, if you enable time zone management, set a time zone on each workstation of your HCL Workload Automation network.

Chapter 22. Integrating with SAP and PeopleSoft

Integrating with SAP and PeopleSoft

Access methods are used to extend the job scheduling functions of HCL Workload Automation to other systems and applications. They run on:

Extended agents (applies only to HCL Workload Automation)

They are logical workstations related to an access method hosted by a physical HCL Workload Automation workstation (not another extended agent). More than one extended agent workstation can be hosted by the same HCL Workload Automation workstation and use the same access method. The extended agent runs on fault-tolerant agents defined using a standard HCL Workload Automation workstation definition, which gives the extended agent a name and identifies the access method. The access method is a program that is run by the hosting workstation whenever HCL Workload Automation submits a job to an external system.

Jobs are defined for an extended agent in the same manner as for other HCL Workload Automation workstations, except that job attributes are dictated by the external system or application.

Information about job running execution is sent to HCL Workload Automation from an extended agent using the job `stdlist` file. A method options file can specify alternate logins to launch jobs and check *opens* file dependencies. For more information, see the *User's Guide and Reference*.

A physical workstation can host a maximum of 255 extended agents.

dynamic agents and HCL Workload Automation agents (z-centric)

They communicate with external systems to start the job and return the status of the job. To run access methods on external applications using dynamic agents, you define a job of type **access method**.

Access methods are available on the following systems and applications.

- SAP R/3
- z/OS
- Custom methods
- unixssh
- unixrsh

The UNIX™ access methods included with HCL Workload Automation, are described in the related section in *Administration Guide*.

If you are working with dynamic agents, for information about defining HCL Workload Automation workstations, see the section that explains how to define workstations in the database in *User's Guide and Reference*. For information about writing access methods, see the section about the access method interface in *User's Guide and Reference*.

Access method interface

The interface between HCL Workload Automation and an access method consists of information passed to the method on the command line, and of messages returned to HCL Workload Automation in **stdout**.

Method command line syntax

The HCL Workload Automation host runs an access method using the following command line syntax:

methodname **-t** *task options* **--** *taskstring*

where:

methodname

Specifies the file name of the access method. All access methods must be stored in the directory: `TWS_home/methods`

-t *task*

Specifies the task to be performed, where *task* is one of the following:

LJ

Launches a job.

MJ

Manages a previously launched job. Use this option to resynchronize if a prior **LJ** task ended unexpectedly.

CF

Extended agents only. Checks the availability of a file. Use this option to check file `opens` dependencies.

GS

Extended agents only. Gets the status of a job. Use this option to check job `follows` dependencies.

options

Specifies the options associated with the task. See Task options on page 1071 for more information.

taskstring

A string of up to 255 characters associated with the task. See Task options on page 1071.

Task options

The task options are listed in Table 129: Method command task options on page 1072. An X means that the option is valid for the task.

Table 129. Method command task options

Task	-c	-n	-p	-r	-s	-d	-l	-o	-j	-q	-w	-S	Task String
LJ	X	X	X	X	X	X	X	X	X			X	*ljstring*
MJ	X	X	X	X	X	X	X	X	X				*mjstring*
CF	X	X	X							X			*cfstring*
GS	X	X	X	X		X					X		*gsstring*

-c *xagent,host,master*

Specifies the names of the agent, the host, and the master domain manager separated by commas.

-n *nodename*

Specifies the node name of the computer associated with the agent, if any. This is defined in the extended agent's workstation definition **Node** field.

-p *portnumber*

Specifies the TCP/IP port number associated with the agent, if any. This is defined in the agent workstation definition **TCP Address** field.

-r *currentrun,specificrun*

Specifies the current run number of HCL Workload Automation and the specific run number associated with the job separated by a comma. The current and specific run numbers might be different if the job was carried forward from an earlier run.

-s *jstream*

Specifies the name of the job's job stream.

-d *scheddate,epoch*

Specifies the job stream date (*yymmdd*) and the epoch equivalent, separated by a comma.

-l *user*

Specifies the job's user name. This is defined in the job definition **Logon** field.

-o *stdlist*

Specifies the full path name of the job's standard list file. Any output from the job must be written to this file.

-j *jobname,id*

Specifies the job's name and the unique identifier assigned by HCL Workload Automation, separated by a comma. The name is defined in the job definition **Job Name** field.

-q *qualifier*

Specifies the qualifier to be used in a test command issued by the method against the file.

-w timeout

Specifies the amount of time, in seconds, that HCL Workload Automation waits to get a reply on an external job before sending a SIGTERM signal to the access method. The default is 300.

-S new name

Specifies that the job is rerun using this name in place of the original job name. Within a job script, you can use the `jobinfo` command to return the job name and run the script differently for each iteration.

-- ljstring

Used with the **LJ** task. The string from the **Script File** or **Command** field of the job definition.

-- mjstring

Used with the **MJ** task. The information provided to the HCL Workload Automation by the method in a message indicating a job state change **%CJ** (for additional details on messages indicating job state change, see Method response messages on page 1073) following to an **LJ** task. Usually, this identifies the job that was launched. For example, a UNIX® method can provide the process identification (PID) of the job it launched, which is then sent by the HCL Workload Automation as part of an **MJ** task.

-- cfstring

Used with the **CF** task. For a file `opens` dependency, the string from the **Opens Files** field of the job stream definition.

-- gsstring

Used with the **GS** task. Specifies the job whose status is checked. The format is as follows:

followsjob[,*jobid*]

where:

followsjob

The string from the **Follows Sched/Job** list of the job stream definition.

jobid

An optional job identifier received by HCL Workload Automation in a **%CJ** response to a previous **GS** task.

Method response messages

Methods return information to HCL Workload Automation in messages written to **stdout**. Each line starting with a percent sign (%) and ending with a new line is interpreted as a message. The messages have the following format:

%CJ *state* [*mjstring* | *jobid*]

%JS [*cputime*]

%RC *rc*

%UT [*errormessage*]

where:

CJ

Changes the job state.

state

The state to which the job is changed. All HCL Workload Automation job states are valid except HOLD and READY. For the **GS** task, the following states are also valid:

ERROR

An error occurred.

EXTRN

Status is unknown.

mjstring

A string of up to 255 characters that HCL Workload Automation will include in any **MJ** task associated with the job.

jobid

A string of up to 64 characters that HCL Workload Automation will include in any **GS** task associated with the job.

JS [*cputime*]

Indicates successful completion of a job and provides its elapsed run time in seconds.

RC *rc*

rc is a number that is interpreted by HCL Workload Automation as the return code of the extended agent job. The return code is taken into account only if a return code condition was specified in the definition of the extended agent job. Otherwise, it is ignored and the successful completion of the extended agent job is indicated by the presence of message **%JS** *[cputime]*. Likewise, if the method does not send the **%RC** message, then the successful completion of the extended agent job is indicated by the presence of message **%JS** *[cputime]*.

UT [*errormessage*]

Indicates that the requested task is not supported by the method. Displays a string of up to 255 characters that HCL Workload Automation will include in its error message.

Method options file

For extended, agents, and HCL Workload Automation Agent you can use a method options file to specify login information and other options.

An options file is a text file located in the methods directory of the HCL Workload Automation installation, containing a set of options to customize the behavior of the access method. The options must be written one per line and have the following format (with no spaces included):

```
option=value
```

All access methods use two types of options files:

Extended agents

Global options file

A common configuration file created by default for each access method installed, whose settings apply to all the extended agent workstations defined for that method. When the global options file is created, it contains only the **LJuser** option, which represents the operating system user ID used to launch the access method. You can customize the global options file by adding the options appropriate to the access method.

Local options file

A configuration file that is specific to each extended agent workstation within a particular installation of an access method. The name of this file is `XANAME_accessmethod.opts`, where:

XANAME

Is the name of the extended agent workstation. The value for *XANAME* must be written in uppercase alphanumeric characters. Double-byte character set (DBCS), Single-byte character set (SBCS), and Bidirectional text are not supported.

accessmethod

Is the name of the access method.

If you do not create a local options file, the global options file is used. Every extended agent workstation, except for z/OS, must have a local options file with its own configuration options.

For example, if the installation of the access method includes two extended agent workstations, `CPU1` and `CPU2`, the names of the local options files are respectively `CPU1_accessmethod.opts` and `CPU2_accessmethod.opts`.

HCL Workload Automation reads the options file, if it exists, before running an access method. For extended agents, if the options file is modified after HCL Workload Automation is started, the changes take effect only when it is stopped and restarted.

HCL Workload Automation Agents and agents

Global options file

A common configuration file created by default for each access method installed, whose settings apply to all the agent workstations defined for that method. When the global options file is created, it contains only the **LJuser** option, which represents the operating system user ID used to run the access method. You can customize the global options file by adding the options appropriate to the access method.

The name of the global options file is `accessmethod.opts`, where access method is the name of the method you are creating.

Local options file

A configuration file that is specific to each access method. The name of this file is `optionsfile_accessmethod.opts`,

In a distributed environment:

- If you are defining a job to run the access method by using the Dynamic Workload Console it is the options file you specified in the **Create new > Job definition > ERP > Access Method** XA Task tab.
- If you are defining the access method by using **composer** it is the options file you specified in the **target** attribute of the job definition.

If you do not create a local options file, the global options file is used.

In a z/OS environment:

- If you are defining a job to run the access method by using the Dynamic Workload Console it is the options file you specified in the **Create new > ERP > Access Method** XA Task tab.
- If you are defining the access method by using the **JOBREC** statement it is the name of the workstation where the access method runs.

If you do not create a local options file, the global options file is used.

If you do not specify an option in the *options_file_accessmethod*.opts file the product uses the value specified for that option in the global option file. If you do not specify them neither in the *options_file_accessmethod*.opts file nor in the global option file the product issues an error message.

The options file must have the same path name as its access method, with an `.opts` file extension. For example, the Windows® path name of an options file for a method named `netmeth` is

```
TWS_home\methods\netmeth.opts
```

HCL Workload Automation reads the options file, if it exists, before running an access method.

The options recognized by HCL Workload Automation are as follows:

LJuser=*username*

Specifies the login to use for the **LJ** and **MJ** tasks. The default is the login from the job definition. See Launch job task (LJ) on page 1077 and Manage job task (MJ) on page 1078.

CFuser=*username*

Extended agents only. Specifies the login to use for the **CF** task. The default for UNIX® is **root**, and for Windows® is the user name of the account in which the product was installed. See the topic about check file tasks (CF) extended agents in the *HCL Workload Automation: User's Guide and Reference*.

GSuser=*username*

Specifies the login to use for the **GS** tasks. The default for UNIX® is **root**, and for Windows® is the user name of the account with which HCL Workload Automation was installed. See Get status task (GS) extended agents only on page 1079.

GStimeout=*seconds*

Specifies the amount of time, in seconds, HCL Workload Automation waits for a response before killing the access method. The default is **300** seconds.

nodename=*node_name*

Specifies the host name or IP address if required by the method you are defining. For the **unixssh** access method, the host name or IP address to connect to the remote engine.

PortNumber=*port_number*

Specifies the port number if required by the method you are defining. For the **unixssh** access method, the port to connect to the remote engine.

For HCL Workload Automation Agents and agents, you can specify the node name and port number also in the `JobManager.ini` file.

If you do not specify them in the *options_file_accessmethod*`.opts` file the product uses the value specified in the global option file. If you do not specify them neither in the *options_file_accessmethod*`.opts` file nor in the global option file the product uses the value specified in the *option_file* stanza of the `JobManager.ini` file.

Note: If the extended agent host is a Windows® computer, these users must be defined as HCL Workload Automation user objects.

Running methods

The following subsections describe the interchange between HCL Workload Automation and an access method.

Launch job task (LJ)

About this task

The **LJ** task instructs the extended agent method to launch a job on an external system or application. Before running the method, HCL Workload Automation establishes a run environment. The **LJuser** parameter is read from the method options file to determine the user account with which to run the method. If the parameter is not present or the options file does not exist, the user account specified in the **Logon** field of the job's definition is used. In addition, the following environment variables are set:

HOME

The login user's home directory.

LOGNAME

The login user's name.

PATH

For UNIX®, it is set to`/bin:/usr/bin`. For Windows®, it is set to`%SYSTEM%\SYSTEM32`.

TWS_PROMOTED_JOB

Set to `YES`, when the job (a mission-critical job or one of its predecessors) is promoted.

TZ

The time zone.

If the method cannot be run, the job is placed in the FAIL state.

Once a method is running, it writes messages to its **stdout** that indicate the state of the job on the external system. The messages are summarized in Table 130: Launch job task (LJ) messages on page 1078.

Table 130. Launch job task (LJ) messages

Task	Method Response	HCL Workload Automation Action
LJ and **MJ**	**%CJ** *state* [*mjstring*]	Sets job state to *state*. Includes *mjstring* in any subsequent **MJ** task.
	%JS [*cputime*]	Sets job state to SUCC.
	Exit code=non-zero	Sets job state to ABEND.
	%UT [*errormessage*] and Exit code=2	Sets job state to ABEND and displays *errormessage*.

A typical sequence consists of one or more **%CJ** messages indicating changes to the job state and then a **%JS** message before the method exits to indicate that the job ended successfully. If the job is unsuccessful, the method must exit without writing the **%JS** message. A method that does not support the **LJ** task, writes a **%UT** message to **stdout** and exits with an exit code of **2**.

Manage job task (MJ)

About this task

The **MJ** task is used to synchronize with a previously launched job if HCL Workload Automation determines that the **LJ** task ended unexpectedly. HCL Workload Automation sets up the environment in the same manner as for the **LJ** task and passes it the *mjstring*. See Launch job task (LJ) on page 1077 for more information.

If the method locates the specified job, it responds with the same messages as an **LJ** task. If the method is unable to locate the job, it exits with a nonzero exit code, causing HCL Workload Automation to place the job in the ABEND state.

Killing a job

About this task

While an **LJ** or **MJ** task is running, the method must trap a SIGTERM signal (signal **15**). The signal is sent when an operator issues a **kill** command from HCL Workload Automation console manager. Upon receiving the signal, the method must attempt to stop (**kill**) the job and then exit without writing a **%JS** message.

Check file task (CF) extended agents only

About this task

The **CF** task requests the extended agent method to check the availability of a file on the external system. Before running the method, HCL Workload Automation establishes a run environment. The **CFuser** parameter is read from the method options file to determine the user account with which to run the method. If the parameter is not present or the options file does not exist, on UNIX® the **root** user is used and, on Windows®, the user name of the account in which HCL Workload Automation was installed is used. If the method cannot be run, the file `opens` dependency is marked as failed, that is, the file status is set to **NO** and any dependent job or job stream is not allowed to run.

Once it is running, the method runs a test command, or the equivalent, against the file using the qualifier passed to it in the **-q** command line option. If the file test is true, the method exits with an exit code of zero. If the file test is false, the method exits with a nonzero exit code. This is summarized in Table 131: Check file task (CF) messages on page 1079.

Table 131. Check file task (CF) messages

Task	Method Response	HCL Workload Automation Action
CF	Exit code=0	Set file state to **YES**.
	Exit code=nonzero	Set file state to **NO**.
	%UT [*errormessage*] and Exit code=2	Set file state to **NO**.

A method that does not support the **CF** task writes a **%UT** message to **stdout** and exits with an exit code of **2**.

Get status task (GS) extended agents only

About this task

The **GS** task tells the extended agent's method to check the status of a job. This is necessary when another job is dependent on the successful completion of an external job. Before running the method, the **GSuser** parameter is read from the method

options file to determine the user account with which to run the method. If the parameter is not present or the options file does not exist, on UNIX® the **root** user is used, and, on Windows®, the user name of the account in which HCL Workload Automation was installed is used. If the method cannot be run, the dependent job or job stream is not allowed to run. If a *jobid* is available from a prior **GS** task, it is passed to the method.

The method checks the state of the specified job, and returns it in a **%CJ** message written to **stdout**. It then exits with an exit code of zero. At a rate set by the ***bm check status*** local option, the method is re-run with a **GS** task until one of the following job states is returned:

abend

The job ended abnormally.

succ

The job completed successfully.

cancl

The job was cancelled.

done

The job is ended, but its success or failure is not known.

fail

The job could not be run.

error

An error occurred in the method while checking job status.

extrn

The job check failed or the job status could not be determined.

Note that **GStimeout** in the method options file specifies how long HCL Workload Automation will wait for a response before killing the method. See Method options file on page 1074 for more information.

Method responses are summarized in Table 132: Get status task (GS) messages on page 1080:

Table 132. Get status task (GS) messages

Task	Method Response	HCL Workload Automation Action
GS	**%CJ** *state* [*jobid*]	Sets job state to *state* and includes *jobid* in any subsequent **GS** task.
	%UT [*errormessage*] and Exit code=2	Job state is unchanged.

A method that does not support the **GS** task writes a **%UT** message to **stdout** and exits with an exit code of **2**.

Cpuinfo command for extended agents only

The **cpuinfo** command can be used in an access method to return information from a workstation definition. See Cpuinfo command for extended agents only on page 1081 for complete command information.

Access methods

Access methods are used to extend the job scheduling functions of HCL Workload Automation to other systems and applications. They run on extended agents, dynamic agents, and HCL Workload Automation agents. They enable communication between external systems (SAP R/3) and HCL Workload Automation and launch jobs and return the status of jobs.

For information about the supported versions of the access methods, see the Supported Software report and browse to the **Data Integration** section.

PeopleTools 8.61 and later require connecting using TLS. To continue working with an unsecure connection without enabling TLS, modify the `psagent` script by adding the **TM_ALLOW_NOTLS** option and setting it to `yes`.

Installing and configuring the access methods

The access methods documented in this guide are packaged with HCL Workload Automation and are automatically installed with the product on dynamic and fault-tolerant agents.

For details about installing an HCL Workload Automation dynamic or fault-tolerant agent, see Planning and Installation.

To use any of the access methods on supported agents, you create an options file, which configures the access method and defines the workstation and the jobs that extend the scheduling capability to external systems or applications.

Setting options for the access methods

An options file is a text file located in the `methods` directory of the HCL Workload Automation installation, containing a set of options to customize the behavior of the access method. The options must be written one per line and have the following format (with no spaces included):

```
option=value
```

All access methods use two types of options files: a global options file and one or more local options files. The names of the local options files are generically referred to as `XA_Unique_ID`_accessmethod_.opts on extended agents and `DYNAMIC_AGENT_FILE`_accessmethod_.opts on dynamic agents. The file names specified for the local options files for both types of agents must respect the following rules:

- Both `XA_Unique_ID` and `DYNAMIC_AGENT_FILE` in the file name must be uppercase alphanumeric characters. See specific requirements about `XA_Unique_ID` in XA_Unique_ID on page 1085.
- Double-byte character set (DBCS), single-byte character set (SBCS), and bidirectional text are not supported. For information about acceptable values for the extended agent workstation name, See Table 133: How to complete the extended agents definition on page 1087.

Dynamic agents and HCL Workload Automation agents

Global options file

The name of the global options file is `accessmethod.opts`, which, depending on your operating system, corresponds to:

For PeopleSoft

`psagent.opts`

For SAP R/3

`r3batch.opts`

Local options file

One or more configuration files that are specific to each access method. The name of this file is *optionsfile_accessmethod*.opts and they are saved to the path *TWA_DATA_DIR*/methods.

In a distributed environment

- If you are defining a job to run the access method by using the Dynamic Workload Console, it is the options file you specify in the **New > Job definition > ERP > Access Method** XA Task tab.
- If you are defining the SAP R/3 job to run the access method by using the Dynamic Workload Console, it is the options file you specify in the **New > Job definition > ERP > SAP Job on Dynamic Workstations** XA Task tab.
- If you are defining the job to run the access method by using **composer**, it is the options file you specify in the **target** attribute of the job definition.

If you do not create a local options file, the global options file is used.

If you do not specify an option in the *optionsfile_accessmethod*.opts file, while the access method is running, the product uses the values specified for that option in the global options file. If you do not specify options either in the *optionsfile_accessmethod*.opts or in the global option file, the product issues an error message.

If the SAP R/3 access method is installed for AGENT1 workstation, with unique identifier, S4HANAR3BW, but you have two external SAP systems on which to schedule jobs, then in the *TWA_DATA_DIR*/methods directory, you create the following options files:

- SAP1_S4HANAR3BW_r3batch.opts
- SAP2_S4HANAR3BW_r3batch.opts

Each file contains the options specific to each external SAP system, for example, the connection information.

For pools and dynamic pools containing *n* agents, you must create an options file for the dynamic pool and copy it in the *TWA_DATA_DIR*/methods of each agent of the pool so that all members of the pool have a local options file with the same name. Then you must create another options file for the specific agent in the same directory. For example, if the SAP R/3 access method is installed for **AGENT1** and **AGENT2** which belong to the dynamic pool **DYN_POOL**, create in the *TWA_DATA_DIR*/methods directory of each agent the following options files:

AGENT1

- FILEOPTS_AGENT1_r3batch.opts
- FILEOPTS_DYN_POOL_r3batch.opts

AGENT2

- `FILEOPTS_AGENT2_r3batch.opts`
- `FILEOPTS_DYN_POOL_r3batch.opts`

Extended agents

All access methods use two types of options file:

Global options file

A common configuration file created by default for each access method installed, whose settings apply to all the extended agent workstations defined for that method. When the global options file is created, it contains only the **LJuser** option, which represents the operating system user ID used to launch the access method. You can customize the global options file by adding the options appropriate to the access method.

The name of the global options file is `accessmethod.opts`, which, depending on your operating system, corresponds to:

For PeopleSoft

`psagent.opts`

For SAP R/3

`r3batch.opts`

For custom access methods

`netmth.opts`

Local options file

A configuration file that is specific to each extended agent workstation within a particular installation of an access method. The name of this file is *`XA_Unique_ID_accessmethod`*`.opts`, where:

XA_Unique_ID

The unique identifier of the workstation in the plan. Since the folder feature was introduced, workstations can be defined in folders, therefore, the workstation name alone, is not sufficient to identify a workstation in the plan, but instead, the name and folder combination is mapped to a unique identifier. In the `localopts` file, the value of the this_cpu option is the unique identifier of the workstation. You can also verify the unique identifier for a workstation by submitting the composer list command with the ;showid filter. Should this result in blank, *XA_Unique_ID* corresponds to the workstation name. You can also retrieve the unique identifier submitting the conman showcpus command with the ;showid filter. For example, if the installation of the `r3batch` access method includes two extended agent workstations, with unique identifiers `S4HANAR3BW` and `07756YBX76Z6AFX2`, then the names of the local options files are `S4HANAR3BW_r3batch.opts` and `07756YBX76Z6AFX2_r3batch.opts`.

accessmethod

Is the name of the access method.

If you do not create a local options file, the global options file is used. Every extended agent workstation, except for z/OS®, must have a local options file with its own configuration options.

The options files must be located in the *`TWA_DATA_DIR`*`/methods` directory. They are read when the supported agent is started. Options are specific to each access method. For details about how to configure each access method, see the following sections:

PeopleSoft

Configuring the PeopleSoft access method on page 1098.

SAP R/3

Configuring the SAP access method on page 1145.

Option value inheritance

This property is currently available for r3batch only. Local options files can inherit existing values from the same options in the global options file `r3batch.opts`. For an access method, the options are listed twice; once as global options and once as local options. If the local options file does not contain a value for the option, then the value for that option in the global options file is used. Otherwise the option value in the local options file is always used.

For example, you might want to define the same value for the **Ljuser** option and a different value for the **retrieve_joblog** option. To do this, you define the **Ljuser** option value in the `r3batch.opts` file. Then you define a different value for the **retrieve_joblog** option in each local options file. This results in the following actions when launching the SAP R/3 job:

- The value for the **Ljuser** option is extracted from the r3batch.opts file.
- The value for the **retrieve_joblog** option is taken from each local options file.

Defining supported agent workstations

A workstation definition is required for each entity of an access method through which HCL Workload Automation schedules and launches jobs.

Creating a workstation using the Dynamic Workload Console

About this task

How to create a workstation definition for supported agents using the Dynamic Workload Console.

Dynamic agents

The agents are automatically registered to the HCL Workload Automation network.

Extended agents

To define an extended agent workstation using the Dynamic Workload Console, perform the following steps:

1. From the Dynamic Workload Console portfolio, click **Administration > Workload Environment Design > Create Workstations**.
2. Select an engine from the list and click **Create Workstation**.
3. In the Workstations properties panel, specify the attributes for the extended agent workstation you are creating. For all the details about available fields and options, see the online help by clicking the "?" in the top-right corner. In the workstation definition, specify the access method and other properties,

as shown in Table 133: How to complete the extended agents definition on page 1087. For further information about the workstation definition properties, see the section about workstation definition in *User's Guide and Reference*.

4. To assign the workstation to an existing domain or to create a new domain, click **Assign to Domain**.
5. Click **Save**.

The following table shows how to complete some specific fields of the workstation properties panel for **extended agents**.

Table 133. How to complete the extended agents definition

Field	Description by Access Method	
	PeopleSoft	**SAP R/3**
Name	The name for the extended agent workstation. For all access methods the name must start with a letter and can contain alphanumeric characters, dashes, and underscores. The maximum length is 16 characters. Workstation names must be unique and cannot be the same as workstation class and domain names. Double-byte character set (DBCS), single-byte character set (SBCS), and bidirectional text are not supported. If a workstation name contains these characters and, as a result, the options file name contains the same name, the workstation cannot be validated by the SAP system. For all the access methods, the name of the options file associated with the extended agent workstation must contain the unique identifier of the extended agent workstation, *XA_Unique_ID*_<access-method>.opts. That is, if the unique identifier for the extended agent workstation named /SAPBUS/S4HANAR3BW is 07756YBX76Z6AFX2, then the options file name must be 07756YBX76Z6AFX2_-r3batch.opts. For information about retrieving the unique identifier for an extended agent workstation, see UNIQUE_ID on page 1085.	
TCP Port	Any number other than 0.	
Access Method	**psagent**	**r3batch**

Table 133. How to complete the extended agents definition

(continued)

Field	Description by Access Method	
	PeopleSoft	**SAP R/3**
		Note: In UNIX™ the name is case sensitive and must be lowercase.

Creating a workstation using the command line

You can define supported agents workstations also using the **composer** command line of HCL Workload Automation.

Dynamic agents

The following example shows a definition for a dynamic agent workstation named *LINUX248* that uses the secure protocol *https* to connect to the Broker server.

```
CPUNAME LINUX248
  DESCRIPTION "This workstation was automatically created."
  OS UNIX
  NODE linux248.romelab.it.abc.com SECUREADDR 31114
  TIMEZONE Europe/Rome
  FOR MAESTRO HOST NC118003_DWB
    AGENTID "FD640FCA740311E18C4EE96D727FA991"
    TYPE AGENT
    PROTOCOL HTTPS
END
```

Extended agents

The following example shows a definition for a z/OS extended agent workstation named *MVSCPU* that uses the `mvsjes` access method.

```
cpuname MVSCPU description "zOS extended agent"
     os other
     node mvsesa36.rome.abc.com
```

```
     tcpaddr 5000
     domain masterdm
     for maestro
        type x-agent
        host ROCIOUS
        access mvsjes
end
```

For details about defining workstations with **composer**, see the *HCL Workload Automation User's Guide and Reference*.

Defining workstations for end-to-end scheduling

About this task

How to create a workstation definition for end-to-end environment.

Scheduling in an end-to-end environment means that in HCL Workload Automation for Z you are scheduling and monitoring jobs that are physically running on HCL Workload Automation workstations.

Extended agents

Extended agent workstations must be defined as fault-tolerant workstations in HCL Workload Automation for Z.

A fault-tolerant workstation is the HCL Workload Automation for Z definition of an existing HCL Workload Automation agent in the distributed network. The HCL Workload Automation agent is where the job associated with the fault-tolerant workstation actually runs in the distributed network.

To define the extended agent workstation in HCL Workload Automation for Z, you must:

1. Define the workstation in the `CPUREC` initialization statement. For an example, see Creating the CPUREC statement for extended agents on page 1090.
2. Add the same workstation definition to the database using ISPF or the Dynamic Workload Console. For a description of how to define the workstation using the Dynamic Workload Console, see *Dynamic Workload Console User's Guide*. For an example, see Defining the workstation with ISPF on page 1091.

HCL Workload Automation agents

To define the agent workstation with z-centric capability in HCL Workload Automation for Z, add the workstation definition to the database using ISPF or the Dynamic Workload Console. For further information, see *Scheduling End-to-end with z-centric Capabilities*.

Creating the CPUREC statement for extended agents

This section is valid only for Extended agents.

Create the `CPUREC` statement for the workstation in the `TOPOLOGY` initialization statement. The `TOPOLOGY` initialization statement is used to define parameters related to the topology of the connected HCL Workload Automation network. Such a network topology statement is made up of one or more (one for each domain) `DOMREC` statements that describe the topology of the distributed network, and by several `CPUREC` statements, one for each fault-tolerant workstation.

The following example shows a `CPUREC` statement for an SAP R/3 extended agent workstation named `R3XA`. The extended agent is hosted by an HCL Workload Automation agent named `TWSA`, which is also the domain manager of `DOMAIN1`.

```
***************TPLGINFO MEMBER ***************************

/*********************************************************************/
/* DOMREC: Domain definition                                         */
/*********************************************************************/
DOMREC   DOMAIN(DOMAIN1)
         DOMMNGR(TWSA)
         DOMPARENT(MASTERDM)
/*********************************************************************/
/* CPUREC: Extended agent workstation definition                     */
/*********************************************************************/
CPUREC   CPUNAME(R3XA)
         CPUOS(OTHER)
         CPUNODE(NODE1)
         CPUDOMAIN(DOMAIN1)
         CPUHOST(TWSA)
         CPUTYPE(XAGENT)
         CPUACCESS(r3batch)
         CPUUSER(TWSuser)
         CPUTZ('Europe/Rome')
/*********************************************************************/
/* CPUREC: Domain manager workstation definition                     */
/*********************************************************************/
CPUREC   CPUNAME(TWSA)
         CPUNODE(NODE1)
         CPUAUTOLINK(ON)
         CPUDOMAIN(DOMAIN1)
         CPUTYPE(FTA)
         CPUUSER(TWSuser)
         CPUTZ('Europe/Rome')
```

The following keywords define `R3XA` as an extended agent:

CPUACCESS

The extended agent access method. For SAP R/3, it is `r3batch`.

CPUHOST

The name of the HCL Workload Automation workstation hosting the extended agent. It cannot be another standard agent or extended agent.

CPUTYPE

The workstation type. For an extended agent, it must be `XAGENT`.

Note: The `CPUREC` statement does not exist for an HCL Workload Automation for z/OS agent workstation.

For further information about `CPUREC` for extended agents, see *Customization and Tuning*.

Defining the workstation with ISPF

About this task

This section shows the ISPF definition for extended agents and agents with z-centric capability.

Extended agents

In ISPF, define the workstation as `computer automatic` and then set the *FT Work station* field to **Y**. The `CPUREC` statement with the three keywords described in Creating the CPUREC statement for extended agents on page 1090 provides the extended agent specification.

Note: Make sure you write the `CPUREC` statement before making the ISPF or Dynamic Workload Console definition, because they have no effect without the `CPUREC` statement.

Figure 27. Defining an Extended Agent workstation

```
-------------- CREATING GENERAL INFORMATION ABOUT A W0RK STATION --------------
 Command ===>

 Enter the command R for resources  A for availability or M for access method
 above, or enter data below:

 WORK STATION NAME    ===> R3XA
 DESCRIPTION          ===> Extended agent for R/3 access method_____________________
 WORK STATION TYPE    ===> C          G  General, C  Computer, P  Printer
 REPORTING ATTR       ===> A          A  Automatic, S  Manual start and completion
                                      C  Completion only, N  Non reporting
 FT Work station      ===> Y          FT Work station, Y or N
 PRINTOUT ROUTING     ===> SYSPRINT   The ddname of daily plan printout data set
 SERVER USAGE         ===> N          Parallel server usage C , P , B or N

 Options:
  SPLITTABLE          ===> N          Interruption of operation allowed, Y or N
  JOB SETUP           ===> N          Editing of JCL allowed, Y or N
  STARTED TASK, STC   ===> N          Started task support, Y or N
  WTO                 ===> N          Automatic WTO, Y or N
  DESTINATION         ===> ________   Name of destination
 Defaults:
  TRANSPORT TIME      ===> 00.00      Time from previous work station  HH.MM
  DURATION            ===> ________   Duration for a normal operation  HH.MM.SS
```

HCL Workload Automation for Z agents

For detailed information and examples about the ISPF definition of HCL Workload Automation for Z agents with z-centric capabilities, see *Scheduling End-to-end with z-centric capabilities*.

Defining jobs for supported agents

To run and monitor a PeopleSoft, SAP R/3, or z/OS job with HCL Workload Automation, the supported agents, or access method require an HCL Workload Automation job definition, where you specify the external job you want to schedule, the workstation (also defined in HCL Workload Automation) on which it is to run, and any recovery actions. To define the job, use either of the following methods:

- Dynamic Workload Console.
- HCL Workload Automation **composer** command line.

If you are scheduling in an end-to-end environment, to define the job, use either of the following methods:

- Dynamic Workload Console.
- HCL Workload Automation for Z ISPF dialogs. You must also create a member in the `SCRIPTLIB` with a `JOBREC` statement for the job.

Jobs defined for supported agents are added to job streams and scheduled in the same way as any other job in HCL Workload Automation and HCL Workload Automation for Z.

Defining jobs with the Dynamic Workload Console

About this task

How to create a job definition for supported agents using the Dynamic Workload Console.

Steps for defining a job for supported agents.

To define jobs, follow these steps:

1. From the Dynamic Workload Console, click the **Design** menu and select Workload Designer.
2. Specify an engine name, either distributed or z/OS. The Workload Designer window opens. Job types and characteristics vary depending on whether you select a distributed or a z/OS engine.
3. Click **Create new** and select **Job definition**.
4. Select the category and type of job you want to create.
   - For SAP jobs, **ERP > SAP Job on XA Workstations** or **SAP Job on Dynamic Workstations**. See Creating SAP Standard R/3 jobs from the Dynamic Workload Console on page 1172.
   - For z/OS and PeopleSoft, **ERP > Access Method**.
5. In the properties panel, specify the attributes for the job definition you are creating. For all the details about available fields and options, see the online help specific for the item.
6. Click **Save** to save the job definition in the database.

Note: The access method for SAP R/3 provides supplementary features if you use the alternative steps described in Creating SAP Standard R/3 jobs from the Dynamic Workload Console on page 1172. You can create native SAP R/3 Standard jobs on a remote SAP system directly from the Dynamic Workload Console.

Defining jobs using the command line

You can also define jobs using the **composer** command line of HCL Workload Automation.

Dynamic agents

The following example describes an HCL Workload Automation job named `DYN_JOB_R3_0001` defined in the folder name `SAPJOBS`, that runs on a dynamic agent workstation named `NC112015_1`. The HCL Workload Automation launches a job in an SAP environment named `JOB_APPS_93`.

```
NC112015_1#/SAPJOBS/DYN_JOB_R3_0001
 TASK
<?xml version="1.0" encoding="UTF-8"?>
<jsdl:jobDefinition
     xmlns:jsdl="http://www.abc.com/xmlns/prod/scheduling/1.0/jsdl"
     xmlns:jsdlxa="http://www.abc.com/xmlns/prod/scheduling/1.0/jsdlxa"
                  name="r3">
  <jsdl:application name="r3" plugin="xajob">
    <jsdlxa:xajob accessMethod="r3batch" target="NW73LIN_r3batch">
            <jsdlxa:taskString>/ -job JOB_APPS_93 -i 14514200 -c c
                 -flag ENABLE_APPL_RC </jsdlxa:taskString>
        </jsdlxa:xajob>
  </jsdl:application>
</jsdl:jobDefinition>
 RECOVERY STOP
```

Extended agents

The following example describes an HCL Workload Automation job named `psjob2` that runs on a PeopleSoft extended agent workstation with unique identifier named `XAPS002`. HCL Workload Automation logs on to UNIX operating system as `psjobs` and launches a job under PeopleSoft. The PeopleSoft process is named `XRFWIN`. If recovery is needed, HCL Workload Automation runs job `recov2` and then continues processing.

```
XAPS002#/mypsjobs/psjob2
  streamlogon psjobs
  scriptname
   -process XRFWIN -type 'SQR Report' -runcontrol 1 -runlocationdescr PSNT
  description "peoplesoft job #2"
  recovery continue after recov2
```

The arguments of `scriptname` differ by application. For details, see:

- Task string parameters for PeopleSoft jobs on page 1107.
- Task string to define SAP jobs on page 1180.

For more information about using the **composer** command line to define jobs, see *User's Guide and Reference*.

Defining jobs for end-to-end scheduling

Extended agents

Extended agent jobs scheduled to run in an end-to-end environment cannot be defined using the Dynamic Workload Console or the HCL Workload Automation command line, but must be added to the `SCRIPTLIB` of HCL Workload Automation for Z.

In the `OPERATIONS` ISPF panel of HCL Workload Automation for Z, extended agent jobs are defined like any other job, but with the specific attribute for a job defined on an extended agent workstation. The following example shows the definition of a job named `SAPJOB`. This is the HCL Workload Automation for Z job that drives the running of on SAP R/3 job (named `BAPRINT46B` as shown in the next example). It shows as an extended agent job because the associated workstation is an extended agent workstation named `R3XA`.

Figure 28. Defining an Extended Agent job for end-to-end scheduling

```
--------------------------------- OPERATIONS -----------------Row 1 to 1 of 1
Command ===>                                                 Scroll ===> PAGE

Enter/Change data in the rows, and/or enter any of the following
row commands:
I(nn) - Insert, R(nn),RR(nn) - Repeat, D(nn),DD - Delete
S - Select operation details, J - Edit JCL
Enter the TEXT command above to include operation text in this list, or,
enter the GRAPH command to view the list graphically.

Application            : APLL1            FTW appl

Row  Oper      Duration  Job name  Internal predecessors           Morepreds
cmd  ws   no.  HH.MM.SS                                          -IntExt-
'''' R3XA 001  00.00.01  SAPJOB___  ___ ___ ___ ___ ___ ___ ___ ___    0  0
****************************** Bottom of data ******************************
```

For each job, create a member in the `SCRIPTLIB` of HCL Workload Automation for Z with details about the job in a `JOBREC` statement. A SAPJOB member was created for the job of the previous example. It contains a `JOBREC` statement like this:

```
JOBREC
  JOBCMD('/-job BAPRINT46B -user MAESTRO -i 14160001 -c C')
  JOBUSR(twsila)
```

The string in `JOBCMD` is read and interpreted by the access method before running the job. The job of this example, `BAPRINT46B`, was previously defined on SAP R/3 and assigned with an ID of 14160001, that was manually written in `JOBCMD`.

The following example is for a PeopleSoft job. The entire string that follows the `JOBCMD` keyword must be enclosed within quotation marks ("), because for PeopleSoft jobs single quotes are already used in the string.

```
JOBREC
   JOBCMD("/ -process XRFWIN -type 'SQR Report' -runcontrol IWS")
   JOBUSR(PsBuild)
```

HCL Workload Automation for Z agents

For information about the jobs definition for agent with z-centric capabilities, see *Scheduling End-to-end with z-centric capabilities*.

The arguments of `JOBCMD` differ by application. For details, see:

- Task string parameters for PeopleSoft jobs on page 1107.
- Task string to define SAP jobs on page 1180 or Defining SAP jobs dynamically on page 1199.

Submitting jobs

About this task

To submit jobs on the supported agent workstation, perform the following steps:

1. Verify that the application system to which the job belongs and the related database is up and running.
2. Launch the job. For details, see:

   Dynamic agents

   - *User's Guide and Reference* for **conman** command line.
   - *Dynamic Workload Console User's Guide* for Dynamic Workload Console.

   Extended agents

   - *User's Guide and Reference* for **conman** command line.
   - *Dynamic Workload Console User's Guide* for Dynamic Workload Console.

   HCL Workload Automation agents

   - *Scheduling End-to-end with z-centric Capabilities* for **ISPF** panel.
   - *Dynamic Workload Console User's Guide* for Dynamic Workload Console.

Access method for PeopleSoft

What you need and what you can do with Access method for PeopleSoft.

Using Access method for PeopleSoft you can run and monitor PeopleSoft jobs from the HCL Workload Automation environment. These jobs can be run as part of a schedule or submitted for ad-hoc job processing. PeopleSoft extended agent or dynamic agent jobs can have all of the same dependencies and recovery options as other HCL Workload Automation jobs. PeopleSoft jobs must be defined in HCL Workload Automation to be run and managed in the HCL Workload Automation environment.

For information about the supported versions of the plug-ins and access methods, run the Supported Software report and select the **Supported Software** tab.

Features

Look at the tasks you can perform by using Access method for PeopleSoft.

Using Access method for PeopleSoft, you can perform the following tasks:

- Use HCL Workload Automation standard job dependencies on PeopleSoft jobs.
- Schedule PeopleSoft jobs to run on specified days, times, and in a prescribed order.
- Define inter-dependencies between PeopleSoft jobs and HCL Workload Automation jobs that run on different applications such as SAP R/3 and Oracle E-Business Suite.
- Define inter-dependencies between PeopleSoft jobs and jobs that run on different operating systems.

Roles and responsibilities

Here you can see the roles and responsibilities of all the actors involved in the process model, and the tasks they perform.

In a typical enterprise, different users contribute to the implementation and operation of the product. Table 134: Roles and responsibilities in Access method for PeopleSoft on page 1097 describes the roles and responsibilities of all those involved in the process model, showing the tasks they perform.

Table 134. Roles and responsibilities in Access method for PeopleSoft

User role	User task
HCL Workload Automation configurator	Defining the configuration options on page 1098
HCL Workload Automation developer	• Defining PeopleSoft jobs in HCL Workload Automation on page 1107 • Configuring the job status mapping policy on page 1110
PeopleSoft administrator	• Creating a batch processing ID in PeopleSoft on page 1103 • Configuring the ITWS_PSXA PeopleSoft project on page 1103 • Uploading the PeopleSoft project on page 1104

Scheduling process for the PeopleSoft supported agents

HCL Workload Automation can launch and monitor jobs in the PeopleSoft process scheduler using a PeopleSoft extended agent or dynamic agent workstation. The PeopleSoft supported agent (extended agent or dynamic agent) is defined in a standard HCL Workload Automation workstation definition. This definition is a logical workstation name and specifies the access method as `psagent`. The access method is used to communicate job requests to the PeopleSoft process scheduler.

To launch a PeopleSoft job, HCL Workload Automation runs the `psagent` method, passing it information about the job. An options file provides the method with the path, the executable, and other information about the PeopleSoft process scheduler and application server used to launch the job. The supported agent t can then access the PeopleSoft process request table and make an entry in the table to launch the job. Job progress and status information is written to the job's standard list file.

For extended agents, there is no need to install Database connectivity on fault-tolerant agents hosting PeopleSoft extended agents because the method currently uses the PeopleSoft 3-tier architecture. You must configure at least one PeopleSoft Application Server for the supported agent to work. The application server must be active to successfully submit jobs to the PeopleSoft process scheduler.

PeopleSoft job tracking in HCL Workload Automation

A PeopleSoft job is a collection of processes that run together as a single unit. HCL Workload Automation jobs can be defined in one of the following ways:

- As PeopleSoft jobs, that is, as a collection of PeopleSoft processes. In this case, the status of the PeopleSoft job is tracked, not the status of the individual processes within the job.
- As PeopleSoft processes. In this case, the status of the individual process is tracked and HCL Workload Automation schedules can be defined to create complex inter-dependencies and recovery options between PeopleSoft processes.

Security

Security for the PeopleSoft jobs is handled by standard HCL Workload Automation security.

Configuring the PeopleSoft access method

This section provides detailed reference information about the PeopleSoft options and how to define them in the options file.

Defining the configuration options

The HCL Workload Automation installation process creates a default global options file for the `psagent` access method, named `psagent.opts`. You can also create the following local files in the path:

On UNIX operating systems

psjoa.jar

```
TWA_DATA_DIR/methods
```

On Windows operating systems

```
TWA_home\methods
```

Extended agent

`XA_Unique_ID_psagent.opts` where `XA_Unique_ID` is the unique identifier for the extended agent workstation. For more details about how to identify the unique ID, see UNIQUE_ID on page 1085.

Dynamic agent

`DYNAMIC_AGENT_FILE_psagent.opts` where *DYNAMIC_AGENT_FILE* is any text string. This string does not necessarily correspond to the name of the dynamic agent workstation since the dynamic agent can have more than one .opts file associated. For more information, see Setting options for the access methods on page 1081.

To edit both options file, you can use any text editor. On dynamic workstations, you can edit the options files from the job definition panels in the Dynamic Workload Console. For examples of options files for this access method, see PeopleSoft options file example on page 1101.

Table 135: Psagent access method options on page 1099 describes the options for the `psagent` access method. Option names are case insensitive. Before you use a manually-created options file, check that all the option names are written correctly, otherwise they will be ignored.

Table 135. Psagent access method options

Table 135. Psagent access method options (continued)

Option	Description
PS_DISTSTATUS	(Optional) Determines whether the distribution status of the PeopleSoft request is taken into account when determining the status of the HCL Workload Automation job. Values are *0* (not taken into account) or *1* (taken into account - the default value).
PSFT_DOMAIN_PWD	(Optional) Specifies the encrypted password (case-sensitive) of the PeopleSoft domain used for the connection to the PeopleSoft application server. For details about how to encrypt the password, see Encrypting PeopleSoft operator passwords on page 1102.
PSFT_OPERATOR_ID	(Mandatory) Specifies the PeopleSoft operator ID used for the connection to the PeopleSoft application server.
PSFT_OPERATOR_PWD	(Mandatory) Specifies the encrypted password (case-sensitive) of the PeopleSoft operator ID used for the connection to the PeopleSoft application server. For details about how to encrypt the password, see Encrypting PeopleSoft operator passwords on page 1102.

PSJOAPATH	(Optional) Specifies the full path name of the `psjoa.jar` file, containing both the path and the psjoa.jar filename.
	
	If this option is not set, the following default path name is used:
	
	`TWA_DATA_DIR/methods/psjoa.jar`
	
	`TWA_home\methods\psjoa.jar`
	
	Ensure that you identify the version of the `psjoa.jar` file that corresponds to the version of PeopleSoft that you are using and you have access to the file.

RUNLOCATION	(Optional) Specifies the default PeopleTools process server that processes the requests.
SERVER_NAME_LIST	(Mandatory) Specifies the list of application servers that the psagent tries to connect to. It is a comma-separated list of addresses in the format: `server:port [,server:port] ...` where: ***server*** Specifies the host name or TCP/IP address of the server

Table 135. Psagent access method options (continued)

Option	Description
	port Specifies the port number the server is listening on.
TWS_MAX_WAIT_TIME	(Optional) Specifies the maximum time that the supported agent waits (timeout) after a failed operation on the PeopleSoft application server before retrying the operation. The default is 10 seconds.
TWS_MIN_WAIT_TIME	(Optional) Specifies the minimum time that the supported agent waits (timeout) after a failed operation on the PeopleSoft application server before retrying the operation. The default is 5 seconds.
TWS_RETRY	(Optional) The maximum number of times that the supported agent attempts to re-run a failed operation on the PeopleSoft application server. The default is 5.
TWSXA_INLINE_CI	(Optional) Specifies the name of the component interface that the `psagent` invokes to submit jobs to PeopleSoft. The default is `ITWS_PROCESSREQUEST`. If you use this default, you must perform the customization steps described in Configuring the ITWS_PSXA PeopleSoft project on page 1103. If you do not plan to schedule jobs containing in-line variables, and you do not want to perform the additional customization steps, you must replace the default value with `PROCESSREQUEST`. This is the component interface invoked by previous versions of the access method; it does not allow the use of in-line variables.
TWSXA_SCHED_METH	(Optional) Specifies the name of the PeopleSoft method invoked by the component interface specified in `TWSXA_INLINE_CI`. Both `ITWS_PROCESSREQUEST` and `PROCESSREQUEST` use the default method `Schedule`. If you are using either of these component interfaces, leave the default. If you are using a different component interface, specify the name of the method called by your component interface, respecting the case of the PeopleSoft object name.

PeopleSoft options file example

Below is a sample options file. It can help you determine your specific site requirements although your options file might be different.

Remember to save the file in the following directory:

```
TWA_DATA_DIR/methods
```

```
TWA_home\methods
```

Example

```
LJuser=TwsUsr
CheckInterval=120
PSFT_OPERATOR_ID=PSHC
PSFT_OPERATOR_PWD=*****
SERVER_NAME_LIST=9.87.120.36:9000
```

If you create the options file manually, you must encrypt the PeopleSoft operator password, as described in Encrypting PeopleSoft operator passwords on page 1102.

Encrypting PeopleSoft operator passwords

When you add or change the PeopleSoft operator password using the Dynamic Workload Console, the password is automatically encrypted and securely stored in the file. For added security, it is displayed on the screen as a series of asterisks.

When you add or change the PeopleSoft user password using a text editor, run the **pwdcrypt** command to encrypt the password before writing it in the file, as follows:

```
pwdcrypt password
```

The program returns the password in encrypted format that you can then copy and paste into the options file.

Connecting to more than one PeopleSoft application server

It might be necessary for the `psagent` method to connect to more than one PeopleSoft application server. For example, a single installation of PeopleSoft might have a TEST, DEMO, and PRODUCTION environment, each with a separate application server. This requires that the `psagent` method uses a separate connect string for each application server.

To support this, you can set up multiple PeopleSoft extended agent workstations that connect to the same method but use different options files. When a workstation starts the method, it first looks for the options file with extended agent workstation unique identifier prepended to `psagent.opts`. For example, a PeopleSoft extended agent with unique identifier `ps847system` would have the following options file:

```
PS847SYSTEM_psagent.opts
```

The **psagent** method searches first for an options file with the extended agent workstation unique identifier, and then for the default `psagent.opts` file. This allows the user to set up an extended agent for each PeopleSoft application server.

To connect to only one application server, use the default name for the options file, **psagent.opts**.

Note: In case you specify some connection properties in your local option files, make sure that the same properties are commented out in your global option file, with the exception of the global property **LJuser**. This action is needed to avoid that warning messages related to duplicate properties are displayed in the job log.

Creating a batch processing ID in PeopleSoft

Create an operator ID in PeopleSoft dedicated to batch scheduling. This operator ID must be granted authority to use the Component Interface in the PeopleTools environment. All the jobs submitted by HCL Workload Automation should use this operator ID.

Configuring the ITWS_PSXA PeopleSoft project

About this task

The configuration steps described in this section are necessary to enable HCL Workload Automation to schedule PeopleSoft jobs that have in-line variables in their definitions.

The `ITWS_PROCESSREQUEST` component interface works around some limitations of the PeopleSoft APIs when invoked from a batch environment. Because of these limitations, HCL Workload Automation cannot schedule jobs defined with in-line bind variables. With current PeopleSoft APIs, data that is stored in the PeopleSoft database and referred to by a `runcontrol ID parameter` that is used to retrieve a `runcontrol data record`, needs to be loaded into the `Component Buffer` before scheduling the API invocation. This cannot be done from a batch environment. Therefore, when invoking the PeopleSoft scheduling APIs from a batch interface, the data related to the `runcontrol ID` is not available for the submission of a job, even though it is available in the database. When unresolved data is present in the submitted job, the PeopleSoft system refuses submission and ends with an error.

The `ITWS_PROCESSREQUEST` component interface enables HCL Workload Automation to schedule PeopleSoft jobs that have in-line variables in their definitions. By invoking this component interface, the access method provides the ability to use data stored in the PeopleSoft database to resolve in-line variable values by taking data from the database and substituting it with variable definitions. It then allows job submission regardless of the use of in-line variable definitions in the jobs. The variable substitution mechanism does not support `work` records, so if the PeopleSoft process uses work records in its parameter list, you find a message similar to the following in the HCL Workload Automation joblog:

```
Error Position:  21
Return:  942 - ORA-00942: table or view does not exist
Statement:
select nvsdlist from PS_NVS_WRK WHERE BUSINESS_UNIT = :1 AND REPORT_ID = :2
Original Statement:
SELECT NVSDLIST FROM PS_NVS_WRK WHERE BUSINESS_UNIT = :1 AND REPORT_ID = :2.
```

To identify work records, use the following PeopleSoft naming conventions:

- A derived work record name ends with '_WRK'.
- A work record definition name for Structured Query Report reports starts with R_

When you use HCL Workload Automation to submit a process that has in-line bind variables, the name of the process type in the PeopleSoft GUI becomes `ITWS_process type`. For example, `SQR Process` becomes `ITWS_SQR Process`.

To schedule a job that contains in-line variables in its definition you must perform the following tasks:

- Leave the value of the `TWSXA_INLINE_CI` option set to `ITWS_PROCESSREQUEST`, that is the default value. See Defining the configuration options on page 1098 for a detailed explanation.
- Upload the PeopleSoft project as described in Uploading the PeopleSoft project on page 1104.

Uploading the PeopleSoft project

About this task

This section describes how to upload a new PeopleSoft project related to PeopleTools 8.44, or later, into the PeopleSoft database. The name of the PeopleSoft project is `ITWS`.

After installing the product, complete the following steps:

1. Mount the PT844 PeopleSoft project directory or copy it to the workstation from where you launch the Application Designer. HCL Workload Automation installs the PeopleSoft project directories, as shown in the following structure:

```
TWS_DIR/methods/
/---PeopleSoft
        /---PT844
         /---ITWS_PSXA
                 ITWS_PSXA.ini
                 ITWS_PSXA.XML
```

```
TWS_DIR\methods\
\---PeopleSoft
        \---PT844
         \---ITWS_PSXA
                 ITWS_PSXA.ini
                 ITWS_PSXA.XML
```

2. Start the Application Designer and from the sign-on window select to start the Application Designer in tier-two mode by entering the following information:

- **Connection Type**: *database used*; for example, Oracle
- **Database Name**: *database instance name*
- **User ID**: *PeopleSoft operator name*; for example, PS
- Password of user ID

3. Using the Application Designer, select **Tools** -> **Copy Project**-> **From file...**
4. Using the browser, edit the full path to specify the folder where the project that you want to load is located.

   The project is contained in the `TWS_DIR/methods/PeopleSoft/PT844` subdirectories on UNIX® and in `TWA_home\methods\PeopleSoft\PT844`. on Windows™.

   After you specify the project folder, a list of projects appears in the **Project Name** field of the Copy Project From File window.

5. Choose `ITWS_PSXA` and click **Open**. If you already configured `ITWS_PSXA` (perhaps after installing a fix pack), a confirmation window enquires if you want to replace the existing one. Click **Yes**.

   The Copy window is displayed showing a list of definition types.

6. Click **Options** to select the new settings.
   a. Click **Report Filter**
   b. Click **Select All**
   c. Click **OK**
   d. Click **Select All**
   e. Click **Copy**. A progress bar is displayed.

   After loading the project, the PeopleSoft Database contains the following objects:
   - ITWS process type definitions
   - ITWS permissions list
   - ITWS component interfaces

7. Create the `ITWS_ROLE` security role. You can use either the PeopleSoft Web GUI or the Application Designer. Follow the steps below:

   From the menu of the PeopleSoft Web GUI:

   a. Select **NavBar** in the upper right corner -> **Menu** -> **PeopleTools** -> **Security** -> **Permission and Roles** -> **Roles**.
   b. Select the **Add a new value** tab
   c. Type or select `ITWS_ROLE` in the **Role Name** field
   d. Select the **Permissions list** tab -> **ITWS** -> **Save**

   From the Application Designer GUI:

   a. Using Maintain Security, edit the `ITWS_ROLE` window
   b. Select the **Permissions list** tab -> **ITWS** -> **Save**

8. Grant `ITWS_ROLE` authority to all users who want to schedule jobs from HCL Workload Automation. You can use either the PeopleSoft Web GUI or the Application Designer. Follow the steps below:

   From the PeopleSoft Web GUI:

a. Select **NavBar** in the upper right corner -> **Menu** -> **PeopleTools** -> **Security** -> **User Profiles**.
b. Type the user name of the user who wants to schedule jobs from HCL Workload Automation.
c. In **User Roles** window click on the + sign in the row where you want to add the role.
d. Select the **Roles** tab.
e. Add `ITWS_ROLE` and save.

From the Application Designer GUI:

a. Using Maintain Security, edit the user name
b. Select the **Roles** tab
c. Add `ITWS_ROLE` and save

9. Add the ITWS process type definitions to the required PeopleTools process scheduler. You can use either the PeopleSoft Web GUI or the Application Designer. Follow the steps below:

   From the PeopleSoft Web GUI:

   a. Select **NavBar** in the upper right corner -> **Menu** -> **PeopleTools** -> **Process Scheduler** -> **Process Scheduler Servers**.
   b. Click on the **Search** button and select the PeopleTools server you plan to use.
   c. In the **Server Definition** tab and in **Process Types run on this Server** window click on the + sign in the row of your choice.
   d. Click on the search icon. A new windows is displayed.
   e. Select all the **ITWS_*** entries (one for each row created).
   f. Click Save.

   From the Application Designer GUI:

   a. Select **Process Scheduler Manager**
   b. Select your PeopleTools server
   c. Add the ITWS_* Type definitions and save

   Note: From the SQL interactive command line, the same task can be performed by the following sample statement, customized for your database environment:

```
INSERT INTO PS_SERVERCLASS SELECT o.SERVERNAME,
o.OPSYS,'ITWS_'||o.PRCSTYPE,o.PRCSPRIORITY,
o.MAXCONCURRENT FROM PS_SERVERCLASS
o WHERE (  SELECT count(*) FROM PS_SERVERCLASS i WHERE
i.SERVERNAME=o.SERVERNAME AND   i.OPSYS=o.OPSYS AND
i.PRCSTYPE='ITWS_'||o.PRCSTYPE ) = 0
AND ( select count(*) from PS_PRCSTYPEDEFN
a where a.PRCSTYPE='ITWS_'||o.PRCSTYPE AND a.OPSYS=o.OPSYS ) > 0
```

10. Restart the process servers.

You do not need to change the existing HCL Workload Automation job definitions, except for the scheduling nVision process, where the `runcontrol ID` must be specified using the *BUSINESS_UNIT.REPORT_ID* convention.

The following is an example of a job definition for the scheduling nVision process:

```
-process 'NVSRUN' -type nVision-Report -runcontrol AUS01.VARIABLE
```

where `NVSRUN` is the process name and `AUS01.VARIABLE` is the *BUSINESS_UNIT.REPORT_ID*.

Defining PeopleSoft jobs

This section provides job definition information for jobs using the extended agent for PeopleSoft.

Defining PeopleSoft jobs in HCL Workload Automation

An HCL Workload Automation job definition is required for every PeopleSoft job you want to manage. An HCL Workload Automation job is associated to an already defined PeopleSoft job and its definition includes:

- The name of the HCL Workload Automation job that runs the PeopleSoft job
- The unique identifier of the extended agent or dynamic workstation or workstation class where the HCL Workload Automation job runs. See UNIQUE_ID on page 1085 for more information about the unique identifier.
- The name of the user launching the job
- Recovery options
- The Script file specifications

For more information, refer to Defining jobs for supported agents on page 1092.

Task string parameters for PeopleSoft jobs

This section describes the task string parameters that control the operation of PeopleSoft jobs. You must specify them in the following places when you define their associated HCL Workload Automation jobs:

- In the **Task string** field of the Task page of the Properties - Job Definition panel, if you use the Dynamic Workload Console
- As arguments of the `scriptname` keyword in the job definition statement, if you use the HCL Workload Automation command line.
- As arguments of the `JOBCMD` keyword in the `JOBREC` statement in the `SCRIPTLIB` of HCL Workload Automation for Z, if you are scheduling in an end-to-end environment. In this case the entire string following the `JOBCMD` keyword must be enclosed within quotation marks (").

  The following is an example of a `JOBREC` statement:

  ```
  JOBREC
    JOBCMD("/-process process_name -type 'process_type' -runcontrol runcontrol_ID")
    JOBUSR(TWS_user_name)
  ```

  where:

  process_name

  The process name for the PeopleSoft job.

  process_type

  The process type for the PeopleSoft job. This entry must be enclosed within single quotes.

runcontrol_ID

The runcontrol ID for the PeopleSoft job.

TWS_user_name

The HCL Workload Automation for Z user who runs the `psagent` access method from the end-to-end scheduling environment.

Table 136: Task string parameters for PeopleSoft jobs on page 1108 describes the parameters to define PeopleSoft jobs.

Table 136. Task string parameters for PeopleSoft jobs

Parameter	Description
-process	The process name for the PeopleSoft job.
-type	The process type for the PeopleSoft job. This entry must be enclosed within single quotes.
-runcontrol	The runcontrol ID for the PeopleSoft job.
-outputdest	The destination of the PeopleSoft job output.
-outputtype	The output type of the PeopleSoft job. Possible values are: • Any • Email • File • None • Printer • Web • Window If you do not specify any value, HCL Workload Automation uses the value associated to the PeopleSoft job you are submitting.

Table 136. Task string parameters for PeopleSoft jobs (continued)

Parameter	Description
	Note: Depending on the PeopleSoft configuration, some combinations of the value of this option with the value of the **outputformat** option are not supported. In this case the PeopleSoft default value is used.
-outputformat	The output format of the PeopleSoft job. Valid values are: **None** PDF **CSV** PS **DOC** RPT **Default** RTF **HTM** TXT **LP** WKS **OTHER** XLS **Note:** Depending on the PeopleSoft configuration, some combinations of the value of this option with the value of the **outputtype** option are not supported. In this case the PeopleSoft default value is used.
-runlocationdescr	The PeopleSoft process scheduler responsible for processing the PeopleSoft job.
-foldername	The name of the report folder used for this job. The folder must have been already created using PeopleSoft Report Manager.

Table 136. Task string parameters for PeopleSoft jobs (continued)

Parameter	Description
tracelvl	Specify the trace setting for the job. Possible values are: **1** Only error messages are written in the trace file. This is the default. **2** Informational messages and warnings are also written in the trace file. **3** A most verbose debug output is written in the trace file. Refer to Configuring the tracing utility on page 1120 for detailed information.

Note: No syntax checking is performed on the output control values (`outputdest`, `outputtype`, `outputformat`, and `foldername`). If the values are not recognized, defaults are used.

The following is an example of a task string specification for a PeopleSoft 8.44 job:

```
-process XRFWIN -type 'SQR Report' -runcontrol 1 -runlocationdescr PSNT
```

Configuring the job status mapping policy

HCL Workload Automation calculates the status of an HCL Workload Automation job based on the PeopleSoft job Run Status and Distribution Status. In PeopleSoft, the run status monitors the running of the job until it reaches a final status; the distribution status monitors the status of the output of the job. If the final status of a PeopleSoft job is neither success nor warning, HCL Workload Automation ignores the distribution status and the HCL Workload Automation job status is ABEND.

If the final status of a PeopleSoft job is success or warning, you can decide whether to use the distribution status of the PeopleSoft job when determining the status of the HCL Workload Automation job by setting the PS_DISTSTATUS option in the options file:

0

The distribution status is ignored and the HCL Workload Automation job status is calculated as shown in Table 138: Relationship between the run status and the HCL Workload Automation job status on page 1112.

1

The distribution status is used and the HCL Workload Automation job status is calculated as shown in Table 137: Relationship between the run status, the distribution status, and the HCL Workload Automation job status on page 1111. This is the default value.

Table 137: Relationship between the run status, the distribution status, and the HCL Workload Automation job status on page 1111 shows the relationship between the run status, the distribution status, and the HCL Workload Automation job status. The return code associated with the status is shown in parentheses. HCL Workload Automation uses this return code to evaluate the return code condition you specified in the **Return Code Mapping Expression** field in the Properties panel of the job definition. For more details about this field, refer to the online help by clicking the "?" in the top-right corner of the panel.

Table 137. Relationship between the run status, the distribution status, and the HCL Workload Automation job status

PeopleSoft job run status	PeopleSoft job distribution status	HCL Workload Automation job status
• Success (9) • Warning (17)	• Posted (5) • None (0)	SUCC
• Success (9) • Warning (17)	• Not Posted (4) • Delete (6)	ABEND
• Success (9) • Warning (17)	• Not Available (1) • Processing (2) • Generated (3) • Posting (7)	EXEC
• Cancel (1) • Delete (2) • Error (3) • Canceled (8) • No Success (10) • Blocked (18) • Restart (19)	Any distribution status	ABEND

Table 138: Relationship between the run status and the HCL Workload Automation job status on page 1112 shows the relationship between the PeopleSoft run status and the HCL Workload Automation job status. The return code associated with the status is shown in parentheses. HCL Workload Automation uses this return code to evaluate the return code

condition you specified in the **Return Code Mapping Expression** field in the Properties panel of the job definition. For more details about this field, refer to the online help by clicking the "?" in the top-right corner of the panel.

Table 138. Relationship between the run status and the HCL Workload Automation job status

PeopleSoft final run status	HCL Workload Automation status
Cancel (1)	ABEND
Delete (2)	ABEND
Error (3)	ABEND
Hold (4)	WAIT
Queued (5)	WAIT
Initiated (6)	INIT
Processing (7)	EXEC
Canceled (8)	ABEND
Success (9)	SUCC
No Success (10)	ABEND
Pending (16)	EXEC
Warning (17)	SUCC
Blocked (18)	ABEND
Restart (19)	ABEND

Note: If HCL Workload Automation fails to retrieve the status of the PeopleSoft job, the HCL Workload Automation job status is **DONE**.

Common serviceability for the access methods

This section provides information common to all the access methods including return code mapping, configuring the tracing utility, and troubleshooting the access method.

The return code mapping feature

The return code mapping feature provides a standard way of mapping messages into return code values. You can also customize the return code mapping. This feature is available for the following access methods:

- PeopleSoft
- SAP R/3

The return code mapping feature provides more granularity when defining the success or failure policies of jobs and improved flexibility in controlling job execution flows based on execution results. Job return code mapping provides the following capabilities:

- Users can define a job final status (successful or failed) based on a condition on the return code of the execution of the program or script of the job.
- The return code can be provided also to the recovery job that is associated with it in the job definition. This causes the recovery job to perform different processing based on the return code.

Parameters

#

Optional comment. All the lines starting with this symbol (#) are not used for mapping.

pattern*n*

Pattern strings delimited by quotation marks (“ and ”). If you use only one pattern string, you can omit the quotation marks. If the pattern string contains a quotation marks character, then it must be escaped by backslash (\). The string can contain the following wildcards and special characters:

Asterisk (*)

Matches an arbitrary number of characters.

Question mark (?)

Matches a single character.

Backslash (\)

Escape character.

RC value

The return code value. This value is sent by the method to HCL Workload Automation by a `%RC nnnn` message.

Creating a return code mapping file

You can create a return code mapping file to customize your own return codes with respect to certain conditions that might affect a job when it runs. Use this file to set the success condition of the job, which HCL Workload Automation uses to assess if the job completes successfully or in error. The return code is sent to HCL Workload Automation in the form of a `%RC nnnn` message. If this message is received, the job state is updated accordingly.

Each method has its own set of files to map the messages into return code values. The mapping files can be either global or local for a workstation.

Return code mapping files that are specific to a workstation are named according to the following scheme:

TWA_DATA_DIR`/methods/rcm/`*accessmethod-type-workstation*`.rcm`

TWA_home`\methods\rcm\`*accessmethod-type-workstation*`.rcm`

Global mapping files have a file name according to the following scheme:

TWA_DATA_DIR`/methods/rcm/`*accessmethod-type*`.rcm`

TWA_home`\methods\rcm\`*accessmethod-type*`.rcm`

For the PeopleSoft access method, *type* is always equal to `rcmap`. For the SAP R/3 access method, *type* is as described in Return code mapping file names for r3batch on page 1118.

Syntax

About this task

Use the following syntax to create the return code mapping file:

```
[#] "pattern1? "pattern2"..."patternn" = RC value
```

Examples

The following is an example of a return code mapping file. The line numbers in bold do not belong to the file, but are shown for reference:

```
1.   # This is an RC mapping file for joblog.
2.
3.   “User * missing  = 102
4.   “\*\*\*?  = 103
5.   “User \
6.   * \
7.   missing  = 102
```

In this example:

- Line **1** is a comment and is not used for mapping.
- Line **2** is blank and is ignored. All blanks preceding or following a pattern string are ignored, as well as those between the equals sign and the return code value.
- Line **3** matches every message starting with the string *User* and ending with the string *missing*.
- Line **4** matches every message starting with three asterisks (*) followed by a blank. When you use the asterisk in this way and not as a wildcard, you must escape it with a backslash.
- Lines **5** through **7** contain a pattern taking several lines. It matches the same messages as the pattern of line **3**.

Considerations

Note the following facts:

- The order of the pattern lines is important because the first matching pattern line is used to build the return code value.
- Empty pattern strings (“”) are ignored by the pattern matching procedure.

For example, the following is a valid pattern sequence. The first line is more restrictive than the second line.

```
“625 “User * missing = 104
“ “User * missing = 102
```

The following pattern sequence is formally valid, but the second pattern line is never used. Because the first line is more general, it is always matched first.

```
“ “User * missing = 102
“625 “User * missing = 104
```

Return code mapping for psagent

For the PeopleSoft access method, you can write return code mapping files associating the internal states listed in Table 139: Job states and return codes for the PeopleSoft access method on page 1116.

When no return code mapping files are defined, or when a string returned by the access method does not satisfy any of the matching patterns of the mapping file, the access method uses the respective standard return codes listed in the tables.

Table 139. Job states and return codes for the PeopleSoft access method

psagent job state	psagent return code
"CANCEL"	1
"DELETE"	2
"ERROR"	3
"HOLD"	4
"QUEUED"	5
"INITIATED"	6
"PROCESSING"	7
"CANCELED"	8
"SUCCESS"	9
"NO SUCCESSPOSTED"	10
"POSTED"	11
"NOT POSTED"	12
"RESEND"	13
"POSTING"	14
"GENERATED"	15

Return code mapping for r3batch

About this task

Using return code mapping with `r3batch` can be useful in overcoming differences in the return code mechanisms of R/3, which returns a mixture of messages and numbers, and of HCL Workload Automation, which handles exclusively numeric return codes. By customizing the return code mapping files listed in Return code mapping file names for r3batch on page 1118, you can map messages from R/3 logs, spool lists, and exceptions from RFC function modules into return code values that HCL Workload Automation can handle.

Note that when you do not use this feature, `r3batch` does not send any return codes to HCL Workload Automation. In this case, HCL Workload Automation displays only the `r3batch` exit code, which cannot be used to set up `rccondsucc` conditions.

The return code mapping mechanism works as follows:

1. `r3batch` reads the output retrieved from the R/3 system (R/3 job log, process chain log, spool list, and so on appended to the `stdlist` of the related HCL Workload Automation job).
2. Following your specifications in the `rcm` files, the R/3 return messages or codes are mapped into your custom return codes and passed on to HCL Workload Automation.
3. These return codes are used together with the `rccondsucc` keyword set in the extended agent job definition and handled accordingly. Return code mapping is meaningful only if you use the return codes to write the expressions that determine job completion. Conversely, in the case of this extended agent, the use of `rccondsucc` is significant only if HCL Workload Automation gets return codes (not exit codes) from the access method.

To use the return code mapping feature:

- Leave the value of the `rcmap` option as `ON` (this is the default).
- Depending on which R/3 logs you want `r3batch` to read and map, leave the default settings of the `retrieve_joblog`, `retrieve_pchainlog`, and `retrieve_spoollist` options as `ON` and manually create the corresponding `rcm` files.
- If you want to map messages from the R/3 syslog, set the `log_r3syslog` option to `ON` and manually create the corresponding `rcm` file.

When setting up your return code mapping for `r3batch`, consider the following:

- You can define any return code numbers for your use because there are no reserved return codes for the access method or for HCL Workload Automation.
- Mapping files are scanned sequentially: the first match found performs the corresponding mapping. When you define a mapping file, write the most restrictive strings first.
- When you define a mapping file, remember that the R/3 log messages are read in their entirety. If you want to map only a part of the entry, you must use the wildcard characters.
- If two lines match two different patterns, then the return code is set to the higher value. In general the return code is set to the highest value among the ones yielded by the matched patterns. This is shown in the following example:

  The job log returned after job PAYT410 has run is:

  ```
  *** ERROR 778 ***  EEWO0778E Failed to modify the job PAYT410 with job id
  ***       05710310.

  *** ERROR 176 ***  EEWO0176E Failed to add step 1.

  *** ERROR 552 ***  EEWO0552E The R/3 job scheduling system has found an
  ***       error for user name * and job name PAYT410. Please check R/3
  ***       syslog.

  *** ERROR 118 ***  EEWO0118E Execution terminated. Could not create and
  ***       start an instance of the R/3 batch job.
  ERROR LEVEL=118
  ```

  and the system log contains the following line:

  ```
  |011:05:12|MAESTRO|SAPMSSY1|EFT|> Step 1 contains illegal values
  ```

The `r3batch-joblog.rcm` file contains the following matching line:

```
"118""*"=100
```

while the `r3batch-syslog.rcm` file contains the following matching line:

```
"*MAESTRO*Step 1 contains illegal values"=9999
```

In this case, the return code sent back to HCL Workload Automation is 9999 because it is the higher of the two matching patterns.

- If no matching takes place, no return code is sent to HCL Workload Automation.

Return code mapping file names for r3batch

program_pattern

`"*"`

message_number_pattern

`"550"`

message_id_pattern

`"*"`

`r3batch-pchainlog.rcm`

Maps messages from the protocol of a Process Chain into return code values. If this file is not present, the messages in the protocol are ignored.

The format of the mapping file is:

```
 message_number_pattern
[message_id_pattern[message_variable1[message_variable2
[message_variable3[message_variable4[message_type]]]]]]=RCvalue
```

`r3batch-spoollist.rcm`

Maps messages in the job spool list of an R/3 job into return code values. If this file is not present, the messages in the spool list are ignored.

The format of the mapping file is:

```
spool_list_row_pattern=RCvalue
```

`r3batch-syslog.rcm`

Maps messages in the syslog of an R/3 system into return code values. The R/3 system log should be checked only when R/3 returns the generic 552 error to `r3batch`.

If this file is not present, the messages in the system log are ignored.

The format of the mapping file is:

```
system_log_row_pattern=RCvalue
```

If you plan to map system log messages, be sure to set the `log_r3syslog` option of `r3batch` to `ON` (the default is `OFF`).

`r3batch-msgrc.rcm`

Maps ABAP exceptions and BAPI return codes of RFC function modules into return code values. If this file is not present, the mapping is done using a hardcoded table.

The format of the mapping file is:

```
message_number=RCvalue
```

`message_number` is the error message number. The last message number is always used. That is, if two error messages are generated, only the second one is checked against the mapping file.

Mapping return codes for intercepted jobs

About this task

To set up return code mapping for intercepted jobs, after defining the appropriate return code conditions in the `r3batch-joblog.rcm` file, do the following:

1. Create a customized template file named `rctemplate.jdf` in the following directory:

   TWA_DATA_DIR`/methods/r3batch_icp/`

   TWA_home`\methods\r3batch_icp\`

   The file must contain the following:

   ```
   alias;rccondsucc "Success Condition"
   ```

   where, the "Success Condition" must match a condition saved in the `rcm` file.
2. Modify the *XANAME*`_r3batch.icp` file, located in the same path, to refer to the `jdf` file you created in the previous step as follows:

   ```
   client job_mask user_mask rctemplate
   ```

   HCL Workload Automation manages the intercepted R/3 job as a docommand job with all the options specified in the customized `jdf` file. You can check if your intercepted job is correctly submitted by reading the job_interceptor joblog.

Configuring the tracing utility

Learn how to configure the trace utility for all the access methods.

HCL Workload Automation logs all the processing information in the following configuration file:

TWA_DATA_DIR`/methods/`*accessmethod*`.properties`

TWA_home`\methods\`*accessmethod*`.properties`

Note: If you delete this file accidentally, HCL Workload Automation creates a new file with all the default values and contains the following comment:

```
# This file was automatically created using the default values.
```

Customizing the .properties file

About this task

Depending on the access method you are working with, customize the trace parameters in the following properties files:

psagent.properties

For the PeopleSoft access method.

r3batch.properties, r3evmon.properties

For the SAP R/3 access method.

With this access method, you can also specify debug and trace parameters in the single job definitions. See Creating SAP Standard R/3 jobs from the Dynamic Workload Console on page 1172 and Task string to define SAP jobs on page 1180.

For each .properties file you can customize the following parameters:

***accessmethod*.trace.tracers.level**

Specify the level of tracing you want to set. Possible values are:

DEBUG_MIN

Only error messages are written in the trace file. This is the default.

DEBUG_MID

Informational messages and warnings are also written in the trace file.

DEBUG_MAX

A most verbose debug output is written in the trace file.

The value you set in the .properties file applies to all the jobs of the corresponding access method. To specify a different trace setting for a particular job, specify the following option in the job definition:

```
-tracelvl=(1|2|3)
```

where:

- `1 = DEBUG_MIN`
- `2 = DEBUG_MID`
- `3 = DEBUG_MAX`

Note: When making changes to the trace level setting, the changes are effective immediately after saving the .properties file. No restart is required.

***accessmethod*.trace.handlers.traceFile.fileDir**

Specifies the path where the trace file is created. Depending on the access method, the default is:

SAP R/3

On UNIX® operating systems

`*TWA_DATA_DIR*/methods/traces`

On Windows™ operating systems

`TWA_home\methods\traces`

All other access methods

On UNIX operating systems

`*TWA_DATA_DIR*/methods`

On Windows operating systems

`*TWA_home*\methods`

Ensure that the new path you specify has already been created as a fully qualified path with write permissions.

Traces are written in XML format. Always use slashes (or backslashes) when you specify a new path, even if you are working on Windows™ operating systems.

The trace files give information about the method execution to the desired level of detail. The minimum trace level is always on, to guarantee a First-Failure Data Capture (FFDC) ability. The trace file name is:

trace-psagent.log

For the PeopleSoft access method.

trace-r3batch.log, trace-*XAname*-r3evmon.log

For the SAP R/3 access method.

***accessmethod*.trace.tracers.logging**

Specifies to enable or disable the trace utility. Possible values are:

true

To enable the trace utility. This is the default value.

false

To disable the trace utility. If you set this parameter to **false**, no traces are written in the trace-*accessmethod*.log file even if there are problems.

r3batch.trace.handlers.traceFile.maxFiles

The maximum number of trace files that are created before the oldest one is deleted. If this parameter is set to 1, the current trace file is never replaced and can grow without limit.

r3batch.trace.handlers.traceFile.maxFileSize

The maximum size (in bytes) that the trace file can reach before it is renamed and a new trace file is created. This parameter is valid only if the r3batch.trace.handlers.traceFile.maxFiles is set to a value greater than 1.

Configuration file example for the SAP access method

The following `r3batch.properties` file is an example of a configuration file for the SAP access method with the following characteristics:

-

  The level of tracing set is DEBUG_MID. This means that not only error messages but also informational messages and warnings are written in the trace file.
- The trace file is created in the `/home/maestro/methods` directory.
- The tracing process creates three trace files, whose maximum size can be 10 MB.

```
r3batch.organization=ABC
r3batch.product=IWS
r3batch.component=R3BATCH
r3batch.trace.tracers.level=DEBUG_MID
r3batch.trace.tracers.listenerNames=r3batch.trace.handlers.traceFile
r3batch.trace.tracers.logging=true
r3batch.trace.handlers.traceFile.fileDir=/home/maestro/methods
r3batch.trace.handlers.traceFile.formatterName=r3batch.trace.formatter
r3batch.trace.handlers.traceFile.maxFileSize=104805100
r3batch.trace.handlers.traceFile.maxFiles=3
```

Integration with SAP



The following sections give you information about HCL Workload Automation for SAP, the SAP access method and job plug-ins, and how to schedule jobs by using the SAP Solution Manager.

HCL Workload Automation is certified for the following SAP integrations:

- SAP Certified Integration with SAP NetWeaver
- SAP Certified Integration with SAP S/4HANA

Through the integrations, HCL Workload Automation can access SAP BW on HANA Process Chains to handle and manage them externally and can invoke and track InfoPackages.

Introducing HCL Workload Automation for SAP

Improve SAP operations and enable business growth with HCL Workload Automation.

Use HCL Workload Automation for SAP, to create, schedule, and control SAP jobs using the job scheduling features of HCL Workload Automation. HCL Workload Automation supported agent workstations help extend the product scheduling capabilities to SAP through the R/3 batch access method . In addition, you can define HCL Workload Automation job plug-ins for SAP BusinessObjects BI and SAP PI Channel. With the SAP Solution Manager integration, you can have the HCL Workload Automation engine run job scheduling tasks available from the Solution Manager user interface.

HCL Workload Automation provides a single and simplified point of planning, control and optimization of end-to-end production services across heterogeneous IT infrastructures. It enables you to control SAP operations from z/OS indifferently.

HCL Workload Automation is certified by SAP for the following SAP interfaces:

- S/4-BC-XBP – BACKGROUND PROCESSING JOB SCHEDULING FOR SAP S/4HANA
- SAP BW/4HANA -Job Scheduling (BW4-SCH)

To understand if a SAP System is compatible with HCL Workload Automation, check if your system exposes the following interfaces:

- BC-XBP 6.10 (V2.0) - Background Processing
- BC-XBP 7.00 (V3.0) - Background Processing
- BW-SCH 3.0 – Business Information Warehouse for SAP BW/4HANA
- BW4-SCH - Business Information Warehouse for SAP BW/4HANA

For detailed information about supported SAP interfaces, prerequisite SAP notes, and supported SAP software versions, see the Supported Software report and select the **Supported Software** tab. In addition, see the dedicated SAP section in the HCL Workload Automation Detailed System Requirements.

Features

Table 140: HCL Workload Automation for SAP features on page 1126 shows the tasks you can perform with HCL Workload Automation for SAP either in a distributed or an end-to-end environment, or both.

Table 140. HCL Workload Automation for SAP features

Feature	Distributed environment	End-to-end
Using HCL Workload Automation standard job dependencies and controls on SAP jobs	✓	✓
Listing jobs, defining jobs, variants, and extended variants using the HCL Workload Automation interface	✓	✓
Defining jobs and variants dynamically at run time	✓	✓
Scheduling SAP jobs to run on specified days and times, and in a prescribed order	✓	✓
Scheduling SAP BusinessObjects Business Intelligence (BI) jobs to gain greater control over your SAP BusinessObjects Business Intelligence (BI) reports through the HCL Workload Automation plug-in for SAP BusinessObjects Business Intelligence (BI).	✓	✓
Scheduling SAP Process Integration (PI) Channel jobs to control communication channels between the Process Integrator and a backend SAP R/3 system.	✓	✓
Scheduling and monitoring job scheduling tasks available from the SAP Solution Manager user interface.	✓	
Defining the national language support options	✓	✓
Using the SAP Business Warehouse Support functions	✓	✓
Customizing job execution return codes	✓	✓
Using SAP logon groups for load balancing and fault-tolerance	✓	✓
Using Business Component-eXternal Interface Background Processing (XBP 2.0 and later) interface support to:	• Collect intercepted jobs • Track child jobs • Keep all job attributes when you rerun a job • Raise events	• Track child jobs • Keep all job attributes when you rerun a job • Raise events

Table 140. HCL Workload Automation for SAP features (continued)

Feature	Distributed environment	End-to-end
Using Business Component-eXternal Interface Background Processing (XBP 3.0) interface support to:	• Create criteria profiles to log raised events, reorganize the event history, and intercept and relaunch jobs, according to the criteria you specify. • SAP application log and application return code • Spool list request and display for jobs that have run. • Temporary variants	• Create criteria profiles to log raised events, reorganize the event history, and intercept and relaunch jobs, according to the criteria you specify. • SAP application log and application return code • Spool list request and display for jobs that have run. • Temporary variants
Assigning an SAP job to a server group, for batch processing	✓	✓
Exporting SAP factory calendars and adding their definitions to the HCL Workload Automation database	✓	
Defining internetwork dependencies and event rules for HCL Workload Automation based on SAP events	✓	
Defining event rules based on IDoc records	✓	
Defining event rules based on CCMS Monitoring Architecture alerts	✓	
Rerunning a job that submits a process chain from a specific process, from failed processes, or as a new instance	✓	✓
Displaying the details of a job that submits a process chain	✓	✓
Enabling job throttling	✓	✓

Access method for SAP

The SAP R/3 batch access method enables communication between an external SAP R/3 system and HCL Workload Automation and provides a single point of entry for automating the launching of jobs, monitoring the status of jobs and managing exceptions and recovery.

Using the SAP access method you can run and monitor SAP jobs from the HCL Workload Automation environment. These jobs can be run as part of a schedule or submitted for ad-hoc job processing. SAP extended agent or dynamic agent jobs can have all of the same dependencies and recovery options as other HCL Workload Automation jobs. SAP jobs must be defined in HCL Workload Automation to be run and managed in the HCL Workload Automation environment.

HCL Workload Automation provides a single and simplified point of planning, control and optimization of end-to-end production services across heterogeneous IT infrastructures. It enables you to control SAP operations from z/OS indifferently.

Scheduling process for the agent workstation hosting the r3batch access method

HCL Workload Automation launches jobs in SAP using HCL Workload Automation jobs defined to run on a supported agent workstation.

Supported agent workstations include:

- dynamic agents
- extended agents
- HCL Workload Automation Agents

The supported agent workstations use the access method, `r3batch`, to pass SAP job-specific information to predefined SAP instances. The access method uses information provided in an options file to connect and launch jobs on an SAP instance.

Multiple extended agent workstations can be defined to use the same host, by using multiple options files. Using the SAP extended agent workstation unique identifier as a key, `r3batch` uses the corresponding options file to determine which instance of SAP will run the job. See UNIQUE_ID on page 1085 for more information about identifying the extended agent workstation unique identifier. It makes a copy of a template job in SAP and marks the job as "`scheduled`". It then monitors the job through to completion, writing job progress and status information to a job standard list on the host workstation.

On dynamic agent workstations, more than one options file can be associated to the workstation.

For more information about job management, refer to the *HCL Workload Automation: User's Guide and Reference*.

For more detailed information about configuration files on extended agents and dynamic agents, see Configuring the SAP access method on page 1145.

Roles and responsibilities

In a typical enterprise, different users contribute to the implementation and operation of the product. Table 141: Roles and responsibilities in HCL Workload Automation for SAP on page 1129 describes the roles and responsibilities of all users in the process model, showing the tasks they perform.

Table 141. Roles and responsibilities in HCL Workload Automation for SAP

User role	User task
HCL Workload Automation administrator	• Creating the HCL Workload Automation RFC user on page 1132 • Creating the authorization profile for the HCL Workload Automation user on page 1132 • Copying the correction and transport files on page 1136 • Importing ABAP/4 function modules into SAP on page 1137
HCL Workload Automation configurator	• Changing the HCL Workload Automation RFC user ID password on page 1141 • Migrating from previous versions on page 1144 • Print parameter and job class issues on page 1143 • Defining the configuration options on page 1147 • Connecting to the SAP system on page 1167 • Creating SAP Standard R/3 jobs from the Dynamic Workload Console on page 1172 • Using the BDC Wait option on page 1222 • Implementing job interception on page 1223 • Defining user authorizations to manage SAP Business Warehouse InfoPackages and process chains on page 1234 • Setting and using job throttling on page 1251 • Exporting SAP factory calendars on page 1257 • Setting National Language support options on page 1295
HCL Workload Automation developer	• Editing a standard SAP job on page 1179 • Task string to define SAP jobs on page 1180

Table 141. Roles and responsibilities in HCL Workload Automation for SAP (continued)

User role	User task
	• Displaying details about a standard SAP job on page 1190 • Verifying the status of a standard SAP job on page 1191 • Deleting a standard SAP job from the SAP database on page 1192 • Balancing SAP workload using server groups on page 1192 • Defining SAP jobs dynamically on page 1199 • Managing SAP R/3 Business Warehouse InfoPackages and process chains on page 1235
HCL Workload Automation developer	• Defining internetwork dependencies and event rules based on SAP background events on page 1261 • Defining event rules based on IDoc records on page 1271 • Defining event rules based on CCMS Monitoring Architecture alerts on page 1281
HCL Workload Automation operator	• Rerunning a standard SAP job on page 1197 • Mapping between HCL Workload Automation and SAP job states on page 1193 • Raising an SAP event on page 1195 • Killing an SAP job instance on page 1195 • Displaying details about a process chain job on page 1241

Configuring user authorization (Security file)

HCL Workload Automation manages security through the use of a configuration file, the security file. In the security file, you specify which scheduling objects a user can manage and how. You define these settings by writing user definitions. A user definition is an association between a name and a set of users, the objects they can access, and the actions they can perform on the specified objects.

For more detailed information about the security file, security file syntax, and how to configure the security file, see "Configuring user authorization (Security file)" in the *Administration Guide*.

The following table displays the access keywords required to grant authorization to access and work with SAP scheduling objects assigned to HCL Workload Automation users.

Table 142. Access keywords for activities with SAP scheduling objects

Activity		Access keywords required
Dynamic Workload Console	Define or search for SAP jobs on an extended agent workstation.	display on the workstation
	Retrieve the spool list on an extended agent workstation.	display on the workstation
	Rerun from a step on an extended agent.	rerun on the job
	Define or search for SAP jobs on a dynamic agent workstation, pool, or dynamic pool.	display and run on the workstation
	Retrieve the spool list on a dynamic agent workstation, pool, or dynamic pool.	display and run on the job
	Rerun from a step on a dynamic agent workstation, pool, or dynamic pool.	rerun on the job

Configuring the SAP environment

You must configure the SAP environment before using the SAP access method.

To communicate and manage the running of jobs on SAP systems using the access method for SAP, complete the following configuration steps in the SAP environment.

The steps require that you have knowledge of an SAP Basis Administrator.

Overview

About this task

Here is an overview of the customization procedure:

1. Create a new user ID for RFC communications in SAP for HCL Workload Automation.
2. Create the authorization profile as described in Creating the authorization profile for the HCL Workload Automation user on page 1132.
3. Copy the correction and transport files from the HCL Workload Automation server to the SAP server.
4. Import the correction and transport files into SAP and verify the installation.

Results



Creating the HCL Workload Automation RFC user

About this task

For HCL Workload Automation to communicate with SAP, you must create a user ID in SAP for HCL Workload Automation batch processing. For security reasons, use a new user ID rather than an existing one.

1. Create a new RFC user ID.
2. Give this new RFC user ID the following attributes:
   - A user type of **CPIC**, **Communications**, or **DIALOG**, depending on the SAP release.
   - A password at least six characters in length. HCL Workload Automation requires this password to start or monitor SAP jobs. If this password changes in SAP, you must update the options file used by `r3batch` with the new password.
   - The appropriate security profiles, depending on your version of SAP.

Creating the authorization profile for the HCL Workload Automation user

The two ways to create the authorization profile for the HCL Workload Automation user.

There are two alternative ways to perform this task:

- Using transaction `su02` and manually creating the profile.
- Using the Profile Generator (transaction `PFCG`).

Using transaction su02 and manually creating the profile

About this task

Perform the following steps:

1. Write a profile name, for example Z_TWS, and a description.
2. Manually add the authorizations according to the following table:

Object	Description	Authorization
S_ADMI_FCD	System authorizations	S_ADMI_ALL
S_APPL_LOG	Application logs	S_APPL_L_E2E
S_BTCH_ADM	Background processing: Background administrator	S_BTCH_ADM
S_BTCH_JOB	Background processing: Operations on background jobs	S_BTCH_ALL
S_BTCH_NAM	Background processing: Background user name	S_BTCH_ALL
S_DEVELOP	ABAP Workbench: full authorization to modify objects of type PROG	E_ABAP_ALL
S_LOG_COM	Authorization to run external commands	S_LOGCOM_ALL
S_PROGRAM	ABAP: program run checks	S_ABAP_ALL
S_RFC	Authoritation. check for RFC access	S_RFC_ALL
S_RZL_ADM	CCMS: System Administration	S_RZL_ALL
S_SPO_ACT	Spool: Actions	S_SPO_ALL
S_SPO_DEV	Spool: Device authorizations	S_SPO_DEV_AL
S_XMI_LOG	Internal access authorizations for XMI log	S_XMILOG_ADM

Object	Description	Authorization
S_XMI_PROD	Authorization for external management interfaces (XMI)	S_XMI_ADMIN

The authorizations are located in the "Basis: Administration" object class.

Depending on the version of SAP R/3, the authorization S_RFC_ALL are located either in the "Cross-application Authorization Objects" or in the "Non-application-specific Authorization Objects" object class.

3. Save the profile.
4. Go to the user maintenance panel and assign the profile to the HCL Workload Automation SAP R/3 user.
5. Save the user data.

Using transaction PFCG (Profile Generator)

About this task

Perform the following steps:

1. Write a name, for example ZTWS, in **Role Name**.
2. Click **Create Role** and write a description for the role, such as "Role for the TWS user."
3. Save the role.
4. Select **Authorizations**.
5. Click **Change Authorization Data**.
6. In the pop-up, select **Templates**.
7. Manually add the following authorization objects:

Object	Description
S_ADMI_FCD	System authorizations
S_APPL_LOG	Application logs
S_BTCH_ADM	Background processing: Background administrator
S_BTCH_JOB	Background processing: Operations on background jobs
S_BTCH_NAM	Background processing: Background user name

Object	Description
S_PROGRAM	ABAP: Program run checks
S_DEVELOP	ABAP Workbench: full authorization to modify objects of type PROG
S_LOG_COM	Authorization to run external commands
S_RFC	Authorization check for RFC access
S_RZL_ADM	CCMS: System Administration
S_SPO_ACT	Spool: Actions
S_SPO_DEV	Spool: Device authorizations
S_XMI_LOG	Internal access authorizations for XMI log
S_XMI_PROD	Authorization for external management interfaces (XMI)

8. Fill in the values according to the following scheme:

Object	Description
S_ADMI_FCD	System authorizations ◦ System administration function: Full authorization
S_APPL_LOG	Activity: Display ◦ Application log Object name: Full authorization ◦ Application log subobject: Full authorization
S_BTCH_ADM	Background processing: Background administrator ◦ Background administrator ID: Full authorization
S_BTCH_JOB	Background processing: Operations on background jobs ◦ Job operations: Full authorization ◦ Summary of jobs for a group: Full authorization
S_BTCH_NAM	Background processing: Background user name ◦ Background user name for authorization check: Full authorization
S_PROGRAM	ABAP: Program run checks ◦ User action ABAP/4 program: Full authorization ◦ Authorization group ABAP/4 program: Full authorization
S_RFC	Authorization check for RFC access ◦ Activity: Full authorization ◦ Name of RFC to be protected: Full authorization ◦ Type of RFC object to be protected: Full authorization
S_RZL_ADM	Activity: Full authorization

Object	Description
S_SPO_ACT	Spool: Actions ◦ Authorization field for spool actions: Full authorization ◦ Value for authorization check: Full authorization
S_SPO_DEV	Spool: Device authorizations ◦ Spool - Long device names: Full authorization
S_XMI_LOG	Internal access authorizations for XMI log ◦ Access method for XMI log: Full authorization
S_XMI_PROD	Authorization for external management interfaces (XMI) ◦ XMI logging - Company name: ABC* ◦ XMI logging - Program name: MAESTRO* ◦ Interface ID: Full authorization

9. Save the authorizations.
10. Generate a profile. Use the same name that you wrote in **Role Name**.
11. Exit the authorization management panel and select **User**.
12. Add the HCL Workload Automation user to the role.
13. Save the role.

Copying the correction and transport files

About this task

The setup file loads four correction and transport files into the HCL Workload Automation home directory. Copy these correction and transport files to the SAP server and import them into the SAP database, as follows:

1. On your SAP database server, log on to the SAP system as an administrator.
2. Copy the control file and data file from the `methods` directory and to the following directories on your SAP database server:

   ```
   copy control_file /usr/sap/trans/cofiles/
   copy data_file /usr/sap/trans/data/
   ```

   The names of *`control_file`* and *`data_file`* vary from release to release. The files are located in the `methods` directory: UNIX®: *`TWA_DATA_DIR`*`\methods`, Windows™: *`TWA_home`*`\methods`, and have the following file names and format:

   For SAP releases earlier than 6.10:

   - `K000`*`xxx`*`.TV1` (control file) and `R000`*`xxx`*`.TV1` (data file)
   - `K900`*`xxx`*`.TV2` (control file) and `R900`*`xxx`*`.TV2` (data file)

For SAP releases 6.10, or later:

- `K9000xx.TV1` (control file) and `R9000xx.TV1` (data file)
- `K9007xx.TV1` (control file) and `R9007xx.TV1` (data file)

where *x* is a digit generated by the SAPsystem.

Specifically, for HCL Workload Automation version 10.2.8 the following files are used:

For SAP releases earlier than 6.10:

- `K000538.TV1` (for standard jobs scheduling)
- `R000538.TV1` (for standard jobs scheduling)
- `K900294.TV2` (for IDoc monitoring and job throttling)
- `R900294.TV2` (for IDoc monitoring and job throttling)

For SAP releases 6.10, or later:

- `K900044.TV1` (for standard jobs scheduling)
- `R900044.TV1` (for standard jobs scheduling)
- `K900751.TV1` (for IDoc monitoring and job throttling)
- `R900751.TV1` (for IDoc monitoring and job throttling)

Importing ABAP/4 function modules into SAP

How to generate, activate and commit new ABAP/4 modules to a SAP system.

About this task

This section describes the procedure to generate, activate, and commit new ABAP/4 function modules to your SAP system and several new internal tables. You do not modify any existing SAP system objects. For information about the supported SAP R/3 releases, see the System Requirements Document at Product Requirements.

The number of ABAP/4 modules that you install with the import process varies from release to release. The modules are installed in the `methods` directory and have the following file names and format:

- `K9000xx.TV1` (function modules for standard jobs scheduling extensions)
- `K9007xx.TV1` (function modules for IDoc monitoring and job throttling)

where *x* is a digit generated by the SAP system. The methods directory is located in:

On UNIX operating systems

`TWA_DATA_DIR/methods`

On Windows operating systems

`TWA_home\methods`

Before importing the ABAP/4 function modules, review the considerations documented in Migrating from previous versions on page 1144.

To import ABAP/4 function modules into SAP:

1. Change to the following directory:

   ```
   cd /usr/sap/trans/bin
   ```

2. Add the transport file to the buffer:

   ```
   tp addtobuffer transport sid
   ```

   where:

   transport

   The transport request file.

   sid

   The SAP system ID.

   For example, if the transport file in the *TWA_home*`\methods` directory is named `K9000`*xxx*`.TV1`, the transport request is tv1K9000*xxx*.

3. Run the **tp tst** command to test the import:

   ```
   tp tst transport sid
   ```

   After running this command, examine the log files in the `/user/sap/trans/log` directory for error messages. Warnings of severity level 4 are normal.

   If there are errors, check with a person experienced in correction and transport, or try using unconditional modes to do the import.

4. Run the following command to import all the files in the buffer:

   ```
   tp import transport sid
   ```

   This command generates the new ABAP/4 modules and commits them to the SAP database. They automatically become active.

   After running this command, examine the log files located in the `/user/sap/trans/log` directory for error messages. Warnings of severity level 4 are normal.

   If a problem is encountered, use unconditional mode when running this step:

   ```
   tp import transport sid U126
   ```

5. When the import is complete, check the log files located in the `/usr/sap/trans/log` directory to verify that the ABAP/4 modules were imported successfully.

   If you apply the standard transport and the IDOC transport, 26 ABAP/4 modules are installed by the import process. For a list of the transport files to be used, refer to Importing ABAP/4 function modules into SAP on page 1137. Table 143: ABAP/4 modules installed on page 1139 lists the ABAP modules installed.

Table 143. ABAP/4 modules installed

ABAP/4 module	Installed?
ENQUEUE_/IBMTWS/EQ_XAPPL	✓
DEQUEUE_/IBMTWS/EQ_XAPPL	✓
/IBMTWS/UNREGISTER_XAPPL	✓
/IBMTWS/GET_XAPPL_REGISTRATION	✓
/IBMTWS/MODIFY_JOB_CLASS	✓
/IBMTWS/REGISTER_XAPPL	✓
J_1O1_BDC_STATUS	✓
J_1O1_DATE_TIME	✓
J_1O1_IDOC_SELECT	✓
J_1O1_JOB_ADJUST_CLIENT	✓
J_1O1_JOB_FIND	✓
J_1O1_JOB_FINDALL	✓
J_1O1_JOB_HAS_EXTENDED_VARIANT	✓
J_1O1_JOB_LOG	✓
J_1O1_RAISE_EVENT	✓
J_1O1_REPORT_ALL_SELECTIONS	✓
J_1O1_REPORT_GET_TEXTPOOL	✓
J_1O1_VARIANT_COPY	✓
J_1O1_VARIANT_CREATE	✓
J_1O1_VARIANT_DELETE	✓
J_1O1_VARIANT_EXISTS	✓
J_1O1_VARIANT_GET_DEFINITION	✓
J_1O1_VARIANT_GET_HELP_VALUES	✓
J_1O1_VARIANT_MAINTAIN_CNT_TBL	✓
J_1O1_VARIANT_MAINTAIN_SEL_TBL	✓
J_1O1_VARIANT_MODIFY	✓

Table 144: ABAP/4 modules contents on page 1140 shows the contents of the ABAP modules for the IDoc records and job throttling feature.

Table 144. ABAP/4 modules contents

Object	Description	Used by...
/IBMTWS/	Type = Development Namespace. For HCL Workload Automation.	Internal use only
/IBMTWS/EQ_XAPPL	Type = Lock Object. Synchronizes the job throttler instances and job interception collector jobs that are running against the same SAP system.	Job throttling Job interception
/IBMTWS/GET_ XAPPL_REGISTRATION	Type = Function Module. It is used to query for existing external application registration data in table IBMTWS/XAPPL, for example the registration data of a job throttler instance or job interception collector.	Job throttling Job interception
/IBMTWS/MODIFY_JOB_CLASS	Type = Function Module. Modifies the job class of an intercepted job that is controlled by the job throttler. For details, see Step 3. Enabling job class inheritance on page 1253.	Job throttling Job interception
/IBMTWS/REGISTER_XAPPL	Type = Function Module. Registers an external application, for example the job throttler.	Job throttling Job interception
/IBMTWS/TWS4APPS	Type = Function group. For HCL Workload Automation.	Internal use only
/IBMTWS/UNREGISTER_XAPPL	Type = Function Module. Unregisters an external application, for example the job throttler.	Job throttling Job interception
/IBMTWS/XAPPL	Type = Table. Stores the registration data of external applications. An external application can be a job throttler instance or a job interception collector.	Job throttling Job interception
J_1O1_IDOC_SELECT	Type = Function Module. Selects IDoc records from SAP internal tables. For details, see Defining event rules based on IDoc records on page 1271.	IDoc event rules
J_1O1_TWS_EDIDC	Type = Data structure in FM interface	Function module J_1O1_IDOC_SELECT

Object	Description	Used by...
J_1O1_TWS_IDOC_ SELECTION	Type = Data structure in FM interface	Function module J_1O1_IDOC_SELECT
J_1O1_TWS_STATE_ SELECTION	Type = Data structure in FM interface	Function module J_1O1_IDOC_SELECT

To uninstall the transport you can use the STMS transaction see: Deleting Imported Requests from the Import Queue.

Changing the HCL Workload Automation RFC user ID password

About this task

If the password of the HCL Workload Automation Remote Function Call (RFC) user ID is modified after the initial installation, the options file used by `r3batch` must be updated to reflect this change.

On UNIX® operating systems, log on as root to the system where HCL Workload Automation is installed.

On Windows™ operating systems, log on as an administrator and start a DOS shell on the system where HCL Workload Automation is installed.

1. Generate an encrypted version of the new password using the enigma command, located in *TWA_home*`/methods`. To do this in a command shell, type:

   ```
   enigma new_password
   ```

   where

   new_password

   is the new password for the HCL Workload Automation RFC user ID.

   The enigma command prints an encrypted version of the password.

2. Copy the encrypted password into the options file, which is located in the following directory:

   On UNIX operating systems

   TWA_DATA_DIR`/methods`

   On Windows operating systems

   TWA_home`\methods`

   The file can be edited with any text editor.

Results

Ensure that you copy the password exactly, preserving uppercase, lowercase, and punctuation. The encrypted password looks similar to:

```
{aes}hyb/LQNyVzIf9oA8/xgY+CSqAuAh7+CvTT7HuDpdiu5YUAH0KJEHJtA=
```

If the encrypted password is not entered correctly, HCL Workload Automation is unable to start or monitor SAP R/3 batch jobs.

Securing data communication

You can increase the security of your SAP system through the use of an external security product. Secure Network Communications (SNC) can integrate the external security product with the SAP system.

Data communication paths between the client and server components of the SAP system that use the SAP protocols RFC or DIAG are more secure with SNC. The security is strengthened through the use of additional security functions provided by an external product that are otherwise not available with SAP systems.

SNC provides security at application level and also end-to-end security. HCL Workload Automation is extended to read SNC configuration parameters and forward them to the SAP RFC communication layer used when logging in to the SAP system. HCL Workload Automation does not provide or ship SNC software but instead enables the use of third-party SNC products to secure the RFC communication.

Levels of protection

You can apply one of the following levels of protection:

Authentication only

This is the minimum level of security protection available with SNC. The system verifies the identity of the communication partners.

Integrity protection

The system detects any changes to the data which might have occurred between the two components communicating.

Privacy protection

This is the maximum level of security protection available with SNC. The system encrypts the messages being transferred so that any attempt to eavesdrop is useless. Privacy protection also includes integrity protection of the data.

Example

The following options in the local options file are used to configure SNC for HCL Workload Automation:

- **r3snclib**: the path and file name of the SNC library.
- **r3sncmode**: enables or disables SNC between r3batch and the SAP R3 system.
- **r3sncmyname**: the name of the user sending the RFC for SNC.
- **r3sncpartnername**: the SNC name of the SAP R3 communication partner (application server).
- **r3sncqop**: the SNC protection level.

See Defining the local options on page 1150 for a description of these options in the local options file.

Print parameter and job class issues

The workstation running the r3batch access method for SAP uses the official RFC interfaces of SAP for job scheduling. When you migrate from previous versions of SAP, there can be problems with print parameters in jobs launched by HCL Workload Automation. This is because of limitations in the RFC interfaces.

These limitations are no longer true with XBP 2.0 and later.

The following is a list of print parameters supported by BAPI XBP 1.0 for SAP release 4.6*x* and later:

- archiving mode
- authorization
- columns
- delete after output
- lines
- number of copies
- output device
- print immediately
- recipient
- sap cover page
- selection cover page
- spool retention period

To resolve the loss of print parameters when copying a job, install the appropriate SAP Support Package as stated in the SAP notes 399449 and 430087.

The same applies to the job class. Official SAP interfaces only allow class C jobs. Installing the SAP Support Package also resolves this issue.

Unicode support

Access method for SAP supports the Unicode standard.

What is Unicode

Unicode was devised to address the problem caused by the profusion of code sets. Since the early days of computer programming hundreds of encodings have been developed, each for small groups of languages and special purposes. As a result, the interpretation of text, input, sorting, display, and storage depends on the knowledge of all the different types of character sets and their encodings. Programs are written to either handle one single encoding at a time and switch between them, or to convert between external and internal encodings.

The problem is that there is no single, authoritative source of precise definitions of many of the encodings and their names. Transferring text from one computer to another often causes some loss of information. Also, if a program has the code and the data to perform conversion between many subsets of traditional encodings, then it needs to hold several Megabytes of data.

Unicode provides a single character set that covers the languages of the world, and a small number of machine-friendly encoding forms and schemes to fit the needs of existing applications and protocols. It is designed for best interoperability with both ASCII and ISO-8859-1, the most widely used character sets, to make it easier for Unicode to be used in applications and protocols.

Unicode makes it possible to access and manipulate characters by unique numbers, their Unicode code points, and use older encodings only for input and output, if at all. The most widely used forms of Unicode are:

- UTF-32, with 32-bit code units, each storing a single code point. It is the most appropriate for encoding single characters.
- UTF-16, with one or two 16-bit code units for each code point. It is the default encoding for Unicode.
- UTF-8, with one to four 8-bit code units (bytes) for each code point. It is used mainly as a direct replacement for older MBCS (multiple byte character set) encodings.

Unicode support on SAP

Starting with SAP version 4.7 (R/3 Enterprise), Unicode is used on all layers of the SAP system:

- UTF-8, UTF-16, and UTF-32 on the database
- UTF-16 on the application server and graphical user interface

`r3batch` uses the UTF-8 code page internally. Because it communicates with SAP at the application server layer, it uses UTF-16 when communicating with Unicode-enabled SAP systems.

To use Unicode support, the following conditions must be met:

- Access method for SAP must run on a supported operating system. See the SAP support section in the HCL Workload Automation Detailed System Requirements. The product does not support Unicode on the other operating systems where it can be installed.
- The SAP systems that communicate with `r3batch` must be running Unicode-enabled SAP versions.

If these conditions are not met, you cannot use Unicode support and must make sure that `r3batch`, the Dynamic Workload Console, and the target SAP system code page settings are aligned. Use the options related to national language support described in SAP supported code pages on page 1296 .

Migrating from previous versions

This version of HCL Workload Automation for SAP supports all the SAP versions listed in HCL Workload Automation Detailed System Requirements. The HCL Workload Automation access method for SAP uses the official SAP RFC interfaces for job scheduling. These are:

- The BC-XBP 6.10 (V2.0) Interface function modules for SAP versions 6.10 and later.
- The BC-XBP 7.00 (V3.0) Interface function modules for SAP versions 7.00, Support Package 16, in addition to the BC-XBP 6.10 (V2.0) function modules.

To avoid conflicts with other vendors, the HCL Workload Automation ABAP modules now belong to the HCL Workload Automation partner namespace J_1O1_*xxx* and /IBMTWS. After you have completed the imports as described in Importing ABAP/4 function modules into SAP on page 1137, the RFC J_1O1_*xxx* function modules and the /IBMTWS function modules are installed on your system.

If you had a previous installation of HCL Workload Automation extended agent for SAP on your system, you can delete the following function modules from your SAP system:

```
Z_MAE2_BDC_STATUS
Z_MAE2_DATE_TIME
Z_MAE2_JOB_COPY
Z_MAE2_JOB_DELETE
Z_MAE2_JOB_FIND
Z_MAE2_JOB_FINDALL
Z_MAE2_JOB_LOG
Z_MAE2_JOB_OPEN
Z_MAE2_JOB_START
Z_MAE2_JOB_STATUS
Z_MAE2_JOB_STOP
```

These are old versions of the ABAP functions, which belong to the customer name space. You can also delete the function group YMA3. It is not necessary to delete the function modules and the function group, but delete them if you want to clean up your system.

To upgrade SAP, perform the following steps:

1. Uninstall the HCL Workload Automation ABAP module.
2. Upgrade SAP.
3. Install the HCL Workload Automation ABAP module.

Configuring the SAP access method

This section provides detailed information about the SAP options file creation.

The files for the SAP access method are located in the following path:

On UNIX operating systems

`TWA_DATA_DIR/methods`

On Windows operating systems

`TWA_home\methods`

If `r3batch` finds the local configuration file for an extended agent or dynamic agent, it ignores the duplicate information contained in `r3batch.opts`. If instead it does not find a local configuration file then it will use `r3batch.opts` global options file.

To successfully use the SAP access method, you must first install the SAP RFC libraries, as described in the System Requirements Document in the SAP Access Method Requirements section.

Dynamic agents

r3batch.opts

A common configuration file for the `r3batch` access method, whose settings affect all the `r3batch` instances. It functions as a "global" configuration file.

DYNAMIC_AGENT_FILE_r3batch.opts

One or more configuration files that are specific to each dynamic agent workstation within a particular installation of a `r3batch` access method. The `DYNAMIC_AGENT_FILE_r3batch.opts` is the name of the options file, where *DYNAMIC_AGENT* is not necessarily the name of the dynamic agent workstation, because the dynamic agent can have more than one `.opts` file associated. If you do not create a local options file, the global options file is used. Every dynamic agent workstation must have one or more local options file with its own configuration options.

Note: The value for *DYNAMIC_AGENT* must be written in uppercase alphanumeric characters. Double-byte character set (DBCS), Single-byte character set (SBCS), and Bidirectional text are not supported.

Note: If you have a **pool** or **dynamic pool** containing *n* agents, you must create an options file for the dynamic pool and copy it in the `TWA_home/methods` of each agent of the pool so that all members of the pool have a local options file with the same name. Then you must create another options file for the specific agent in the same directory.

For example, if the SAP R/3 access method is installed for **`AGENT1`** and **`AGENT2`** that belong to the dynamic pool ***DYN_POOL***, you need to create the following options files in the path `TWA_home/methods` on each agent:

AGENT 1

- `FILE_AGENT1_r3batch.opts`
- `FILE_DYN_POOL_r3batch.opts`

AGENT2

- `FILE_AGENT2_r3batch.opts`
- `FILE_DYN_POOL_r3batch.opts`

On dynamic workstations, you can create a new options file or edit an existing options files from a graphical user interface panel available on the General tab of the SAP job definition panels in the Dynamic Workload Console.

Extended agents

r3batch.opts

A common configuration file for the `r3batch` access method, whose settings affect all the `r3batch` instances. It functions as a “global” configuration file.

***XA_Unique_ID*_r3batch.opts**

A configuration file that is specific to each HCL Workload Automation extended agent workstation that uses the `r3batch` access method. Its options affect only the `r3batch` instance that is used by that particular workstation. It functions as a “local” configuration file.

Note: *`XA_Unique_ID`* is the unique identifier for the extended agent workstation. See UNIQUE_ID on page 1085 for more details about identifying the unique identifier for an extended agent workstation.

For example, to define two extended agent workstations with unique identifiers, `07756YBX76Z6AFX2` and `S4HANAR3BW`, that access two SAP systems, `SAP1` and `SAP2`, with the `r3batch` access method, you must define the following three configuration files:

- Global `r3batch.opts`
- Local file `07756YBX76Z6AFX2_r3batch.opts`
- Local file `S4HANAR3BW_r3batch.opts`

Defining the configuration options

This section describes the options you can configure in `r3batch.opts` and in *`XANAME`*`_r3batch.opts`.

Defining the global options

Table 145: r3batch global configuration options on page 1148 lists the options that can be specified only in the *global* configuration file `r3batch.opts`.

Table 145. r3batch global configuration options

Option	Description	Default
dep_sem_proj	(Optional) The project ID for the external dependency semaphore used for handling SAP R/3 background processing events as external follows dependencies.	`d`
icp_sem_proj	(Optional) The project ID for the job interception semaphore.	`c`
job_sem_proj	(Optional) The project ID for the job semaphore.	`a`
max_jobs_to_release_for_user	Defines the maximum number of jobs released for each user each time the release job is submitted. If this option is less than or equal to 0, the option is ignored and all jobs are released when the release job is submitted. This option is present both in global and local options. If you define it in both configuration files, the local value overrides the global one.	`ON`
primm_enable	(Optional) Enables (`ON`) the SAP print parameter PRIMM (Print Immediately) for all jobs.	`OFF`
prnew_enable	(Optional) Enables (`ON`) the SAP print parameter PRNEW (New Spool Request) for all jobs.	`OFF`
prrel_enable	(Optional) Enables (`ON`) the SAP print parameter PRREL (Immediately delete the spool output after printing) for all jobs.	`OFF`
prsap_enable	(Optional) Enables (`ON`) the SAP print parameter PRSAP (Print SAP Cover Page) for all jobs. The default value is OFF.	`OFF`
prunx_enable	(Optional) Enables (`ON`) the SAP print parameter PRUNX (Print Operating System Cover Page) for all jobs.	`OFF`
release_all_intercepted_jobs_for_request	Releases jobs for each user on a cyclic basis, based on the number of jobs specified in the **max_jobs_to_release_for_user** option. The default value is **ON**, which means that all jobs are submitted for each user:	`0`0

Table 145. r3batch global configuration options (continued)

Option	Description	Default
	• If the **max_jobs_to_release_for_user** option is less than or equal to **0**, all jobs are released for each user. • If the **max_jobs_to_release_for_user** option is higher than **0**, the specified number of jobs is submitted for each user on a cyclic basis. For example, if **max_jobs_to_release_for_user**=**5**, the first **5** jobs are submitted for each user, then the following 5 jobs for each user, and so on, until all jobs for all users are submitted. If this option is set to **OFF**, it releases for each user only the number of jobs specified in the **max_jobs_to_release_for_user** option. The remaining jobs are submitted only when a new release job is submitted: • If the **max_jobs_to_release_for_user** option is less than or equal to **0**, all jobs are released for each user. • If the **max_jobs_to_release_for_user** option is higher than **0**, only the specified number of jobs is submitted, then no other job is submitted until the new release job. If **max_jobs_to_release_for_user**=**5**, the first **5** jobs are submitted for each user, then no other job is submitted until the new release job. This option is present both in global and local options. If you define it in both configuration files, the local value overrides the global one.	
var_sem_proj	(Optional) The project ID for the variant semaphore.	b

Note: If the **max_jobs_to_release_for_user** option is higher than 0, only the specified number of jobs is submitted, then no other job is submitted until the new release job. If **max_jobs_to_release_for_user**=5, the first 5 jobs are submitted for each user, then no other job is submitted until the new release job.

Modifying the default values of the semaphore options is particularly useful when the IDs that are generated would be the same as the IDs already used by other applications.

On UNIX and Linux, to resolve the problem of duplicated IDs, HCL Workload Automation for SAP uses system-5 semaphores to synchronize critical ABAP function module calls. It uses one semaphore for job-related tasks and another one for tasks related to variant maintenance.

To synchronize on the same semaphore, the communication partners must use the same identifier. There are several ways to choose this identifier. HCL Workload Automation for SAP uses two parameters: a path name and a project ID (which is a character value). The path name parameter is the fully qualified path to the options file. The project ID is taken from

the options described in Table 145: r3batch global configuration options on page 1148. If these options are omitted, HCL Workload Automation for SAP uses default values, which work for most installations.

Note:

1. The semaphore options must be edited directly in the global options file using a text editor; you cannot use the options editor to modify these values.
2. If two semaphore options are assigned the same value, all the semaphore values are reset according to the following rule:

   job_sem_proj

   It keeps the value assigned, or its default value.

   var_sem_proj

   It is reset to the first character that, in the ASCII table, follows the value assigned to `var_sem_proj`.

   icp_sem_proj

   It is reset to the second character that, in the ASCII table, follows the value assigned to `var_sem_proj`.

   dep_sem_proj

   It is reset to the third character that, in the ASCII table, follows the value assigned to `var_sem_proj`.

Defining the local options

Table 146: r3batch local configuration options on page 1150 lists the options that you can specify only in the *local* configuration files.

Table 146. r3batch local configuration options

Option	Description
bapi_sync_level	(Optional) Specifies the synchronization level between the SAP function modules BAPI_XBP_JOB_COPY and BAPI_XBP_JOB_START_ASAP. Allowed values are:

Table 146. r3batch local configuration options (continued)

Option	Description
	high All RFC calls between BAPI_XBP_JOB_START_ASAP and BAPI_XBP_JOB_COPY are synchronized. This is the default. **medium** The RFC calls to BAPI_XBP_JOB_START_ASAP are synchronized. **low** The RFC calls are not synchronized.
blank_libpath	(Optional) Clears (`ON`) the operating system variables LD_LIBRARY_PATH and LIBPATH. The default value is `OFF`.
fn_cache_enabled	(Optional) Enables or disables the file cache on the agent. Can be ON (default value).
fn_cache_purge_interval	(Optional) Specifies the time of validity (in days) of the cached files. If it is left unspecified or set equal to or less than 0, the files are valid indefinitely.
get_job_status_retry	(Optional) Sets the number of times a Remote Function Call must be attempted to retrieve the actual status of an SAP Job. Allowed values are in the range from 1 to 9999. The default value is 5.
get_job_status_retry_delay	(Optional) Sets the number of seconds between two consecutive calls of a Remote Function Call. Allowed values are in the range from 1 to 9999.
job_duration	(Optional) Enables (`ON`) that the CPU time value in the production plan report that is run from the Dynamic Workload Console is set to the actual duration of the SAP job. Default value is `OFF`. To retrieve the job duration from the SAP system, ensure that the authorization profile contains the following authorization objects: • S_DEVELOP • S_TCODE with parameter SE38 (only for SAP 6.40 and 7.00) For details about the authorization profile, see Creating the authorization profile for the HCL Workload Automation user on page 1132.
max_jobs_to_release_for_user	Defines the maximum number of jobs released for each user each time the release job is submitted. If this option is less than or equal to 0, the option is ignored and all jobs are released when the release job is submitted. This option is present both in global and local options. If you define it in both configuration files, the local value overrides the global one.

Table 146. r3batch local configuration options (continued)

Option	Description
primm_enable	(Optional) Enables (`ON`) the SAP print parameter PRIMM (Print Immediately) for all jobs. The default value is `OFF`.
prnew_enable	(Optional) Enables (`ON`) the SAP print parameter PRNEW (New Spool Request) for all jobs. The default value is `OFF`.
prrel_enable	(Optional) Enables (`ON`) the SAP print parameter PRREL (Print Release) for all jobs. The default value is `OFF`.
prsap_enable	(Optional) Enables (`ON`) the SAP print parameter PRSAP (Print SAP Cover Page) for all jobs. The default value is `OFF`.
prunx_enable	(Optional) Enables (`ON`) the SAP print parameter PRUNX (Print Operating System Cover Page) for all jobs. The default value is `OFF`.
release_all_intercepted_jobs_for_request	Releases jobs for each user on a cyclic basis, based on the number of jobs specified in the **max_jobs_to_release_for_user** option. The default value is **ON**, which means that all jobs are submitted for each user: • If the **max_jobs_to_release_for_user** option is less than or equal to **0**, all jobs are released for each user. • If the **max_jobs_to_release_for_user** option is higher than **0**, the specified number of jobs is submitted for each user on a cyclic basis. For example, if **max_jobs_to_release_for_user=5**, the first **5** jobs are submitted for each user, then the following 5 jobs for each user, and so on, until all jobs for all users are submitted. If this option is set to **OFF**, it releases for each user only the number of jobs specified in the **max_jobs_to_release_for_user** option. The remaining jobs are submitted only when a new release job is submitted: • If the **max_jobs_to_release_for_user** option is less than or equal to **0**, all jobs are released for each user. • If the **max_jobs_to_release_for_user** option is higher than **0**, only the specified number of jobs is submitted, then no other job is submitted until

Table 146. r3batch local configuration options (continued)

Option	Description
	the new release job. If **max_jobs_to_release_for_user**=**5**, the first **5** jobs are submitted for each user, then no other job is submitted until the new release job. This option is present both in global and local options. If you define it in both configuration files, the local value overrides the global one.
r3client	(Mandatory) The SAP client number.
r3gateway	(Optional) The host name of the SAP gateway.
r3group	(Optional) The name of the SAP logon group.
r3gwservice	(Optional) The service number of the SAP gateway.
r3host	(Mandatory) The host name of the SAP message server when using logon groups, or the host name of the application server in all other cases. If this server can be reached through one or more SAP gateways, use a string in the format `/H/gateway/H/` for each of them.
r3instance	(Mandatory) The SAP instance number. If **r3group** is set, this option is ignored.
r3password	(Mandatory) The password for the **r3user**. Ensure that you enter the same password when creating this user in the SAP system. It can be a maximum of 40 characters for non-encrypted passwords. If the password is encrypted, the maximum length is 72 characters. The value is case sensitive. For information about how to encrypt the password see Encrypting SAP user passwords on page 1166.
r3sid	(Mandatory) The SAP system ID.
r3snclib	(Optional) Specifies the path and file name of the SNC library. This option becomes mandatory if **r3sncmode** is activated (1).
r3sncmode	(Optional) Enables (1), or disables (0), secure network communication (SNC) between r3batch and the SAP R3 system. The default setting is (0). Refer to the SAP documentation for more information about using the SAP cryptographic Library for SNC.
r3sncmyname	(Optional) Specifies the name of the user sending the RFC for secure network communication (SNC).

Table 146. r3batch local configuration options (continued)

Option	Description
r3sncpartnername	(Optional) Specifies the SNC name of the SAP R3 communication partner (application server). This option becomes mandatory if **r3sncmode** is activated (1).
r3sncqop	(Optional) Specifies the secure network communication (SNC) protection level.
r3user	(Mandatory) The name of the SAP user with which the access method connects to the SAP system. It must have the appropriate privileges for running background jobs. It is sometimes also called the Maestro™ User ID.
report_list_max_limit	(Optional) Sets the maximum number of ABAP reports which can be loaded. The default value is -1, which means no limit.

Defining the common options

Table 147: r3batch common configuration options on page 1154 lists additional options that you can specify in either configuration file.

Table 147. r3batch common configuration options

Option	Description	Default
bdc_job_status_failed	(Optional) How HCL Workload Automation sets the completion status of a job running BDC sessions, according to a possible BDC processing failure. The allowed values are: ***n*** If at least *n* BDC sessions failed (where *n* is an integer greater than 0), HCL Workload Automation sets the job completion status as failed. **all** If all the BDC sessions failed, HCL Workload Automation sets the job completion status as failed.	`ignore`

Table 147. r3batch common configuration options (continued)

Option	Description	Default
	ignore When all the BDC sessions complete, regardless of their status, HCL Workload Automation sets the job completion status as successful. This is the default. **Note:** This option is ignored if you defined the job by setting the **nobdc** or **nobdcwait** option. For details about these options, see Task string to define SAP jobs on page 1180.	
ccms_alert_history	(Optional) Enables (`ON`) or disables (`OFF`) the product to retrieve all the matching CCMS alerts, included those that were generated before the monitoring process started. The default value is `OFF`, meaning that the product retrieves only the CCMS alerts that are generated after the monitoring process started. **Note:** This option takes effect the first time you start the CCMS alert monitoring. If you initially set it to `OFF` and later you want to retrieve the alerts generated before the monitoring process started, stop the monitoring and delete the *XAname*`_r3xalmon.cfg` file located in *TWA_DATA_DIR*`/methods/r3evmon_cfg` on UNIX® and `TWA_home\methods\r3evmon_cfg` on Windows™. In the options file, set `ccms_alert_history=on` and start the monitoring process again.	`OFF`
commit_dependency	(Optional) Enables (`ON`) or disables (`OFF`) the product to commit internetwork dependencies after processing. If you enable this option, internetwork dependencies are committed immediately by default. If you disable or delete this option, the **-commit** parameter set in the internetwork dependency definition is applied. For details about the **-commit** parameter, see Table 157: Parameters to define an SAP internetwork dependency on page 1263.	`OFF`
enable_appl_rc	(Optional) Enables (`ON`) or disables (`OFF`) the mapping of the application return code to the HCL Workload Automation return code.	`OFF`

Table 147. r3batch common configuration options (continued)

Option	Description	Default
	Note: This feature does not modify the exit code of the access method. For more details, refer to the **rccondsucc** keyword in the job definition documented in *HCL Workload Automation: User's Guide and Reference*.	
evmon_interval	(Optional) The polling rate (in seconds) that the `r3evmon` process applies to monitor the list of events.	`60`
ifuser	(Optional) The ID of the user who runs the access method to retrieve job information.	None
idoc_no_history	(Optional) Enables (`ON`) or disables (`OFF`) the product to retrieve only IDoc data that is generated after the monitoring process started. If you specify `OFF`, all matching IDocs are retrieved, including those that were generated before the monitoring process started. When processing this option, `r3evmon` uses the *XAname*`_r3idocmon.cfg` file to retrieve the date and time for the next monitoring loop.	`ON`
idoc_shallow_result	(Optional) Enables (`ON`) or disables (`OFF`) the product to retrieve only the most recent matching IDocs. For example, suppose you set `idoc_shallow_result=ON`. If the status of an IDoc changes several times during the monitoring interval and the same status, matching an event rule condition, occurs more than once in the sequence of statuses, only the most recent matching IDoc is retrieved. If you specify `OFF`, all matching IDocs are retrieved.	`ON`
jobdef	(Optional) If enabled, you can use the Dynamic Workload Console to define jobs, in addition to the command line. Specify `r3batch` to enable the option, and any other value to disable it.	`r3batch`
job_interceptable	(Optional) Enables (`ON`) or disables (`OFF`) the job launched by r3batch to be intercepted by SAP. If enabled, when r3batch launches a job and the SAP job interception feature is enabled, the job can be intercepted if it matches previously defined criteria. If disabled, the job launched by r3batch cannot be intercepted by SAP.	`OFF`
ljuser	(Optional) The ID of the user who runs the access method to launch jobs (LJ tasks) and manage jobs (MJ tasks).	None

Table 147. r3batch common configuration options (continued)

Option	Description	Default
log_r3syslog	(Optional) Enables (`ON`) or disables (`OFF`) the access method to write the latest entries from the SAP syslog to its trace file when an RFC returns with a general error.	`OFF`
long_interval	(Optional) The maximum interval, in seconds, between status checks. It cannot be greater than 3600 seconds. See also *short_interval*.	`3600`
max_n0_counter	(Optional) The maximum value of the N0 counter. If the N0 counter reaches the specified value, it starts again from 0.	`2^15 - 1`
max_name_counter	(Optional) The maximum value of the variant name counter. If the name counter reaches the specified value, it starts again from 0.	`40`
n0_counter_policy	(Optional) The N0 counter policy: **step** The N0 counter is increased once for every step. **job** The N0 counter is increased once for every job.	`job`
name_counter_policy	(Optional) The name counter policy: **step** The name counter is increased once for every step. **job** The name counter is increased once for every job.	`job`
nojobdefs	(Optional) Disables (`1`) or enables (`0`) the definition of new SAP jobs using the Dynamic Workload Console. If this option is set to 1, you must create the job definitions in the SAP job before creating the HCL Workload Automation job that is going to schedule them.	`0`
oldcopy	(Optional) Enables (`1`) or disables (`0`) the access method to use the old way of copying jobs, even though the function module BAPI_XBP_JOB_COPY is present on the SAP system.	`0`
pchain_recover	(Optional) The action taken by HCL Workload Automation when you rerun a job that submits a process chain. The allowed values are: **rerun** HCL Workload Automation creates another process chain instance and submits it to be run again.	`rerun`

Table 147. r3batch common configuration options (continued)

Option	Description	Default
	restart HCL Workload Automation restarts the original process chain from the failing processes to the end. For details about rerunning a process chain job, refer to Rerunning a process chain job on page 1243.	
pchain_details	(Optional) Enables (`ON`) or disables (`OFF`) the display of details about an SAP process chain that you scheduled as an HCL Workload Automation job.	`OFF`
pchainlog_bapi_msg	(Optional) Enables (`ON`) or disables (`OFF`) the product to retrieve additional messages from the BAPI calls from the SAP Business Warehouse process chains and appends them to the stdlist of HCL Workload Automation.	`ON`
pchainlog_level	(Optional) Supplements the option **retrieve_pchainlog**. Specifies which level of process chain logs you want to retrieve. Allowed values are: **1** Only the first level of process chain is logged. ***level_number*** Process chains are logged down to the level of chain you indicate here. For example, if you indicate 2 only the first two levels are logged. **all** All process chains are logged.	If you omit this option, and leave **retrieve_pchainlog** set to `ON`, the default is level `1`.
pchainlog_verbosity	(Optional) Supplements the option **retrieve_pchainlog**. Specifies which type of process chain logs you want to retrieve. Allowed values are: **chains_only** Logs only the process chains. **chains_and_failed_proc** In addition to the process chains, logs all failed processes.	If you omit this option, and leave **retrieve_pchainlog** set to `ON`, the default is `complete`.

Table 147. r3batch common configuration options (continued)

Option	Description	Default
	complete Logs all process chains and processes. **Note:** This option affects the entire process chain; verbosity cannot be reduced for individual processes.	
pc_launch_child	(Optional) Enables (`ON`) or disables (`OFF`) the product to launch child jobs that are in scheduled state. **Note:** You can use this option only if you activated the parent-child feature on the SAP system. On the XBP 2.0 or later SAP system, you can activate this feature by using the INITXBP2 ABAP report.	`OFF`
placeholder_abap_step	(Optional) If XBP version 2.0 is used, the name of the ABAP report used as the dummy step in the SAP placeholder job that is created to monitor an SAP event defined as external dependency.	If this option is not specified, either as global or local option, the default `BTCTEST` is used.
qos_disable	(Optional) Enables (`ON`) or disables (`OFF`) the creation of the environment variable QOS_DISABLE on Microsoft™ Windows™ systems that use the Quality of Service (QoS) feature, before `r3batch` opens an RFC connection. Without this option, because of problems in the implementation of the QoS service, the connection between `r3batch` and the SAP RFC library does not work.	`OFF`
r3auditlevel	(Optional) The audit level for the XBP. A number from 0 (low) to 3 (high).	`3`
rcmap	(Optional) Enables (`ON`) or disables (`OFF`) the return code mapping capabilities of Access method for SAP.	`ON`
retrieve_applinfo	(Optional) Enables (`ON`) or disables (`OFF`) the retrieval and appending of the SAP application log to the stdlist of HCL Workload Automation.	`OFF`
retrieve_ipaklog	(Optional) Enables (`ON`) or disables (`OFF`) the retrieval and appending of the SAP BW InfoPackage logs to the `stdlist` of HCL Workload Automation.	`ON`

Table 147. r3batch common configuration options (continued)

Option	Description	Default
	Note: The retrieval and appending of SAP BW InfoPackage job logs to the `stdlist` might be time-consuming for jobs that produce large logs.	
retrieve_joblog	(Optional) Enables (`ON`) or disables (`OFF`) the retrieval and appending of the SAP job logs to the `stdlist` of HCL Workload Automation. **Note:** 1. The retrieval and appending of job logs to the `stdlist` might be time-consuming for jobs that produce large logs. 2. If you disable the retrieval of the job logs, you also disable the return code mapping function for the log entries. 3. This option does not affect the `BDC Wait` feature.	`ON`
retrieve_pchainlog	(Optional) Enables (`ON`) or disables (`OFF`) the retrieval and appending of the SAP BW process chain logs to the `stdlist` of HCL Workload Automation. **Note:** 1. The retrieval and appending of SAP BW process chain logs to the `stdlist` might be time-consuming for jobs that produce large logs. 2. If you disable the retrieval of the SAP BW process chain logs, you also disable the return code mapping function for the log entries. 3. This option on its own retrieves the log of only the first level of a process chain. To retrieve more complete logs, use this option with the **pchainlog_level** and **pchainlog_verbosity** options.	`ON`
retrieve_spoollist	(Optional) Enables (`ON`) or disables (`OFF`) the retrieval and appending of the SAP job spool lists to the `stdlist` of HCL Workload Automation.	`ON`

Table 147. r3batch common configuration options (continued)

Option	Description	Default
	Note: 1. The retrieval and appending of SAP job spool lists to the `stdlist` might be time-consuming for jobs that produce large spool lists. 2. If you disable the retrieval of the SAP job spool lists, you also disable the return code mapping function for the spool list entries.	
retry	(Optional) The retry count for SAP function module calls. Specify an integer greater than 0.	`5`
rfc_interval	(Optional) The polling rate (in milliseconds) with which r3batch listens for results of RFC requests. The rate cannot exceed 1,000 milliseconds. Consider that the lower the value of the rfc_interval option, the higher the frequency with which RFC request results are collected and, as a consequence, CPU consumption on the r3batch system is high.	`10`
rfc_open_delay	(Optional) The maximum number of seconds to wait between two consecutive calls before opening an RFC connection.	`1800`
rfc_open_retry	(Optional) The retry count for opening an RFC connection to the SAP system. Specify an integer greater than 0 to limit the number of retries, or `-1` for an unlimited number of retries.	`5`
rfc_timeout	(Optional) The time (in seconds) that `r3batch` waits before canceling a non-responding RFC communication. Allowed values are in the range from 0 to 9999; 0 means no timeout.	`600`
short_interval	(Optional) The minimum interval, in seconds, between status checks. It cannot be less than 2 seconds. Setting this option to low values makes the notification of status changes faster, but increases the load on the hosting machine. See also *long_interval*.	`10`
throttling_enable_ job_class_inheritance	(Optional) Enables (`ON`) or disables (`OFF`) the inheritance of priority class. `ON` means that the intercepted job inherits the priority class of its progenitor job, if it is higher than its own class; otherwise it keeps its	`ON`

Table 147. r3batch common configuration options (continued)

Option	Description	Default
	own class. `OFF` means that the intercepted job keeps its own class, regardless of its progenitor's class. **Note:** By setting this option, the parent-child feature is automatically enabled on the SAP system.	
throttling_enable_ job_interception	(Optional) Enables (`ON`) the job interception feature at job throttler startup, or keeps the current setting (`OFF`). `ON` means that when the job throttler starts, it enables the job interception feature on the SAP system. When the job throttler is stopped, the job interception feature is also automatically restored to the setting that was previously configured on the SAP system. `OFF` means that the job interception feature is kept as it is currently set in the SAP system.	`ON`
throttling_job_ interception_ version	Specifies the BC-XBP interface version to be used when the job throttler starts. Valid values are: • 2 • 3 The default BC-XBP interface version that is used is `2` (version 2.0).	`2`
throttling_interval	(Optional) The interval (in seconds) between each job throttling run.	`5`
throttling_max_ connections	(Optional) The maximum number of connections (connection pool size) that the job throttler can open to communicate with the SAP system. The minimum value is 3.	`5`
throttling_release_all_ on_exit	(Optional) Enables (`ON`) or disables (`OFF`) the release of all intercepted jobs. `ON` means that when the job throttler is stopped, it releases all the intercepted jobs. `OFF` means that when the job throttler is stopped, it does not release the intercepted jobs therefore the jobs remain intercepted, in scheduled state.	`ON`
throttling_send_ ccms_data	(Optional) Enables (`ON`) or disables (`OFF`) the sending of data from job throttling to the SAP CCMS Monitoring Architecture.	`OFF`

Table 147. r3batch common configuration options (continued)

Option	Description	Default
	`ON` means that the job throttler sends its status data to CCMS continuously. `OFF` means that the job throttler does not send its status to CCMS.	
throttling_send_ ccms_rate	(Optional) Rate (in number of runs) at which the job throttler sends its status data to the SAP CCMS monitoring architecture. The minimum value is `1`, meaning that the job throttler sends the data at every run.	`1`
twsmeth_cp	(Optional) The code page that `r3batch` uses to write its output. This option must be consistent with **twsmeth_lang**. It can be any of the existing TIS codepages.	The code page used by the HCL Workload Automation workstation that hosts the `r3batch` access method.
twsmeth_lang	(Optional) The language used to report messages. This option must be consistent with **twsmeth_cp**.	The language of the locale of the workstation that hosts the `r3batch` access method.
twsxa_cp	(Optional) The encoding used by `r3batch` to establish RFC connections with SAP systems. Use this option if `r3batch` is not Unicode-enabled. Possible values are: • 1100 • 1103 • 8000 • 8300 • 8400	`1100`
twsxa_lang	(Optional) The language used to log in to SAP systems. Specify one of the following (DE, EN, and JA can be set from the Option Editor. The other languages can be set using any text editor): **DE** German **EN** English	`EN`

Table 147. r3batch common configuration options (continued)

Option	Description	Default
	ES Spanish **FR** French **IT** Italian **JA** Japanese **KO** Korean **pt_BR** Brazilian Portuguese **zh_CN** Simplified Chinese **zh_TW** Traditional Chinese 	
use_fips	(Optional) Enables (ON) or disables (OFF) the FIPS mode of operation for HCL Workload Automation.	`OFF`
utf8cmdline	(Optional) Enables (`1`) or disables (`0`) the encoding of extended parameters in UTF-8 format. The default value is `0`.	**Note:** If you have both global and local options files and you want to change

Table 147. r3batch common configuration options (continued)

Option	Description	Default
		the default value for **utf8cmdline**, modify the local options file because this overrides the global options.
variant_delay	(Optional) The time, in seconds, that r3batch allows the SAP system to clean up the structures used for communication between r3batch and the SAP system. This option is valid when you launch a job that uses extended variants and requires a copy of a job template. Use this option only when you want to reduce r3batch response time, because it increases the load on the hosting machine. Higher values of **variant_delay** increase the response time and decrease the load. Allowed values are in the range from 0 to 3600.	`10`
variant_selection_ screen	(Optional) Specifies the functional interface used to read report selection screens. Specify one of the following: **Custom** To communicate with the SAP system using the HCL Workload Automation custom function module. **SAP** To communicate with the SAP system using the XBP 3.0 function module.	Custom
xbpversion	(Optional) The XBP version used on the target SAP system. Specify an integer value. This value overwrites the XBP version automatically determined during RFC logon.	The XBP version determined by r3batch during RFC

Table 147. r3batch common configuration options (continued)

Option	Description	Default
	Note: For details about XBP 3.0 and SAP NetWeaver 2004s with SP9, refer to the SAP Note 977462.	logon from the SAP system.

SAP option file example

Below is an example of an options file for SAP. It can help you determine your specific site requirements, although your options file might be different.

```
r3client=100
r3host=/H/tiraix64.lab.rome.abc.com
r3instance=00
r3password={aes}hyb/LQNyVzIf9oA8/xgY+CSqAuAh7+CvTT7HuDpdiu5YUAH0KJEHJtA=
r3sid=GS7
r3user=twstest
long_interval=120
r3auditlevel=3
short_interval=10
twsxa_lang=EN
```

Encrypting SAP user passwords

When you add your entries in the options file from the Dynamic Workload Console, the password value is automatically encrypted before it is written in the file. If you modify the file with a text editor, run the **enigma** program to encrypt the password before writing it in the file, as follows:

```
enigma your_password
```

You can include the password on the command line or enter it in response to a prompt. The program returns an encrypted version that you can then enter in the options file.

Configuration options usage

The format of the configuration file is the following:

```
option1=value1
option2=value2
option3=value3
...
```

with no blanks before the option, after the value, or before or after the equals (=) character.

You can put all the common information, such as the `LJuser`, `IFuser`, `JobDef`, and `LogFileName` options in `r3batch.opts`, while you can put tailored data for the target SAP system of the extended agent or dynamic agent (for example, `SAP1`) in a local configuration file (for example, `XA1_r3batch.opts`).

You can put a local option in the global configuration file if you want to give the same option to all the `r3batch` instances. For example, if the SAP user name is the same in all your SAP systems, you can place the `r3user` option in the global file without duplicating that information in all the local configuration files.

A global option, such as `job_sem_proj`, only has effect in the global configuration file. If you put global options in a local file they have no effect.

`r3batch` reads the global configuration file first and then the local file. Every option (except the global options) contained in the local configuration file will override those in the global file. For example, if both the global and the local configuration files contain the `r3user` option, `r3batch` uses the one in the local file.

There are six mandatory options that `r3batch` requires:

- `r3client`
- `r3host`
- `r3instance`
- `r3password`
- `r3sid`
- `r3user`

You can put them all in the local configuration file or you can spread them between the global and the local files. For example, you could put `r3user` and `r3password` in the global configuration file and `r3sid`, `r3instance`, `r3client`, and `r3host` in the local one.

The `r3user` option is both local and mandatory. It must be put either in the global configuration file or the local configuration file.



Note: These configuration files are not created during the installation process.

Connecting to the SAP system

The Access method for SAP uses the SAP remote connection call (RFC) library to connect to the SAP system. The connection address for an SAP system is denoted as a `connection string`.

To successfully use the SAP R/3 access method, you must first install the SAP Remote Function Call (RFC) libraries, as described in the in the SAP section.

Connecting to a specific application server

To connect to a specific application server, you enter strings which, according to the complexity of the networks, might be more or less complex and contain passwords to secure the routers.

In its basic form, a connection string consists of the host name (or IP name) of an SAP application server; for example:

```
/H/hemlock.romlab.rome.abc.com
```

This type of connection string works only in very simple network environments, where all application servers can be reached directly through TCP/IP. Usually, modern companies use more complex network topologies, with a number of small subnetworks, which cannot communicate directly through TCP/IP. To support this type of network, the SAP RFC library supports SAP routers, which are placed at the boundaries of the subnetworks and act as proxies. For this type of network, the connection string is a composite of basic connection strings for each SAP router, followed by the basic connection string for the target SAP system; for example:

```
 /H/litespeed/H/amsaix33/H/hemlock.romlab.rome.abc.com
```

Moreover, you can secure the SAP routers with passwords, to prevent unauthorized access. In this case, the basic connection string for the SAP router is followed by `/P/` and the password of the router.

Note: The SAP RFC library limits the length of the connection string to a maximum of 128 characters. This is a real limitation in complex network environments. As a workaround, it is recommended to use simple host names, without the domain name whenever possible. Alternatively, you can use the IP address, but this is not recommended, because it is difficult to maintain.

HCL Workload Automation for SAP supports both types of connection strings, basic and composite, where:

r3host

The connection string.

r3instance

The SAP instance number.

r3sid

The SAP system ID.

For example:

```
r3host=/H/litespeed/H/amsaix33/H/hemlock.romlab.rome.abc.com
r3instance=00
r3sid=TV1
```

Connecting to a logon group

About this task

In large SAP installations, the application servers are usually configured in logon groups for load balancing and fault-tolerance purposes. Load balancing is done by a dedicated server, called the `message server`. The message server automatically assigns users to the application server with the least workload of the logon group it controls.

Ensure that the file services (on UNIX: `/etc/services` on Windows: `C:\Windows\system32\drivers\etc\services`) contain an entry for the message server port of the SAP system to which `r3batch` connects. The entry has the following format:

```
sapmsSID 36system_number/tcp
```

where *SID* is the SAP system ID, and *system_number* is the SAP system number.

Set the following options to configure `r3batch` to connect to a logon group:

r3host

The hostname of the message server.

r3group

The name of the logon group.

r3sid

The SAP system ID.

For example:

```
r3host=pwdf0647.wdf.sap-ag.de
r3group=PUBLIC
r3sid=QB6
```

Configuring SAP event monitoring

This section provides detailed information about how to configure your system to monitor SAP events:

- Prerequisite to defining event rules based on SAP events on page 1169
- Monitoring SAP events on page 1170

Prerequisite to defining event rules based on SAP events

About this task

To be able to define event rules based on one or more SAP events, stop the HCL Workload Automation WebSphere Application Server Liberty Base and copy the following file (located on the system where you installed HCL Workload Automation:

On UNIX operating systems

`TWA_DATA_DIR/methods/SAPPlugin/SapMonitorPlugIn.jar`

On Windows operating systems

`TWA_home\methods\SAPPlugin\SapMonitorPlugIn.jar`

to the following directory of the master domain manager and of its backup nodes:

On UNIX operating systems

`TWA_DATA_DIR/eventPlugIn`

On Windows operating systems

`TWA_home\eventPlugIn`

For the changes to take effect, stop and restart the HCL Workload Automation WebSphere Application Server Liberty Base. If the master domain manager is connected to the Dynamic Workload Console, stop and restart also the Dynamic Workload Console Application Server.

Monitoring SAP events

Whenever you define an event rule based on an SAP event in your HCL Workload Automation plan, that event is monitored by HCL Workload Automation. Monitoring SAP events is allowed only if you use XPB version 3.0, or later.

HCL Workload Automation monitors two types of SAP event:

Events defined by the SAP system

The events that are triggered automatically by system changes, for example when a new operation mode is activated. This type of event cannot be modified by the user.

Events defined by the user

The events that are triggered by ABAP or external processes, for example when a process triggers an SAP event to signal that external data has arrived and must be read by the SAP system. For details about how to trigger events by external processes, refer to Raising an SAP event on page 1195.

If you modify the `r3batch` option files, to make the changes effective you must stop and restart the extended agent monitoring processes with the following command. For UNIX only, this command must be entered by the owner of the HCL Workload Automation installation:

Command syntax

```
r3evman { start | stop }
```

Where:

start | stop

The action to perform:

start

Starts monitoring SAP events.

stop

Stops monitoring SAP events.

Defining SAP jobs

You must define some jobs to be able to run jobs on an SAP workstation from HCL Workload Automation.

To define and manage jobs on an SAP workstation from HCL Workload Automation, you must define the following:

Jobs in SAP that you want to run under HCL Workload Automation control

You can define these jobs using standard SAP tools or using the Dynamic Workload Console.

Jobs in HCL Workload Automation that correspond to the jobs in SAP

The HCL Workload Automation job definitions are used in scheduling and defining dependencies, but the SAP jobs are actually run.

You can define SAP job definitions from the Dynamic Workload Console and then have HCL Workload Automation launch the jobs in SAP R/3 using jobs defined on the following workstations that support the r3batch access method:

- An HCL Workload Automation extended agent workstation. A workstation that is hosted by a fault-tolerant agent or master workstation.
- A dynamic agent workstation.

- A dynamic pool.
- A z-centric workstation.

You can manage your SAP environment from both:

- An HCL Workload Automation distributed environment
- An HCL Workload Automation for Z environment.

The SAP job definitions can reference the following types of SAP jobs:

- Standard R/3
- Business Warehouse Process Chains
- Business Warehouse InfoPackages

For information about Business Warehouse Process Chains and Business Warehouse InfoPackages, see Using Business Information Warehouse on page 1233.

Creating SAP Standard R/3 jobs from the Dynamic Workload Console

How to create and manage an SAP job that is associated to an HCL Workload Automation job that manages it.

About this task

You can easily create and manage Standard R/3 jobs on a remote SAP system entirely from the Dynamic Workload Console, and then continue to manage the remote SAP job from HCL Workload Automation.

The HCL Workload Automation job definition, available for both distributed and z/OS environments, maps to the newly created job on the SAP system. The SAP job can run on extended agent workstations, dynamic agent workstations, pools, dynamic pools, and workstations depending on the type of job definition you choose to create.

Note: Using this procedure to create a new HCL Workload Automation Agent SAP Standard R/3 job, you cannot manage variants. To manage variants, use the SAP graphical user interface or use the **List Jobs on SAP** entry from the navigation tree of the Dynamic Workload Console.

To create a new SAP Standard R/3 job on a remote SAP system that maps to an HCL Workload Automation job definition, you have to associate your SAP Standard R/3 jobs to HCL Workload Automation jobs and you can do as follows:

- Creating an SAP job on a remote SAP system: Creating an SAP job from the Dynamic Workload Console on page 1173
- Creating an HCL Workload Automation job and associating it to an SAP job: Create an HCL Workload Automation job and associate it to an SAP job on page 1173

When performing operations that require a connection to a remote SAP system, you must configure the SAP connection data. The connection is made through an HCL Workload Automation workstation with the r3batch access method installed. Each workstation can have one or more options files that can be used to customize the behavior of the r3batch access

method, except for extended agent workstations, where only one options file can be defined and therefore a selection is not required. For information about setting the SAP connection data, see Setting the SAP data connection on page 1174.

Creating an SAP job from the Dynamic Workload Console

How to create an SAP job definition on a remote SAP system from the Dynamic Workload Console.

About this task

You can also create and save SAP Standard R/3 jobs directly on the remote SAP system from HCL Workload Automation, as you would from the SAP graphical user interface. To create Standard R/3 jobs on the SAP system from the Dynamic Workload Console, perform the following steps:

1. From the **Design** menu, select **Manage Jobs on SAP**.
2. In the Job Filter Criteria, select **Standard Job** and specify the workstation name. This parameter is mandatory because it identifies the remote SAP system.
3. Specify the workstation where the SAP job runs. This is the workstation with the r3batch access method that communicates with the remote SAP system.
4. If the workstation is not an extended agent workstation, you must also specify the options file to be used.
5. Click **Display** to view a list of the Standard R/3 jobs for the specified workstation.
6. Click **New** to create a new Standard R/3 job and enter the required information.
7. Click **Save** to create the job on the SAP system.

What to do next

After creating the new SAP job on the SAP from the Dynamic Workload Console, you must reference it in an HCL Workload Automation SAP Standard R/3 job if you want to manage the job from within HCL Workload Automation as explained in Create an HCL Workload Automation job and associate it to an SAP job on page 1173.

Create an HCL Workload Automation job and associate it to an SAP job

Create an HCL Workload Automation job definition and map it to a new or existing SAP job to manage it.

About this task

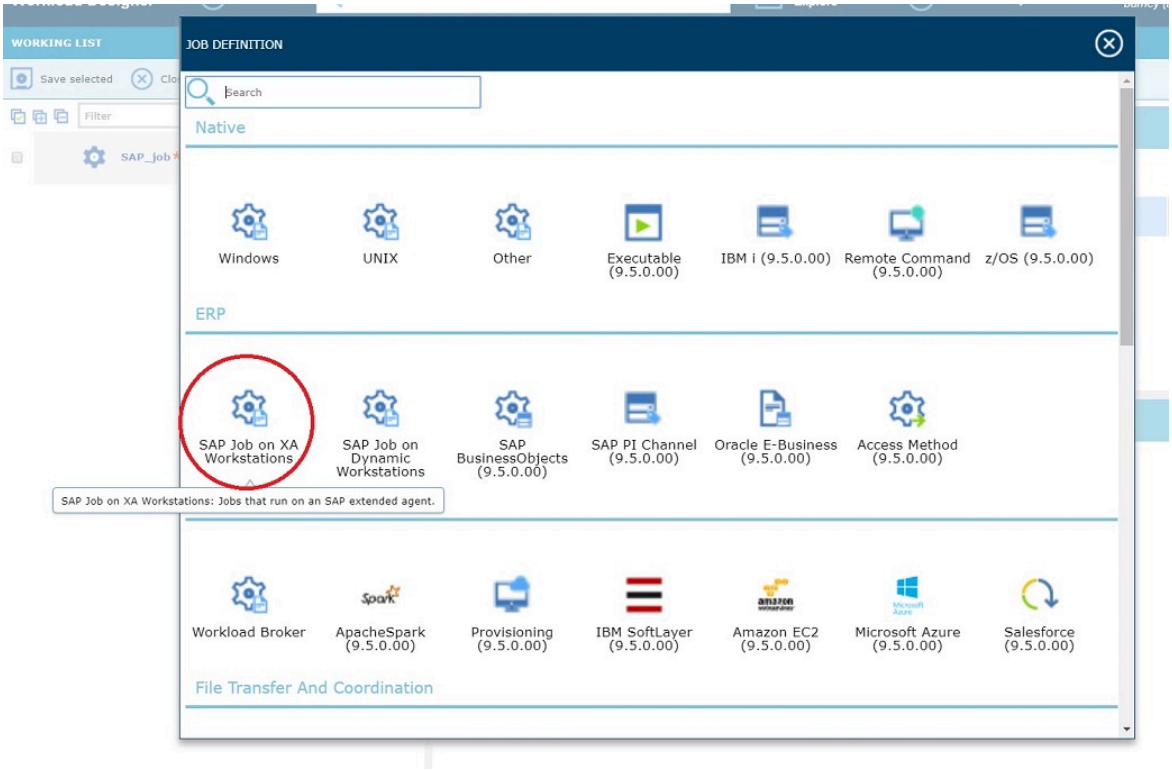
To create a new HCL Workload Automation job and then associate it to a new SAP job, follow these steps:

1. From the **Design** menu, click Workload Designer page.
2. Select an engine. The Workload Designer window is displayed.
3. Click **Create New**, select **Job definition** and in the **ERP** category choose:
    - **SAP Job on Dynamic Workstations**

      For distributed systems only. This job definition can run on dynamic agent workstations, dynamic pools, and HCL Workload Automation Agent workstations.

SAP Job on XA Workstations

This job definition can run on extended agent workstations, which are workstations hosted by fault-tolerant agents or master workstations.



- **SAP**

  SAP

  For z/OS systems only. This job definition references an existing job on the SAP system and can run on dynamic agent workstations, dynamic pools, and HCL Workload Automation Agent.

4. In the Properties pane, specify the properties for the SAP job definition you are creating using the tabs available. The tabs for each type of SAP job definition are similar, but there are some differences depending on the type of engine you selected and the type of workstation on which the job runs. For more detailed information about the UI elements on each tab, see the Dynamic Workload Console online help.
5. In the **Task** section, specify the HCL Workload Automation job that you want to associate to the SAP job.
6. Click **Save** to create the SAP job definition in the HCL Workload Automation database.

Setting the SAP data connection

You can configure a default connection to be used when performing actions that access the remote SAP system.

About this task

There are several operations you can perform which require connection details to establish a link to a remote SAP system. The connection is made through an HCL Workload Automation workstation with the r3batch access method installed used to

communicate with the SAP system. Each workstation can have one or more options files that can be used to customize the behavior of the r3batch access method, except for extended agent workstations, where only one options file can be defined and therefore a selection is not required.

For example, you can use **Workload Designer** to create HCL Workload Automation job definitions that reference remote SAP jobs, or you can create a SAP job on a remote SAP system. You can also search for SAP jobs on the remote system from the **Working List** and **Quick Open** panes.

To configure a default SAP data connection to be used when creating items in the Workload Designer that require a SAP connection, perform the following steps:

1. From the **Design** menu, select **Manage SAP Criteria Profile**.
2. Select an engine.
3. In **Workstation**, enter the name of the workstation that communicates with the SAP system. If you do not know the name of the workstation, click the lookup icon and specify a search criteria to find the workstation.
4. In **Options file**, enter the file name of the options file or click the lookup icon to search for options files that reside on the specified workstation and select one.
5. Click **Go** to establish the connection.

Results
A default SAP connection is now configured. It will be used each time an item that requires access to a SAP system is defined.

Managing SAP variants using the Dynamic Workload Console

Managing variants using the Dynamic Workload Console.

About this task

This section describes how to manage variants using the Dynamic Workload Console:

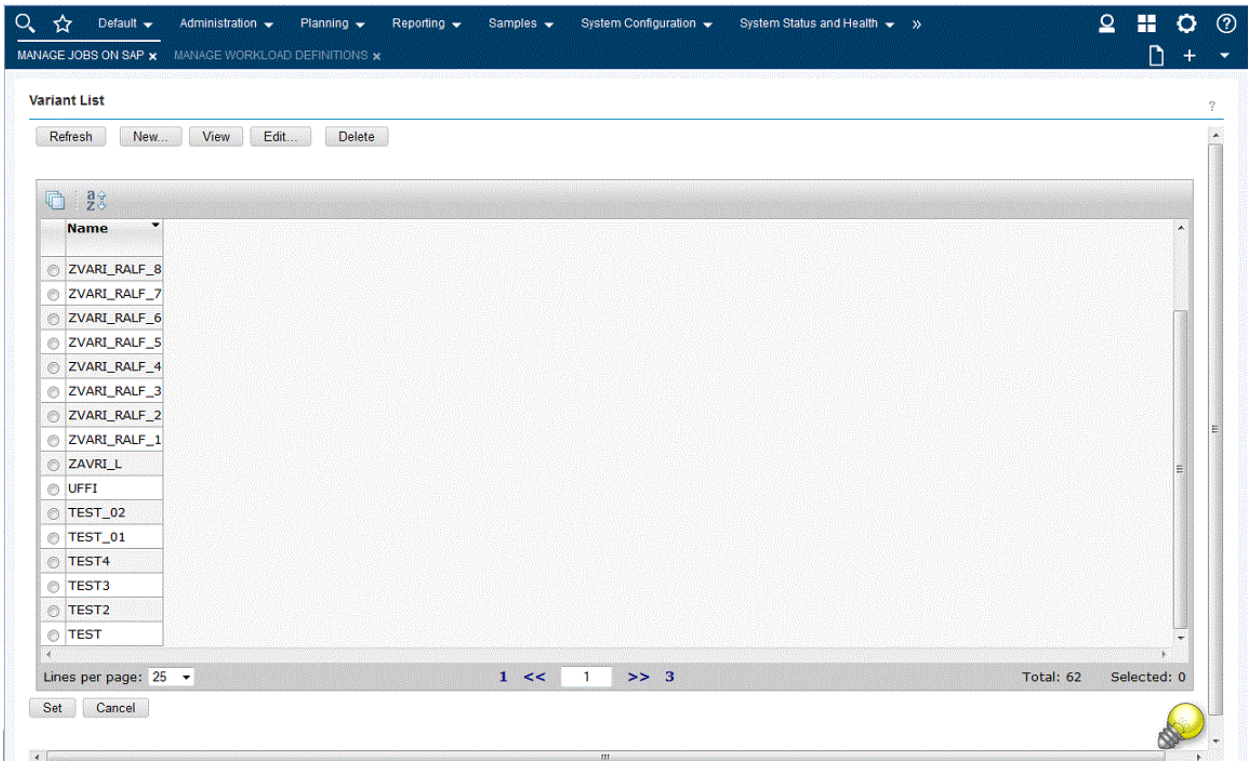
1. Click **Design > SAP > Manage Jobs on SAP** from the portfolio.
2. Specify an engine connection.
3. In **Workstation name**, type the name of the workstation where the SAP job runs. This is the workstation with the r3batch access method that communicates with the remote SAP system. If you do not know the name of the workstation, click (...) browse to enter your filter criteria and click **Search**. If you enter a string representing part of the workstation name, it must be followed by the asterisk (*) wildcard character. Both the question mark (?) and asterisk (*) are supported as wildcards. You can also simply use the asterisk wildcard character (*) to display all workstations. Optionally, specify any of the other search criteria available and click **Search**.
   Result
   From the results displayed, select the workstation and click **OK**.
4. In **Options file**, specify an options file that resides on the specified workstation. Each workstation can have one or more options files that can be used to customize the behavior of the r3batch access method, except for extended agent workstations, where only one options file can exist and therefore does not need to be specified. For the

workstation specified, enter the file name of the options file or click the browse (...) button to search for options files that reside on the specified workstation and select one.

5. Click **Display**. The list of available jobs on the remote SAP system for the specified engine is displayed.
6. A list of SAP jobs on the remote SAP system are displayed.
7. Select a SAP job from the list and click **Edit**.
8. On the **SAP Steps** page, select a program of type ABAP from the list and click **Edit**. The properties for the ABAP program are displayed.
9. In the **Variant** field, click the ellipsis (**...**) icon to display the Variant List panel. This panel lists all the variants associated with the ABAP specified in the **Name** field.

   Result

   Figure 30. The Variant List panel



10. From this panel, you can take the following actions:

    Refresh

    To refresh the content of the variant list with the information contained in the SAP database.

    New

    To create a new variant as described in Creating or editing a variant on page 1177.

    View

    To display information on an existing variant.

Edit

To modify information on an existing variant as described in Creating or editing a variant on page 1177.

Delete

To delete a variant.

Set

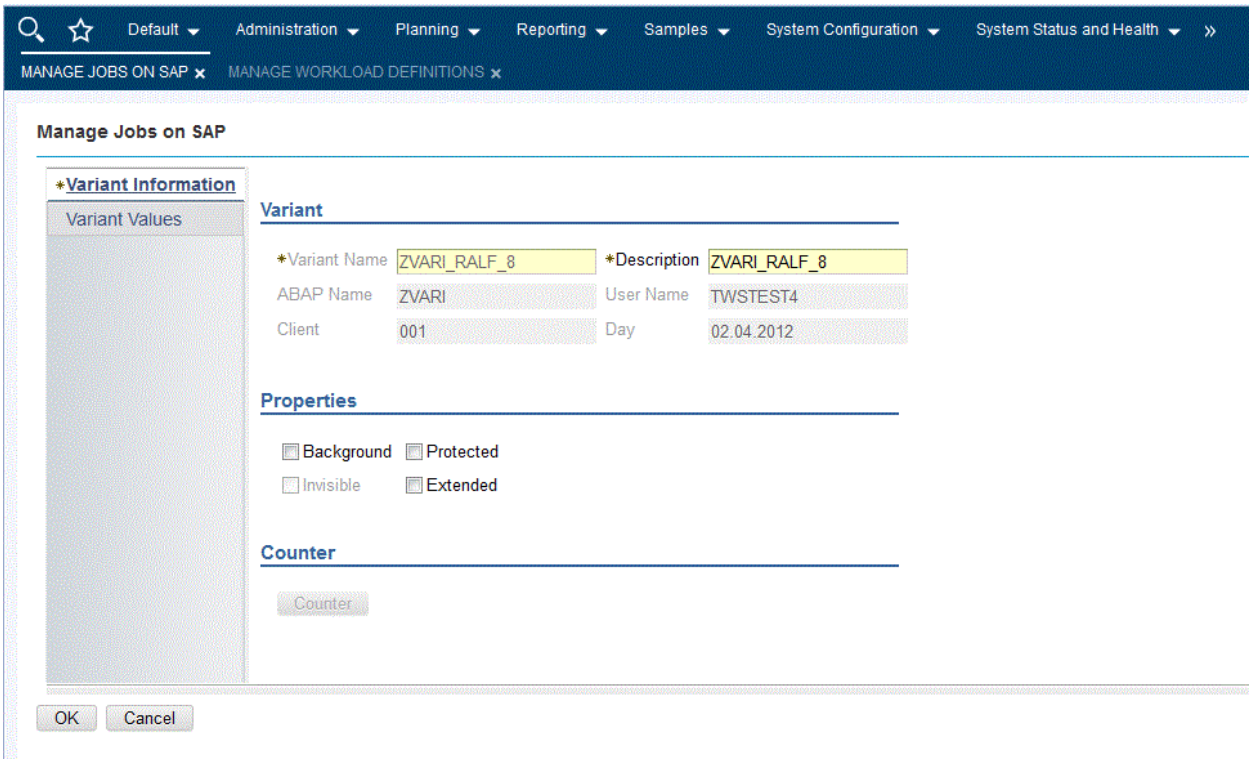
To associate the value chosen from the list to the ABAP.

Creating or editing a variant

About this task

You can create or edit a variant from the **Variant List** panel. To display the **Variant List** panel, see Managing SAP variants using the Dynamic Workload Console on page 1175.

1. In the Variant List panel, click **New** or **Edit**. The Variant Information page is displayed by default. If you are editing an existing variant, the fields and selections are not empty.
   Figure 31. The Variant Information page of the Variant List panel



2. The panel consists of the following pages:
   - Variant Information:
     a. Enter or modify the variant name and description.
     b. Optionally, check a **Properties** box:

Background

The variant can only be used in background processing.

Protected

The variant is protected against being changed by other users.

Invisible

The variant will not be displayed in the F4 value list on the SAP GUI. Not available for the BC-XBP 3.0 interface.

Extended

Allows for the use of placeholders and counters as variant values. If you check this box, **Counter** becomes available.

For extended variants, you can use placeholders and counters that eliminate the error-prone task of adjusting values and therefore minimize the effort for variant maintenance. Placeholders and counters are preprocessed by HCL Workload Automation and the values are automatically adjusted when the job is launched. Supported placeholders and counters are:

Table 148. Placeholders and counters for extended variants

Symbol	Meaning	Syntax
$S	Timestamp	YYYYMMDDHHMM
$D	Day of the month	DD
$_D	Date	YYYYMMDD
$M	Month	MM
$Y	Year	YY
$_Y	Year	YYYY
$H	Hour	HH
$T	Minute	MM
$_T	Time	HHMMSS
$N*x*	Counters	10 counters: $N0 - $N9 ($N = $N0)
$(*date expression*)	Date expression	Like the `datecalc` command. Enclosed within $(and).
$[*arithmetic expression*]	Arithmetic expression	Arithmetic expressions allowing for +, -, *, and " operations between integers and counters.

- Variant Values:

  In the Variant Values page, the fields and values are dynamically built through `r3batch` depending on the characteristics of the variant or step and are identical to the ones in the equivalent SAP panel.

Editing a standard SAP job

Before you begin

You can edit SAP Standard R/3 jobs in two different ways in HCL Workload Automation.

- The Dynamic Workload Console contains the **Manage Jobs on SAP** entry in the portfolio for creating and editing SAP Standard R/3 jobs on remote SAP systems.
- From the Workload Designer you can create and edit remote SAP jobs. See Creating SAP Standard R/3 jobs from the Dynamic Workload Console on page 1172.

About this task

To edit a SAP standard R/3 job, follow these steps:

1. Click **Design > SAP > Manage Jobs on SAP**.
2. Select the name of the engine connection from which you want to work with SAP jobs.
3. Leave the default setting in the **SAP Job Type** section to **Standard R/3 Job**.
4. In **Workstation name**, type the name of the workstation where the SAP job runs. This is the workstation with the r3batch access method that communicates with the remote SAP system. If you do not know the name of the workstation, click (...) browse to enter your filter criteria and click **Search**. If you enter a string representing part of the workstation name, it must be followed by the asterisk (*) wildcard character. Both the question mark (?) and asterisk (*) are supported as wildcards. You can also simply use the asterisk wildcard character (*) to display all workstations. Optionally, specify any of the other search criteria available and click **Search**.

   Result

   From the results displayed, select the workstation and click **OK**.
5. In **Options file**, specify an options file that resides on the specified workstation. Each workstation can have one or more options files that can be used to customize the behavior of the r3batch access method, except for extended agent workstations, where only one options file can exist and therefore does not need to be specified. For the workstation specified, enter the file name of the options file or click the browse (...) button to search for options files that reside on the specified workstation and select one.
6. Click **Display**. The list of available jobs on the remote SAP system for the specified engine is displayed.
7. Select the job you want to modify in the list and click **Edit**. The **List Jobs on SAP** panel is displayed.
8. Edit the properties on the **R/3 Job Definition** and **R/3 Steps** pages as appropriate. Refer to the contextual online help available for more detailed information about the UI elements available on each page.

   Note:

- On the **R/3 Job Definition** page, when you modify the **Job Class**, **Target Host**, or **Server Group** and click **OK**, the **Job ID** is maintained and remains synchronized with the one associated to the current job. Instead, when you modify the **Job Name** and click **OK**, the **Job ID** is automatically replaced with the one associated to the new job name.
- On the **R/3 Steps** page, for each step you modify, the new step information is saved in the SAP database. For each step you add or delete, the **Job ID** is maintained and remains synchronized with the one associated to the modified step.

9. Click **OK** to save your changes.

Task string to define SAP jobs

This section describes the task string parameters that define and control the running of SAP jobs. You can specify them in the following places when you define their associated HCL Workload Automation jobs:

- In the **SAP Command Line** section of the Task page of the Submit Ad Hoc Jobs action from the Dynamic Workload Console.
- In the **SAP Command Line** field of the More Options page of the SAP job definition, if you use the Dynamic Workload Console and selected a **SAP** job definition.
- As arguments of the `scriptname` keyword in the job definition statement, if you use the HCL Workload Automation command line.
- As arguments of the `JOBCMD` keyword in the `JOBREC` statement in the `SCRIPTLIB` of HCL Workload Automation for Z, if you are scheduling in an end-to-end environment. The following is an example of a `JOBREC` statement:

```
JOBREC
  JOBCMD('/-job job_name -user user_name -i job_ID -c class_value')
  JOBUSR(TWS_user_name)
```

where:

class_value

The priority with which the job runs in the SAP system. For details, see Table 149: Task string parameters for SAP jobs on page 1181.

job_ID

The unique SAP job ID. For details, see Table 149: Task string parameters for SAP jobs on page 1181.

job_name

The name of the SAP job to run. For details, see Table 149: Task string parameters for SAP jobs on page 1181.

user_name

The SAP user who owns the target job. For details, see Table 149: Task string parameters for SAP jobs on page 1181.

TWS_user_name

The HCL Workload Automation for Z user who runs the `r3batch` access method from the end-to-end scheduling environment.

The string syntax is the following:

Job definition syntax

```
-jobjob_name [{ -i | -id} job_ID ][ -useruser_name ][{ -host | -ts} host_name ][ -sgserver_group ]
[ -clientsource_client ][ -exec_clienttarget_client ][ -rfc_clientrfc_logon_client ][ -cclass_value ][
 -bdc_job_status_failed bdc_processing ][{ -nobdc | -nobdcwait }][ -bapi_sync_level{ high | medium | low }][
 -sstarting_step_number ][ -s Step_number attribute_name [ =attribute_value ]][ -vStep_number variant_name ]
[ -vtxtStep_number variant_description ][ -vparStep_number name=variant_value ][ -vselStep_number name={ i |
e } #operation#lowest [ #highest ]][ -vtemp Step_number ][ -recipientR/3_login_name ][ -rectyperecipient_type ]
[ -flag {reccp | recbl}][ -flag recex ][ -flag recnf ][ -flag{ im | immed }][ -flag{ enable_applinfo |
disable_applinfo }][ -flag{ enable_appl_rc | disable_appl_rc }][ -flag{ enable_joblog | disable_joblog }][
 -flag{ enable_job_interceptable | disable_job_interceptable }][ -flag{ enable_spoollist | disable_spoollist }][
 -flagpc_launch ][{ -debug }][ -tracelvl{ 1 | 2 | 3 }][{ -rfctrace }]
```

Table 149: Task string parameters for SAP jobs on page 1181 describes the parameters for the task string to define SAP jobs.

Note:

1. You can specify both `-i` or `-id` and `-user` in the same job definition, but the user name is ignored.
2. When you specify the job ID, both `-client` and `-exec_client` are ignored because the ID is unique for the entire SAP system.
3. Typically, the `-debug` and `-trace` options are for debugging the extended agent and should not be used in standard production.

Table 149. Task string parameters for SAP jobs

Section	Parameters	Description	GUI Support
JOB	-job *job_name*	The name of the job to run. This parameter is mandatory.	✓

Table 149. Task string parameters for SAP jobs

(continued)

Section	Parameters	Description	GUI Support
	-i *job_ID* -id *job_ID*	The unique SAP job ID. Specify this parameter if you are submitting a job that refers to a predefined job template stored in the SAP database for which you want to change a parameter.	✓
	-user *user_name*	The SAP user who owns the target job. Use this parameter when the target SAP system has only one job with the specified name for the specified user. This parameter has no effect if a job ID is specified in the job definition.	✓
	-host *host_name* -ts *host_name*	The name of the SAP workstation where the job is to be run. *host_name* has the format *hostname_SAPsystemname_SAPsystemnumber*. For example, the name of a host might be amss80a0_gs7_90 These parameters are mutually exclusive with -sg.	✓
	-sg *server_group*	The name of the SAP server group where the job is to be run. Use this parameter to run the job on an application server that belongs to the group. The server group must exist on the SAP system, otherwise an error code is returned and the job is not launched. This parameter is case-sensitive and can be up to 20 characters. It is mutually exclusive with -host and -ts.	✓
JOB	-client *source_client*	The number that identifies the SAP client where the job definition is to be found, regardless of the client number defined by the `r3client` keyword in the options file. This parameter has no effect if a job ID is specified in the job definition.	
	-exec_client *target_client*	The number that identifies the SAP client where the job is to be run, regardless of the client number defined by the r3client keyword in the options file. This parameter has no effect if a job ID is specified in the job definition.	

Table 149. Task string parameters for SAP jobs

(continued)

Section	Parameters	Description	GUI Support
	-rfc_client *rfc_logon_client*	The number that identifies the SAP client to be used for RFC logon. This value overwrites the value specified by the r3client keyword in the corresponding r3batch options file.	
	-c *class_value*	The priority with which the job runs in the SAP system. Possible values are: **A** High priority **B** Medium priority **C** Low priority. This is the default value.	✓
	-bdc_job_status_failed *bdc_processing*	How HCL Workload Automation sets the completion status of a job running BDC sessions, according to a possible BDC processing failure. The allowed values are: ***n*** If at least *n* BDC sessions failed (where *n* is an integer greater than 0), HCL Workload Automation sets the job completion status as failed. **all** If all the BDC sessions failed, HCL Workload Automation sets the job completion status as failed. **ignore** When all the BDC sessions complete, regardless of their status, HCL Workload Automation sets the job completion status as successful. This is the default. If **-nobdc** or **-nobdcwait** is set, this option is ignored.	✓

Table 149. Task string parameters for SAP jobs

(continued)

Section	Parameters	Description	GUI Support
	-nobdc -nobdcwait	Disables the BDC Wait option (enabled by default) to have the job considered as completed even if not all its BDC sessions have ended.	✓
	-bapi_sync_level	Specifies the synchronization level between the SAP function modules BAPI_XBP_JOB_COPY and BAPI_XBP_JOB_START_ASAP. Allowed values are: **high** All RFC calls between BAPI_XBP_JOB_START_ASAP and BAPI_XBP_JOB_COPY are synchronized. This is the default. **medium** The RFC calls to BAPI_XBP_JOB_START_ASAP are synchronized. **low** The RFC calls are not synchronized.	
STEP	-s *starting_step_number*	The number of the starting step.	✓
	-s*Step_number attribute_name=attribute_value*	The step number and its attributes, where: ***step_number*** The number of the step being defined. Each step is identified by a sequential number (1, 2, 3, ...*n*) using the *step number* ***attribute_name*** The name of the attribute. ***attribute_value*** The value of the attribute. It is optional for some attributes. Attributes can be defined in any order, but cannot be repeated for the same step. Attribute validation is performed before the job is created in the SAP system. If	✓

Table 149. Task string parameters for SAP jobs

(continued)

Section	Parameters	Description	GUI Support

		the validation fails, the HCL Workload Automation job goes into the ABEND state. For a detailed description of each attribute and its value, see Defining attributes for ABAP steps on page 1210 and Defining attributes for external programs and external commands steps on page 1214.
		
		For example, the following step (step 8) is an ABAP module running the report MYPGM and has two attributes, only one of which has a value.
		`-s8 type=A -s8 program=MYPGM`
		`-s8 pr_cover="My title" -s8 pr_immed`	

VARIANT	-v*step_number* name	The variant name for the specified step number.	✓
	-vtxt*step_number variant_description*	The textual description of the variant, in the HCL Workload Automation logon language (customizable with the `TWSXA_LANG` option of `r3batch`). The maximum length is 30 characters.	✓
	-vpar*step_number* name=*value*	For ABAP modules only. The value for a variant parameter for the specified step number. This parameter is mandatory when creating a new variant. See Defining attributes for ABAP steps on page 1210 for a complete list of the supported attributes for ABAP steps.	✓
	-vsel*step_number* name=*sign#operation#lowest*[*#highest*]	For ABAP modules only. The value for a variant selection option for the specified step number. **sign** Sign of the operation. Possible values are: **I** Include **E** Exclude **operation** Possible values are:	✓

Table 149. Task string parameters for SAP jobs

(continued)

Section	Parameters	Description	GUI Support
		EQ Equals **NE** Not equal to **BT** Between **NB** Not between **LT** Less than **LE** Less than or equal to **GT** Greater than **GE** Greater than or equal to **CP** Contains pattern **NP** Does not contain pattern **lowest** Low value of the selection. You can use up to 45 characters. **highest** High value of the selection. You can use up to 45 characters. This attribute is optional. For a complete list of the supported attributes for ABAP steps, seeDefining attributes for ABAP steps on page 1210.	

Table 149. Task string parameters for SAP jobs

(continued)

Section	Parameters	Description	GUI Support
	-vtemp*step_number*	For ABAP modules only. Specifies to assign a temporary variant to the specified step number. Temporary variants are created ad-hoc by the SAP system and assigned to the job instance when it is run. The lifecycle of the temporary variant is determined by the SAP system. If the job is deleted by SAP, then the temporary variant is deleted. See Examples: Dynamically defining and updating SAP jobs on page 1216 to refer to examples that demonstrate the behavior of temporary variants.	
SPOOL	-recipient *name*	The login name of an SAP user.	
	-flag {reccp|recbl}	Specifies how the spool list is sent to the recipient. Possible values are: **reccp** The spool list is sent as a copy. **recbl** The spool list is sent as a blind copy.	
	-flag recex	Specifies that the spool list is sent as an express message to the recipient.	
	-flag recnf	Specifies that the recipient is not allowed to forward the spool list.	
	-rectype type	Specifies the recipient type. Possible values are: ' ' SAP user (default value) **'B'** SAP user **'C'** Shared distribution list **'D'** X.500 address	

Table 149. Task string parameters for SAP jobs

(continued)

Section	Parameters	Description	GUI Support
		'G' Organization object/ID **'H'** Organization unit **'I'** SAP object **'L'** Telex number **'O'** SAPoffice user **'P'** Private distribution list **'R'** SAP user in another SAP system **'U'** Internet address **'1'** Other recipient type	
FLAGS	-flag im -flag immed	Specifies to launch job immediately, meaning that if there are no spare work processes, the job fails.	✓
	-flag enable_applinfo -flag disable_applinfo	Enables or disables the retrieval and appending of the SAP application log to the stdlist of HCL Workload Automation.	✓
	-flag enable_appl_rc -flag disable_appl_rc	Enables or disables the mapping of the SAP application return code to the HCL Workload Automation return code. The SAP application return code is mapped only if **-flag enable_applinfo** is set and the application log contains the application return code.	

Table 149. Task string parameters for SAP jobs

(continued)

Section	Parameters	Description	GUI Support
	-flag enable_joblog -flag disable_joblog	Enables or disables retrieval of the joblog.	✓
	-flag enable_job_interceptable -flag disable_job_interceptable	Enables or disables the job launched by r3batch to be intercepted by SAP. If enabled, when r3batch launches a job and the SAP job interception feature is enabled, the job can be intercepted if it matches previously defined criteria. If disabled, the job launched by r3batch cannot be intercepted by SAP. This setting overwrites the setting in the common options file.	✓
	-flag enable_spoollist -flag disable_spoollist	Enables or disables retrieval of the spool lists of the job.	✓
-flag pc_launch	Specifies to launch child jobs that are in scheduled state. **ON** The product launches child jobs that are in scheduled state. **OFF** The product does not launch child jobs that are in scheduled state. This is the default value. **Note:** You can use this option only if you activated the parent-child feature on the SAP system. On the XBP 2.0 (or later)SAP system you can activate this feature using the INITXBP2 ABAP report		
TRACING	-debug	Enables maximum trace level.	✓

Table 149. Task string parameters for SAP jobs

(continued)

Section	Parameters	Description	GUI Support
	-tracelvl 1|2|3	Specifies the trace setting for the job. Possible values are: **1** Only error messages are written in the trace file. This is the default. **2** Informational messages and warnings are also written in the trace file. **3** A most verbose debug output is written in the trace file. For detailed information, refer to Configuring the tracing utility on page 1120.	✓
	-rfctrace -trace	Enables RFC trace.	

The following is an example for an SAP job named `BVTTEST` with ID `03102401` and user `myuser`:

```
-job BVTTEST -i 03102401 -user myuser -debug
```

Managing SAP jobs

This section describes how to manage SAP jobs.

Displaying details about a standard SAP job

About this task

Perform the following steps to display details for standard jobs on specific workstations.

For information about how to display details about a job that submits an SAP process chain, refer to Displaying details about a process chain job on page 1241.

1. Click **Design > SAP > Manage Jobs on SAP**.
2. In **Engine name**, select the name of the HCL Workload Automation engine connection from which you want to view SAP job details.
3. In **Workstation name**, type the name of the workstation where the SAP job runs. This is the workstation with the r3batch access method that communicates with the remote SAP system. If you do not know the name of the workstation, click (...) browse to enter your filter criteria and click **Search**. If you enter a string representing part of the workstation name, it must be followed by the asterisk (*) wildcard character. Both the question mark (?) and asterisk (*) are supported as wildcards. You can also simply use the asterisk wildcard character (*) to display all workstations. Optionally, specify any of the other search criteria available and click **Search**.
   Result
   From the results displayed, select the workstation and click **OK**.
4. In **Options file**, specify an options file that resides on the specified workstation. Each workstation can have one or more options files that can be used to customize the behavior of the r3batch access method, except for extended agent workstations, where only one options file can exist and therefore does not need to be specified. For the workstation specified, enter the file name of the options file or click the browse (...) button to search for options files that reside on the specified workstation and select one.
5. Click **Display**. The list of available jobs for the specified engine is displayed.
6. Select the job for which you want to display the details and click **Details**. The List Jobs on SAP panel is displayed containing job and time information.
7. When you have finished viewing the details for the job, click **OK** to return to the list of SAP jobs on the workstation specified.

Verifying the status of a standard SAP job

About this task

To verify the status of a standard SAP job, perform the following steps:

1. Click **Design > SAP > Manage Jobs on SAP**.
2. In **Engine name**, select the name of the HCL Workload Automation engine connection from which you want to verify the status of an SAP job.
3. In **Workstation name**, type the name of the workstation where the SAP job runs. This is the workstation with the r3batch access method that communicates with the remote SAP system. If you do not know the name of the workstation, click (...) browse to enter your filter criteria and click **Search**. If you enter a string representing part of the workstation name, it must be followed by the asterisk (*) wildcard character. Both the question mark (?) and asterisk (*) are supported as wildcards. You can also simply use the asterisk wildcard character (*) to display all workstations. Optionally, specify any of the other search criteria available and click **Search**.
   Result
   From the results displayed, select the workstation and click **OK**.
4. In **Options file**, specify an options file that resides on the specified workstation. Each workstation can have one or more options files that can be used to customize the behavior of the r3batch access method, except for extended agent workstations, where only one options file can exist and therefore does not need to be specified. For the

workstation specified, enter the file name of the options file or click the browse (...) button to search for options files that reside on the specified workstation and select one.
5. Click **Display**. The list of available jobs for the specified engine is displayed.
6. Select the job for which you want to verify the status and click **Status**. The current status for the SAP job is displayed, as well as the database name where the job is installed.
7. When you have finished verifying the status for the job, click **OK** to return to the list of SAP jobs on the workstation specified.

Deleting a standard SAP job from the SAP database

About this task

To delete a standard SAP job from the SAP database, perform the following steps:

1. Click **Design > SAP > Manage Jobs on SAP**.
2. In **Engine name**, select the name of the HCL Workload Automation engine connection from which you want to delete the SAP job.
3. In **Workstation name**, type the name of the workstation where the SAP job runs. This is the workstation with the r3batch access method that communicates with the remote SAP system. If you do not know the name of the workstation, click (...) browse to enter your filter criteria and click **Search**. If you enter a string representing part of the workstation name, it must be followed by the asterisk (*) wildcard character. Both the question mark (?) and asterisk (*) are supported as wildcards. You can also simply use the asterisk wildcard character (*) to display all workstations. Optionally, specify any of the other search criteria available and click **Search**.
   Result
   From the results displayed, select the workstation and click **OK**.
4. In **Options file**, specify an options file that resides on the specified workstation. Each workstation can have one or more options files that can be used to customize the behavior of the r3batch access method, except for extended agent workstations, where only one options file can exist and therefore does not need to be specified. For the workstation specified, enter the file name of the options file or click the browse (...) button to search for options files that reside on the specified workstation and select one.
5. Click **Display**. The list of available jobs for the specified engine is displayed.
6. Select the job or jobs you want to delete and click **Delete**. A confirmation message prompts you to confirm the delete action.
7. When the delete action is complete, click **OK** to return to the list of SAP jobs on the workstation specified.

Balancing SAP workload using server groups

SAP jobs run on application servers that host work processes of type *batch*. Critical batch jobs are run in specific time frames, on specific application servers. With SAP Basis version 6.10 and later, application servers can be assigned to server groups. With HCL Workload Automation you can assign a server group to a job. In this way, when a job is launched, the SAP system runs it on an application server that belongs to that group, balancing the workload among the various application servers in the group.

If the application servers defined in a group are modified in the SAP system, the job defined as belonging to that server group is not affected and does not need to be modified. The batch execution targets are reorganized in the SAP system without having to change job definitions in HCL Workload Automation.

This function is supported with the following versions of SAP:

- SAP Basis 6.10, with Service Pack 40
- SAP Basis 6.20, with Service Pack 41
- SAP Basis 6.40, with Service Pack 04
- SAP Basis 7.00, and later

Mapping between HCL Workload Automation and SAP job states

When an SAP job is launched by HCL Workload Automation, you can monitor its progress. The status transitions in HCL Workload Automation (internal status) and the corresponding SAP status are listed in Table 150: Status transitions in HCL Workload Automation (internal status) and the corresponding SAP R/3 status on page 1193.

Table 150. Status transitions in HCL Workload Automation (internal status) and the corresponding SAP R/3 status

HCL Workload Automation Job State	SAP Job State
INTRO	Not Available
WAIT	Ready, Release
EXEC	Active

Table 150. Status transitions in HCL Workload Automation (internal status) and the corresponding SAP R/3 status (continued)

HCL Workload Automation Job State	SAP Job State
SUCC	Finished
ABEND	Canceled

The INTRO state indicates that HCL Workload Automation is in the process of introducing the job, but in SAP, the job has not yet entered the ready state. Because it takes some time to get a job queued and into the ready column, the INTRO state might last a few minutes if the SAP system is particularly busy.

Even if a job is finished in SAP, HCL Workload Automation keeps it in the EXEC state if its BDC sessions are not complete and you have not selected the **Disable BDC Wait** option. For details about this option, see Using the BDC Wait option on page 1222.

Managing spools

Browse spool lists on request without having to download the entire spool which can occupy significant space on the file system.

Spool lists can be very large so rather than download them as part of a job run, you can request to browse the spool list, chunks at a time, even if you have disabled the option, retrieve_spoollist, to append the spool list to the HCL Workload Automation joblog.

From the Dynamic Workload Console, you can list the spool data available for SAP Standard R/3 jobs that have run. Each spool is identified by the following information:

- The spool number.
- The related step number.
- The name of the spool request.
- The title of the spool request.
- The total number of pages for the spool information.
- The user who executed the SAP job related to the spool.
- The date the spool was created based on the Coordinated Universal Time (UTC) time standard.
- The client for which the spool was created.

Browsing spool data

You can list the spool data available for SAP Standard R/3 jobs that have run and browse the contents of the spool.

About this task

To browse spool data for a specific job that has run:

1. From the **Monitoring and Reporting** menu, click **Orchestration Monitor**.
2. In the **Monitor Workload** input fields enter the engine name, the plan, and any filtering data that helps you filter the selection of jobs (you can also select **Edit** for a guided selection of filtering criteria) and select **Run**.
3. In the output table select an SAP Standard R/3 job and click **More Actions > Show Spool List**. The list of spool data available for the selected job is displayed.
4. Select a spool and click **Spool**.

Results

By default, the first ten pages of the spool are made available. You can change this default by editing the number of pages specified in **Pages for screen**. Use the page functions to jump to a specific page number, jump to the last page of the spool, jump to the first page of the spool, or move forward or back through the number of pages indicated by **Pages for screen**.

Killing an SAP job instance

About this task

This section describes how to kill an HCL Workload Automation job that submits either a standard SAP job or an SAP process chain.

To kill an SAP job instance, do the following:

The HCL Workload Automation job status is set to ABEND. The SAP job or process chain is set to canceled in the SAP system.

Note: If you kill a process chain job, the SAP system stops as soon as the process that is currently running completes.

1. Use the **Monitor Workload** query of the Dynamic Workload Console to display a list of defined job instances containing the job you want to kill. From the **Monitoring and Reporting** menu, click **Orchestration Monitor**.
2. In the **Monitor Workload** input fields enter the engine name, the plan, and any filtering data that helps you filter the selection of jobs (you can also select **Edit** for a guided selection of filtering criteria) and select **Run**.
3. The **Monitor Jobs** panel is displayed. Select the job instance you want to kill and click **More Actions > Kill**.

Raising an SAP event

About this task

You can raise events on XBP 2.0 (or later) SAP jobs in the HCL Workload Automation database in one of the following ways:

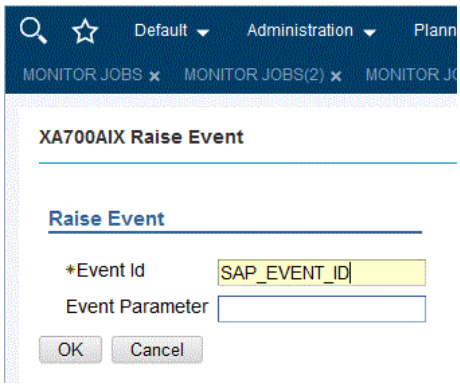
Using the Monitor Workload in the Dynamic Workload Console

Perform the following steps:

1. On the SAP system, create a job that has as start condition a SAP event. When you create this job, its status is released.
2. Check that this job was not intercepted by the interception function.

3. Log in to the Dynamic Workload Console.
4. From the **Monitoring and Reporting** menu, click **Orchestration Monitor**.
5. In the Monitor Workload window select the engine, enter **Workstation** in the Object Type field, and select the plan to display the list of workstations you want to monitor. Click **Run**.

   A list of workstations is displayed.
6. Select a workstation that has been defined to connect to a remote SAP system.
7. From the toolbar, select **More Actions > Raise Event**. The Raise Event panel opens.
   Figure 33. The Raise Event panel



8. The panel consists of the following:

   Event ID

   The identifier of the event that is to be raised.

   Event Parameter

   The parameter of the event that is to be raised.
9. Click **OK**. The event is raised.

Creating a job that launches a Windows™ or UNIX™ command that raises an event

Perform the following steps:

1. From the Design menu, click Workload Designer page.
2. Specify an engine name, either distributed or z/OS. The Workload Designer window opens. Job types and characteristics vary depending on whether you select a distributed or a z/OS engine.
3. Click **Create new** and select **Job definition**.
4. Select the **Native** category and then either **Windows** or **UNIX** job.
5. Use the **General** section to provide general information about the new job definition.
6. Use the **Task** section to provide task information for the job.
7. In the **Task** section, select **Command** and in the command string type the following command that raises the event:

```
<data_dir>/methods/r3event -c workstation_name -u user_name
-e SAP_event_ID -p parameter
```

where:

workstation_name

The name of the workstation where the SAP R/3 job is defined.

user_name

The name of the SAP user with which the access method connects to the SAP system. This is the name specified in the r3user option.

SAP_event_ID

The identifier of the event.

parameter

The parameter defined for the event.

8. Save the job definition.

What to do next

See Defining conditions and criteria on page 1218 for information about how to define criteria that manages which raised events to log.

Rerunning a standard SAP job

You can rerun a standard SAP job from the start, or from a specific numeric step of the SAP instruction.

About this task

To rerun a standard SAP job, you can use one of the following user interfaces:

conman

For details, refer to the *HCL Workload Automation: User's Guide and Reference*.

Dynamic Workload Console

Dynamic Workload Console

For details about how to rerun a job that submits an SAP process chain, refer to Rerunning a process chain job on page 1243.

For an SAP extended agent, a `step` is the numeric step of the SAP instruction from which a job can be restarted. Before you rerun an SAP job with HCL Workload Automation, you have the option of providing a step name for the job. This affects `r3batch` in the following ways:

- If you use a step name that is up to 9 digits (or 8 digits preceded by a character) in length, this name is used as the starting step number for the rerunning job.
- If you use any different format, the name is ignored and the job is rerun starting from the first step.

For example, to rerun a job from the third step, you can use: `A03`, `3`, `00003`, or `H3`.

In z/OS environments, you need to set the status of the job to **Ready** before you can rerun the job.

Note: By default, if you specify a job step to rerun, the new job is assigned the name of the step you indicated. To keep the original job name, set the HCL Workload Automation global option **enRetainNameOnRerunFrom** to `yes`. This option works only when used with the following arguments: `rr jobselect;from=[wkstat#]job`. For details about these arguments, see *HCL Workload Automation: User's Guide and Reference*, *Managing objects in the plan - conman*, *Conman commands*, *rerun*. For details about this option, see *HCL Workload Automation: Planning and Installation*.

When `r3batch` reruns a job from its first step, either because you specified it as the starting step or because no starting step was specified, it uses the `new copy` feature, if applicable. If the starting step is greater than one, `r3batch` uses the `old copy` to rerun the job. For a description about the difference between the new and old copy of a rerunning job, refer to Old copy and new copy of a rerunning job on page 1199.

To rerun a SAP Standard R/3 job from the Dynamic Workload Console, perform the following steps:

1. In the Dynamic Workload Console select **Monitoring & Reporting > Workload Monitoring > Orchestration Monitor**.
2. In the Orchestration Monitor input fields select **Job** as the Object Type, the engine name, the plan, and any filtering data that helps you filter the selection of jobs (you can also select **Edit** for a guided selection of filtering criteria) and select **Run**.
3. A list of jobs is displayed. Select an SAP Standard R/3 job.
4. Rerun the job.

   Distributed environment

   a. Click **Rerun...**. The General properties for the rerun operation are displayed.
   b. Optionally, you can choose to not rerun the same job but instead, substitute the selected SAP job with a different job definition and run it. Type the job definition name in the **From Job Definition** field, or use the browse button to search for it and select it.
   c. Optionally, type the workstation name of the workstation on which you want to rerun the job in the **Workstation Name** field.
   d. Optionally, in **Step**, enter a specific numeric step of the SAP instruction from which you want to rerun the job rather than rerunning the whole job.
   e. Optionally, specify the start and finish time for the job.
   f. Click **Rerun**.

   The job reruns immediately or at the specified start time.

   z/OS environment

   In a z/OS environment, an alias for the job name is not required so the job reruns with the same name. The list of jobs always reports the latest action performed on the job.

   a. Before you can rerun a job, you must change the status of the job to **Ready**. Select a job and click **Set Status**.
   b. In **Change Status**, select **Ready**.
   c. Click **OK** to return to the list of jobs.

   The job reruns immediately and the internal status reports **Started**.

Old copy and new copy of a rerunning job

When the access method for SAP launches a job, it makes a copy of a template job and runs it.

The `new copy` feature is available for SAP versions 3.1i, and later. It copies an entire job, preserving steps, job class, and all print and archive parameters. It is performed by using a new SAP function module that is part of the SAP Support Package as stated in the SAP Notes 399449 and 430087.

The `old copy` feature, instead, is based on standard SAP function modules, and creates a new SAP job and adds the steps with a loop that starts from the step name or number you specified. Be aware that, unless you have XBP 2.0 or later:

- The `old copy` does not preserve all the print and archive parameters.
- The job class of the copy is always set to class `C`.

Refer to Print parameter and job class issues on page 1143 to learn how to resolve the problem of lost job class and print and archive parameters.

SAP Note 758829 is required to ensure correct operation of the `new copy` and `old copy` features. See also Table 174: Miscellaneous troubleshooting items on page 1297.

Defining SAP jobs dynamically

This section describes how to create and submit SAP jobs dynamically without creating or referencing predefined job templates.

When you launch a job created as described in Creating SAP Standard R/3 jobs from the Dynamic Workload Console on page 1172 and Task string to define SAP jobs on page 1180, HCL Workload Automation makes a copy of the predefined job (also known as a template job) and runs the copy. If you want to run the job on several SAP systems, you must manually create the template job on each system.

To create and submit SAP jobs dynamically, without creating or referencing predefined job templates, submit:

- In the SAP system, a job that does not reference an existing template in the SAP R/3 database.
- A job that references a predefined job template stored in the SAP R/3 database for which you want to change a parameter.

To take full advantage of this feature, make sure that you have XBP version 2.0 or later installed, because earlier versions of XBP do not support the full set of print and archive parameters, or provide a way to set the job class or the spool list recipient.

Task string to define SAP jobs dynamically

This section describes the task string that controls the running of SAP jobs. You can build an entire job definition by using the six main sections concerning SAP job parameters. These sections are grouped in Table 151: Task string parameters for SAP jobs (dynamic definition) on page 1201 and are related to the:

- Job
- Job steps
- Variants associated with the steps (for ABAP modules only)
- Spool list recipients associated with the job
- Flags associated with the job
- Tracing specifications for the job

You can specify them in the following places when you define their associated HCL Workload Automation jobs:

- In the **SAP Command Line** section of the Task page of the Submit Ad Hoc Jobs action from the Dynamic Workload Console.
- In the **SAP Command Line** field of the More Options page of the SAP job definition, if you use the Dynamic Workload Console and selected a **SAP** job definition.
- As arguments of the `scriptname` keyword in the job definition statement, if you use the HCL Workload Automation command line.
- As arguments of the `JOBCMD` keyword in the `JOBREC` statement in the `SCRIPTLIB` of HCL Workload Automation for Z, if you are scheduling in an end-to-end environment. The following is an example of a `JOBREC` statement:

```
JOBREC
  JOBCMD('/-job job_name -user user_name -i job_ID -c class_value')
  JOBUSR(TWS_user_name)
```

To define and submit an SAP job dynamically, use the following syntax:

Job definition syntax

-job *job_name* [{ **-i | -id** } *job_ID*] **-flag type=exec** [{ **-host | -ts** } *host_name*] [**-sg** *server_group*] [**-client** *source_client*] [**-exec_client** *target_client*] [**-rfc_client** *rfc_logon_client*] [**-c** *class_value*] [**-bdc_job_status_failed** *bdc_processing*] [{ **-nobdc | -nobdcwait** } [**-bapi_sync_level** { **high | medium | low** }] [**-s** *starting_step_number*]] [**-s** *Step_number attribute_name* [= *attribute_value*]] [**-v** *Step_number variant_name*] [**-vtxt** *Step_number variant_description*] [**-vpar** *Step_number* **name** = *variant_value*] [**-vsel** *Step_number* **name** = { **i | e** } # *operation* # *lowest* [# *highest*]] [**-vtemp** *Step_number*] [**-recipient** *R/3_login_name*] [**-rectype** *recipient_type*] [**-flag** { **reccp | recbl** }] [**-flag recex**] [**-flag recnf**] [**-flag** { **im | immed** }] [**-flag** { **enable_applinfo | disable_applinfo** }] [**-flag** { **enable_appl_rc | disable_appl_rc** }] [**-flag** { **enable_joblog | disable_joblog** }] [**-flag** { **enable_job_interceptable | disable_job_interceptable** }] [**-flag** { **enable_spoollist | disable_spoollist** }] [**-flag pc_launch**] [{ **-debug** }] [**-tracelvl** { **1 | 2 | 3** }] [{ **-rfctrace** }] [**-rfc_client** *rfc_logon_client*]

The following is an example of a definition for the SAPTEST job:

```
-job SAPTEST -C A -s1 program=BTCTEST -s1 type=A -s1 pr_release
-s2 report=BTCTEST -s2 variant=BVT -s2 type=A -flag type=exec
-vpar2 TESTNAME=test -vtxt2 Test
```

Table 151: Task string parameters for SAP jobs (dynamic definition) on page 1201 describes the parameters for the task string to define SAP jobs dynamically.

Note: The parameter values are case sensitive.

Table 151. Task string parameters for SAP jobs (dynamic definition)

Section	Parameters	Description
JOB	-job *job_name*	The name of the job to be run. This parameter is mandatory.
	-i *job_ID* -id *job_ID*	The unique SAP job ID. Specify this parameter if you are submitting a job that refers to a predefined job template stored in the SAP database for which you want to change a parameter.

Table 151. Task string parameters for SAP jobs (dynamic definition)

(continued)

Section	Parameters	Description
	-host *host_name* -ts *host_name*	The name of the SAP workstation where the job is to be run. *host_name* has the format *hostname_SAPsystemname_SAPsystemnumber*. For example, the name of a host might be amss80a0_gs7_90 These parameters are mutually exclusive with -sg.
	-sg *server_group*	The name of the SAP server group where the job is to be run. Use this parameter to run the job on an application server that belongs to the group. The server group must exist on the SAP system, otherwise an error code is returned and the job is not launched. This parameter is case-sensitive and can be up to 20 characters. It is mutually exclusive with -host and -ts.
	-client *source_client*	The number that identifies the SAP client where the job definition is to be found, regardless of the client number defined by the **r3client** key in the options file. This parameter has no effect if a job ID is specified in the job definition.
	-exec_client *target_client*	The number that identifies the SAP client where the job is to be run, regardless of the client number defined by the `r3client` key in the options file. This parameter requires that the client-dependent data (such as the user name and report variants) exists on both the source and target clients. This parameter has no effect if a job ID is specified in the job definition.
	-rfc_client *rfc_logon_client*	The number that identifies the SAP client to be used for RFC logon. This value overwrites the value specified by the r3client keyword in the corresponding r3batch options file.
	-c *class_value*	The priority with which the job runs in the SAP system. Possible values are: **A** High priority

Table 151. Task string parameters for SAP jobs (dynamic definition)

(continued)

Section	Parameters	Description
		B Medium priority **C** Low priority. This is the default value.
	-flag type=exec	Specify this parameter to enable the dynamic definition of the SAP job. This parameter is mandatory.
JOB	-bdc_job_status_failed *bdc_processing*	How HCL Workload Automation sets the completion status of a job running BDC sessions, according to a possible BDC processing failure. The allowed values are: ***n*** If at least *n* BDC sessions failed (where *n* is an integer greater than 0), HCL Workload Automation sets the job completion status as failed. **all** If all the BDC sessions failed, HCL Workload Automation sets the job completion status as failed. **ignore** When all the BDC sessions complete, regardless of their status, HCL Workload Automation sets the job completion status as successful. This is the default value. If **-nobdc** or **-nobdcwait** is set, this option is ignored.
	-nobdc -nobdcwait	Disables the BDC Wait option (enabled by default) to have the job considered as completed even if not all its BDC sessions have ended.
	-bapi_sync_level	Specifies the synchronization level between the SAP function modules BAPI_XBP_JOB_COPY and BAPI_XBP_JOB_START_ASAP. Allowed values are:

Table 151. Task string parameters for SAP jobs (dynamic definition)

(continued)

Section	Parameters	Description
		high All RFC calls between BAPI_XBP_JOB_START_ASAP and BAPI_XBP_JOB_COPY are synchronized. This is the default. **medium** The RFC calls to BAPI_XBP_JOB_START_ASAP are synchronized. **low** The RFC calls are not synchronized.
STEP	-s *starting_step_number*	The number of the starting step.
	-s*step_number attribute_name=attribute_value*	The step number and its attributes, where: ***step_number*** The number of the step being defined. Each step is identified by a sequential number (1, 2, 3, ...*n*) using the *step number*. ***attribute_name*** The name of the attribute. ***attribute_value*** The value of the attribute. It is optional for some attributes. Attributes can be defined in any order, but cannot be repeated for the same step. Attribute validation is performed before the job is created in the SAP system. If the validation fails, the HCL Workload Automation job goes into the ABEND state. For a detailed description of each attribute and its values, see Defining attributes for ABAP steps on page 1210 and Defining attributes for external programs and external commands steps on page 1214. For example, the following step (step 8) is an ABAP module running the report "MYPGM" and has two attributes, only one of which has a value.

Table 151. Task string parameters for SAP jobs (dynamic definition)

(continued)

Section	Parameters	Description
		`-s8 type=A -s8 program=MYPGM` `-s8 pr_cover="My title" -s8 pr_immed`
VARIANT[1]	-v*step_number* name	The variant name for the specified step number.
	-vtxt*step_number variant_description*	The textual description of the variant, in the HCL Workload Automation logon language (customizable with the `TWSXA_LANG` option of `r3batch`). The maximum length is 30 characters. Not valid for temporary variants.
	-vpar*step_number* name=*value*	For ABAP modules only. The value for a variant parameter for the specified step number. This parameter is mandatory when creating a new variant. For a complete list of the supported attributes for ABAP steps, see Defining attributes for ABAP steps on page 1210.
	-vsel*step_number* name=*sign#operation#lowest*[*#highest*]	For ABAP modules only. The value for a variant selection option for the specified step number. **sign** Sign of the operation. Possible values are: **I** Include **E** Exclude **operation** Possible values are: **EQ** Equals **NE** Not equal to **BT** Between **NB** Not between

Table 151. Task string parameters for SAP jobs (dynamic definition)

(continued)

Section	Parameters	Description
		LT Less than **LE** Less than or equal to **GT** Greater than **GE** Greater than or equal to **CP** Contains pattern **NP** Does not contain pattern **lowest** Low value of the selection. You can use up to 45 characters. **highest** High value of the selection. You can use up to 45 characters. This attribute is optional. For a complete list of the supported attributes for ABAP steps, see Defining attributes for ABAP steps on page 1210.
	-vtemp*step_number*	For ABAP modules only. Specifies to assign a temporary variant to the specified step number. Temporary variants are created ad-hoc by the SAP system and assigned to the job instance when it is run. The lifecycle of the temporary variant is determined by the SAP system. If the job is deleted by SAP, then the temporary variant is deleted. See Examples: Dynamically defining and updating SAP jobs on page 1216 to refer to examples that demonstrate the behavior of temporary variants.
SPOOL	-recipient *name*	The login name of an SAP user.

Table 151. Task string parameters for SAP jobs (dynamic definition)

(continued)

Section	Parameters	Description
	-flag {reccp|recbl}	Specifies how the spool list is sent to the recipient. Possible values are: **reccp** The spool list is sent as a copy. **recbl** The spool list is sent as a blind copy.
	-flag recex	Specifies that the spool list is sent as an express message to the recipient.
	-flag recnf	Specifies that the recipient is not allowed to forward the spool list.
	-rectype *type*	Specifies the recipient type. Possible values are: **blank** SAP user (default value) **B** SAP user **C** Shared distribution list **D** X.500 address **G** Organization object/ID **H** Organization unit **I** SAP object **L** Telex number **O** SAPoffice user

Table 151. Task string parameters for SAP jobs (dynamic definition)

(continued)

Section	Parameters	Description
		P Private distribution list **R** SAP user in another SAP system **U** Internet address **1** Other recipient type
FLAGS	-flag im -flag immed	Specifies to launch the job immediately, meaning that if there are no spare work processes, the job fails.
	-flag enable_applinfo -flag disable_applinfo	Enables or disables the retrieval and appending of the SAP application log to the stdlist of HCL Workload Automation.
	-flag enable_appl_rc -flag disable_appl_rc	Enables or disables the mapping of the SAP application return code to the HCL Workload Automation return code. The SAP application return code is mapped only if **-flag enable_applinfo** is set and the application log contains the application return code.
	-flag enable_joblog -flag disable_joblog	Enables or disables retrieval of the joblog.
	-flag enable_joblog -flag disable_joblog	Enables or disables retrieval of the joblog.
	-flag enable_job_interceptable -flag disable_job_interceptable	Enables or disables the job launched by r3batch to be intercepted by SAP. If enabled, when r3batch launches a job and the SAP job interception feature is enabled, the job can be intercepted if it matches previously defined criteria. If disabled, the job launched by r3batch cannot be intercepted by SAP. This setting overwrites the setting in the common options file.

Table 151. Task string parameters for SAP jobs (dynamic definition)

(continued)

Section	Parameters	Description
-flag pc_launch	Specifies to launch child jobs that are in scheduled state. **ON** The product launches child jobs that are in scheduled state. **OFF** The product does not launch child jobs that are in scheduled state. This is the default value. **Note:** You can use this option only if you activated the parent-child feature on the SAP system. On the XBP 2.0 (or later) SAP system, you activate this feature by using the INITXBP2 ABAP report.	
TRACING	-debug	Enables maximum trace level.
	-tracelvl 1|2|3	Specifies the trace setting for the job. Possible values are: **1** Only error messages are written in the trace file. This is the default. **2** Informational messages and warnings are also written in the trace file. **3** A most verbose debug output is written in the trace file. For more details, refer to Configuring the tracing utility on page 1120.
		Enables RFC trace.

Table 151. Task string parameters for SAP jobs (dynamic definition)

(continued)

Section	Parameters	Description
	-rfctrace -trace	

Note: See Examples: Dynamically defining and updating SAP jobs on page 1216 to refer to examples that demonstrate the behavior of variants and temporary variants.

1. The following rules apply when you create or update SAP jobs dynamically:
   - To create or reference a variant within an ABAP step, you can use one of the following equivalent syntaxes:
     - -s1 Variant=Var1
     - -s1 Parameter=Var1
     - -v1 Var1
   - If a variant does not exist, it is created with the parameters specified in the job definition statement. In this case, all the required attributes of the variant must be given a value. You cannot create empty variants. For example, if you specify **-vtemp1**, with no value assigned, an empty temporary variant is erroneously created.
   - If a variant is already present in the SAP system, its values are modified according to the command line parameters. If the existing variant is an extended one, a new instance of it is created with resolved placeholders and updated counters. This new variant instance is then updated using the values from the command line. Finally, the job step is run using this variant instance.
   - All changes to the variant values are permanent. That is, HCL Workload Automation neither restores the old values of the variants, nor deletes the variants created after the job is run. HCL Workload Automation does not change the case of the variant values.

Defining attributes for ABAP steps

To create and submit SAP jobs dynamically, look at the table and define the attributes for ABAP steps.

Table 152: Supported attributes for ABAP step definition on page 1211 shows a complete list of the supported attributes for ABAP step module definition:

Table 152. Supported attributes for ABAP step definition

Attribute name	Synonym	Description	Required
type	typ	Specify the step type. Possible values are: • A • ABAP The product performs a check for correct attribute values prior to launching the job.	✓
program		Specify the ABAP program name.	✓
parameter		Specify the ABAP variant name.	✓
user	authcknam	Specify the user of the step.	✓
language	lang	Specify the step language. This attribute accepts language names in either the ISO format (two characters, for example DE, EN) or the R/3 format (one character, for example D, E). If this attribute is not specified, the login language of the access method is used (customize using the option **twsxa_lang** in the r3batch options files). The product performs a check for a valid language prior to launching the job.	✓
pr_dest	printer pdest	Print Parameter: Specify the printer for the output.	
pr_copies	prcop	Print Parameter: Specify the number of copies. The value of this attribute must be numeric. A corresponding check is performed prior to launching the job.	
pr_lines	linct	Print Parameter: Specify the page length. The value of this attribute must be numeric. A corresponding check is performed prior to launching the job.	
pr_columns	linsz	Print Parameter: Specify the page width.	

Table 152. Supported attributes for ABAP step definition (continued)

Attribute name	Synonym	Description	Required
		The value of this attribute must be numeric. A corresponding check is performed prior to launching the job.	
pr_auth	prber	Print Parameter: Authorization	
pr_arcmode	armod	Print Parameter: Archiving mode	
pr_sapbanner	prsap	Print Parameter: SAP cover page	
pr_exp	pexpi	Print Parameter: Spool retention period The value of this attribute must be a single digit. A corresponding check is performed prior to launching the job.	
pr_recip	prrec	Print Parameter: Recipient	
pr_spoolname	plist	Print Parameter: Name of spool request [1]	
pr_format	paart	Print Parameter: Print formatting [1]	
pr_dep	prabt	Print Parameter: Department on cover page [1]	
pr_spoolds	prdsn	Print Parameter: Name of spool data set [1]	
pr_spoolprio	priot	Print Parameter: Spool request priority [1]	
pr_immed	primm	Print Parameter: Print immediately [2]	
pr_release	prrel	Print Parameter: Delete after printing [2]	
pr_banner	prbig	Print Parameter: Selection cover page [2]	
pr_newspool	prnew	Print Parameter: New spool request [1] [2]	
pr_cover	prtxt	Print Parameter: Text for cover page [1]. If the string contains spaces it must be enclosed between single quotes.	
pr_hostcover	prunx	Print Parameter: Host spool cover page [1] . Possible values are: **' '** Blank. Does not use any cover page. **'X'** Prints the host cover page. **'D'** Prints the default host cover page.	
al_sapobject	sap_object	SAP ArchiveLink: Object type of business object	

Table 152. Supported attributes for ABAP step definition (continued)

Attribute name	Synonym	Description	Required
al_object	object	SAP ArchiveLink: Document type	
al_info	info	SAP ArchiveLink: Info field	
al_id	archiv_id	SAP ArchiveLink: Target storage system [1]	
al_doctype	doc_type	SAP ArchiveLink: Document class [1]	
al_rpchost	rpc_host	SAP ArchiveLink: PRC host [1]	
al_rpcserv	rpc_servic	SAP ArchiveLink: RPC service / RFC destination [1]	
al_iface	interface	SAP ArchiveLink: Name of communication connection component [1]	
al_client	mandant	SAP ArchiveLink: Client [1]	
al_report		SAP ArchiveLink: Report name [1]	
al_text	arctext	SAP ArchiveLink: Text information field [1]	
al_date	datum	SAP ArchiveLink: Archiving date [1]	
al_user	arcuser	SAP ArchiveLink: Data element for user [1]	
al_printer		SAP ArchiveLink: Target printer [1]	
al_format	formular	SAP ArchiveLink: Output format [1]	
al_path	archivpath	SAP ArchiveLink: Standard archive path [1]	
al_protocol	protokoll	SAP ArchiveLink: Storage connection protocol [1]	
al_version		SAP ArchiveLink: Version number [1]	

Note:

1. This attribute is available for BC-XBP 2.0 and later.
2. This attribute is a flag, that is, it does not have a value, for example: `-s2 pr_release`.

HCL Workload Automation performs the following syntax validation on job attributes:

- Only valid attributes are allowed.
- Checks if a particular attribute requires a value.
- The values of the following attributes are checked:
  - type
  - language

- pr_copies
- pr_lines
- pr_colums

Validation is performed before the job is created in the SAP system. If the validation fails, the HCL Workload Automation job goes into the ABEND state.

Defining attributes for external programs and external commands steps

Table 153: Supported attributes for external programs and external commands step definition on page 1214 shows a complete list of the supported attributes for external programs and external commands step definition.

Table 153. Supported attributes for external programs and external commands step definition

Attribute name	Synonym	Description	Required
type	typ	The step type can assume one of the following values: **For external programs** • X • EXTPRG **For external commands** • C • EXTCMD	✓

Table 153. Supported attributes for external programs and external commands step definition (continued)

Attribute name	Synonym	Description	Required
		Before launching the job, the product performs a check for correct attribute values.	
report		ABAP program name or name of the external program or command.	✓
parameter		Parameters for the external program or command.	
user	authcknam	User of the step.	
language	lang	Step language. This attribute accepts language names in either the ISO format (two characters, for example DE, EN) or the R/3 format (one character, for example D, E). If this attribute is not specified, the login language of the access method is used (customize using the **twsxa_lang** option in the `r3batch` option files). The product performs a check for a valid language prior to launching the job.	
targethost	xpgtgtsys	Target host for the external program or command. This name must be exactly the same as the name shown in the External Operating System Commands table in the SAP system (transaction `sm69`).	
os	opsystem	Operating system for the external command. This name must be exactly the same as the name shown in the External Operating System Commands table in the SAP system (transaction `sm69`).	
termcntl	waitforterm	Control flag: if an external command or program is to be run synchronously. [2]	
tracecntl		Control flag: if SAP tracing level 3 is activated for tracing SAPXPG, the program that starts an external command or program. [1] [2]	
stdoutcntl		Control flag: indicates if standard output from an external command or program is to be written to the job log. [1] [2]	
stderrcntl		Control flag: indicates if standard error from an external command or program is to be written to the job log. [1] [2]	

Note:

1. This attribute is available for BC-XBP 2.0 and later.
2. This attribute is a flag, that is, it does not have a value, for example: `-s2 pr_release`.

HCL Workload Automation performs the following syntax validation on job attributes:

- Only valid attributes are allowed.
- Checks if a particular attribute requires a value.
- The values of the following attributes are checked:
  - type
  - language
  - pr_copies
  - pr_lines
  - pr_colums

Validation is performed before the job is created in the SAP system. If the validation fails, the HCL Workload Automation job goes into the ABEND state.

Specifying job parameters using variable substitution

Parameters can be provided at run time using the variable substitution feature. For example, the value appears as:

```
-s1 report=&VARNAME
```

The variable substitution process occurs while HCL Workload Automation is creating the symphony file.

Examples: Dynamically defining and updating SAP jobs

This section describes some usage examples of this feature:

Job definition and run scenario using the -flag type=exec parameter

The following example creates and runs a 3-step job. The first step runs the ABAP MYPROG1 using variant VAR01 and associated variant parameter. Step 2 has a step user defined. Step 3 uses the same ABAP as step 1 with no associated variant.

The only requirement is that the elements referred to are known in the SAP system (user, program). If the variant does not exist, there should be a set of values to define the content of the variant for its creation (pairs of `-vparN -vselN` parameters for the parameters and selections of the ABAP program).

```
-job TESTJOB01 -c A
-s1 type=A -s1 program=MYPROG1
-v1 VAR01 -vpar1 TESTNAME=TST
-s2 report=SPOOLX1 -s2 user=PRTUSER
-s3 type=A -s3 program=MYPROG1 -flag type=exec
```

The job returns job ID 12345678

Job copy and overwrite the job created in the previous step

The following job statement references the job created in the previous example. A new copy of the job is made and the parameters specified in the invocation are used to update the definition. In this case the variant for step 1 is modified and a new external program step (Step 4) is added.

```
-job TESTJOB01 -i 12345678
-s1 variant=VAR01A
-vpar1 TESTNAME=TST2
-s4 type=X -s4 report=niping -s4 parameter=-t
-flag type=exec
```

Copy and overwrite a job referencing an existing job template

The following example shows a job creation referencing a job template (previously created without using this feature). A template job called TEMPLAJOB already exists on the SAP system with an ID of 56780123. It is a single ABAP step job to which we now add some print parameters.

```
-job TEMPLAJOB
-I 56780123 -s1 pr_immed
-flag type=exec
```

A temporary variant is created using the information indicated in the expression

The following is the syntax to be used:

```
-vpar1 <parameter_name>=<parameter_value> ...
-vsel1 <selection_option_name>
... -vtemp1
```

The following example shows how you can submit a job that creates a temporary variant that is assigned to step number 1, and assigns a value to a variant parameter for step number 1:

```
-job TESTJOB01 -C A -flag type=exec -user R3USER
-s1 type=A -s1 program=MYPROG1
-vtemp1 -vpar1 TESTNAME=TST
```

The following example shows how you can submit a job that creates a temporary variant that is assigned to step number 1, assigns a value to a variant parameter for step number 1, and assigns a value to a variant selection option (date) for step number 1:

```
-job TESTJOB01 -C A -flag type=exec -user R3USER
-s1 type=A -s1 program=MYPROG1
-vtemp1 -vpar1 FILENAME=FLN
-vsel1 date=E#BT#20110101#20110412
```

Assign a temporary variant to the specified step number

The following is the syntax to be used:

```
-v1 <temporary_variant_name> -vtemp1
```

The following is an example of how you can submit a job to assign a temporary variant, which has already been created (as in the previous example), and assign a value to step number 1:

```
-job TESTJOB01 -C A -flag type=exec -user R3USER
-s1 type=A -s1 program=MYPROG1
-vtemp1 -v1 &000000000001
```

The value for a temporary variant that already exists is substituted with the value indicated in the expression

The following is the syntax to be used:

```
-v1 <temporary_variant_name> -vpar1 <parameter_name>=<parameter_value> ...
-vsel1 <selection_option_name> ... -vtemp1
```

The following is an example of how you can submit a job that substitutes the value of a temporary variant, which must already exist, with a new value. The temporary variant must exist, otherwise, the expression returns an error.

```
-job TESTJOB01 -C A -flag type=exec -user R3USER
-s1 type=A -s1 program=MYPROG1
-vtemp1 -v1 &000000000001 -vpar1 TESTNAME=TST2
```

Defining conditions and criteria

HCL Workload Automation accesses the Computer Center Management System (CCMS) Background Processing components of SAP systems through the BC-XBP interface to provide additional capabilities from the Dynamic Workload Console, one of those being the Criteria Manager.

HCL Workload Automation supports the BC-XBP 3.0 interface which provides functions to control R/3 batch jobs.

The Criteria Manager is a tool that enables you to define conditions and criteria that, when combined, form complex dependencies that you can use in the following contexts:

- Managing raised events in the SAP event history.
- Managing reorganization tasks against the SAP event history.
- Intercepting jobs.

If you have other types of criteria defined on your SAP system, then you can perform other actions in addition to those listed in this section.

The criteria profile

The Criteria Manager enables you to define a criteria profile which is a container for a combination of criteria. The criteria profile can be of various types and each criteria type has a standard set of selection criterion. For each criteria, you can specify a single value, a range of values by indicating a lower and upper limit, and multiple values. The following is the standard set of selection criterion for each criteria profile type. In addition to these, you can also see any other types of criteria profiles you have defined on your SAP system:

Event History

EVENTID

The identifier of the event defined in the SAP system.

EVENTPARM

The parameter of the event defined in the SAP system.

PARMID

The identifier of the parameter of the event defined in the SAP system.

Event History Reorg

Event State

The state of the event.

Event Timestamp

The timestamp for the event.

Interception

Job Name

A name identifying the job.

Job Class

The class assigned to the job that represents the priority with which the job runs in the SAP system.

The criteria hierarchy

You create and combine criteria in a criteria hierarchy. The criteria hierarchy is a set of all the criteria that must be fulfilled for a specific action to take place in the specific context. For example, you can define a criteria hierarchy to log all raised events in the SAP event history with an event name that begins with "CRITICAL_EVENT" and with an event argument equal to 150.

The criteria in the hierarchy is grouped in nodes and relationships between the nodes are determined by the logical operators AND or OR. You can nest nodes in other nodes.

To have the criteria profile begin processing, the criteria profile must be activated. Only one criteria profile of the same type can be active at one time.

Example

An example

See Example: Defining which raised events to log on page 1220 for an example that demonstrates how to build a criteria hierarchy to manage the logging of raised events in the SAP event history.

Example: Defining which raised events to log

The event history stores all events that are raised by the system. You can define specific criteria so that only raised events that match certain criteria are logged.

The event history enables HCL Workload Automation to consume events that are raised by the SAP system.

Checking the log of raised events gives you access to the following information:

- Verify that an event was raised in the system.
- Verify if the event was processed.

In the example that follows, an event history criteria profile is created that contains the definition of the criteria, the criteria hierarchy, that events must fulfill to be logged in the event history. The criteria profile must then be activated so that it can begin processing events according to the criteria.

The criteria profile, `Event profile 1`, contains a criteria hierarchy that logs only those events in the event history with event name that begins with `CRITICAL_EVENT` and event argument equal to "789".

Create the criteria profile

A criteria profile contains the definition of the criteria you want to set for logging raised events.

About this task

Create a criteria profile, `Event profile 1`, of type, Event History, to contain the criteria hierarchy.

1. In the Navigation bar at the top, click **Design > SAP > Manage SAP Criteria Profiles**.
2. In **Engine name**, select the name of the HCL Workload Automation engine connection from which you want to work with SAP jobs.
3. In **Workstation name**, type the name of the workstation where the SAP job runs. This is the workstation with the r3batch access method that communicates with the remote SAP system. If you do not know the name of the workstation, click the **Lookup Workstations** icon to enter your filter criteria and click **Search**. If you enter a string representing part of the workstation name, it must be followed by the asterisk (*) wildcard character. Both the question mark (?) and asterisk (*) are supported as wildcards. You can also simply use the asterisk wildcard character (*) to display all workstations. Optionally, specify any of the other search criteria available and click **Search**.
   Result
   From the results displayed, select the workstation and click **OK**.
4. In **Options file**, specify an options file that resides on the specified workstation. Each workstation can have one or more options files that can be used to customize the behavior of the r3batch access method, except for extended agent workstations, where only one options file can exist and therefore does not need to be specified. For the workstation specified, enter the file name of the options file or click pick icon to search for options files that reside on the specified workstation and select one.
5. Click **Go**.
6. From the Criteria Manager main view, click **New** to create a criteria profile.
7. Select **Event History** as the type of criteria profile you want to create.

8. Enter descriptive text that enables you to easily identify the criteria profile in the table of criteria profiles. Type `Event profile 1`. Avoid using special characters such as, < (less than), > (greater than), or the ' (apostrophe) in this field.
9. Click **Save**.

Results

The criteria profile is displayed in the list of criteria profiles and it is not yet active.

What to do next

Next, begin building the criteria hierarchy. The criteria profile is the container for the criteria hierarchy.

Build the criteria hierarchy

The criteria hierarchy is stored in the criteria profile and is made up of criteria. A group of criteria is contained in a node.

Before you begin

The criteria hierarchy is made up of a combination of nodes and criteria. A node contains a group of criteria where the relationship between the criteria is determined by an AND or an OR relation. You can nest nodes in other nodes. By default, a top level AND node is created in the criteria hierarchy. You can create other AND or OR nodes nested in this parent node. You can also add one or more criterion to the nodes. Add an AND node when all of the criteria defined in the node must be fulfilled. Add an OR node when at least one of the criteria defined in the node must be fulfilled.

About this task

In this example, define a criterion that logs all events whose name begins with `CRITICAL_EVENT` and with event argument equal to `789`.

1. Click  to create a new criterion in the default AND node.
2. In **Description**, type `Criterion 1`.
3. In **EVENTID**, click  to specify the value for the EVENTID field.
4. Leave the default value **Select** to indicate to use the selection criterion specified when processing events.
5. In **Options**, select **Pattern** and in **Single Value or Lower Limit**, type `CRITICAL_EVENT*`.
   Result
   This sets the condition for the event name.
6. In **EVENTPARM**, click  to specify the value for the EVENTPARM field.
7. Leave the default value **Select** to indicate to use the selection criterion specified when processing events.
8. In **Options**, select **Equal to** and in **Single Value or Lower Limit**, type `789`.
   Result
   This sets the condition for the event argument.
9. Click **Save** to save the criterion definition.

Results

The criteria profile now contains a criterion that specifies which raised events must be logged. You can continue to create another criteria in the same parent node or you can nest either an AND or an OR node in the parent node to determine the

logical relation between the criteria that the nested node will contain. Add an AND node within which you can create one or more criteria where all the criteria specified in the node must be fulfilled, or add an OR node within which you can create one or more criteria where at least one of the criteria specified must be fulfilled.

What to do next

To apply this criteria profile so that it begins processing events according to the criteria defined, you must activate the criteria profile.

Activate the criteria profile

To apply the `Event profile 1` criteria profile so that it begins processing raised events according to the criteria specified in the criteria hierarchy, you must activate the criteria profile.

About this task

A criteria profile can either be active or not active. For a criteria profile to take effect, the profile must be activated. Only one criteria profile of the same type can be active at one time. Criteria profiles cannot be edited if they are in the active state. Follow the procedure to activate the `Event profile 1` criteria profile.

1. Select the `Event profile 1` criteria profile from the table of criteria profiles.
2. Select **Activate** from the toolbar. 

Results

The status of the criteria profile is updated to show that it is now active. The criteria profile can now begin to process raised events according to the specifications of the criteria hierarchy and log them to the event history. If another criteria profile of the same criteria type was active, its status changes to inactive.

Using the BDC Wait option

By using the Batch Data Collector (BDC) Wait option, you can specify that a SAP job launched by HCL Workload Automation is not to be considered complete until all of its BDC sessions have completed.

About this task

The Batch Data Collector (BDC) Wait option prevents other HCL Workload Automation jobs that are dependent on the SAP job from being launched until all of the related BDC sessions for the SAP job have ended.

To use the option, a SAP job must write informational messages in its job log. This can be done by modifying the SAP function module BDC_OPEN_GROUP as follows:

```
FUNCTION BDC_OPEN_GROUP.
...
CALL 'BDC_OPEN_GROUP' ID 'CLIENT'      FIELD CLIENT
                      ID 'GROUP'       FIELD GROUP
                      ID 'USER'        FIELD USER
                      ID 'KEEP'        FIELD KEEP
                      ID 'HOLDDATE'    FIELD HOLDDATE
                 ID 'DESTINATION' FIELD DEST
                 ID 'QID'         FIELD QID
                 ID 'RECORD'      FIELD RECORD
```

```
                ID 'PROG'        FIELD PROG.
*
IF SY-SUBRC EQ 0.
  BQID   = QID.
  BUSER  = SY-MSGV1.
  BGROUP = GROUP.
* CALL FUNCTION 'DB_COMMIT'.
  CALL FUNCTION 'ENQUEUE_BDC_QID'
  EXPORTING   DATATYP   = 'BDC '
  GROUPID    = BGROUP
  QID              = BQID
  EXCEPTIONS  FOREIGN_LOCK  = 98
  SYSTEM_FAILURE   = 99.

  IF SY-SUBRC EQ 0.
     message i368(00) with 'BDCWAIT: ' qid.
  ENDIF.

ENDIF.
*
PERFORM FEHLER_BEHANDLUNG USING SY-SUBRC.
*
*
ENDFUNCTION.
```

Note: The actual parameters of the call of the C function `(CALL 'BDC_OPEN_GROUP' ID ...)` might vary depending on the SAP release. With this approach, you obtain a global change in your SAP system.

The completion status of a SAP job launched by HCL Workload Automation is based on the value you set for the **bdc_job_status_failed** option. By default, this option is set to `ignore`, meaning that the job is considered successfully completed when the BDC sessions are finished, regardless of their success or failure. For details about the **bdc_job_status_failed** option, refer to Table 149: Task string parameters for SAP jobs on page 1181.

Job interception and parent-child features

This section describes how the job interception and parent-child features of BC-XBP 2.0 and 3.0 are supported by HCL Workload Automation.

Note: The process of defining relaunch criteria and collecting and relaunching intercepted jobs is supported only in distributed environments and not in z/OS environments.

Implementing job interception

The high-level steps required to implement job interception.

About this task

Job interception is a feature of both the BC-XBP 2.0 and BC-XBP 3.0 interfaces. It enables HCL Workload Automation to have a very sophisticated control over the jobs launched by SAP users from the SAP graphical interface.

The job interception mechanism becomes active when the SAP job scheduler is about to start an SAP job (that is, when the `start` conditions of an SAP job are fulfilled). It checks the job parameters (`job name`, `creator`, `client`) against the entries in the SAP table `TBCICPT1`, and when the job parameters match the criteria, the SAP job is set back to the `scheduled` status and is marked with a special flag, denoting that the job has been intercepted. The criteria defined in the criteria table establishes which job are intercepted.

If HCL Workload Automation has been set up to handle job interception, it periodically runs its own job to retrieve a list of intercepted jobs and reschedules them to be relaunched. This job can be referred to as the *interception collector* job.

Job interception with the BC-XBP 2.0 interface is based on the single extended agent workstation, whereas with the BC-XBP 3.0 interface, job interception is based on the currently active job interception criteria profile.

Note:

- Jobs launched by HCL Workload Automation, or by any other external scheduler using the BC-XBP interface, can be intercepted provided the **job_interceptable** option in the common options file is set to `ON`, and the **-flag enable_job_interceptable** keyword is included in the job definition.
- Ensure that the job interception and job throttling features are not running at the same time. The interception collector jobs fail if a job throttler instance is running. To stop the job throttler, refer to Step 5. Starting and stopping the job throttling feature on page 1254.

The following are the high-level steps required to implement job interception for both the BC-XBP 2.0 and 3.0 interfaces.

Job interception and the BC-XBP 2.0 interface

About this task

To set up HCL Workload Automation to handle job interception in an SAP environment with the BC-XBP 2.0 interface, implement the following steps:

1. Install the BC-XBP 2.0 interface. Refer to SAP Note 604496 to know if your SAP system already has the BC-XBP 2.0 interface, or which SAP support package you need to install to enable it.
2. Define an HCL Workload Automation job to periodically collect the intercepted SAP jobs.
3. Specify interception criteria in the SAP system.
4. Specify interception criteria in HCL Workload Automation from the Monitor Workstations portlet on the Dynamic Workload Console. The criteria is set at workstation level.
5. Activate the job interception feature of the BC-XBP 2.0 interface.

Job interception and the BC-XBP 3.0 interface

About this task

To set up HCL Workload Automation to handle job interception in an SAP environment with the BC-XBP 3.0 interface, implement the following steps:

1. Verify if the BC-XBP 3.0 interface is already installed on the SAP system.
2. Define an HCL Workload Automation job to periodically collect the intercepted SAP jobs.
3. Specify interception criteria in the SAP system.
4. Specify interception criteria in HCL Workload Automation from the Manage SAP Criteria Profiles portlet on the Dynamic Workload Console.
5. Activate the job interception feature of the BC-XBP 3.0 interface.

Collecting intercepted jobs periodically for BC-XBP 2.0

With the BC-XBP 2.0 interface, you can configure the job interception collector using an HCL Workload Automation job that periodically retrieves intercepted jobs and relaunches them.

About this task

Define an HCL Workload Automation job that uses the SAP interception collector task to collect intercepted jobs and restart them.

To define an HCL Workload Automation job that collects intercepted job and relaunches them, use the following syntax:

```
ENGINE_NAME_HOSTING_XA#[folder/]JOBNAME
  SCRIPTNAME "TWA_home/methods/r3batch -t HIJ -c UNIQUE_ID"
  DESCRIPTION "Collects intercepted jobs on SAP XA XA_Unique_ID"
  STREAMLOGON TWSuser
  RECOVERY STOP
```

Where:

ENGINE_NAME_HOSTING_XA

The name of the engine workstation hosting the XA workstation with the r3batch access method that communicates with the SAP system.

JOBNAME

Name of the HCL Workload Automation job and the folder in which it is defined, if any.

TWA_home

Fully qualified path to your HCL Workload Automation installation.

XA_Unique_ID

The unique identifier for the extended agent workstation. See UNIQUE_ID on page 1085 for more details about retrieving the unique identifier.

–t HIJ

This is the SAP task type to run the job interception collector. HIJ stands for `Handle Intercepted Jobs`.

TWSuser

Name of the HCL Workload Automation user that launches the access method.

The interception collector job runs at periodical intervals; for example, every 10 minutes. It retrieves all the jobs that have been intercepted since the last run of the interception collector, and launches them again according to a template.

Collecting intercepted jobs periodically for BC-XBP 3.0

With the BC-XBP 3.0 interface, you can configure the job interception collector using an HCL Workload Automation job that periodically retrieves intercepted jobs and relaunches them.

About this task

Because intercepted jobs remain in the `Released` and then `Intercepted` status until they are relaunched, you need to use the SAP interception collector task to collect and relaunch them.

To use the job interception collector BC-XBP 3.0 to collect and restart jobs, create the **r3batch_icp** folder in the *TWS_home*`/methods/` path with the correct access rights.

To define an HCL Workload Automation job that collects and relaunches jobs use the following syntax:

```
ENGINE_NAME_HOSTING_XA#[folder/]JOBNAME
DOCOMMAND "TWA_home/methods/r3batch -t HIJ -c UNIQUE_ID -- \
   "-profile_id profile_ID_number\""
STREAMLOGON TWSuser
DESCRIPTION "Collects intercepted jobs on SAP XA XA_Unique_ID"
TASKTYPE UNIX
RECOVERY STOP
```

where,

ENGINE_NAME_HOSTING_XA

The name of the engine workstation hosting the XA workstation with the r3batch access method that communicates with the SAP system.

JOBNAME

Name of the HCL Workload Automation job and the folder within which it is defined, if any.

TWA_home

Fully qualified path to your HCL Workload Automation installation.

XA_Unique_ID

The unique identifier for the extended agent workstation. See UNIQUE_ID on page 1085 for more details about retrieving the unique identifier.

–t HIJ

This is the SAP task type to run the job interception collector. HIJ stands for `Handle Intercepted Jobs`.

- profile_id *profile_ID_number*

Specifies the identification number of the interception criteria profile on the SAP system for XBP 3.0.

TWSuser

Name of the HCL Workload Automation user that launches the access method.

The interception collector job runs at periodical intervals; for example, every 10 minutes. It retrieves all the jobs that have been intercepted since the last run of the interception collector, and launches them again according to a template.

Note: If the interception collector is configured for XBP 3.0 job interception, but the XBP 2.0 interface is configured on the SAP system, the collector fails. Ensure the XBP interface versions are synchronized.

Setting interception criteria on the SAP system

About this task

In SAP, the interception criteria are held in table `TBCICPT1`. Only jobs that match the criteria of this table are intercepted, when their `start` conditions are fulfilled. All the other jobs are run normally.

You can maintain the entries in this table by using transaction `se16` and setting the following:

- Client number
- Job mask
- User mask

Setting interception criteria on HCL Workload Automation

About this task

In HCL Workload Automation, interception criteria are defined and used by setting:

Table criteria

For BC-XBP 2.0

You use the **Monitor Workload** of Dynamic Workload Console to set table criteria.

For details about how you set table criteria, see Setting SAP table criteria on the extended agent workstation on page 1227.

For BC-XBP 3.0

You set table criteria from the **Administration > Workload Design > Manage SAP Criteria Profiles** panel from the Dynamic Workload Console.

For details about how you set table criteria, see Setting SAP criteria in the job interception criteria profile on page 1229.

Template files (optional)

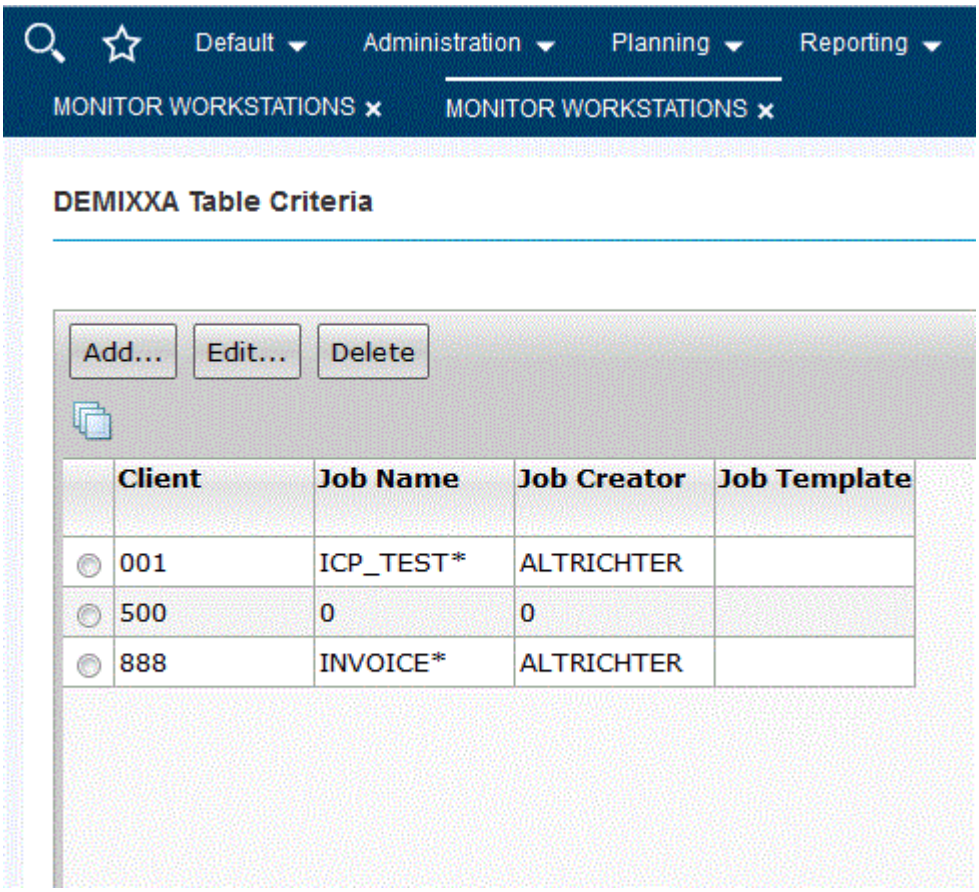
For details about how you create template files, see Using template files on page 1230.

Setting SAP table criteria on the extended agent workstation

About this task

To set table criteria with the BC-XBP 2.0 interface on an SAP job using the **Monitor Workload** of the Dynamic Workload Console, follow these steps:

1. Log in to the Dynamic Workload Console.
2. From the **Monitoring and Reporting** menu, click **Orchestration Monitor**.
3. In the **Monitor Workload** window select the engine, enter **Workstation** in the **Object Type** field, and select the plan to display the list of workstations you want to monitor. Click **Run**.
4. Select an extended agent workstation in the table of displayed workstations, and click **More Actions > Table Criteria...** from the toolbar.
5. The **Table Criteria** panel displays. From this panel you can add, delete, edit, or refresh criteria.
   Figure 35. The Table Criteria panel



6. Specify the criteria:
   a. In **Client**, specify the client workstation of the SAP job.
   b. In **Job Name**, specify a filter to match a set of SAP jobs. Use the asterisk (*) wildcard character to match a set of jobs.
   c. In **Job Creator**, specify a filter to match a set of SAP job creator. Use the asterisk (*) wildcard character to match a set of jobs.

d. Optionally, in **Job Template**, specify the template file that contains instructions for the interception collector about how to run the intercepted SAP job under control of HCL Workload Automation. For more information about template files, see Using template files on page 1230.
e. In **Job Class**, specify the class assigned to the job that represents the priority with which the job runs on the SAP system.

7. Click **OK**.

Setting SAP criteria in the job interception criteria profile

Setting criteria to intercept jobs and relaunch them.

About this task

To set the criteria that defines which SAP jobs to intercept and relaunch with the BC-XBP 3.0 interface using the Dynamic Workload Console, perform the following steps:

1. In the Navigation bar at the top, click **Design > SAP > Manage SAP Criteria Profiles**.
2. In **Workstation name**, type the name of the workstation where the SAP job runs. This is the workstation with the r3batch access method that communicates with the remote SAP system. If you do not know the name of the workstation, click the **Lookup Workstations** icon to enter your filter criteria and click **Search**. If you enter a string representing part of the workstation name, it must be followed by the asterisk (*) wildcard character. Both the question mark (?) and asterisk (*) are supported as wildcards. You can also simply use the asterisk wildcard character (*) to display all workstations. Optionally, specify any of the other search criteria available and click **Search**.
   Result
   From the results displayed, select the workstation and click **OK**.
3. From the Criteria Manager main view, click **New** to create a criteria profile.
4. Select **Interception** as the type of criteria profile you want to create.
5. Enter descriptive text that enables you to easily identify the criteria profile in the table of criteria profiles. Avoid using special characters such as, <(less than), > (greater than), or the ' (apostrophe) in this field.
6. Click **Save**.
   Result
   The criteria profile is displayed in the list of criteria profiles and it is not yet active.
7. On the **Details** tab in the upper-right pane, define the criteria that intercepted jobs must match. For example, to intercept jobs with a job name beginning with "ICP", specify the following criteria:

   a. Click  to define a new criterion.

   b. In **Description**, type `Criterion 1`.

   c. In **JOB NAME**, click  to specify the value for the JOB NAME field.

   d. Leave the default value **Select** to indicate to use the selection criterion specified when intercepting jobs.

   e. In **Options**, select **Pattern** and in **Single Value or Lower Limit**, type `ICP*`.

Result
This sets the condition for the job name.

f. Click **Save** to save the criterion definition.

8. Define the criteria that must be matched to relaunch intercepted jobs. Click the **Job Relaunch Criteria** tab.
   a. Click  to define a new criteria that determines which jobs are relaunched.
   b. In **Client**, specify the client workstation of the SAP job.
   c. In **Job Name**, specify a filter to match a set of SAP jobs. Use the asterisk (*) wildcard character to match a set of jobs.
   d. In **Job Creator**, specify a filter to match a set of SAP job creator. Use the asterisk (*) wildcard character to match a set of jobs.
   e. Optionally, in **Job Template**, specify the template file that contains instructions for the interception collector about how to run the intercepted SAP job under control of HCL Workload Automation. For more information about template files, see Using template files on page 1230.
   f. In **Job Class**, specify the class assigned to the job that represents the priority with which the job runs on the SAP system.
9. Click **OK**.
10. You can continue to define more criteria and then save the criteria profile.
11. When you are done defining the criteria, save the criteria profile.
12. Select the criteria profile and then click **Activate** from the toolbar. 

Results
The status of the criteria profile is updated to show that it is now active. The criteria profile can now begin to intercept jobs according to the specifications of the criteria hierarchy and relaunch them as defined in the HCL Workload Automation job. If another criteria profile of the same criteria type was active, its status changes to inactive.

Using template files

About this task

A template is a file with extension `.jdf` located in the same directory as the interception criteria file (*`TWA_DATA_DIR`*`/methods/r3batch_icp`). The template file contains instructions for the interception collector about how to run the intercepted SAP job under control of HCL Workload Automation. Its syntax corresponds to the syntax of `docommand` in conman. You can use any text editor to maintain this file. Ensure that the user, LJUser, is able to read and write to this file.

If the user template file is empty, a template file named `default.jdf` is used. If `default.jdf` does not exist, the following instructions are used:

```
alias=SAP_$RUN_$JOBNAME_$JOBCOUNT
```

This means that the intercepted SAP jobs are to be restarted immediately, because of the absence of the `at=` job option. Their HCL Workload Automation names are composed of the string `SAP_`, the current run number of the interception collector, and the name and ID of the SAP job.

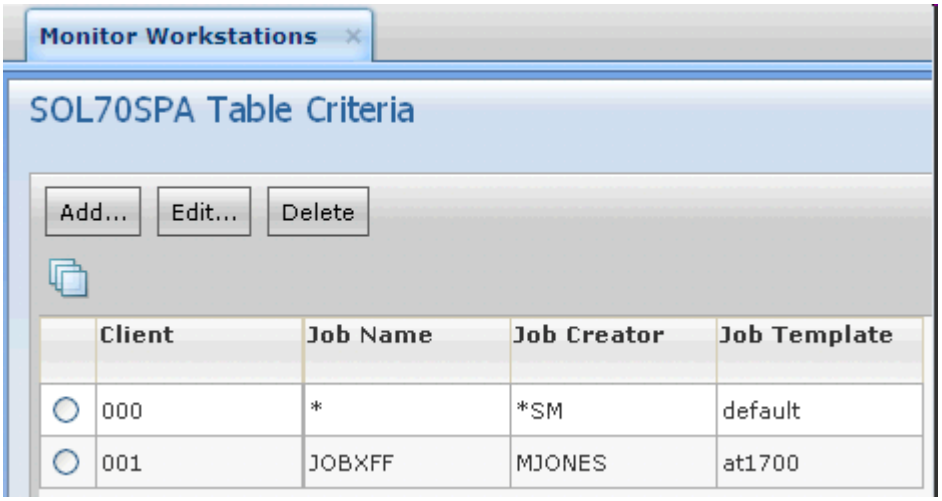
The instruction set for restarting an intercepted SAP job is retrieved in the following order:

1. From the template file, if an existing template is specified in the interception criteria file.
2. From the default template file, if the template is specified in the interception criteria file but does not exist, or if the template is not specified in the interception criteria file.
3. From the default instruction set, if the default template file does not exist.

Job interception example

The following example demonstrates how different template files can be used to determine when an intercepted SAP job is restarted. The interception criteria table contains the following entries:

Figure 36. The Table Criteria panel



The table criteria specified, implies the following:

Client 000

All jobs started in client `000` by SAP users whose user name begins with `sm`, will be intercepted. The interception collector restarts the jobs using the instructions from the default template file `default.jdf`. If the default template file does not exist, then the SAP jobs are restarted immediately as specified in the default instruction set:

```
alias=SAP_$RUN_$JOBNAME_$JOBCOUNT
```

Client 001

The job named, JOBXFF, started in client `001` by SAP user named, MJONES, will be intercepted. The interception collector restarts the jobs using the instructions from the template file `at1700.jdf`. The SAP jobs are restarted at 17:00 with a random name, because of the `alias` command. The template file `at1700.jdf` contains the following entry:

```
alias;at=1700
```

Using placeholders

In the template files you can use a number of placeholders that are replaced by the interception collector at run time. They are listed in Table 154: Placeholders for job interception template files on page 1232.

Table 154. Placeholders for job interception template files

Placeholder	Description
$CPU	Name of the extended agent workstation where the interception collector runs.
$CLIENT	Client number of the intercepted SAP job.
$JOBNAME	Name of the intercepted SAP job.
$JOBCOUNT	Job ID of the intercepted SAP job.
$USER	Name of the user who launched the SAP job.
$JOBNUM	Job number of the interception collector.
$RUN	Current run number of the interception collector.
$SCHED	Schedule name of the interception collector.
$RAND	Random number.

The template:

```
alias=ICP_$RAND_$JOBNAME_$JOBCOUNT_$CLIENT;at=1000
```

instructs the interception collector to restart the SAP job named `DEMO_JOB` with job ID `12345678` on client `100` at `10:00` as HCL Workload Automation job `ICP_1432_DEMO_JOB_12345678_100.`

Activating the job interception feature

Activate the job interception feature for the appropriate BC-XBP interface.

About this task

To enable the job interception feature: .

1. Run ABAP report `INITXBP2`.
   Result
   This report shows you the current status of the job interception and parent-child features, and allows you to toggle the status of both features.
2. Select the BC-XBP interface version as appropriate:
   - **Activate 3.0**
   - **Activate 2.0**
3. Save the changes.

The parent-child feature

In some situations, an SAP R/3 job dynamically spawns a number of other jobs; for example, to distribute the workload to the free application servers. Prominent examples are the mass activity jobs of the SAP R/3 `FI-CA` component. Before BC-XBP 2.0, it was difficult for external schedulers to handle this situation, because the business process does not usually end with the end of the initial job (parent job), but with the end of all subjobs (child jobs).

The BC-XBP 2.0 interface allows you to determine if a job has launched subjobs, together with their names and IDs, and so it is now possible to track them.

To activate this feature, use the `INITXBP2` ABAP report, which you can also use to toggle the status of job interception.

When the parent-child feature is active, HCL Workload Automation considers an SAP R/3 job as finished only after all its child jobs have ended. The status of the HCL Workload Automation job remains as `EXEC` while the parent job or any of its child jobs are running.

The status of the HCL Workload Automation job becomes `SUCC` if the parent job and all child jobs end successfully. If any of the jobs ended with an error, the status of the HCL Workload Automation job becomes `ABEND`.

Note: The parent-child feature can interfere with job interception because, although the parent job cannot be intercepted, any of its child jobs can be intercepted if they match the interception criteria. In this case, the HCL Workload Automation job remains in the `EXEC` status until the intercepted child job has been relaunched and has ended.

The joblogs of the child jobs are appended in the HCL Workload Automation `stdlist` after the joblog of the parent job.

Using Business Information Warehouse

Business Information Warehouse (BIW) is a data warehouse solution tailored to SAP.

Business Information Warehouse (BIW) allows business reporting and decision support.

To use the InfoPackages component, you must have the SAP Business Warehouse Systems, version 2.0B or later installed.

To use the Process Chains component, you must have the SAP Business Warehouse Systems, version 3.0B or later installed.

The Support Package 9 (SAPKW31009) for SAP Business Warehouse version 3.1 is required so that `SAP` can launch process chains.

Business Warehouse components

SAP supports two main Business Warehouse components, InfoPackages and Process Chains.

An InfoPackage is the entry point for the loading process from a specific **InfoSource** (a logical container of data source, generically named **InfoObject**). Technically, an InfoPackage is an SAP job whose aim is to load data. Like any other SAP job, it contains job-pecific parameters such as start time, and dependencies.

A Process Chain is a complex chain of different processes and their relationships. The processes within a process chain are not limited to data load processes, or InfoPackages, but also include:

- Attribute/Hierarchy Change run
- Aggregate rollup
- ABAP program
- Another process chain
- Customer build process

Defining user authorizations to manage SAP Business Warehouse InfoPackages and process chains

What you need to use SAP R/3 Business Warehouse InfoPackages and process chains.

Access method for SAP can manage SAP Business Warehouse InfoPackages and process chains. To use the SAP Business Warehouse functions, you must define an HCL Workload Automation user within SAP with full authorization for the ABAP Workbench object S_DEVELOP.

The user must also belong to the following profiles:

- S_BI-WHM_RFC (for Business Information Warehouse version 7.0, or later)
- S_RS_ALL
- Z_MAESTRO

Managing SAP R/3 Business Warehouse InfoPackages and process chains

You can manage existing InfoPackages and process chains on SAP systems from SAP.

Business Warehouse InfoPackages and process chains can only be created from the SAP R/3 environment. However, the Dynamic Workload Console supports pick lists of InfoPackages and process chains, so that you can also define HCL Workload Automation jobs for these existing objects.

You can create HCL Workload Automation job definitions that map to SAP jobs that already exist on SAP systems in the following environments:

- Distributed
- z/OS

The SAP jobs can run on extended agent workstations, dynamic agent workstations, dynamic pools, and z-centric workstations depending on the type of job definition you choose to create.

This section describes how to perform tasks such as creating the HCL Workload Automation job definitions that map to SAP jobs, how to display the details of these jobs, and how to rerun a process chain job.

Creating an HCL Workload Automation job that contains InfoPackages or process chains

Creating a job with InfoPackages or process chains.

About this task

This section describes how to create an HCL Workload Automation SAP job definition that references a Business Warehouse InfoPackage or Process Chain SAP job.

SAP job definitions can be created using both a distributed or z/OS engine and they can be scheduled to run on the following workstations with the r3batch access method:

- An HCL Workload Automation extended agent workstation. A workstation that is hosted by a fault-tolerant agent or master workstation.
- A dynamic agent workstation.
- A dynamic pool.
- A z-centric workstation.

Refer to the Dynamic Workload Console online help for a complete description of all UI elements for both engine types and all supported workstation types.

Take into consideration that:

- To be able to schedule InfoPackages using HCL Workload Automation, the scheduling options of the InfoPackage must have:
  - Start type set to **Start later in background process**.
  - Start time set to **Immediate**.
- To be able to control process chains using HCL Workload Automation, the scheduling options of the process chain must be **Start Using Meta Chain or API**. If the process chain is set to **Direct Scheduling**, it starts immediately when activated in the SAP system or transported to another SAP system.
-

  If you are using an operating system that does **not** support Unicode, set the **TWSXA_LANG** option. For details about the operating systems that support Unicode, see Unicode support on SAP on page 1144. For details about the **TWSXA_LANG** option, see Setting National Language support options on page 1295.

You can create a SAP job definition to reference an InfoPackage or process chain using the Dynamic Workload Console.

The following procedure creates an HCL Workload Automation SAP job definition and references an InfoPackage or process chain in the HCL Workload Automation database:

1. Click **IBM Workload Scheduler**>**Design**>**Workload Designer**.
2. Select a an engine. The **Workload Designer** is displayed.
3. From the Working List pane, click:
   - z/OS engine: **New > ERP**
   - Distributed engine: **New > Job Definition > ERP**
4. Select the SAP job definition in accordance with the engine and type of agent on which the job runs.

   z/OS engine

   SAP

   This job definition references an existing job on the SAP system and can run on dynamic agent workstations, dynamic pools, and z-centric workstations.

   Distributed engine

   SAP Job on Dynamic Workstations

   This job definition can run on dynamic agent workstations, dynamic pools, and z-centric workstations.

   SAP Job on XA Workstations

   This job definition can run on extended agent workstations. A workstation that is hosted by a fault-tolerant agent or master workstation.

5. In the Workspace pane, specify the properties for the job definition you are creating using the tabs available. The tabs for each type of SAP job definition are similar, but there are some differences depending on the type of engine you selected and the type of workstation on which the job runs. For more detailed information about the UI elements on each tab, see the Dynamic Workload Console online help.

   The **General** page requires information regarding the workstation that connects to the remote SAP system. If a default SAP connection is already configured, then these fields are alreaday prefilled, otherwise, you can specify the required information on the General page or you can configure a default connection to be used each time it is required in a definition, see Setting the SAP data connection on page 1174 for more information.

   On the **Task** page, in **Subtype**, specify either **BW Process Chain** or **BW InfoPackage**.

6. Click **Save** to add the SAP job definition to the HCL Workload Automation database.

Task string to define Business Warehouse InfoPackages and process chain jobs

This section describes the task string parameters that control the running of the Business Warehouse InfoPackages and process chain jobs. You must specify them in the following places when you define their associated HCL Workload Automation jobs:

- If you use the Dynamic Workload Console, in the **SAP command line** field of the Task page of the SAP job definition panel.
- As arguments of the `scriptname` keyword in the job definition statement, if you use the HCL Workload Automation command line.
- As arguments of the `JOBCMD` keyword in the `JOBREC` statement in the `SCRIPTLIB` of HCL Workload Automation for Z, if you are scheduling in an end-to-end environment.

The string syntax is the following:

Job definition syntax

```
-jobjob_name -i{ipak_ | pchain_}[ -debug][ -trace][ -flag{imm | immed}][ -flag{enable_pchainlog | disable_pchainlog}][ -flag{enable_ipaklog | disable_ipaklog}][ -flag{level_all_pchainlog | level_n_pchainlog}][ -flag{pchainlog_chains_only | pchainlog_chains_and_failed_proc | pchainlog_complete}][ -flag{enable_pchainlog_bapi_msg | disable_pchainlog_bapi_msg}][ -flag{enable_pchain_details | disable_pchain_details}][ -flag{pchain_rerun | pchain_restart | pchain_refresh}]
```

The parameters are described in Table 149: Task string parameters for SAP jobs on page 1181.

Table 155. Task string parameters for SAP jobs

Parameter	Description	GUI Support
-job *job_name*	The name of the task to be run. It is either an InfoPackage technical field name, or a process chain name. This parameter is mandatory.	✓
-i {ipak_ | pchain_}	One of the following: **ipak_** Target job is an InfoPackage **pchain_** Target job is a process chain	✓
-debug	Turns on the most verbose `r3batch` trace. This option is for debugging the extended agent and should not be used in standard production.	✓
-trace	Turns on the `SAP RFC` trace. When you use this option, a trace file is created in the HCL Workload Automation `methods` directory. In UNIX, this trace file is called `dev_rfc`. In Windows, the file is called `rfc`*`xxxxx_xxxxx`*`.trc`. The `methods` directory is located in: **On UNIX operating systems** *`TWA_DATA_DIR`*`/methods` **On Windows operating systems** *`TWA_home`*`\methods` This option is for debugging the extended agent and should not be used in standard production. Ensure that you delete the trace option from the job after you have performed debug procedures. The trace file can become very large and unmanageable.	✓
-flag {imm | immed}	Specifies to launch the job immediately, meaning that if there are no spare work processes, the job fails.	✓
-flag {enable_pchainlog | disable_pchainlog}	Enables or disables retrieval and appending of the process chain job log in the HCL Workload Automation `stdlist`. Disable if the size of the log affects performance. A related configuration option can be set for this purpose	✓

Table 155. Task string parameters for SAP jobs (continued)

Parameter	Description	GUI Support
	at a more general level. See Table 147: r3batch common configuration options on page 1154.	
-flag {enable_ipaklog | disable_ipaklog}	Enables or disables retrieval and appending of the InfoPackage job log in the HCL Workload Automation `stdlist`. Disable if the size of the log affects performance. A related configuration option can be set for this purpose at a more general level. See Table 147: r3batch common configuration options on page 1154.	
-flag {level_*n*_pchainlog | level_all_pchainlog}	Allows for retrieval of process chain logs down to the process chain level you specify. **level_*n*_pchainlog** Specifies that the process chains are logged down to, and including, the level represented by number *n*. **level_all_pchainlog** Specifies that all the process chains are logged. The default is `level_1_pchainlog`. A related configuration option can be set for this purpose at a more general level. See Table 147: r3batch common configuration options on page 1154.	
-flag {pchainlog_chains_only | pchainlog_chains_and_failed_proc | pchainlog_complete}	Specifies what type of process chain-related logs will be retrieved. **pchainlog_chains_only** Only the process chains are logged. **pchainlog_chains_and_failed_proc** In addition to the process chains, all the processes that failed are also logged. **pchainlog_complete** The process chains and all processes are logged.	

Table 155. Task string parameters for SAP jobs (continued)

Parameter	Description	GUI Support
	The default is **pchainlog_complete**. A related configuration option can be set for this purpose at a more general level. See Table 147: r3batch common configuration options on page 1154.	
-flag {enable_pchainlog_bapi_msg | disable_pchainlog_bapi_msg}	Enables or disables retrieval of additional messages from the BAPI calls from the SAP Business Warehouse process chains and appends them to the HCL Workload Automation `stdlist`.	
-flag {enable_pchain_details | disable_pchain_details}	Enables or disables the display of details about the process chain job. A related configuration option can be set for this purpose at a more general level. See Table 147: r3batch common configuration options on page 1154.	✓
-flag {pchain_rerun | pchain_restart | pchain_refresh}	Determines the action that HCL Workload Automation performs when you rerun a job that submits a process chain. **pchain_rerun** HCL Workload Automation creates another process chain instance and submits it to be run again. **pchain_restart** HCL Workload Automation restarts the original process chain from the failing processes to the end. **pchain_refresh** HCL Workload Automation updates the status and details of the original process chain. For more details about rerunning a process chain, refer to Rerunning a process chain job on page 1243.	✓

Note: Typically, the `-debug` and `-trace` options are for debugging the extended agent and should not be used in standard production.

The following is an example for an InfoPackage job whose technical field name is `ZPAK_3LZ3JRF29AJDQM65ZJBJF5OMY`:

```
-job ZPAK_3LZ3JRF29AJDQM65ZJBJF5OMY -i ipak_
```

Displaying details about Business Warehouse InfoPackages

About this task

To display details about a Business Warehouse InfoPackage, perform the following steps:

1. From the **Design** menu, click **Manage Jobs on SAP**.
2. In **Engine name**, select the name of the HCL Workload Automation engine connection from which you want to view SAP job details.
3. In **SAP Job Type**, select **Business Warehouse InfoPackage**.
4. In **Workstation name**, specify the workstation where the SAP job runs. If you do not know the object name, click the **...** (Browse) button. In the **Name** and **Location** panel, enter some characters of the object name (asterisk is supported as a wildcard) and click **Start**. From the displayed list, select the workstation you want to use, and click **OK**.
5. Click **Display**. The list of available jobs of type Business Warehouse InfoPackage for the specified engine is displayed.
6. Select the job for which you want to display the details and click **Details**.
7. When you have finished viewing the details for the job, click **OK** to return to the list of SAP jobs on the workstation specified.

Displaying details about a process chain job

You can view the details for a process chain job including any local subchains contained in the process chain.

Before you begin

Ensure you have performed the following steps before running this procedure:

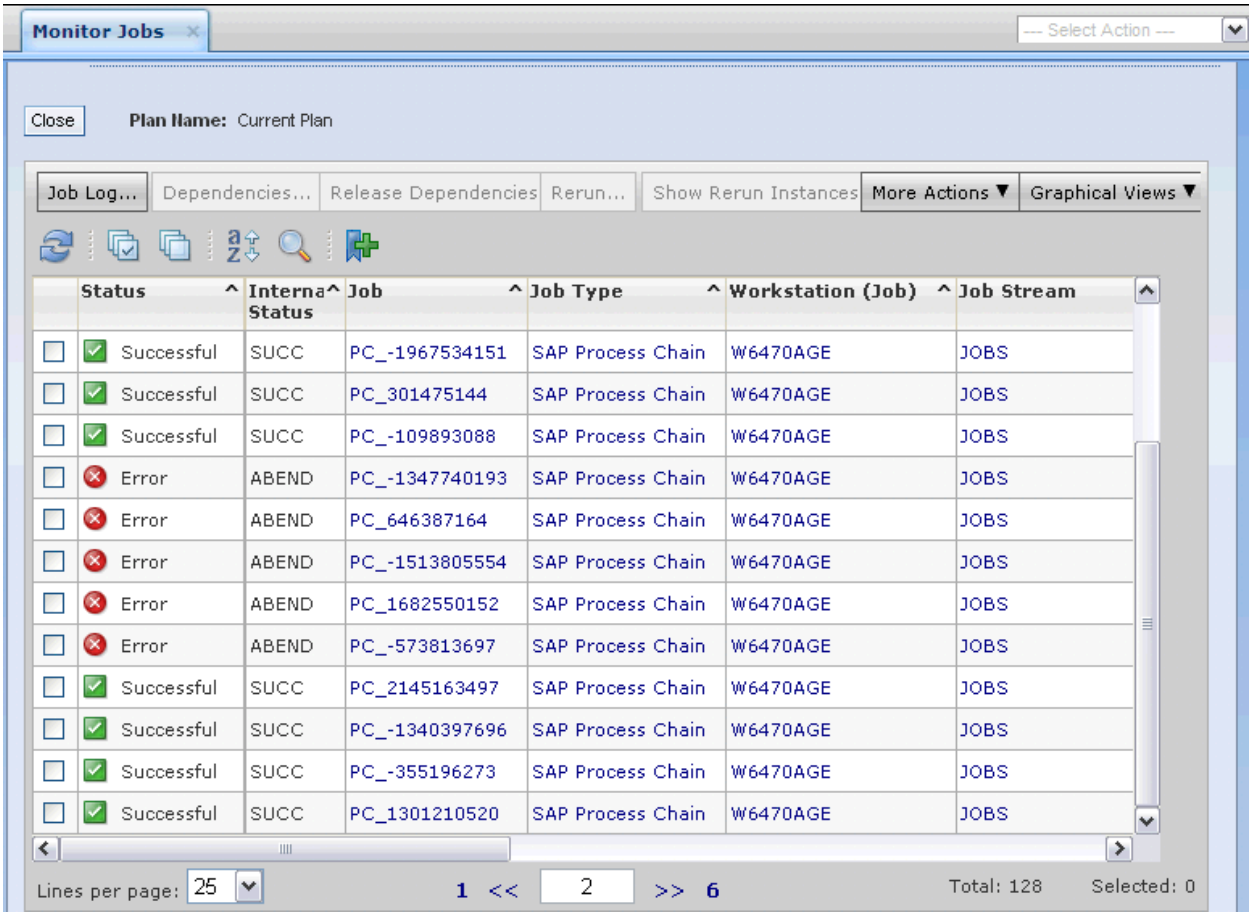
- Set the **pchain_details** option to `ON` in the common options file. For more information about this option, refer to Defining the common options on page 1154.
- In a distributed environment, customize the Browse Jobs tasks that you created *before* installing HCL Workload Automation 8.4 Fix Pack 1 to show the **Job Type** column. For details about how to customize the task properties, refer to the Dynamic Workload Console online help.
- In a z/OS environment, you must customize the task properties to display the **Advanced Job Type** column that indicates the job type. For details about how to customize the task properties, refer to the Dynamic Workload Console online help.

About this task

To display details about an SAP Process Chain that you scheduled as an HCL Workload Automation job, perform the following steps from the Dynamic Workload Console.

1. Click **Monitoring and Reporting > Workload Monitoring > Monitor Workload**.
2. In **Engine name**, select the name of the HCL Workload Automation engine connection from where you want to work with SAP jobs.
3. In **Object Type**, leave the default selection **Job**.
4. Click **Run**.
5. The table of results corresponding to the search criteria is displayed:

   Figure 38. Dynamic Workload Console - Table of results



6. Select a process chain job. For each process chain job, a hyperlink named **SAP Process Chain** is displayed.

   Distributed environment

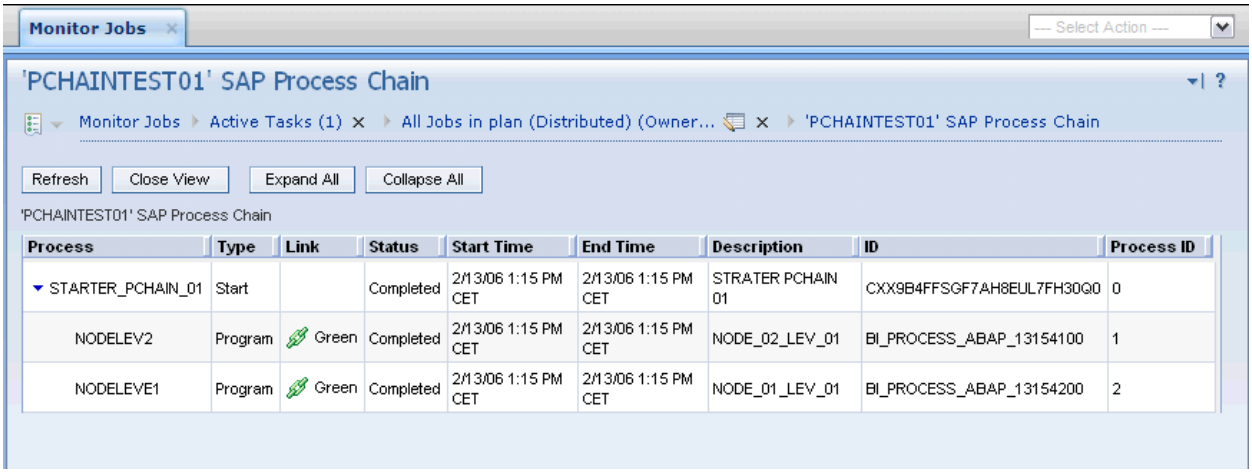
   The **Job Type** column displays **SAP Process Chain** to help you identify SAP process chain jobs.

   z/OS environment

   The **Advanced Job Type** column displays **SAP Process Chain** to help you identify SAP process chain jobs.

   Click the hyperlink for the job whose details you want to display.
7. The details for the process chain are displayed:

Figure 39. Dynamic Workload Console - Details of a process chain job



HCL Workload Automation monitors the process chain job until the job completes. The details shown reflect the last monitoring process performed. Perform a restart of the process chain indicating a refresh operation to synchronize the details with those on the remote SAP system to have the most updated information possible. If the process chain contains local subchains, a hyperlink is displayed for each one. Click the hyperlink you want, to display details about the corresponding subchain job. Alternatively, you can display the process chain details by clicking the hyperlink for the job and display the job properties panel. Click the hyperlink shown under **SAP Job Details**. The details for the process chain are displayed.

Rerunning a process chain job

Process chain jobs can be rerun from the start, rerunning the entire process chain, or they can be restarted from a specific process. Restarting a process enables you to restart without rerunning the whole process chain again. You can choose to either restart from the failed processes in a process chain, or restart a specific process indicating the related process ID.

To rerun an SAP job that submits a process chain, you can use one of the following user interfaces:

conman

For details, refer to the *User's Guide and Reference*.

Dynamic Workload Console

See Procedure for rerunning a process chain job on page 1247 for information about performing this task from the console.

For information about rerunning an SAP Standard R/3 job, see Rerunning a standard SAP job on page 1197.

In general, when you rerun a process chain job, the new job is assigned the name of the alias you specify. To keep the original job name, set the HCL Workload Automation global option **enRetainNameOnRerunFrom** to `yes`. For details about this option, see *Administration Guide*.

On extended agents, an alias is mandatory for each action you perform on the process chain job and the action itself, is the prefix of the alias name. For example, if you choose to restart a process chain from the failed processes, and assign PCHAIN1 as the alias for the process chain job, then the new job name is `Restart_PCHAIN1`.

In a z/OS environment, the process chain job maintains the same name and the Monitor Jobs view always displays the status for the last action performed on the job. Every time a rerun is performed on a process chain job, a new instance is generated each with a different ID.

Note:

1. By default, if you do not specify any setting, rerunning a process chain job corresponds to submitting a new process chain instance.
2. If you kill an HCL Workload Automation job that submits a process chain, the process chain is removed from schedule in the SAP Business Information Warehouse system. To restart the same process chain instance with r3batch, you require at least the following SAP Business Information Warehouse versions:
   - 3.0 with SP25
   - 3.1 with SP19
   - 3.5 with SP10
   - 7.0

   If your version of SAP Business Information Warehouse is earlier, you can restart the process chain only manually, through the SAP graphical interface.

   Table 156: Actions performed when you rerun a process chain job on page 1244 shows the action performed when you rerun an HCL Workload Automation job that submits a process chain, depending on the settings you specify. These are the actions performed when you submit the rerun operation using the Rerun button from the Monitor Jobs view.

Table 156. Actions performed when you rerun a process chain job

Action performed	Description and setting
A new process chain instance is submitted	HCL Workload Automation creates another process chain instance and submits it to be run again. This action occurs when: • On extended agents, you specify `RERUN`*`value`* as the step to rerun, where *`value`* is any value you want. This setting overrides the settings in the job definition and options file, if any. In an end-to-end environment, you can perform this action on a centralized job by adding the following parameter to the script file: `-flag pchain_rerun`

Table 156. Actions performed when you rerun a process chain job (continued)

Action performed	Description and setting
	• In the job definition, you set `-flag pchain_rerun`. This setting overrides the setting in the options file, if any. For a description of this parameter, see Table 155: Task string parameters for SAP jobs on page 1238. • In the options file, you set the **pchain_recover** option to `rerun`. For a description of this option, refer to Table 147: r3batch common configuration options on page 1154.
The original process chain is rerun from the failed processes	HCL Workload Automation restarts the original process chain from the failed processes to the end. In this way, after you detected the error that caused the failure and performed the recovery action, you can rerun the process chain job from the failed processes and have its run completed. This action is performed only if at least one process in the process chain did not complete successfully. It occurs when: • On extended agents, you specify `RESTART`*`value`* as the step to rerun, where *`value`* is any value you want. This setting overrides the settings in the job definition and options file, if any. In an end-to-end environment, you can perform this action on a centralized job by adding the following parameter to the script file: `-flag pchain_restart` • In the job definition, you set `-flag pchain_restart`. This setting overrides the setting in the options file, if any. For a description of this parameter, see Table 155: Task string parameters for SAP jobs on page 1238. • In the options file, you set the **pchain_recover** option to `restart`. For a description of this option, refer to Table 147: r3batch common configuration options on page 1154.
The process that you specify is restarted	HCL Workload Automation restarts the process of the original process chain that you specify, and monitors the process chain run until its final state. On extended agents, this action occurs when you specify `PROCESS`*`processID`* as the step to rerun, where *`processID`* is the identifier of the process you want. For example, if the process ID is `3`, you must specify `PROCESS3` as the step. You can view the process IDs in the following ways:

Table 156. Actions performed when you rerun a process chain job (continued)

Action performed	Description and setting

	• Dynamic Workload Console, version 8.5 or later. From the panel where the details about the process chain are displayed, see the column named Process ID. For details about how to display the process chain details, refer to Displaying details about a process chain job on page 1241.
	• HCL Workload Automation job log, as follows:
	`+++ EEWO1071I Start of process chain PCHAIN1`
	`Process Chain PCHAIN1 (Log ID:D3C0ZWAYESD58PXOYPEOGNZK7).`
	`-------------------------`
	`Process Type: TRIGGER.`
	`Process Variant: PCHAIN1_STARTER.`
	`Actual State: F.`
	`...`
	`>> **Process ID: 3.**`
	`Process Type: ABAP.`
	`Process Variant: Z_PCHAIN1_NODE3.`
	`Actual State: F.`
	`Instance: D3C0ZXL3IJ8LR5O9Q1D9A4Y4N.`
	
	`>> **Process ID: 4.**`
	`Process Type: ABAP.`
	`Process Variant: Z_PCHAIN1_NODE1.`
	`Actual State: .`
	`Instance: D3C0ZZKS0RR88DKRJQ09Z1WW7.`
	
	`+++ EEWO1072I End of process chain PCHAIN1`
	The following list shows the meaning of the alphabetic value used as the actual state in the job log:
	Actual state
	Meaning
	A
	Active
	F
	Completed
	G
	Successfully completed
	P
	Planned
	Q
	Released

Table 156. Actions performed when you rerun a process chain job (continued)

Action performed	Description and setting
	R Ended with errors **S** Skipped **X** Canceled **Y** Ready ***blank*** Undefined In an end-to-end environment, you can perform this action on a centralized job by adding the following parameter to the script file: `-pchain_pid` *processID*
The status and details of the original process chain are updated	HCL Workload Automation monitors the original process chain until its final status. This action occurs when: • On extended agents, you specify `REFRESH`*`value`* as the step to rerun, where *`value`* is any value you want. This setting overrides the setting in the job definition, if any. In an end-to-end environment, you can perform this action on a centralized job by adding the following parameter to the script file: `-flag pchain_refresh` • In the job definition, you set `-flag pchain_refresh`. For a description of this parameter, see Table 155: Task string parameters for SAP jobs on page 1238.

Procedure for rerunning a process chain job

You can rerun all of the processes in the process chain from the Dynamic Workload Console or you can rerun at a process level.

Before you begin

In z/OS environments, you need to set the status of the job to **Ready** before you can rerun the job.

1. Select a job and click **Set Status**.
2. In **Change Status**, select **Ready**.
3. Click **OK** to return to the list of jobs.

About this task

To rerun a process chain SAP job, perform the following steps:

1. Click **Monitoring and Reporting > Workload Monitoring > Monitor Workload**.
2. In **Engine name**, select the name of the HCL Workload Automation engine connection from where you want to work with SAP jobs.
3. In **Object Type**, leave the default selection **Job**.
4. Click **Run**.
5. A list of jobs is displayed. Select a process chain job.

   Distributed environment

   The **Job Type** column displays **SAP Process Chain** to help you identify SAP process chain jobs.

   z/OS environment

   The **Advanced Job Type** column displays **SAP Process Chain** to help you identify SAP process chain jobs. To display the **Advanced Job Type** column in the table, edit the **Task Properties** and in **Column Definition**, add the **Advanced Job Type** column to the **Selected Columns** list. Move the column up to define the order of the column in the table and make it more visible.

6. Rerun the job.

   a. Click **More Actions > Restart Process Chain**.

   b. Select the action you want to perform on the selected process chain:

      Rerun

      Reruns the entire process chain. The process chain ID on the SAP system remains the same, as well as the job identifier on z/OS systems.

      Specify an alias to identify the new job. In distributed systems the rerun process chain is identified with this alias name prefixed by `RERUN`.

      Refresh

      Refreshes the Dynamic Workload Console view with the latest updates on the remote SAP system so that the two views are synchronized.

      Specify an alias to identify the new job. In distributed systems the refreshed process chain is identified with this alias name prefixed by `REFRESH`.

Restart from the failed processes

Action available only for process chains in error state. Rerun only some steps of the process chain, starting from the failed processes.

Specify an alias to identify the new job. In distributed systems the restarted process chain is identified with this alias name prefixed by `RESTART`.

Restart from a specific process

Action available only for process chains in error state. Rerun only some steps of the process chain, starting from the process specified in the **SAP Process ID** field. You can find the process ID by opening the job log or viewing the job type details from the table of results of your monitor job task.

In distributed systems the restarted process chain is identified with this alias prefixed by `PROCESS`.

7. Click **OK** to perform the selected action on the process chain.

Results

The job reruns immediately.

Business scenario: rerunning the original process chain job from the failed process

As a scheduling administrator, you are responsible for managing batch jobs in both SAP and non-SAP systems. The workflow is one or more job streams in HCL Workload Automation. A job stream contains jobs that collect and prepare data for month-end closing over all sales channels. The month-end closing report requires data to be collected from several sales and distribution systems. Data is collected using local and remote process chains in the SAP Business Intelligence system. The process chains include a set of Infopackages, ABAP reports, and operating system jobs to sort the report data by a logical hierarchy.

To administer from a single point of control, you link the SAP process chains to HCL Workload Automation through HCL Workload Automation.

During batch processing, an HCL Workload Automation job comprising a process chain, failed. Optionally, you can see which processes failed either from the Dynamic Workload Console (for details, see Displaying details about a process chain job on page 1241) or in the job log. You ask the SAP administrator to fix the cause of the error, then, on an extended agent, you rerun the HCL Workload Automation job by setting the step as `RESTART`*`value`*. In this way, the original process chain is restarted from the failed processes and continues until the ending step.

Alternatively, you can select the process chain job from the Monitor Jobs view on the Dynamic Workload Console and then select **More Actions > Restart Process Chain** and then select the **Restart from the failed processes** option.

Business scenario: restarting a specific process of the process chain

You might decide to restart a single process as a preparation step before restarting the failed processes of a process chain. A failed process might have corrupted some data, so you run the single process to restore the data and set up the required system state before you rerun the other processes in the process chain.

Suppose you are using InfoPackages and process chains to extract data from one or several sources and you want to transform this data into managerial reports, for example by using aggregate functions. If the process that transforms this data fails, it might corrupt the data that the preceding InfoPacakge process had successfully extracted. After fixing the problem with the transformation process, you must restart the InfoPackage extraction process to reload the data, even though this extraction process had completed successfully before. Restart the failed transformation process only after the data has been reloaded, either by restarting the failed processes of the process chain or by restarting just the failed transformation process.

On an extended agent, from the Monitor Jobs view on the Dynamic Workload Console, select the process chain and click Rerun, then specify `PROCESS`*`processID`* as the step to rerun, where *processID* is the identifier of the process you want to restart.

To restart a specific process of the process chain, from the Monitor Jobs view on the Dynamic Workload Console, select the process chain and click **More Actions > Restart Process Chain** and then select the **Restart from a specific process** option, specifying the process ID in the **SAP Process ID** field.

Job throttling feature

Learn how the job throttling feature helps you to improve the efficiency of your scheduling on SAP systems and reduce the batch window for your SAP jobs to a minimum.

Using advanced XBP 2.0 and 3.0 functions, such as the job interception and parent-child, the job throttler ensures that the SAP system is not overloaded and the number of released jobs does not exceed the total number of SAP background work processes in the system.

You can also configure the job throttler to send data related to its activity to the SAP Computing Center Monitoring System (CCMS) for monitoring purposes.

Business scenario

You manage your Internet sales through an application software that verifies that data is correct, checks the availability of the item, and validates the order. To process all the orders received, you scheduled an HCL Workload Automation job to run every 12 hours, connect to SAP, and generate a child job for every order to process. Child jobs are in charge of creating shipping bills, checking destination address, and forwarding the orders to the appropriate carrier, thus optimizing the delivery

process. A potential overload of the system might occur during peak times, for example over Christmas, and could risk the late delivery of orders, damaging your business. To manage the submission of jobs and activate an advanced management of their priority class (for both parent and child jobs), enable the job throttling feature.

Additionally, you might want to set a policy so that an SAP CCMS alert is raised each time the number of jobs to be released under the control of the job throttler exceeds a certain threshold. To do this, you enable the job throttler to send data to the SAP CCMS monitoring architecture. At job throttler startup, an MTE that monitors the number of jobs to be released by the job throttler is created. By including the MTE in a monitoring set and specifying the related threshold, you are alerted each time the threshold is exceeded.

Software prerequisites

To use job throttling, you must have the SAP JCo 3.0.2 libraries or later (`dll` and `jar` files) installed in the `<data_dir>/methods/throttling/lib` directory. To download JCo 3.0.x, visit the SAP Service Marketplace web site.

Setting and using job throttling

The job throttler enqueues intercepted jobs and releases them when the background work processes that they need on the SAP server or SAP server group are available. The queue of intercepted jobs is sorted by scheduling time and priority of SAP jobs. When the SAP parent-child feature is enabled, child jobs inherit their progenitor's priority so that new urgent jobs are run before other planned jobs.

The following sections describe the steps to operate job throttling.

Step 1. Setting the options in the options file

About this task

To define the behavior of the job throttling feature, set the following options in the options file. For detailed information about the options, see Table 147: r3batch common configuration options on page 1154.

- `throttling_enable_job_class_inheritance`
- `throttling_enable_job_interception`
- `throttling_interval`
- `throttling_max_connections`
- `throttling_release_all_on_exit`

Step 2. Enabling and configuring the job interception feature

About this task

As a prerequisite, the job throttler requires that the job interception feature is enabled in the SAP system. To enable and configure job interception, follow these steps.

Note: Ensure that the job throttling and job interception features are not running at the same time. The job throttler cannot start if interception collector jobs are running.

1. Enable job interception, either automatically or manually, as follows:

   Automatic activation (meaning that the job throttler enables the job interception on SAP system)

   In the options file of the workstation with the r3batch access method you are using, set `throttling_enable_job_interception=on` (this is the default).

   Manual activation

   a. In the SAP system, run the INITXBP2 ABAP program in the transaction `se38` and enable job interception.

   b. In the options file of the workstation with the r3batch access method you are using, set `throttling_enable_job_interception=off`.

   Note: When you stop the job throttler, the setting for the job interception feature that was previously configured on the SAP system is restored.

2. In the SAP system, configure the job interception criteria as follows:
   a. Launch the transaction `se16` to access the table TBCICPT1, where the interception settings are maintained.

   b. Set the job name, creator, and client related to the jobs you want to intercept. To intercept all SAP jobs, specify the wildcard * (asterisk) for the job name, creator, and client.

   c. Save your settings and close the dialog.

SAP will intercept all the jobs matching the selection criteria, and the job throttling will release all the jobs that were intercepted.

Step 3. Enabling job class inheritance

About this task

You can configure the job throttler to have the intercepted job inherit the priority class from its progenitor (the top-level job in the hierarchy), if the progenitor class is higher than the intercepted job class. To do this, in the options file set `throttling_enable_job_class_inheritance=on`; this setting automatically enables the parent-child feature on the SAP system.

Note: When you stop the job throttler, the setting for the parent-child feature that was previously configured on the SAP system is restored.

Step 4. Configuring the logging properties

About this task

You can configure the trace properties of the job throttler by editing the logging configuration file `jobthrottling.properties` located in `<data_dir>/methods/throttling/properties`.

To configure the trace level, follow the procedure.

1. Set the trace level property. The supported trace levels are: `DEBUG_MIN`, `DEBUG_MID`, and`DEBUG_MAX`, where `DEBUG_MAX` is the most verbose trace level.
2. Save the changes.

Results

When making changes to the trace level setting, the changes are effective immediately after saving the .properties file. Other changes might require a restart to make them effective.

What to do next

You can also configure the name, number, and size of the trace file. By default, the job throttler generates a maximum of 3 files of 5 MB in the `<data_dir>/methods/traces` directory.

Note: The job throttler creates the `<data_dir>/methods/traces` directory as soon as it is started.

Step 5. Starting and stopping the job throttling feature

About this task

To start job throttling, run the `jobthrottling` executable file related to the operating system you are using. Optionally, you can create an HCL Workload Automation job that starts the job throttler.

Note: On Windows systems using a single-byte character language, to start job throttling from a command prompt ensure that the DOS shell font is *not* Lucida Console. Ensure also that you set the HCL Workload Automation environment by entering the following command:

```
TWA_home\tws_env.cmd
```

From a command prompt, enter:

UNIX operating systems

```
TWA_home/methods/jobthrottling.sh {XA_Unique_ID|base_options_filename} [-scratch]
```

Windows operating systems

```
TWA_home\methods\jobthrottling.bat {XA_Unique_ID|base_options_filename} [-scratch]
```

Where:

XA_Unique_ID

The unique identifier for the extended agent workstation you are using. See UNIQUE_ID on page 1085 for details about retrieving the unique identifier for a workstation.

base_options_filename

For dynamic and z-centric agents, the file name of the options file without the extension, defined on the engine workstation hosting the workstation with the r3batch access method.

-scratch

If you enabled the job throttler to send data to CCMS (for details, see Sending data from job throttling to the CCMS Monitoring Architecture on page 1255), the job throttler starts and resets the attribute MTE named `JT total released jobs` to 0. If you do not specify **-scratch**, the job throttler starts and increments the `JT total released jobs`.

This parameter is optional, and has effect only if the job throttler sent its data to CCMS at least once before.

To know the syntax for the `jobthrottling` command, run the command as follows:

```
jobthrottling -u
```

To stop the job throttler, enter the following command (optionally, you can create an HCL Workload Automation job that stops the job throttler):

UNIX operating systems

TWA_home/methods/stop-jobthrottling.sh {*XA_Unique_ID*|*base_options_filename*}

Windows operating systems

TWA_home\methods\stop-jobthrottling.bat {*XA_Unique_ID*|*base_options_filename*}

Alternatively, you can enter the following command (you must be connected as *TWSUser* and have read and write permissions on the `txt` file):

```
echo shutdown > TWA_home/methods/{XA_Unique_ID|base_options_filename}_jobthrottling_cmd.txt
```

The job throttler stops:

- When the timestamp of {*XA_Unique_ID*|*base_options_filename*}_jobthrottling_cmd.txt is later than the time when the job throttler started.
- Within the time interval you specified in the `throttling_interval` option.

Sending data from job throttling to the CCMS Monitoring Architecture

About this task

You can configure the job throttler to send data related to its activity to the SAP Computing Center Monitoring System (CCMS) for monitoring purposes. Sending data from the job throttler to CCMS is supported if you have at least the SAP Web Application Server 6.20, Support Package 12 installed.

In the options file, set the following options (for details, see Table 147: r3batch common configuration options on page 1154):

```
throttling_send_ccms_data
throttling_send_ccms_rate
```

In this way, at job throttler startup the following monitoring tree elements (MTE) are created:

- A context MTE named **ITWS for Apps**.
- An object MTE with the same name as the HCL Workload Automation extended agent where the job throttler is running. This object MTE belongs to the context MTE **ITWS for Apps**.
- The following attribute MTEs:

JT total released jobs

The total number of jobs that the job throttler has released since startup. This value depends on the `-scratch` option you set at job throttler startup; for details, see Step 5. Starting and stopping the job throttling feature on page 1254.

JT queue

The number of enqueued intercepted jobs to be released.

JT released jobs per cycle

The number of released jobs in the latest run. This value depends on the `throttling_send_ccms_rate` setting; for details, see Table 147: r3batch common configuration options on page 1154.

Note: By default `throttling_release_all_on_exit` is set to `ON`, meaning that when you stop the job throttler, all the intercepted jobs are released. However, these jobs are not considered when updating the **JT total released jobs**, **JT queue**, and **JT released jobs per cycle** MTEs.

To begin monitoring, include the MTEs in the monitoring set you want, and set the thresholds to generate an alert.

You can define an HCL Workload Automation event rule based on the CCMS alerts; for detailed information, refer to Defining event rules based on CCMS Monitoring Architecture alerts on page 1281.

For example, to define an event that monitors the attribute MTE **JT total released jobs**, on the extended agent workstation with unique identifier `SAP_XA`, connected to the SAP system ID `T01`, specify the following information:

XA Workstation

`SAP_XA`

MTE SAP System ID

`T01`

MTE Monitoring Context Name

`ITWS for Apps`

MTE Monitoring Object Name

`SAP_XA`

MTE Monitoring Attribute Name:

`JT total released jobs`

Deleting the monitoring tree elements

About this task

After you stopped the job throttling feature, if you configured it to send its status data to CCMS, you can delete one or more MTEs that were created. To do this:

1. From the SAP GUI, invoke the transaction `rz20` to display a list of monitor sets.
2. Locate the monitor set named SAP CCMS Technical Expert Monitors, and expand it.
3. Locate the monitor named All Monitor Contexts, and double-click it to open it.
4. From the action menu, select **Extras -> Activate Maintenance Functions**.
5. Locate the MTE named **ITWS for Apps** and select it.
6. Right-click the MTE and select **Delete**. You are prompted to choose one of the delete options.
7. Select the option you want. The MTE is deleted accordingly.

Note: Deleting **ITWS for Apps** from the All Monitor Contexts monitor, deletes also all the copies that you might have created in other monitors.

Exporting SAP factory calendars

This section describes how to export SAP factory calendars into a file format that can be processed by the HCL Workload Automation **composer** command line, to add the exported calendar definitions to the HCL Workload Automation database.

Business scenario

About this task

You might want to configure your HCL Workload Automation scheduling activities based on the schedule calendar in your SAP system. To do this, use the `r3batch` export function to export the SAP calendar definitions into a file whose format is compatible with the HCL Workload Automation **composer** command line. Based on the parameters you specify, you create a file that contains only the SAP calendar definitions that meet your scheduling requirements. Use this file as input for the **composer add** command, to import the calendar definitions into the HCL Workload Automation database. Your HCL Workload Automation and SAP calendars are now synchronized.

To keep the HCL Workload Automation and SAP calendar definitions synchronized and avoid duplicating data maintenance in the two environments, you can schedule to export the calendar definitions from SAP and import them to HCL Workload Automation on a regular basis using a dedicated job.

Exporting and importing SAP factory calendars

Refer to the following sections:

- Exporting factory calendars on page 1258 for an explanation about how you use the **r3batch** export function to access and download factory calendars available in an SAP system. The main purpose of this function is to create an output file that can be used by the **composer** to synchronize HCL Workload Automation calendars with existing SAP factory calendars, integrating the calendar definitions from SAP into HCL Workload Automation.
- Importing factory calendars on page 1260 for an explanation about how you import the exported calendar definitions into the HCL Workload Automation database.

For details about the HCL Workload Automation calendar definitions, see *User's Guide and Reference*.

Exporting factory calendars

About this task

To export an SAP R/3 calendar, from `TWA_home/methods` (where *TWA_home* is the complete path where you installed HCL Workload Automation) enter the following command:

Command syntax

```
r3batch -t RSC -c XA_Unique_ID -- " -calendar_ID calendarID -year_from yyyy -year_to yyyy [ { -getworkdays | -getfreedays } ] [ -tws_name tws_cal_name ] [ -tws_description tws_cal_desc ] [ -filename output_filename ] "
```

Where:

-t RSC

The identifier of the task to be performed, in this case RSC (Retrieve SAP R/3 Calendars). This parameter is required.

-c *XA_Unique_ID*

The unique identifier for the extended agent workstation connected to the SAP R/3 system where the calendar data to export is located. The SAP R/3 system must be configured as a workstation to HCL Workload Automation. This parameter is required. For information about retrieving the unique identifier for the extended agent workstation, see UNIQUE_ID on page 1085.

-calendar_id *calendarID*

The identifier of the SAP R/3 calendar to be exported, which consists of two alphanumeric characters. This parameter is required.

-year_from *yyyy*

The year of the calendar from when to start exporting dates, in the format *yyyy*. This parameter is required.

-year_to *yyyy*

The year of the calendar when to stop exporting dates, in the format *yyyy*. This parameter is required.

-getworkdays | -getfreedays

Specify getworkdays to create the HCL Workload Automation calendar definition based on the working days of the SAP R/3 calendar. In this way, each date of a working day is stored in the output file.

Specify getfreedays to create the HCL Workload Automation calendar definition based on the holidays of the SAP R/3 calendar. Each date of a non-working day is stored in the output file.

These parameters are optional and mutually exclusive. If you do not specify either, the default is getworkdays.

-tws_name *tws_cal_name*

The HCL Workload Automation name for the exported SAP R/3 factory calendar. It is stored in the output file.

You can specify up to eight alphanumeric characters. This parameter is optional, the default is SAP*XX_calendarID*, where:

XX

Corresponds to WK if the calendar includes only working days or FR if the calendar includes only non-working days.

calendarID

The identifier of the SAP R/3 calendar.

For example, the default HCL Workload Automation name for an exported calendar, whose identifier is 04, that includes only working days, is SAPWK_04.

-tws_description *tws_cal_desc*

The description of the HCL Workload Automation calendar. It is stored in the output file. You can specify up to 120 alphanumeric characters. If the description contains blanks, it must be enclosed between single quotes. This parameter is optional.

-filename *output_filename*

The name of the output file that is to contain the calendar definitions. This file is written in a scheduling language that can be processed by the **composer** when you add the calendar data to the HCL Workload Automation database.

You can specify a file name with its complete or partial path; if you do not specify any path, the file is created in the current directory. If the path you specify does not exist, it is created, provided that you have the appropriate access rights. Otherwise, the command returns an error message and is not performed.

You can specify up to the maximum number of characters allowed by your operating system. If the name of the file contains blanks, it must be enclosed between single quotes. If another file with the same name exists, it is overwritten.

This parameter is optional. The default value is *tws_name*.txt, where *tws_name* is the value you set for the tws_name parameter.

The following is an example of an SAP R/3 factory calendar export command:

```
r3batch -t RSC -c horse10 -- " -calendar_id 01 -year_from 2007
-year_to 2010 -tws_name CAL1 -tws_description 'SAP Calendar 01'
-getworkdays -filename 'my dir/calendar_01.dat' "
```

This command exports the SAP R/3 calendar named 01, located on the SAP R/3 system named horse10. The dates exported begin from year 2007, until year 2010, considering only working days. The HCL Workload Automation name used for the calendar is CAL1, and the description written in the output file is SAP Calendar 01. The output file is named `calendar_01.dat`, stored in `<data_dir>/methods/my dir`, and its content looks like the following

```
$CALENDAR
    CAL1
    "SAP Calendar 01"
    01/02/2007 01/03/2007 01/04/2007 01/05/2007 01/08/2007 01/09/2007 01/10/2007
    01/11/2007 01/12/2007 01/15/2007 01/16/2007 01/17/2007 01/18/2007 01/19/2007
    01/22/2007 01/23/2007 01/24/2007 01/25/2007 01/26/2007 01/29/2007 01/30/2007
    01/31/2007 02/01/2007 02/02/2007 02/05/2007 02/06/2007 02/07/2007 02/08/2007
     .......
    11/24/2010 11/25/2010 11/26/2010 11/29/2010 11/30/2010 12/01/2010 12/02/2010
    12/03/2010 12/06/2010 12/07/2010 12/08/2010 12/09/2010 12/10/2010 12/13/2010
    12/14/2010 12/15/2010 12/16/2010 12/17/2010 12/20/2010 12/21/2010 12/22/2010
    12/23/2010 12/24/2010 12/27/2010 12/28/2010 12/29/2010 12/30/2010 12/31/2010
```

Importing factory calendars

About this task

To import the exported calendar definitions into the HCL Workload Automation database, copy the output file from the extended agent for SAP R/3 to the master workstation and from the **composer** command line on the master workstation, enter the following command:

```
-add output_filename
```

where `output_filename` is the name of the exported file, with its complete path.

For example, to import the `tws_calendar_01.dat` file exported in the previous example, copy the file to the master workstation. From the **composer** command line on the master workstation, enter:

```
-add TWA_home/methods/my dir/tws_calendar_01.dat
```

where *TWA_home* is the complete path where you installed HCL Workload Automation.

Defining internetwork dependencies and event rules based on SAP background events

This section describes how to define internetwork dependencies and event rules for HCL Workload Automation based on SAP background events.

Note: To be able to define and monitor event rules, you must configure your environment as described in Configuring SAP event monitoring on page 1169.

Defining internetwork dependencies based on SAP background events

Dependencies are prerequisites that must be satisfied before a job or job stream can start. Internetwork dependencies are dependencies checked by the extended agent workstation to which they belong. In response to an internetwork dependency, the SAP extended agent checks for the occurrence of the SAP background event specified in the dependency. As soon as the SAP event is raised, the SAP extended agent commits the event and instructs HCL Workload Automation to resolve the corresponding internetwork dependency.

For more details about internetwork dependencies, refer to the *HCL Workload Automation: User's Guide and Reference*. For more details about how to raise SAP events, see Raising an SAP event on page 1195.

To define SAP background events as internetwork dependencies, XBP versions 2.0 and 3.0 are supported, with the following differences:

XBP version 2.0

SAP background events can release HCL Workload Automation internetwork dependencies only if the dependencies are created or checked *before* the SAP event is raised. An event history is ignored, therefore an SAP event raised before the internetwork dependency is created, is not considered.

Note:

Because an SAP event history is ignored, for each SAP background event to be checked, a placeholder SAP job is created. This is a dummy job whose running depends on the SAP background event, therefore an SAP event is considered raised as soon as the corresponding placeholder job has completed.

XBP version 3.0 (supported by SAP NetWeaver 7.0 with SP 9, or later)

Only the SAP background events stored in the SAP event history table are considered by HCL Workload Automation to check for internetwork dependencies resolution. As a prerequisite, the SAP administrator must create the appropriate event history profiles and criteria on the target SAP system.

To avoid performance reduction, run reorganization tasks against the SAP event history.

Note:

Some SAP systems providing XBP version 3.0 still return XBP version as 2.0. To check if your SAP system provides XBP 3.0, invoke the transaction `se37` and search for the function module `BAPI_XBP_BTC_EVTHISTORY_GET`. If your system contains the module, set the `xbpversion` option to 3. In this way, **r3batch** will ignore the XBP value returned by the SAP system. For details about the `xbpversion` option, refer to Table 147: r3batch common configuration options on page 1154.

To define an SAP background event as an internetwork dependency, use the following parameters:

Table 157. Parameters to define an SAP internetwork dependency

Parameter	Description	GUI support
-evtid *sap_event_name*	The name of the SAP background event, up to 32 characters. If the name contains blanks, enclose it between single quotes. This parameter is required.	✓
-evtpar *sap_event_parm*	The SAP event parameter, up to 64 characters. If the parameter contains blanks, enclose it between single quotes. This parameter is optional.	✓
-commit	Defines that the SAP background event is committed immediately after the internetwork dependency has been resolved. If you do not specify `-commit`, the event must be committed by running the `r3batch` task PI. The default is that `-commit` is not specified. For details about the PI task, refer to Committing SAP background events by an external task on page 1264. In addition to this parameter, you can set as default that the system commits internetwork dependencies immediately by specifying `commit_dependency=on` in the options file. For details about the commit_dependency option, see Table 147: r3batch common configuration options on page 1154. **Note:** With XBP version 2.0, defining two internetwork dependencies on the same SAP event might lead to an error, if `-commit` is specified. For example, suppose you define an internetwork dependency for the SAP event `SAPEVT`, with or without setting `-commit`. After this definition, the SAP event `SAPEVT` is raised. Then you define a second internetwork dependency based on `SAPEVT`, specifying `-commit`. The second dependency immediately commits the SAP event, with the consequence that the first dependency becomes impossible to resolve. Therefore, when the first job checks for the internetwork dependency, an error is issued.	✓

The following example shows how to define an internetwork dependency based on the SAP background event named `SAP_TEST` with the parameter `12345678`. After its processing, the event is not immediately committed.

```
-evtid SAP_TEST -evtpar 12345678
```

The resulting internetwork dependency looks like the following, where `SAPWS` is the name of the extended agent workstation that connects to the SAP background processing system where the event runs:

```
follows SAPWS::"-evtid SAP_TEST -evtpar 12345678"
```

The following example shows how to define an internetwork dependency based on the SAP background event named `SAP_TEST`, without parameter. As soon as the internetwork dependency is resolved, the event is committed.

```
-evtid SAP_TEST -commit
```

The resulting internetwork dependency looks like the following, where `SAPWS` is the name of the extended agent workstation that connects to the SAP background processing system where the event runs:

```
follows SAPWS::"-evtid SAP_TEST -evtpar 12345678"
```

Table 158: Internetwork dependency definition and possible resolution on page 1264 shows the correspondence between the definition and possible resolution of an internetwork dependency that depends on an SAP event, with or without parameters assigned. In this table, `SAP_TEST` is used as the event name and `12345678` or `ABCDEFG` as the event parameter.

Table 158. Internetwork dependency definition and possible resolution

HCL Workload Automation internetwork dependency specified	SAP event raised in SAP system	SAP event pa rameter	HCL Workload Automation internetwork dependency resolved
-evtid SAP_TEST	none	none	No
-evtid SAP_TEST	END_OF_JOB	none	No
-evtid SAP_TEST	SAP_TEST	none	Yes
-evtid SAP_TEST	SAP_TEST	12345678	Yes
-evtid SAP_TEST -evtpar 12345678	SAP_TEST	none	No
-evtid SAP_TEST -evtpar 12345678	SAP_TEST	12345678	Yes
-evtid SAP_TEST -evtpar 12345678	SAP_TEST	ABCDEFG	No

Committing SAP background events by an external task

About this task

SAP events defined as HCL Workload Automation internetwork dependencies, by default are not automatically committed after their processing. You can modify this default by specifying the `-commit` parameter. Otherwise, if you leave the default, you must commit the processed event by using the external task Put Information (PI).

The PI task commits all the processed events that meet the given criteria. For this reason, it is recommended that you run this task at the end of the working day. By doing so, internetwork dependencies that are already resolved are not reset and the objects depending on them are not blocked until they are resolved again.

From a command line, enter the following command:

Command syntax

-r3batch -t PI -c *XA_Unique_ID* **-- " -t CE -evtid***sap_event_name* [**-evtpar***sap_event_parm*] **"**

Where:

-t PI

The identifier of the task to be performed, in this case PI (Put Information). This parameter is required.

-c *XA_Unique_ID*

The unique identifier for the extended agent workstation connected to the SAP background processing system where the event is run. This parameter is required. For information about retrieving the unique identifier for the extended agent workstation, see UNIQUE_ID on page 1085.

-t CE

The identifier of the task to be performed, in this case CE (Commit Event). This parameter is required.

-evtid *sap_event_name*

The name of the SAP R/3 event running on the background processing system. If the name contains blanks, enclose it between single quotes. This parameter is required.

-evtpar *sap_event_parm*

The parameter of the SAP event running on the background processing system. If the parameter contains blanks, enclose it between single quotes. This parameter is optional. If you do not specify it, all the SAP events with the name you specified, with or without a parameter, are committed on the target system.

The following is an example of how to commit the SAP event named `SAP_TEST`, with parameter `1234567`, with extended agent workstation with unique identifier `horse10` connected to the background processing system:

```
r3batch -t PI -c horse10 -- " -t CE -evtid SAP_TEST -evtpar 1234567"
```

Defining internetwork dependencies based on SAP background events with the Dynamic Workload Console

About this task

To define an SAP background event as an internetwork dependency with the Dynamic Workload Console, perform the following steps:

1. Launch the **Workload Designer** from the Dynamic Workload Console. From the **Design** menu, click **Workload Designer** page.
2. Search for and open the job stream you want to manage.
    a. Select the **Job stream card** to display all job streams present in the selected folder. Otherwise, you can select the **Job stream card** and then type the job stream name in the **Search** bar.
    b. Select the job stream and click **Edit**. The job stream and its contents are displayed in the **Workspace area**.
3. In the job stream subrow, click **Add Dependency** and select **Internetwork**.
4. Specify the properties for the internetwork dependency.
    a. In the **Network Agent** field, enter the name of the agent workstation connected to the SAP background processing system where the event runs.
    b. In the **Dependency** field, enter the parameters to define the internetwork dependency. For a description of the parameters allowed, refer to Table 157: Parameters to define an SAP internetwork dependency on page 1263.
5. Click **Save** to save the changes to the job stream.

Results

The local job or job stream now has a dependency on a SAP background event. You can also perform this procedure from the graphical view available from the job stream row in the **Workspace area**. For more information about adding dependencies and editing objects in the graphical view, refer to the Dynamic Workload Console User's Guide.

Defining event rules based on SAP background events

A scheduling event rule defines a set of actions to run when specific event conditions occur. The definition of an event rule correlates events and triggers actions.

An event rule is identified by a rule name and by a set of attributes that specify if the rule is active, the time frame of its validity, and other information required to decide when actions are triggered. It includes information related to the specific events (`eventCondition`) that the rule must detect and the specific actions it is to trigger upon their detection or timeout (`ruleAction`). Complex rules might include multiple events and multiple actions.

If you are using XBP 3.0, only the SAP background events that are stored in the event history table are considered by HCL Workload Automation.

To define event rules, you can use either of the following:

The composer command line

You edit the rules with an XML editor of your choice. For details about how to use the composer to define event rules, see the *User's Guide and Reference*.

The Dynamic Workload Console

For information about creating an event rule, see the section about creating an event rule in *Dynamic Workload Console User's Guide*.

The SAP background event is identified by the following information:

SAP Event ID

The name identifying the SAP event. Wildcards are not allowed.

If you are using the Dynamic Workload Console, you can type the event name in the SAP Event ID field. This field does not support wildcard characters (* and %), nor the following special characters: asterisk (*), question mark (?), and backslash (\). Note that for supported special characters, the escape character (\) must not be used.

Alternatively, you can use the lookup function to search for and select the event name. When specifying the string to search for that represents the SAP Event ID, wildcard characters are supported, (* and %). For example, if you specify "myevent*", then results can include events such as "myevent", "myevent%", and "myevents".

Event parameter

The parameter associated with the SAP event, if any. Wildcards are not allowed.

If you are using the Dynamic Workload Console, the following special characters are not supported when specifying the event parameter: asterisk (*), question mark (?), and backslash (\).

Extended or dynamic agent workstation

The name of the extended agent workstation or the name of the dynamic agent workstation running event monitoring.

Note:

1. If you specify a pattern with the wildcard asterisk (*), all the agents whose name matches the pattern will monitor the specified event.
2. As a best practice, define that an event belonging to an SAP system is monitored by one agent workstation only. If the same SAP event is monitored by more than one agent, you might either be notified multiple times for the same event occurrence or the first agent that notifies the event occurrence makes that event unavailable to the other agents.
3. If you modify the extended agent configuration in the `r3batch` option files, to make the changes effective you must stop and restart the agent.
4. For dynamic agents you can specify the name of a local options file. In the Properties section of the Create Event Rules window of the Dynamic Workload Console a lookup button provides a list of all the local options files associated with that agent. If you do not specify the name of a local options file, the global options file is used by default in the rule definition.

SAP events matching criteria

The SAP background events specified in the event rule are matched with the events raised in the SAP system, according to the following criteria. Depending on the parameters you set:

The SAP event ID and parameter are specified in the event rule

To match, the SAP event ID and parameter must be the same as the event ID and event parameter raised in the SAP system. Also, the event state must be N (New). SAP events with a different parameter or without any parameter are ignored.

The information collected about the matching SAP event is sent by the `r3evmon` process to HCL Workload Automation. If the notification is successfully sent, the event is committed on the SAP system and its state changed to C (Confirmed).

For example, you define an event rule in your HCL Workload Automation plan based on the following SAP event:

SAP event ID

SAP_TEST

SAP event parameter

ABCDEF

Workstation

An extended agent named GENIUS

According to these settings, a file named `GENIUS_r3evmon.cfg` is created on `GENIUS`. It contains the following `!R3EVENT` keyword:

```
!R3EVENT 0008SAP_TEST0006ABCDEF
```

Monitoring of the `SAP_TEST` event with parameter `ABCDEF` is automatically started. Suppose that the following SAP events were raised on the SAP system:

Table 159. History table of the SAP events raised

EVENT GUID	SAP EVENT ID	EVENT PARM	EVENT SERVER	EVENT TIMESTAMP	EVENT STATE	PROCESS STATE	COUNT OF JOBS
1234	SAP_TEST	ABC123	...	20070925 13:00	C	OK	1
2345	SAP_TEST	ABCD	...	20070925 14:00	N	OK	2
3456	SAP_TEST		...	20070925 15:00	N	OK	3
4567	SAP_TEST	ABCDEF	...	20070925 16:00	N	OK	4

Only the following SAP event is notified to HCL Workload Automation:

Table 160. SAP event matching with the event rule defined

EVENT GUID	SAP EVENT ID	EVENT PARM	EVENT SERVER	EVENT TIMESTAMP	EVENT STATE	PROCESS STATE	COUNT OF JOBS
4567	SAP_TEST	ABCDEF	...	20070925 16:00	N	OK	4

If the notification is successfully sent, the event is committed on the SAP system and its state changed to C (Confirmed).

Only the SAP event ID is specified in the event rule

To match, the SAP event ID must be the same as the ID of the events raised in the SAP system whose state is N (New). The parameters of the SAP events, whether specified or not, are not taken into account.

The information collected about all the matching SAP events is sent by the `r3evmon` process to HCL Workload Automation. Each event successfully notified is committed on the SAP system and its status changed to C (Confirmed).

For example, you define an event rule in your HCL Workload Automation plan based on the following SAP event:

SAP event ID

SAP_TEST

Workstation

GENIUS

According to these settings, a file named `GENIUS_r3evmon.cfg` is created on `GENIUS`. It contains the following `!R3EVENT` keyword:

```
!R3EVENT 0008SAP_TEST
```

Monitoring of the `SAP_TEST` event is automatically started. Suppose that the following SAP events were raised on the SAP system:

Table 161. History table of the SAP events raised

EVENT GUID	SAP EVENT ID	EVENT PARM	EVENT SERVER	EVENT TIMESTAMP	EVENT STATE	PROCESS STATE	COUNT OF JOBS
1234	SAP_TEST	ABC123	...	20070925 13:00	C	OK	1
2345	SAP_TEST	ABCD	...	20070925 14:00	N	OK	2
3456	SAP_TEST		...	20070925 15:00	N	OK	3
4567	SAP_TEST	ABCDEF	...	20070925 16:00	N	OK	4

Only the following SAP events are notified to HCL Workload Automation:

Table 162. SAP events matching with the event rule defined

EVENT GUID	SAP EVENT ID	EVENT PARM	EVENT SERVER	EVENT TIMESTAMP	EVENT STATE	PROCESS STATE	COUNT OF JOBS
2345	SAP_TEST	ABCD	...	20070925 14:00	N	OK	2
3456	SAP_TEST		...	20070925 15:00	N	OK	3
4567	SAP_TEST	ABCDEF	...	20070925 16:00	N	OK	4

Each event whose notification is successfully sent is committed on the SAP system and its state changed to C (Confirmed).

Setting a filter for SAP background events in the security file

In the security file, you can filter the SAP background events that can be used to define event rules. By doing this, you restrict the use of certain SAP events to specific users. For example, assume that you want your USA department to manage only the SAP events whose ID begins with `SAP_USA`, and your Italy department to manage all events except those beginning with `SAP_USA`. In the security file that defines the user access for the USA department, define the `CUSTOM` keyword for the `EVENT` object as follows:

```
EVENT      PROVIDER=@ +CUSTOM=SAP_USA@  ACCESS=USE
```

where:

`PROVIDER=@`

Specifies that the user can use the events coming from any provider.

`+CUSTOM=SAP_USA@`

Specifies that the user can use only the SAP events whose ID begins with `SAP_USA`.

This keyword applies only to the SAP provider (SapMonitor).

`ACCESS=USE`

Sets the user access to the object to `USE`.

In the security file that defines the user access for the Italy department, define the `CUSTOM` keyword for the `EVENT` object as follows:

```
EVENT      PROVIDER=@ ~CUSTOM=SAP_USA@  ACCESS=USE
```

where:

`PROVIDER=@`

Specifies that the user can use the events coming from any provider.

`~CUSTOM=SAP_USA@`

Specifies that the user can use all SAP events, except those whose ID begins with `SAP_USA`.

This keyword applies only to the SAP provider (SapMonitor).

`ACCESS=USE`

Sets the user access to the object to `USE`.

For more details about the security file and how to set up user authorizations, see the *HCL Workload Automation: Administration Guide*.

Defining event rules based on IDoc records

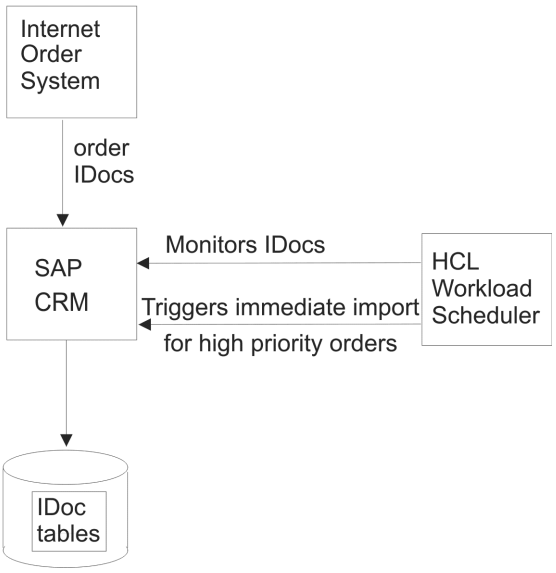
You can use HCL Workload Automation to monitor Intermediate Document (IDoc) records in SAP systems and forward events to the HCL Workload Automation event integration framework.

To do this, you define an event condition that contains the criteria that the IDocs must match to be forwarded to HCL Workload Automation. When the event condition occurs, the action that you associated with it (for example, running a job) is performed.

Business scenario

You connected your Internet sales application to your SAP Customer Relationship Management (CRM) system, which receives the orders as incoming IDocs. The orders are classified as emergency and ordinary, and therefore have different IDoc message types. You want the emergency orders to be imported into the CRM system directly, and the ordinary orders to be processed in batch mode. To do this, in HCL Workload Automation, you define an event rule that monitors the IDoc message types corresponding to emergency orders and sends an event to HCL Workload Automation. In HCL Workload Automation, you define a job to be released when this type of event is received and is linked to an SAP job that runs an import ABAP report for these specific types of IDocs.

Figure 42. Managing high priority IDocs overview



Creating event rules based on IDocs

About this task

To define event rules based on IDocs, specify the fields to be used as matching criteria during IDoc monitoring. For details about these fields, refer to Events matching criteria on page 1272. To create the event rules, you can use either of the following:

The composer command line

You edit the rules with an XML editor of your choice. For a general explanation about how to use the composer to define event rules, see the *HCL Workload Automation: User's Guide and Reference*. The event condition requires:

- `SAPMonitor` as event monitor provider.
- `IDOCEventGenerated` as event type.

For a list of the values that you can specify in the **attributeFilter** name when defining the event condition, refer to Table 165: Parameters of IDOCEventGenerated event type on page 1275.

The Dynamic Workload Console

For information about creating an event rule, see the section about creating an event rule in *Dynamic Workload Console User's Guide*.

Note:

1. To be able to define and monitor event rules, ensure that you configured your environment as described in Configuring SAP event monitoring on page 1169.
2. To configure how HCL Workload Automation retrieves the IDoc monitors, set **idoc_no_history** and **idoc_shallow_result** in the options file. For details about these options, refer to Defining the common options on page 1154.

Events matching criteria

Table 163: HCL Workload Automation fields used to define event rules based on IDocs on page 1272 lists the HCL Workload Automation fields corresponding to the fields in the IDoc record that you want to search. During monitoring, each IDoc matching the search criteria generates an event that is sent to HCL Workload Automation.

Table 163. HCL Workload Automation fields used to define event rules based on IDocs

Composer property	Console property	IDoc field
SAPClient	SAP client	MANDT
SAPIDocStatus	Status	STATUS
SAPDirectionIDocTransmission	Direction	DIRECT
SAPReceiverPort	Receiver port	RCVPOR

Table 163. HCL Workload Automation fields used to define event rules based on IDocs (continued)

Composer property	Console property	IDoc field
SAPReceiverPartnerFunction	Receiver partner function	RCVPFC
SAPReceiverPartnerType	Receiver partner type	RCVPRT
SAPReceiverPartnerNumber	Receiver partner number	RCVPRN
SAPSenderPort	Sender port	SNDPOR
SAPSenderPartnerType	Sender partner type	SNDPRT
SAPSenderPartnerFunction	Sender partner function	SNDPFC
SAPSenderPartnerNumber	Sender partner number	SNDPRN
SAPLogicalMessageType	Logical message type	MESTYP
SAPNameOfBasicType	Name of basic type	IDOCTP
SAPLogicalMessageCode	Logical message code	MESCOD
SAPLogicalMessageFunction	Logical message function	MESFCT
SAPTestFlag	Test flag	TEST
SAPOutputMode	Output mode	OUTMOD

Optionally, you can define also correlation rules by using the fields listed in Table 164: HCL Workload Automation fields used to define correlation rules for IDoc events on page 1273. Date and time values are specified in GMT time zone.

Table 164. HCL Workload Automation fields used to define correlation rules for IDoc events

Composer property	Console property	IDoc field
SAPIDocNumber	IDoc number	DOCNUM
SAPReleaseForIDoc	IDoc SAP release	DOCREL
SAPIDocType	IDoc type	DOCTYP
SAPReceiverAddress	Receiver SADR address	RCVSAD
SAPReceiverSADRClient	Receiver SADR client	RCVSMN
SAPFlagForInternationalReceiverAddress	Receiver SADR flag	RCVSNA
SAPReceiverCommunicationType	Receiver SADR communication type	RCVSCA
SAPDefaultFlagForReceiverAddress	Receiver SADR default flag	RCVSDF
SAPReceiverAddressSequentialNumber	Receiver SADR sequential number	RCVSLF
SAPReceiverLogicalAddress	Receiver logical address	RCVLAD

Table 164. HCL Workload Automation fields used to define correlation rules for IDoc events (continued)

Composer property	Console property	IDoc field
SAPEDIStandard	EDI Standard	STD
SAPEDIStandardVersion	EDI standard version	STDVRS
SAPEDIMessageType	EDI message type	STDMES
SAPSenderAddress	Sender SADR address	SNDSAD
SAPSenderSADRClient	Sender SADR client	SNDSMN
SAPFlagForInternationalSenderAddress	Sender SADR flag	SNDSNA
SAPSenderCommunicationType	Sender SADR communication type	SNDSCA
SAPDefaultFlagForSenderAddress	Sender SADR default flag	SNDSDF
SAPSenderAddressSequentialNumber	Sender SADR sequential number	SNDSLF
SAPSenderLogicalAddress	Sender logical address	SNDLAD
SAPReferenceToInterchangeFile	Interchange file reference	REFINT
SAPReferenceToMessageGroup	Message group reference	REFGRP
SAPReferenceToMessage	Message reference	REFMES
SAPEDIArchiveKey	EDI archive key	ARCKEY
SAPIDocCreationDate	IDoc creation date	CREDAT
SAPIDocCreationTime	IDoc creation time	CRETIM
SAPExtension	Extension	CIMTYP
SAPEDIALESerializationField	EDI/ALE Serialization field	SERIAL
SAPOverridingInInboundProcessing	Overriding in inbound processing	EXPRSS
SAPIDocChangeDate	IDoc last update date	UPDDAT
SAPIDocChangeTime	IDoc last update time	UPDTIM

Based on the defined rule, the `r3evmon` process of HCL Workload Automation monitors the events related to IDoc records according to a polling rate. To customize this polling rate, use the **evmon_interval** option; for details, see Defining the common options on page 1154.

Table 165: Parameters of IDOCEventGenerated event type on page 1275 lists the values that you can specify as attribute filter name when defining the event condition.

Table 165. Parameters of IDOCEventGenerated event type

Property name	Description	Type	Filtering allowed	Required	Multiple values allowed	Wildcard allowed	Length (min-max)	
SAPClient	SAP client number	numeric (0-9)	✓	✓		✓	1	3
SAPIDocStatus	IDoc status information For a list of allowed values, refer to Table 166: Standard outbound IDoc statuses on page 1276 and Table 167: Standard inbound IDoc statuses on page 1278.	numeric	✓	✓	✓		1	2
SAPDirectionIDocTransmission	IDoc direction	numeric Value can be 1 (outbound) or 2 (inbound).	✓	✓			1	1
SAPReceiverPort	Receiver port. SAP system, EDI subsystem	string	✓				1	10
SAPReceiverPartnerFunction	Partner function of receiver	string	✓				1	2
SAPReceiverPartnerType	Partner type of receiver	string	✓				1	2
SAPReceiverPartnerNumber	Partner number of receiver	string	✓				1	10
SAPSenderPort	Sender port. SAP system, EDI subsystem	string	✓				1	10
SAPSenderPartnerType	Partner type of sender	string	✓				1	2
SAPSenderPartnerFunction	Partner function of sender	string	✓				1	2
SAPSenderPartnerNumber	Partner number of sender	string	✓				1	10
SAPLogicalMessageType	Logical message type	string	✓			✓	1	30
SAPNameOfBasicType	Name of basic type	string	✓			✓	1	30
SAPLogicalMessageCode	Logical message code	string	✓				1	3
SAPLogicalMessageFunction	Logical message function	string	✓				1	3
SAPTestFlag	Test flag	string	✓				1	1
SAPOutputMode	Output Mode	string	✓				1	1

Table 165. Parameters of IDOCEventGenerated event type

(continued)

Property name	Description	Type	Filtering allowed	Required	Multiple values allowed	Wildcard allowed	Length (min-max)	
		Value can be 2 (immediate sending) or 4 (collected sending).						

Table 166: Standard outbound IDoc statuses on page 1276 lists the standard outbound IDoc statuses and Table 167: Standard inbound IDoc statuses on page 1278 lists the standard inbound IDoc statuses. Optionally, you can activate a check to prevent event rule definitions with inconsistent IDoc status list and direction. If you activate the check and specify inconsistent values when defining a rule (for example, 02 as status and 2 as direction), you receive an error message and you cannot save the rule definition. To activate the check, perform the following steps:

1. In the *TWA_home*`>\eventPlugIn` directory on Windows™ and in the *TWA_DATA_DIR*`/eventPlugIndirectory` on UNIX®, create the `SapMonitorPlugIn.properties` file.
2. Edit `SapMonitorPlugIn.properties` to set the following configuration property:

   ```
   TWSPlugIn.event.idoc.consistency.check = true
   ```

3. From conman, stop and restart the event processing server by using, respectively, the **stopeventprocessor** and **starteventprocessor** commands.

The default value is `false`.

To have predictable event action results, when defining event rules consider using only non-transitory statuses that allow user checks.

Table 166. Standard outbound IDoc statuses

Status	Description
01	IDoc generated
02	Error passing data to port
03	Data passed to port
04	Error within control information of EDI subsystem
05	Error during translation
06	Translation
07	Error during syntax check

Table 166. Standard outbound IDoc statuses (continued)

Status	Description
08	Syntax check
09	Error during interchange
10	Interchange handling
11	Error during dispatch
12	Dispatch OK
13	Retransmission OK
14	Interchange acknowledgement positive
15	Interchange acknowledgement negative
16	Functional acknowledgement positive
17	Functional acknowledgement negative
18	Triggering EDI subsystem OK
19	Data transfer for test OK
20	Error triggering EDI subsystem
22	Dispatch OK, acknowledgement still due
23	Error during retransmission
24	Control information of EDI subsystem OK
25	Processing despite syntax error
26	Error during syntax check of IDoc
27	Error in dispatch level (ALE service)
29	Error in ALE service
30	IDoc ready for dispatch (ALE service)
31	Error no further processing
32	IDoc was edited
33	Original of an IDoc which was edited
34	Error in control record of IDoc
36	Electronic signature not performed (timeout)
37	IDoc added incorrectly
38	IDoc archived

Table 166. Standard outbound IDoc statuses (continued)

Status	Description
39	IDoc is in the target system (ALE service)
40	Application document not created in target system
41	Application document created in target system
42	IDoc was created by test transaction

Table 167. Standard inbound IDoc statuses

Status	Description
50	IDoc added
51	Application document not posted
52	Application document not fully posted
53	Application document posted
54	Error during formal application check
55	Formal application check OK
56	IDoc with errors added
57	Error during application check
58	IDoc copy from R/2 connection
60	Error during syntax check of IDoc
61	Processing despite syntax error
62	IDoc passed to application
63	Error passing IDoc to application
64	IDoc ready to be transferred to application
65	Error in ALE service
66	IDoc is waiting for predecessor IDoc (serialization)
68	Error - no further processing
69	IDoc was edited
70	Original of an IDoc which was edited
71	IDoc reloaded from archive
73	IDoc archived

Table 167. Standard inbound IDoc statuses (continued)

Status	Description
74	IDoc was created by test transaction

For example, you define a rule with the following attributes:

Workstation

A dynamic agent named SAPCPU

SAP client number

001

IDoc status list

56,60

IDoc direction

2 (inbound)

After saving the rule according to these settings, when the rule becomes active a file named `SAPCPU_r3evmon.cfg` is created on `SAPCPU`. It contains the following `!IDOC` keyword:

```
!IDOC 0003001000556,600001200000000000000000000000000000000000000000000000000000000000000000
```

IDoc monitoring is automatically started. When the event condition is verified, the action defined in the rule is triggered

For an explanation of the `!IDOC` keyword format, refer to Table 174: Miscellaneous troubleshooting items on page 1297.

Examples of event rules based on IDocs

The following example applies to the scenario described in Business scenario on page 1271. It shows an event rule that triggers an import ABAP report when an IDoc is added with a message type corresponding to emergency orders.

```
<?xml version="1.0" encoding="UTF-8"?>
<eventRuleSet xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
              xmlns="http://www.abc.com/xmlns/prod/tws/1.0/event-management/rules"
              xsi:schemaLocation="http://www.abc.com/xmlns/prod/tws/1.0/
                  event-management/rules EventRules.xsd">
       <eventRule name="scenario1_IDoc" ruleType="filter" isDraft="no">
         <eventCondition name="IDocEventRaised1" eventProvider="SapMonitor"
                         eventType="IDocEventGenerated">
            <scope>
                  001 ON SAPCU WITH 2
            <scope>
            <filteringPredicate> <attributeFilter name="Workstation" operator="eq">
                  <value>SAPCPU</value>
               </attributeFilter>
               <attributeFilter name="SAPClient" operator="eq">
                   <value>001</value>
               </attributeFilter>
               <attributeFilter name="SAPIDocStatus" operator="eq">
                   <value>50</value>
               </attributeFilter>
```

```
                <attributeFilter name="SAPDirectionIDocTransmission" operator="eq">
                    <value>2</value>
                </attributeFilter>
                <attributeFilter name="SAPLogicalMessageType" operator="eq">
                    <value>EORD1</value>
                </attributeFilter>
            </filteringPredicate>
          </eventCondition>
          <action actionProvider="TWSaction" actionType="sbj"
                  responseType="onDetection">
            <description>Trigger immediate report for high priority orders
            </description>
            <parameter name="JobDefinitionWorkstationName">
              <value>MASTER84</value>
            </parameter>
            <parameter name="JobDefinitionName">
              <value>triggerimport</value>
            </parameter>
          </action>
        </eventRule>
</eventRuleSet>
```

The following example shows an event rule defined to create a ticket for failing IDocs in the SAP Solution Manager or any other problem management system: when an IDoc with a syntax error is detected, the engine submits a job to create a ticket for the failing IDoc.

```
<?xml version="1.0" encoding="UTF-8"?>
<eventRuleSet xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
              xmlns="http://www.abc.com/xmlns/prod/tws/1.0/event-management/rules"
              xsi:schemaLocation="http://www.abc.com/xmlns/prod/tws/1.0/
                  event-management/rules EventRules.xsd">
        <eventRule name="scenario1_IDoc" ruleType="filter" isDraft="no">
          <eventCondition name="IDocEventRaised1" eventProvider="SapMonitor"
                          eventType="IDocEventGenerated">
             <filteringPredicate> attributeFilter name="Workstation" operator="eq">
                    <value>SAPCPU</value>
                </attributeFilter>
                <attributeFilter name="SAPClient" operator="eq">
                    <value>001</value>
                </attributeFilter>
                <attributeFilter name="SAPIDocStatus" operator="eq">
                    <value>60</value>
                <attributeFilter name="SAPDirectionIDocTransmission" operator="eq">
                    <value>2</value>
                </attributeFilter>
                </attributeFilter>
                <attributeFilter name="SAPLogicalMessageType" operator="eq">
                    <value>MYORD1</value>
                </attributeFilter>
            </filteringPredicate>
          </eventCondition>
          <action actionProvider="TWSaction" actionType="sbj"
                  responseType="onDetection">
            <description>Create a ticket for failing IDocs
            </description>
            <parameter name="JobDefinitionWorkstationName">
              <value>MASTER84</value>
            </parameter>
```

```
            <parameter name="JobDefinitionName">
              <value>createticket</value>
            </parameter>
          </action>
        </eventRule></eventRuleSet>
```

Defining event rules based on CCMS Monitoring Architecture alerts

Use CCMS functions to check the performance of the various SAP system components, diagnose potential problems, and be alerted about error and warning conditions.

The SAP Computing Center Monitoring System (CCMS) is a centralized monitoring architecture that provides a set of monitors for monitoring the SAP environment. Using the CCMS functions you can check the performance of the various SAP system components, diagnose potential problems, and be alerted about error and warning conditions. The monitors provide you with the information you require to fine tune the SAP system and the operating modes, and hence optimize system performance.

With HCL Workload Automation, you can integrate the CCMS monitoring functions into your management infrastructure by defining event rules based on the alerts raised in the SAP system.

Business scenarios

The following sections describe:

- Business scenario: defining an event rule to process alerts related to IDocs on page 1281
- Business scenario: defining an event rule to process alerts related to operating system on page 1281

Business scenario: defining an event rule to process alerts related to IDocs

You connected your Internet sales application to your SAP Customer Relationship Management (CRM) system, which receives the orders as incoming IDocs. You want to import the orders into the CRM system when their number exceeds a specified threshold, therefore you configured your SAP CCMS monitoring architecture to generate an alert when the number of incoming IDocs exceeds a certain value. To automatically start a task that imports the orders:

1. In your SAP CCMS monitoring architecture, identify the element related to the alert that you configured for the incoming order IDocs.
2. In HCL Workload Automation, define an event rule, to be active during the timeframe when inbound order traffic is heavy, which monitors the element identified in step 1. As soon as an alert is generated for the element, a CCMS event is sent to HCL Workload Automation.
3. In HCL Workload Automation, define a job to be submitted when the CCMS event is received, to run an SAP job that runs an import ABAP report for the order IDocs.

Business scenario: defining an event rule to process alerts related to operating system

As an HCL Workload Automation administrator, you are in charge of taking the appropriate action in the HCL Workload Automation plan when a critical situation occurs in the SAP system. You have an SAP extended agent workstation dedicated

to submit Business Intelligence tasks, whose activity you want to suspend every time the SAP BI system faces a critical situation (for example, the SAP system is running out of space). To do this:

1. In your SAP CCMS monitoring architecture, identify the element related to the SAP system you want to monitor.
2. In HCL Workload Automation, define an event rule that monitors the element and sends an event to HCL Workload Automation when an alert is generated for it. Associate with this event an action that sets the limit of the agent workstation to 0, and sends a mail to the SAP administrator to notify the details of the critical situation.
3. As soon as the SAP administrator resolves the problem, you set the limit of the agent workstation back to its original value to resume the scheduling activities.

Creating event rules based on CCMS alerts

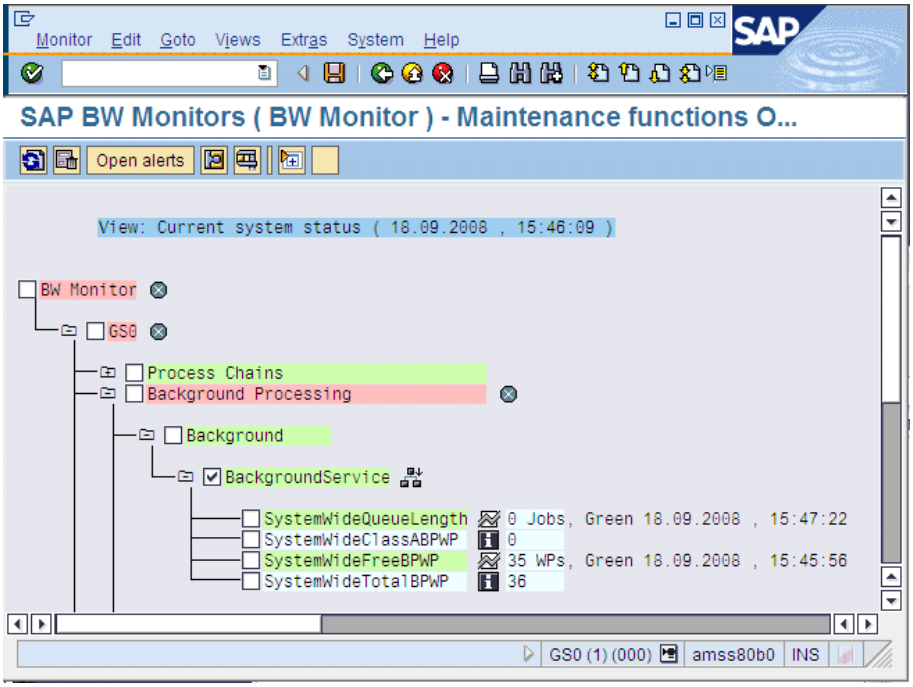
About this task

SAP systems are shipped with a predefined set of monitors, grouped in *monitor sets*. A monitor set contains a list of monitors, each monitor contains a set of *monitoring trees*. A monitor is a set of *monitoring tree elements* (MTEs) that are arranged in a hierarchical structure, named *alert monitoring tree*. You can define event rules based on the alert generated for a specific MTE.

Note: To be able to define and monitor event rules, ensure that you configured your environment as described in Configuring SAP event monitoring on page 1169.

Figure 43: A monitor and its MTEs - © SAP AG 2009. All rights reserved. on page 1282 shows the monitor named **BW Monitor** (belonging to the monitor set **SAP BW Monitor**) and its associated monitor tree elements (MTEs).

Figure 43. A monitor and its MTEs - © SAP AG 2009. All rights reserved.



To configure how HCL Workload Automation retrieves the CCMS alerts, set **ccms_alert_history** in the options file. For details about this option, refer to Defining the common options on page 1154.

To create event rules, you can use either of the following:

The composer command line

You edit the rules with an XML editor of your choice. For a general explanation about how to use the composer to define event rules, see the *User's Guide and Reference*.

The Dynamic Workload Console

For information about creating an event rule, see the section about creating an event rule in *Dynamic Workload Console User's Guide*.

To define the CCMS event for your rule, specify the following information. For more details about how you separate the MTE name into the individual HCL Workload Automation fields, see Mapping between the MTE name and HCL Workload Automation fields on page 1285.

Extended or dynamic agent workstation

The name of the extended agent workstation or the name of the dynamic agent workstation running event monitoring.

Note:

1. If you specify a pattern with the wildcard asterisk (*), all the agents whose name matches the pattern will monitor the specified event.
2. As a best practice, define that an event belonging to an SAP system is monitored by one agent workstation only. If the same SAP event is monitored by more than one agent, you might either be notified multiple times for the same event occurrence or the first agent that notifies the event occurrence makes that event unavailable to the other agents.
3. If you modify the extended agent configuration in the `r3batch` option files, to make the changes effective you must stop and restart the agent.

4. For dynamic agents you can specify the name of a local options file. In the Properties section of the Create Event Rules window of the Dynamic Workload Console a lookup button provides a list of all the local options files associated with that agent. If you do not specify the name of a local options file, the global options file is used by default in the rule definition.

MTE SAP System ID

Name of the SAP system where the MTE is located (for example, `GS0` in Figure 43: A monitor and its MTEs - © SAP AG 2009. All rights reserved. on page 1282). This field is required. Wildcards are not allowed, you can specify up to eight characters.

MTE Monitoring Context Name

Name of the monitoring context to which the MTE belongs. This field is required. A monitoring context is a logically connected group of monitoring objects that are ordered together under one summary in the monitoring tree (for example, `Background` in Figure 43: A monitor and its MTEs - © SAP AG 2009. All rights reserved. on page 1282).

Wildcards are not allowed, you can specify up to 40 characters.

MTE Monitoring Object Name

Name of the monitoring object in the alert monitor. This field is required. A monitoring object is a component or property of the system that is to be monitored (for example, `BackgroundService` in Figure 43: A monitor and its MTEs - © SAP AG 2009. All rights reserved. on page 1282). If you choose not to specify a value, you must leave the value NULL, which is the default.

Wildcards are not allowed, you can specify up to 40 characters.

MTE Monitoring Attribute Name

Name of the monitoring attribute in the alert monitor. In the monitoring tree, a monitoring attribute is always an end node in the hierarchy (for example, `SystemWideFreeBPWP` in Figure 43: A monitor and its MTEs - © SAP AG 2009. All rights reserved. on page 1282). This field is required. If you choose not to specify a value, you must leave the value NULL, which is the default.

Wildcards are not allowed, you can specify up to 40 characters.

Alert Value

Numeric value that indicates the color of the alert generated for the MTE. This field is optional. You can specify one or a combination of the following values:

1

Green, meaning Everything OK.

2

Yellow, meaning Warning.

3

Red, meaning Problem or error.

If you do not specify any value, all the alerts generated for the MTE are considered.

Alert Severity

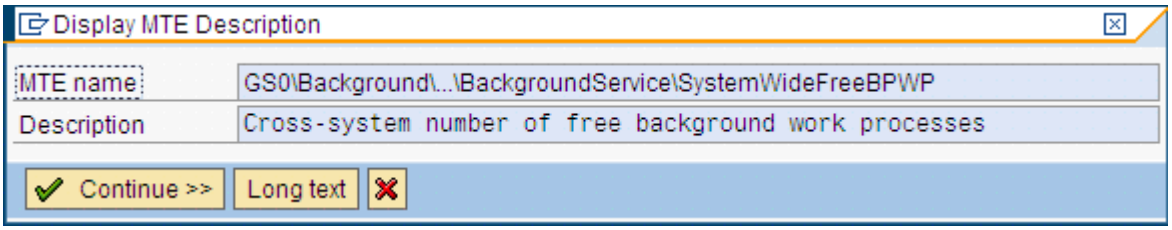
Severity of the alert. It can be a number between 0 (lowest) and 255 (highest), or a range among these values. This field is optional. Alert severity is assigned during alert configuration; the SAP standard configuration is 50.

Mapping between the MTE name and HCL Workload Automation fields

About this task

Within SAP, MTEs are identified by a name made up of several tokens, separated by backslashes (\). To display the complete MTE name, select the MTE and click **Properties** or press **F1**:

Figure 44. Name and description of an MTE - © SAP AG 2009. All rights reserved.



According to the type of MTE that you want to monitor, you must fill in each HCL Workload Automation field with a specific token of the MTE name (to know your MTE type, select the MTE and click **Legend**):

If you are using Dynamic Workload Console V8.5.1, or later

1. In the **Event Rule Editor** panel, from the **Properties** section, click **Autofill MTE Tokens**. The **MTE Name** window opens.
2. In the **MTE Name** field, write the name of the MTE to monitor and click **OK**. You are returned to the **Event Rule Editor** panel, where the HCL Workload Automation fields are filled in accordingly.

If you are using Dynamic Workload Console prior to V8.5.1

Refer to the instructions provided in the following sections:

- Context MTE on page 1285
- Object MTE on page 1287
- Attribute MTE on page 1287

Note: Virtual MTEs cannot be monitored.

Context MTE

A context MTE is the uppermost node of a monitoring tree; it contains all the associated object MTEs and attribute MTEs. Context nodes can be either of the following types:

Root

Belongs only to the All Monitoring Contexts monitor. According to the SAP version you are using, a root context MTE name can have either of the following formats:

```
tokenA\tokenB\...
- OR -
tokenA\tokenB
```

For example:

```
T10\SystemConfiguration\...
```

Refer to Table 168: Mapping between root context MTE name and HCL Workload Automation fields on page 1286 for an explanation about how you report this type of MTE in the HCL Workload Automation fields:

Table 168. Mapping between root context MTE name and HCL Workload Automation fields

HCL Workload Automation field	Token of MTE name	In this example...
MTE SAP System ID	*tokenA*	`T10`
MTE Monitoring Context Name	*tokenB*	`SystemConfiguration`
MTE Monitoring Object Name	N/A	`NULL`
MTE Monitoring Attribute Name	N/A	`NULL`

Summary

According to the SAP version you are using, a summary context MTE name can have either of the following formats:

```
tokenA\tokenB\...\tokenC\...
- OR -
tokenA\tokenB\tokenC
```

For example:

```
T10\SystemConfiguration\...\InstalledSupportPackages\...
```

Refer to Table 169: Mapping between summary context MTE name and HCL Workload Automation fields on page 1286 for an explanation about how you report this type of MTE in the HCL Workload Automation fields:

Table 169. Mapping between summary context MTE name and HCL Workload Automation fields

HCL Workload Automation field	Token of MTE name	In this example...
MTE SAP System ID	*tokenA*	`T10`
MTE Monitoring Context Name	*tokenB*	`SystemConfiguration`

Table 169. Mapping between summary context MTE name and HCL Workload Automation fields (continued)

HCL Workload Automation field	Token of MTE name	In this example...
MTE Monitoring Object Name	*tokenC*	`InstalledSupportPackages`
MTE Monitoring Attribute Name	N/A	`NULL`

Object MTE

According to the SAP version you are using, an object MTE name can have either of the following formats:

```
tokenA\tokenB\tokenC\tokenD
- OR -
tokenA\tokenB\...\tokenD
```

For example:

```
PR0\amsp53_PR0_11\R3Services\Background\
```

Refer to Table 170: Mapping between object MTE name and HCL Workload Automation fields on page 1287 for an explanation about how you report this type of MTE in the HCL Workload Automation fields:

Table 170. Mapping between object MTE name and HCL Workload Automation fields

HCL Workload Automation field	Token of MTE name	In this example...
MTE SAP System ID	*tokenA*	`PR0`
MTE Monitoring Context Name	*tokenB*	`amsp53_PR0_11`
MTE Monitoring Object Name	*tokenD*	`Background`
MTE Monitoring Attribute Name	N/A	`NULL`

Attribute MTE

According to the SAP version you are using, an attribute MTE name can have either of the following formats:

```
tokenA\tokenB\tokenC\tokenD\tokenE
- OR -
tokenA\tokenB\...\tokenD\tokenE
```

For example:

```
PR0\amsp53_PR0_11\R3Services\Background\AbortedJobs
```

Refer to Table 171: Mapping between attribute MTE name and HCL Workload Automation fields on page 1288 for an explanation about how you report this type of MTE in the HCL Workload Automation fields:

Table 171. Mapping between attribute MTE name and HCL Workload Automation fields

HCL Workload Automation field	Token of MTE name	In this example...
MTE SAP System ID	*tokenA*	`PR0`
MTE Monitoring Context Name	*tokenB*	`amsp53_PR0_11`
MTE Monitoring Object Name	*tokenD*	`Background`
MTE Monitoring Attribute Name	*tokenE*	`AbortedJobs`

Setting correlation rules and action parameters

Optionally, you can use the alert properties listed in Table 172: Alert properties for correlations on page 1288 to:

- Define correlation rules between CCMS events.
- Specify additional parameters for the action that is associated with the event rule.

Date and time values are specified in GMT time zone.

Table 172. Alert properties for correlations

Table 172. Alert properties for correlations (continued)

CCMS alert property	Console property	Composer property
OBJECTNAME for event definition	MTE Monitoring Object Name	InputMonObjectName
FIELDNAME for event definition	MTE Monitoring Attribute Name	InputMonFieldName
ALERTDATE	Alert Date	AlertDate
ALERTTIME	Alert Time	AlertTime
ALINDEX	Alert Index	AlertIndex
ALSYSID	Alert SAP System ID	AlertSAPSystemID
ALUNIQNUM	Alert UID	AlertUID
EXTINDEX	Alert MT Ext Index	AlertMTExtIndex
FIELDNAME	Alert Monitoring Attribute Name	AlertMonFieldName
MANDT	Client	Client
MSCGLID	Alert Msg Container ID	AlertMsgContainerID
MSEGNAME	Alert MTE Segment	AlertMTESegment
MSG	Alert Message	AlertMessage
MSGCLASS	XMI Ext Company Name	XMIExtCompanyName
MSGID	XMI Log Msg ID	XMILogMsgID
MTCLASS	Alert MT Class	AlertMTClass
MTINDEX	Alert MT Index	AlertMTIndex
MTMCNAME	Alert Monitoring Context Name	AlertMTEContext
MTNUMRANGE	Alert MTE Range	AlertMTERange
MTSYSID	Alert MTE System	AlertMTESys
MTUID	Alert MT Type ID	AlertMTTypeID
OBJECTNAME	Alert Monitoring Object Name	AlertMonObjName
RC	Alert Return Code	AlertReturnCode
REPORTEDBY	Alert Reported By	AlertReportedBy
SEVERITY	Alert Severity	AlertSeverity
STATCHGBY	Alert Changed By	AlertChangedBy
STATCHGDAT	Alert Change Date	AlertChangeDate
STATCHGTIM	Alert Change Time	AlertChangeTime

Table 172. Alert properties for correlations (continued)

CCMS alert property	Console property	Composer property
STATUS	Alert Status	AlertStatus
USERID	User ID	UserID
VALUE	Alert Value	AlertValue

Getting alert status and committing alerts by an external task

Learn how to get CCMS alert status and commit CCMS alerts.

Refer to the following sections for details about:

- Getting CCMS alert status on page 1290
- Committing CCMS alerts on page 1291

Getting CCMS alert status

About this task

To get the current status of a CCMS alert from HCL Workload Automation, use the external task Get Information (GI). To replace the command arguments with the actual values, refer to the output returned by the event rule you defined. For details about the correspondence between the CCMS properties and the Console and composer properties, see Table 172: Alert properties for correlations on page 1288.

From a command line, enter the following command:



-c *Agent_name*

The name of the dynamic agent workstation, or the unique identifier for the extended agent workstation connected to the SAP system where the MTE for which the alert was raised is located. For information about retrieving the unique identifier for the extended agent workstation, see UNIQUE_ID on page 1085.

-t GAS

Identifier of the task to be performed, in this case GAS (Get Alert Status).

-alsysid *sap_system_ID*

Identifier of the SAP system where the MTE for which the alert was raised is located. If the name contains blanks, enclose it between single quotes.

-msegname *alert_monitoring_segment*

Name of the alert monitoring segment. You can specify from 1 to 40 characters.

-aluniqnum *alert_UID*

Unique identifier of the alert, made up of 10 characters.

-alindex *alert_index*

Alert index, made up of 10 characters.

-alertdate *alert_date*

Date of the alert, in the format *yyyymmdd*.

-alerttime *alert_time*

Time of the alert, in the format *hhmmss*.

The following is an example of how to retrieve the current status of a CCMS alert:

```
r3batch -t GI -c horse10 -- " -t GAS -alsysid T10
        -msegname SAP_CCMS_horse10_T10_00 -aluniqnum 0017780869
        -alindex 0000000104 -alertdate 20081007 -alerttime 040356"
```

You are returned the current status of the alert.

Committing CCMS alerts

About this task

The CCMS alerts that you defined as HCL Workload Automation events are not automatically committed after their processing. To commit an alert that was processed by HCL Workload Automation, use the external task Put Information (PI).

To replace the command arguments with the actual values, refer to the output returned by the event rule you defined. For details about the correspondence between the CCMS properties and the Console and composer properties, see Table 172: Alert properties for correlations on page 1288.

From a command line, enter the following command:

Command syntax

```
-r3batch -t PI -c XA_Unique_ID -- " -t CA -alsysidsap_system_ID -msgnamealert_mte_segment -
aluniqnumalert_UID -alindexalert_index -alertdatealert_date -alerttimealert_time "
```

Where:

-t PI

Identifier of the task to be performed, in this case PI (Put Information).

-c *Agent_name*

The name of the dynamic agent workstation or the unique identifier for the extended agent workstation connected to the SAP system where the MTE for which the alert was raised is located. For information about retrieving the unique identifier for the extended agent workstation, see UNIQUE_ID on page 1085.

-t CA

Identifier of the task to be performed, in this case CA (Commit Alert).

-alsysid *sap_system_ID*

Identifier of the SAP system where the MTE for which the alert was raised is located. If the name contains blanks, enclose it between single quotes.

-msegname *alert_monitoring_segment*

Name of the alert monitoring segment. You can specify from 1 to 40 characters.

-aluniqnum *alert_UID*

Unique identifier of the alert, made up of 10 characters.

-alindex *alert_index*

Alert index, made up of 10 characters.

-alertdate *alert_date*

Date of the alert, in the format *yyyymmdd*.

-alerttime *alert_time*

Time of the alert, in the format *hhmmss*.

The following is an example of how to commit a CCMS alert:

```
r3batch -t PI -c horse10 -- " -t CA -alsysid T10
        -msegname SAP_CCMS_horse10_T10_00 -aluniqnum 0017780869
        -alindex 0000000104 -alertdate 20081007 -alerttime 040356"
```

You are returned with the message `The CCMS alert was successfully confirmed`.

Example of an event rule based on CCMS alerts

The following example shows an event rule defined to monitor the yellow alerts raised on the MTE named `GS0\ALE/EDI GS0(000) Log.sys TVALE\Inbound IDoc ORDER_IDOC\Inbound: IDoc generated`. The MTE is configured to generate a yellow alert when the number of IDocs representing orders ready to process exceeds a specified threshold. If this condition occurs, the following actions are triggered:

- An HCL Workload Automation job is submitted to process the order IDocs.
- An HCL Workload Automation job, with priority 10, is submitted to confirm the alert.

```
<?xml version="1.0" encoding="UTF-8"?>
<eventRuleSet xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
  xmlns="http://www.abc.com/xmlns/prod/tws/1.0/event-management/rules"
  xsi:schemaLocation="http://www.abc.com/xmlns/prod/tws/1.0/event-management/rules
      http://www.abc.com/xmlns/prod/tws/1.0/event-management/rules/EventRules.xsd">
     <eventRule name="SCENARIO1_XAL" ruleType="filter" isDraft="yes">
     <eventCondition name="MTEEventRaised1" eventProvider="SapMonitor"
                     eventType="CCMSEventGenerated">
         <filteringPredicate>
              <attributeFilter name="Workstation" operator="eq"
                    <value>SAP_XA</value>
              </attributeFilter>
              <attributeFilter name="InputSAPSystemID" operator="eq"
                    <value>GS0</value>
              </attributeFilter>
              <attributeFilter name="InputMonitoringContextName" operator="eq"
                    <value>ALE/EDI GS0(000) Log.sys TVALE</value>
              </attributeFilter>
              <attributeFilter name="InputMonObjectName" operator="eq"
                    <value>Inbound IDoc ORDER_IDOC</value>
              </attributeFilter>
              <attributeFilter name="InputMonFieldName" operator="eq"
                    <value>Inbound: IDoc generated</value>
              </attributeFilter>
         </filteringPredicate>
     </eventCondition>
     <action actionProvider="TWSAction" actionType="sbj" responseType="onDetection">
              <parameter name="JobUseUniqueAlias">
                    <value>false</value>
              </parameter>
              <parameter name="JobDefinitionWorkstationName">
                    <value>SAP_XA/value>
              </parameter>
              <parameter name="JobAlias">
                    <value>IDOC_%{MTEEventRaised1.AlertUID}/value>
              </parameter>
              <parameter name="JobDefinitionName">
                    <value>PROCESS_ORDER/value>
              </parameter>
```

```
     </action>
     <action actionProvider="TWSAction" actionType="sbd" responseType="onDetection">
              <parameter name="JobUseUniqueAlias">
                    <value>false/value>
              </parameter>
              <parameter name="JobWorkstationName">
                    <value>TWS_HOST_FTA/value>
              </parameter>
              <parameter name="JobTask">
                    <value>C:\TWA_home\methods\r3batch -t PI
                           -c %{MTEEventRaised1.Workstation} -- "
                           -t CA -ALSYSID %{MTEEventRaised1.AlertSAPSystemID}
                           -MSEGNAME %{MTEEventRaised1.AlertMTESegment}
                           -ALUNIQNUM %{MTEEventRaised1.AlertUID}
                           -ALINDEX %{MTEEventRaised1.AlertIndex}
                           -ALERTDATE %{MTEEventRaised1.AlertDate}
                           -ALERTTIME %{MTEEventRaised1.AlertTime} "
                    </value>
               </parameter>
              <parameter name="JobPriority">
                    <value>10</value>
              </parameter>
              <parameter name="JobType">
                    <value>Command</value>
              </parameter>
              <parameter name="JobAlias">
                    <value>CONFIRM_%{MTEEventRaised1.AlertUID}</value>
              </parameter>
              <parameter name="JobStreamName">
                    <value>CONFIRM_STREAM</value>
              </parameter>
              <parameter name="JobLogin">
                    <value>twsuser</value>
              </parameter>
     </action>
     </eventRule>
</eventRuleSet>
```

National Language support

The National Language support feature allows you to install SAP on a localized HCL Workload Automation workstation and use localized characters for HCL Workload Automation job names, job streams, and SAP variants.

Using the local and global configuration files, you can set up SAP to use different code pages and languages for both its output and its connection with a remote SAP system.

As described in Unicode support on page 1143, this version of Access method for SAP features Unicode, which is widely supported by SAP systems since version 4.7. However, if either the workstation running SAP or the target SAP systems do not support Unicode, this section describes how you configure code pages and national languages for SAP.

Setting National Language support options

The following options control the code page and national language used by Access method for SAP, when Unicode support is not used:

TWSXA_CP

The code page used to establish the connection between `r3batch` and the target SAP system.

If you are running a non-Unicode version of `r3batch`, set this option to the code page installed on the SAP system (for a list of the valid code pages, refer to SAP supported code pages on page 1296). The default value is the SAP code page 1100, similar to the standard ISO8859-1. In all other cases, this option is ignored.

TWSXA_LANG

The language that `r3batch` uses to log in. It can be one of the following (DE, EN, and JA can be set from the Option Editor. The other languages can be set using any text editor):

- Brazilian Portuguese (pt_BR)
- English (EN, the default value)
- French (FR)
- German (DE)
- Italian (IT)
- Japanese (JA)
- Korean (KO)
- Simplified Chinese (zh_CN)
- Spanish (ES)
- Traditional Chinese (zh_TW)



TWSMETH_CP

The code page that `r3batch` uses for its output. The default is the code page used by the HCL Workload Automation workstation that hosts `r3batch`.

Ensure that the `TWSMETH_CP` and `TWSMETH_LANG` options are consistent.

TWSMETH_LANG

The catalog language used by `r3batch`. The default is the language used by the HCL Workload Automation workstation that hosts `r3batch`.

Ensure that the `TWSMETH_CP` and `TWSMETH_LANG` options are consistent.

SAP supported code pages

To communicate with SAP systems, Access method for SAP uses the following code pages. Use these values to set option `TWSXA_CP`, only when `r3batch` does not support Unicode.

Table 173. SAP supported code pages

SAP code pages	Description
1100	8859-1, this is the default value
1103	MS 850
8000	SJIS: Shift JIS
8300	BIG5: Traditional Chinese
8400	GBK: Simplified Chinese

Troubleshooting

Learn what to do if you get any problems while installing or using HCL Workload Automation access methods or plug-ins.

Troubleshooting the SAP connection

If you are unable to submit SAP jobs using HCL Workload Automation after the R/3 configuration, perform the following tests:

- Ensure that you can ping the SAP system from the HCL Workload Automation system. This shows basic network connectivity.
- Note that using the SAP routers to access the R/3 system could exceed the size of internal buffers of the RFC library used to store the hostname of the SAP system. When this occurs, the hostname gets truncated, causing the connection to the R/3 system to fail. To work around this problem, do not fully qualify the name of the SAP routers or alternatively use the IP addresses.
- Run the following **telnet** command to verify connectivity:

```
telnet systemname 33xx
```

where *systemname* is the system name or IP address of the SAP server and *xx* is the SAP instance.

If the command fails to complete, this means that communication between `r3batch` and the SAP application server is down.

- Log on to the SAP system as an administrator and verify that the HCL Workload Automation RFC user (created in the Creating the HCL Workload Automation RFC user on page 1132) exists.

- If the SAP gateway truncates the connection string, replace the host name with the IP address.
- If `r3batch` runs on an AIX® system that does not use U.S. English, make sure that the U.S. Language Environment® is installed on both the HCL Workload Automation workstation and the SAP database workstation. Otherwise the error `BAD TEXTENV` (or a similar error message) might appear in the `dev_rfc` trace file and connections to SAP fail.

Other known problems

Table 174: Miscellaneous troubleshooting items on page 1297 lists miscellaneous troubleshooting problems.

Table 174. Miscellaneous troubleshooting items

Area	Item
`r3batch` and `r3event`: output contains unreadable characters	**Symptom**: When you enter the `r3batch` and `r3event` commands interactively (for example, to export an SAP calendar) the output is returned in UTF-8 format.
	Solution: To resolve this problem, you can either use a shell that supports the UTF-8 code page or redirect the output to a file and open it with a text editor that supports the UTF-8 format.
`r3batch:` SAP jobs contain quotation marks (") or reverse quotes (`)	**Symptoms**: SAP jobs whose names contain quotation marks or reverse quotes are not displayed in the pick list of the Dynamic Workload Console. -OR- You have an HCL Workload Automation job that tries to submit an SAP job whose name contains quotation marks, but it abends with an error. The following message might be displayed:

Table 174. Miscellaneous troubleshooting items

(continued)

Area	Item
	`EEWO0439E The required options are not specified either in the global or in the local options file.`
	Solution: In your SAP system, make a copy of the SAP job and assign it a name that does not contain quotation marks or reverse quotes.
`r3batch:` SAP job containing Arabic characters.	**Symptom**: An SAP job abends when the job contains Arabic characters.
	Solution: If you run an SAP job that contains Arabic characters, you must set the local codepage of the agent workstation hosting the r3batch access method to the Arabic codepage. Refer to the **twsmeth_cp** keyword in the common options file, Defining the common options on page 1154.

`r3batch:` error messages submitting a job on dynamic agents.	**Symptom**: When working with dynamic workstations and performing actions such as: displaying a process chain, restarting a process chain, or retrieving the spool list, the following messages might be displayed from the Dynamic Workload Console:
	`EEWO0439E The required options are not specified either in the global or in the local options file.`
	`EEWO1065W The environment variable UNISON_JOB is not set. The process chain cannot be restarted.`

	Solution: These messages might indicate that the requested action is not supported on dynamic workstations. Refer to the *HCL Workload Automation Release Notes*® for more information about HCL Workload Automation features and minimum required versions for compatibility.
`r3batch:` r3batch hangs when performing actions from the Dynamic Workload Console.	**Symptoms**: r3batch hangs when performing actions from the Dynamic Workload Console such selecting from a pick list, submitting a job, or similar actions that require connection to the SAP system. The HCL Workload Automation joblog might also contain multiple "Timer expired" messages.
	Solution: This problem is caused by the HCL Workload Automation logging and tracing component. Modify the `r3batch.properties` files. Locate the **r3batch.trace.handlers.traceFile.MPFileSemKeyproperties** setting, and then either comment this property setting out or use a different value. Choose any numeric value and retry the operation.
`r3batch:` Submit same process chain in parallel fails.	**Symptom**: The SAP system returns an error message RFC_ERROR_SYSTEM_FAILURE when starting an SAP process chain.
	Solution: Verify if the corrections stated in SAP note 1723482 are applied to your SAP Business Warehouse system or avoid running the same process chain more than once simultaneously.
r3batch: When you restart the process of a subchain, the status of the	**Symptom**: When you restart the process of a subchain, the status of the original process chain is not changed to active.
	Solution: Refer to SAP Note 1075876.

Table 174. Miscellaneous troubleshooting items

(continued)

Area	Item
original process chain is not changed to active	
`r3batch:` Refresh an SAP process chain after a kill action on a running job instance.	**Symptom**: If you perform a kill action on an HCL Workload Automation job instance running on a dynamic workstation which monitors an SAP process chain, and then subsequently perform a Refresh operation on this job, the job fails.
	Solution: You cannot perform a Refresh operation after having performed a kill action on an HCL Workload Automation job instance running on a dynamic workstation which monitors an SAP process chain. Verify the status of the SAP process chain on the SAP system, and then set the HCL Workload Automation job status accordingly.
`r3batch:` Wrong characters are displayed in the Criteria Manager profile description.	**Symptom**: Special characters such as, < (less than), > (greater than), or the ' (apostrophe) specified in the **Description** field of the **Create criteria profile** dialog are displayed incorrectly.
	Solution: Avoid using special characters in the **Description** field when creating a new criteria profile.
`r3evmon:` monitoring events is not started, stopped, or performed	**Symptom**: You cannot start or stop event monitoring, or event monitoring is not performed.
	Solution: Ensure that *TWSuser* is the owner of the following files, and that the user has read and write permissions: • `/TWA_DATA_DIR/pids/XAname_r3evmon.pid` • `/TWA_DATA_DIR/EIF/XAname_r3evmoncache.dat` • `/TWA_DATA_DIR/EIF/XAname_r3evmoneif.conf` • `/TWA_DATA_DIR/methods/r3evmon_cfg/XAname_r3evmon.cfg` • `/TWA_DATA_DIR/methods/r3evmon_cfg/XAname_r3idocmon.cfg` • `/TWA_DATA_DIR/methods/r3evmon_cfg/XAname_r3xalmon.cfg` • `/TWA_DATA_DIR/methods/r3evmon_cfg/XAname_r3evmon.lck` On Windows™ workstations, these files are located in the TWA_home directory and not in the *TWA_DATA_DIR* directory.
`r3batch:` monitoring SAP events is not performed	**Symptom**: The SAP event on which the event rule is based is neither monitored nor committed.
	Solution: Ensure that the extended agent workstation you specified in the SAP event definition exists. When you define an SAP event within an event rule, no check on the extended agent workstation

Table 174. Miscellaneous troubleshooting items

(continued)

Area	Item
	is made: if the workstation does not exist, the event rule is saved and activated but it will never be resolved.
`r3batch:` monitoring SAP events is not performed	**Symptom**: With XBP 3.0, the SAP event is raised but HCL Workload Automation is not notified and therefore does not act as expected.
	Solution: Ensure that the SAP event was not excluded from logging in the SAP event history table.
`r3batch:` monitoring SAP events is not performed	**Symptom**: The SAP events on which the event rule is based are not monitored nor committed.

	Solution: The SAP events being monitored are listed in the following file:
	`TWA_DATA_DIR/monconf/XAname_r3evmon.cfg`
	where `XAname` is the name of the SAP extended agent workstation.
	Check that the file is updated and contains the current monitoring plan. The SAP events are indicated by the following keyword (one for each SAP event on the same extended agent):
	`!R3EVENT SAP_event_name_lengthSAP_event_name[SAP_event_parm_lengthSAP_event_parm]`
	where:
	SAP_event_name_length
	The length of the SAP event name to monitor, in the format *nnnn*. For example, `0008`, if the event name is `SAP_TEST`.
	SAP_event_name
	The name of the SAP event to monitor.
	SAP_event_parm_length
	The length of the parameter associated with the SAP event to monitor, if any. The format is *nnnn*. For example, `0007`, if the event name is `SAP_PAR`.
	SAP_event_parm
	The parameter associated with the SAP event to monitor, if any. This value is optional, but omitting it identifies an SAP event with no parameter associated. For details about how the events are matched between `r3evmon.cfg` and the SAP system, see SAP events matching criteria on page 1267.

Table 174. Miscellaneous troubleshooting items

(continued)

Area	Item
	For each configuration file, an `r3evmon` process is started to monitor the SAP events listed. To start an `r3evmon` monitoring process for a specific extended agent workstation, enter the following command. **Note:** 1. For UNIX® only, `r3evmon` must be entered by the owner of the HCL Workload Automation installation: 2. If you run `r3evmon` from a Windows™ DOS shell, the command prompt is not returned until the process completes. `r3evmon -t SEM -c XA_Unique_ID -- "[-EIFSRV EIF_server -EIFPORT EIF_port]"` where: ***XA_Unique_ID*** The unique identifier of the extended agent workstation. For information about retrieving the unique identifier for the extended agent workstation, see UNIQUE_ID on page 1085. ***EIF_server*** The host name or IP address of the master domain manager. ***EIF_port*** The port that the master domain manager uses to receive the event notification.
`r3batch`:IDoc monitoring is not performed	**Symptom**: The events on which the event rule is based are not monitored or no event is generated during IDoc monitoring.

	Solution: The events being monitored are listed in the following file:
	
	`TWA_DATA_DIR/monconf/XAname_r3evmon.cfg`
	
	where `XAname` is the name of the SAP extended agent workstation. It is the same file that is used to monitor SAP events in general.
	
	Check that the file is updated and contains the current monitoring plan. The events corresponding to the `IDOCEventGenerated` event type are indicated by the following keyword (one for each event on the same extended agent):
	
	`!IDOC nnnn<Client Number>nnnn<IDoc Status List>nnnn<Direction>nnnn<Receiver Port>`
	`nnnn<Receiver Partner Type>nnnn<Partner Function of Receiver>`
	`nnnn<Partner Number of Receiver>nnnn<Sender Port>nnnn<Sender Partner Type>`
	`nnnn<Partner Function of Sender>nnnn<Partner Number of Sender>`

Table 174. Miscellaneous troubleshooting items

(continued)

Area	Item
	nnnn<Message Type>nnnn<IDoc Type>nnnn<Logical Message Variant> *nnnn<Logical Message Function>nnnn<Test Flag>nnnn<Output Mode>* where: ***nnnn*** The length of the IDoc field. For example, `0005` indicates the value of an IDoc status list corresponding to `56,60`. **< >** Contains the value of the field associated with the IDoc to be monitored. For a list of the supported IDoc fields, refer to Table 163: HCL Workload Automation fields used to define event rules based on IDocs on page 1272. For each configuration file, an `r3evmon` process is started to monitor the events listed. Make sure that an `r3evmon` monitoring process is started for the involved extended agent workstation.
`r3evmon:` monitoring SAP and IDoc events increases memory consumption	**Symptom**: Memory consumption increases continuously during monitoring of IDoc and standard SAP events.
	Solution: Refer to SAP Notes® 1021071 and 1109413.
`r3batch:` Duplicated events generated during IDoc monitoring	**Symptom**: The action defined in an event rule with `IDOCEventGenerated` event type is unexpectedly repeated.
	Solution: Reset the start date and time for the next monitoring loop. These values are stored in the following file: *<data_dir>*`/methods/r3evmon_cfg/`*XAname*`_r3idocmon.cfg` where *`XAname`* is the name of the SAP extended agent workstation. Therefore you can either: • Stop `r3evmon`, delete the *XAname*`_r3idocmon.cfg` file and then start `r3evmon` again. - OR - • Stop `r3evmon`, set the date and time in the *XAname*`_r3idocmon.cfg` file to the values you want, and start`r3evmon` again. Use the following format for the start date and time:

Table 174. Miscellaneous troubleshooting items

(continued)

Area	Item

	`start_date=YYYYMMDD`
	`start_time=HHMMSS`
	
	For example:
	`start_date=20080307`
	`start_time=115749`
	
	Check the value of the **idoc_no_history** option:
	• If it is set to OFF and no *XAname*`_r3idocmon.cfg` file exists, then all matching IDocs are retrieved, not only the current ones.
	• If it is set to ON (default value), check the date and time in the *XAname*`_r3idocmon.cfg` file.

`r3batch:` No event is generated during IDoc monitoring	**Symptom**: The expected event actions are not triggered.
	Solution: Check the value of the **idoc_no_history** option; if it is set to ON (default value), check the date and time in the *XAname*`_r3idocmon.cfg` file.

Error defining an internetwork dependency based on SAP event	**Symptom**: If you work with XBP 2.0, when you try to define an internetwork dependency based on an SAP event, the following error message is displayed:
	`*** ERROR 778 *** EEWO0778E An internal error has occurred. The program could not modify the following job:`
	`Job name:`
	`Job ID:`
	`%CJ ERROR`

	Solution: Perform the following steps: 1. Check if the BCTEST report is defined in your SAP system by invoking either one of the following transactions: **`sa38`** Enter BTC* and click the picklist button. In the panel that opens, click the picklist button and check if BTCTEST is shown in the list that is displayed. **`se38`**

Table 174. Miscellaneous troubleshooting items

(continued)

Area	Item
	Enter BTC* and click the picklist button. Check if BTCTEST is shown in the list that is displayed. 2. If report BTCTEST is not found in the list, you can either: ◦ Choose another existing report, and, in the local options file, set the placeholder_abap_step option to the name you chose. Because the report assigned to the placeholder job is run when the corresponding event is raised, ensure that you choose a dummy report. For details about the placeholder_abap_step option, see Table 147: r3batch common configuration options on page 1154. - OR - ◦ Set the placeholder_abap_step option to a custom developed ABAP code of your choice.

`r3batch:` error message when scheduling SAP jobs	**Symptom**: When creating an SAP job, the following message is displayed while trying to view the details of an ABAP's variant:
	`AWSO0101E`
	`Missing ABAP routine.`
	`J_101_REPORT_ALL_SELECTIONS please install the latest ABAP routine for Maestro!!`

	Solution: This defect is caused by an error in an SAP function module. SAP describes this problem and possible solutions in the SAP Notes® 0351293 and 0116354.
`r3batch:` modify job step error	You change print parameters with the `BAPI_XBP_MODIFY_JOB_STEP` function module, and subsequently, they are incorrect. As a consequence, `r3batch` gets error 221: `MSG_CANNOT_GET_PRIARC_PARAMS: "Retrieving new print and archive parameters failed"` The problem is solved by installing SAP Note 758829.
`r3batch:` modify job step error	The BAPI_XBP_MODIFY_JOB_STEP function module always uses the name of the logged-on user as the name for the step user. In this case, when submitting a job with the `-vX` options, `r3batch` creates a job by copying all the data from the original template, except the variant name of the first step (which is provided as the option parameter). This procedure is usually referred to as the "old copy". However, when adding a step to a new job, the XBP 2.0 interface ignores the user parameter passed by `r3batch`. The problem is solved by installing SAP note 758829.
`r3batch:` does not start after	**Symptom**: After installing or upgrading the SAP R/3 access method to version 8.5 on a Windows™ operating system, you try to start `r3batch` but nothing happens. The following message is displayed:

Table 174. Miscellaneous troubleshooting items

(continued)

Area	Item
installation on Windows™	`The application failed to initialize properly.` `Click on OK to terminate the application.`
	Solution: Ensure that you applied the SAP Note 684106 to install the required Microsoft™ DLLs.
`r3batch:` HCL Workload Automation environment variables are not resolved when specified in the task string for an R/3 batch job.	**Symptom**: When HCL Workload Automation environment variables are used in the task string for an R/3 batch job and the job is launched, the environment variables are not resolved. The exact string used to specify the variable is used instead.

	Solution: To leverage HCL Workload Automation environment variables, you must modify the access method as follows:
	1. In the *`TWA_DATA_DIR`*`/methods` directory, create a file named, r3batch.cmd (on Windows™) or r3batch.sh (on UNIX®) as required, containing the following content:
	`@echo off`
	`set METHODSPATH=%~dp0`
	`call "%METHODSPATH:"=%r3batch.exe" %*`
	2. Modify the CPU XAGENT definition from r3batch to r3batch.cmd. An example follows:
	`CPUNAME NW1`
	`DESCRIPTION "r3batch"`
	`OS OTHER`
	`NODE none TCPADDR 31111`
	`FOR MAESTRO HOST STROMBOLI ACCESS "r3batch.cmd"`
	`TYPE X-AGENT`
	`AUTOLINK OFF`
	`BEHINDFIREWALL OFF`
	`FULLSTATUS OFF`
	`END`
	3. To modify the CPU access method in the Symphony® file, run JnextPlan as follows:
	`JnextPlan -for 0000`

@longlink file present in installation directory	**Symptom**: After installing HCL Workload Automation on a computer with an AIX® operating system where a master domain manager is already installed, a @longlink file containing the following is present in the installation directory:
	`methods/_tools/_jvm_64/lib/desktop/icons/HighContrastInverse/48x48/mimetypes/`
	`gnome-mime-text-x-java.png`

	Solution: The file can be ignored. It does not present any problems for the proper functioning of the product.

Table 174. Miscellaneous troubleshooting items

(continued)

Area	Item
Job throttling does not start	**Symptom**: When you start the job throttling feature, nothing happens and the following error message is displayed: `EEWOJTR0207E Error, another job throttler instance is already running against the same SAP system. Foreign job throttler registration is: Client ID="clientID", Name="TWS4APPS_JOBTHROTTLER",Host="hostname", UID "UniqueID"`
	Cause and Solution: Possible causes are: • You are running job interception collector jobs, but the job interception and job throttling features cannot run at the same time. Choose which feature to start. For detailed information, refer to Job interception and parent-child features on page 1223 and Job throttling feature on page 1250. • Another job throttler instance is running against the same SAP system. You can start only one job throttler instance. • A previous job throttler instance created an exclusive lock object on the SAP system that could have become permanent. To verify it, use transaction `sm12` and query for the lock object named `TWS4APPS_JOBTHROTTLER`. If the lock object exists, and you are not running any job throttler or job interception instance, remove the lock manually and restart the job throttler.
Job throttling does not start	**Symptom**: When you start the job throttling feature, nothing happens and the following error message is displayed: `EEWOJT0209E Error, the password format is not valid.`
	Cause and Solution: Your password is encrypted in old format. To encrypt the password with the correct encryption version, use the **enigma** or **pwdcrypt** programs. For details about how to encrypt the password, see Encrypting SAP user passwords on page 1166.
Job throttling does not stop	**Symptom**: When you stop the job throttling feature, nothing happens.
	Cause and Solution: You are connected as a *TWSUser* who does not have write permission on the `XAname_jobthrottling_cmd.txt` file. To solve this problem, delete the `XAname_jobthrottling_cmd.txt` file and enter the command again. For detailed information about stopping the job throttler, refer to Step 5. Starting and stopping the job throttling feature on page 1254.
Job throttling: alerts for MTEs are not generated according to the	**Symptom**: Alerts for the MTEs created by the job throttler are generated without respecting the threshold values that are set.
	Cause and Solution: You started a new job throttler instance, which, being enabled to send data to CCMS, created the related MTEs. When you include the MTEs in your monitoring set, the threshold

Table 174. Miscellaneous troubleshooting items

(continued)

Area	Item
threshold values set	values are automatically set according to the existing MTE class. Nevertheless, alerts are generated without respecting these values. To solve this problem, edit the MTE properties and save them again, even if you do not change anything.
Job throttling: saving MTE properties generates an informational message	**Symptom**: When you edit and save the properties of MTEs generated by the job throttler, the following informational message is displayed: `Message does not exist.`
	Cause and Solution: In the pop-up window that displays the message, click **Continue** and close the Properties window. Your settings are saved.
The system cannot intercept jobs	**Symptom**: Although the job interception feature is active on the SAP system, the intercepted jobs are kept in scheduled state.
	Cause and Solution: The job throttler feature or the Java™ Virtual Machine used by the job throttler might still be active. On each extended agent where the job throttler was started at least once, ensure that: 1. You stopped the feature. For details, see Step 5. Starting and stopping the job throttling feature on page 1254. 2. The Java™ Virtual Machine used by the job throttler was stopped by the process. To search for Java™ processes, use: **On Windows™** The Process Explorer **On UNIX®** The command `ps -ef | grep throttling` If a Java™ Virtual Machine instance related to the job throttler is found, kill it.
`access method executables: r3batch, r3event, psagent`: permission	**Symptom**: The job log reports multiple "Permission denied" messages.
	Cause and Solution: The root cause might be that the access method executable, for example, r3batch, is submitted by the `root` user and not the `twsuser`. This creates directories and files with the wrong ownership and file permissions. Verify the ownership of the following directories and files if

Table 174. Miscellaneous troubleshooting items

(continued)

Area	Item
denied messages in the job log.	you are running the product on UNIX® platforms. Ensure that the `twsuser` is the owner of the files and that the user has both read and write permissions on the files, and execute permission on the directories. *TWA_DATA_DIR*/methods/traces *TWA_DATA_DIR*/methods/traces/*.log
`psagent`: misleading message displayed if the local options file has no right permissions	**Symptom**: The job log shows the following message: `EEW00439E You did not specify the required options either in the global or in the local options file.` but all the mandatory options were correctly set in the options file.
	Solution: Check that the options file has `read` and `write` permissions available to the user who is trying to launch the job.
No messages written in the job log	**Symptom**: HCL Workload Automation does not write any messages in the job log if the file system for tracing is full or the ljuser does not have the correct permission to write in the trace directory.

The submission of a PeopleSoft job fails	**Symptom**: The submission of a PeopleSoft job fails and the HCL Workload Automation job log contains a Java™ exception similar to the following:
	`Exception in thread "3194" java.lang.ExceptionInInitializerError`
	`at bea.jolt.JoltSessionAttributes.<clinit>(JoltSessionAttributes.java:183)`
	`at psft.pt8.net.JoltSessionPool.createConnection(JoltSessionPool.java:363)`
	`at psft.pt8.net.JoltSessionPool.getJoltSession(JoltSessionPool.java:220)`

	Cause and Solution: The `psjoa.jar` path contains special characters. Define a path without special characters.

The submission of an Oracle job fails	**Symptom**: The submission of an Oracle job fails and the HCL Workload Automation job log shows the following information:
	`EEWP0017 Child MCMLJ exited normally.`
	`Exit code: 1.EEWP0027 Error - Launch job failed`

Table 174. Miscellaneous troubleshooting items

(continued)

Area	Item
	Solution: Submitting an Oracle job might fail because there is a connection problem to the Oracle database. Verify that your Oracle naming methods are set correctly. For details about how to configure naming methods, refer to the *Oracle Net Services Administrator's Guide*.
To upgrade the SAP environment, perform the following steps:	1. Delete the TWS ABAP module. 2. Upgrade SAP. 3. Install TWS ABAP module.

Scheduling jobs on HCL Workload Automation from SAP Solution Manager

HCL Workload Automation and SAP Solution Manager are integrated to allow the HCL Workload Automation engine to run the job scheduling tasks available from the Solution Manager user interface.

The integration is provided by the SMSE Adapter, which runs on the master domain manager. The SMSE Adapter uses the SAP Solution Manager Scheduling Enabler (SMSE) interface provided by SAP to enable *external schedulers* to run the scheduling for Solution Manager.

With this integration, when you schedule a job from the `Scheduling` panel of Solution Manager, HCL Workload Automation takes charge of the job scheduling, monitoring, and management tasks, as well as of job triggering and notification.

Under these conditions HCL Workload Automation acts as an RFC-Server with a common interface for scheduling jobs. It is identified through an RFC-Destination, registered in the SMSE. The interaction between Solution Manager and HCL Workload Automation is based on a PUSH mechanism implemented by the SMSE interface, whereby the master domain manager responds to requests solicited by the Solution Manager job scheduling functions.

Qualified as *external scheduler* by Solution Manager, the registered master domain managers, identified by their RFC destination names, can be called or administered from the `Process Scheduling Adapter` menu item in the Solution Manager GUI.

The jobs scheduled from Solution Manager on HCL Workload Automation must have been previously defined in the HCL Workload Automation database.

A job scheduled from the `Schedule Jobs` or `Job Documentation` panels in Solution Manager to be run by HCL Workload Automation, is automatically mapped in a job stream that is expressly created to include the job.

Registering the master domain manager on SAP Solution Manager

The first step to run the integration is to register the master domain manager on the SAP Solution Manager system.

To register master domain manager on the SAP Solution Manager system, you must:

1. Have established a connection based on RFC or Web Services between the master and the Solution Manager system.
2. Have the SAP JCo 3.1 Patch 3 (sapjco31P_3) libraries (`jar` files and, according to your operating system, `dll`, `so`, or `sl`) installed in the `<data_dir>/methods/smseadapter/lib` directory on the master domain manager. To download 3.1 Patch 3 (sapjco31P_3), visit the Sap Service Marketplace.

   Attention: The libraries require the Microsoft Visual C++ Redistributable Package (vcredist) installed.

3. Configure the `smseadapter.properties` file located in the `<data_dir>/methods/smseadapter/lib` directory on the master.

   The file contains a `SMSE_ADAPTER_CONNECTION_n` section that can be duplicated depending on the number of connections that you want to define. You can in fact set more connection definitions for the same master, where, for example, the following can vary:
   - The SAP Solution Manager system.
   - The agent that is to run the workload.
   - The SAP user name.

   Note: A master domain manager can have only one active connection at a time via the `smseadpter`. If the adapter finds more that one section with the `startAdapter` property set to `true` (or not set to `false`), it uses the first section of properties and ignores the others.

4. Stop and start WebSphere Application Server Liberty Base. For further information, see Application server - starting and stopping.

The following is an example of `smseadapter.properties` file:

```
[SMSE_ADAPTER_CONNECTION_1]
startAdapter =
ashost =
sysnr =
client =
sid =
user =
passwd =
lang =
destination =
setDestinationAsDefault =
jobStreamNamePrefix =
agentName =
notificationThreadCheckInterval =
adminConsoleHost =
adminConsolePort =
adminConsoleUser =
adminConsoleUserPassword =
maxRegistrationAttempts =
registrationAttemptInterval =
```

This section can be repeated as many times as needed in the `smseadapter.properties` file.

The properties are:

Table 175. Properties for the `smseadapter.properties` file.

Property	Description	Required	Notes
SMSE_ADAPTER_CONNECTION_1	This is the section header. If you have more sections the last digit should differ from one section and another. If two sections contain identical property values, only the first section read is considered, the other is ignored.	✓	
startAdapter	Specifies whether to connect or not to SAP Solution Manager. Can be true or false. Must be set to true to make the connection work. Set to false to temporarily suspend the connection.	✓	The default is true.
ashost	The host name of the SAP Solution Manager server on which the master domain manager registers. For example, /H/7.142.153.8/H/7.142.154.114.	✓	The master domain manager can connect to one Solution Manager system at a time.
sysnr	The SAP system number of the system that the master registers on. This value must have two digits. For example, 00.	✓	
client	The SAP client number. For example, 001.	✓	
sid	The SAP system identifier (SID) that the master registers on. For example, SM1.	✓	
user	The SAP user name that will be used during the notification process to log into SAP Solution Manager. For example, twsadmin.	✓	
passwd	The SAP password that will be used during the notification process to log into SAP Solution Manager. You can enter it in clear or in encrypted forms.	✓	To encrypt the password use the `enigma` program located in the `methods` folder on the master.
lang	The SAP logon language. For example, EN.	✓	
destination	A name entered here to identify the RFC Destination that will be used to connect	✓	This name defines the logical connection between the Solution Manager system

Table 175. Properties for the `smseadapter.properties` file.

(continued)

Property	Description	Required	Notes
	to SAP Solution Manager. For example, IWSM2. **Note:** The destination name must be univocal.		and the master domain manager, referred to in Solution Manager as the `external scheduler`. The complete destination name will then be formed by: *destination@mdm_name* For example: IWSM2@MAS93WIN **Note:** The destination name must be univocal.
setDestinationAsDefault	Set to `true` to make this destination the default one. The default is `false`.		Use this property in a context where a Solution Manager system has more than one active destination defined (that is, more registered masters), to set the default external scheduler. If you do not set a default, and you have more external schedulers registered on an SM system, you will have to specify the destination at scheduling time.
jobStreamNamePrefix	A prefix of at least four letters that is to be added to the names of the job streams created when jobs are submitted. The first character must be a letter while the remaining characters can be alphanumeric.		The default prefix is SOLMAN.
agentName	The name of the IWS agent that will run the jobs. When you search for the job definition in the Scheduling dialog, the Search utility returns the names of the jobs defined to run on this agent.		If no agent name is specified, the Search utility returns the names of the jobs defined to run on all the agents attached to the master domain manager (unless you use filtering).
notificationThreadCheckInterval	The time interval, in seconds, between checks made by the notification thread on the status changes of a job. The default is 5 seconds.		The thread notifies Solution Manager with the status changes of a job.
adminConsoleURL	The protocol used (http or https) and the host name and port of the Dynamic Workload Console		The next four properties, all related to the Dynamic Workload Console, are optional, but if you specify one, you must specify all.

Table 175. Properties for the `smseadapter.properties` file.

(continued)

Property	Description	Required	Notes
	attached to the master. For example, `https://mydwc:port_number/abc/console`.		
adminConsoleUser	The username that logs onto the Dynamic Workload Console attached to the master.		
adminConsoleUserPassword	The password of the username that logs onto the Dynamic Workload Console attached to the master.		
maxRegistrationAttempts	The maximum number of attempts to connect to SAP. By default, it is set to 5 times.		
registrationAttemptInterval	The time after which a new attempt to connect to SAP is performed. By default, it is set to 5 seconds.		

Note: If the language configured for the master domain manager is different from the language configured for the Solution Manager system, the messages issued in the Solution Manager user interface may be displayed in mixed languages.

Scheduling

The `Job Management Administration` panel of Solution Manager has two entry points for scheduling jobs:

- The `Schedule Jobs` item in `Common Tasks`, a direct way of scheduling, where you pick the job from the HCL Workload Automation database and you set the scheduling options and time definitions.
- The `Job Documentation` object, where you can create and edit job documentation, schedule, monitor, and manage jobs.

The jobs scheduled from Solution Manager on HCL Workload Automation must have been previously defined in the HCL Workload Automation database.

A job scheduled from the `Schedule Jobs` or `Job Documentation` panels in Solution Manager to be run by HCL Workload Automation, is automatically mapped in a job stream that is expressly created to include the job. The job stream is (automatically) defined in the HCL Workload Automation database with a specific prefix defined in the smseadapter.properties file on page 1312.

Best practice for job security and stability

To ensure system stability and prevent known security conflicts, set the *streamlogon* user to match the agent installation owner (`wauser`) or `root` when submitting SAP jobs on the extended agent.

By aligning the *streamlogon* user with the owner of the TWSDATA directory, you avoid the permissions problem that occurs when the job attempts to write necessary diagnostic files.

If environmental constraints require using a *streamlogon* user that differs from the agent installation owner, the job successfully initializes but is expected to fail with *ABEND (Return Code 139)* upon execution.

This failure occurs because the non-default *streamlogon* user lacks the required write permissions to create and access essential log and trace files used by the `r3batch` method.

Required manual workaround

To resolve the *RC=139* error and allow the SAP job to complete, the administrator must manually adjust the file system permissions on the agent host **before** running the job. The administrator must grant full write access (`chmod 777`) to the *streamlogon* user for the following locations:

1. The `stdlist` folder within the DATA_DIR.
2. The `trace-r3batch.log` file located in the *TWA_DATA_DIR*`/methods/trace/` directory.

Scheduling jobs directly

In the Scheduling panel, before you can proceed to enter the job name and the scheduling details, you are asked to specify the identity of the scheduling system and the scheduler type, which must be SMSE. You can then specify the name of the job definition, and the job type, which can be any of the job types supported by HCL Workload Automation. The job is qualified by Solution Manager as an `external` job.

Select the `Status message` check box to enable monitoring tasks for the job.

In the `Start Conditions` section select when and how frequently the job will run and optionally make selections in the `Repeat every` and `Time Period` groups. Your selections are then mapped to matching run cycles, `valid from` and `valid to` dates, working days or non- working days, and time dependencies on HCL Workload Automation.

Note: The Extended Window start condition is not supported. All other start conditions are supported.

Use the tabs available on the uppermost part of the panel to manage jobs; for example, to copy, reschedule, release, kill, or cancel a job, create and see external notes, and check external logs.

If a scheduled job has not been started, you can change its start conditions or parameters and click **Schedule/Change Externally** again. Alternatively, you can change the start conditions and select **Reschedule** to reset the job to a new start time. In either case, HCL Workload Automation deletes the current job instance (that has not been started) and creates another one with the new characteristics.

On the other hand, you can click **Cancel** on a non-completed job that was already started. In this case, HCL Workload Automation deletes the running instance as expected.

As soon as the job is scheduled with success, the external job ID and the status are updated and you can view the job instance on the Dynamic Workload Console.

Scheduling from job documentation

With the Job Documentation option of the Job Management Administration panel, you can also create job documentation for jobs defined in HCL Workload Automation and scheduled from Solution Manager. From the Job Documentation menu you can view and edit job details, including job steps, basic business information, and scheduling information.

To create job documentation:

1. In the Job Documentation view, create job documentation **for a Job with detail UI**.
2. In the General pane of the new job documentation creation page, enter a job documentation name and select **Other** as *Job Type*. Selecting **Other**, *External Scheduler* is automatically selected.
3. Add a step in the Step Overview table.
4. Select a job definition type from a list of job types available from HCL Workload Automation.
5. Click **Save** on top of the job documentation creation page.
6. Select the **Systems** tab in the job documentation creation page and add a solution documentation, a logical component group, or a technical scenario for the new job documentation in the *Logical Component Groups, Solution Documentation and Technical Scenarios* table. Click **Save**.
7. Select **Scheduling** in the Systems table to set up scheduling definitions for the job associated with the new job documentation.

   This action displays the same Scheduling panel described in Scheduling jobs directly on page 1314.

   You can also select **Configure Monitoring** in the Systems table to set up monitoring specifications for the job.

Monitoring

Job status retrieval and job monitoring tasks are run by HCL Workload Automation, but you can configure and view them from the Solution Manager `Job Documentation` and `Monitoring` views. In Solution Manager to monitor a job you must configure a Business project monitoring object (BPmon). When monitoring data is requested for a job, Solution Manager through the SMSE adapter requests HCL Workload Automation for updates on the job status, logs, alerts.

To view the status of a job in the Solution Manager `Job Documentation` view, provided you selected the `Status message` check box in the `Scheduling` page, follow these steps:

1. Open the job documentation view for the job.
2. Select the `Systems` and the `Scheduling` tabs.
3. In the `Scheduling` page select the `External Log` button.

   The job log is displayed in a pop-up window.

4. Select the `Refresh` button of the `External Job Status` field in the `Scheduling` page.

   The current status of the job is displayed in the field.

To configure monitoring for a scheduled job with the `Status message` check box selected, go to the `Job Management Administration` panel of Solution Manager and open the `Job Documentation` view related to the job. There, select the `Systems` tab and in the ensuing page select `Configure Monitoring`.

1. In the `Identification` section of the `Job Monitoring Setup` window, input all mandatory fields.
2. Select the `Alert Setting and Data Collection` tab and configure alerts to your convenience.
3. Fill in the mandatory fields in the `Incidents and Notifications` and `Monitoring Activities` tabs.
4. Select the `Mass Generate and Activate` button on top to save and activate the monitoring object.

With the Push mechanism HCL Workload Automation forwards to Solution Manager the status changes that a job instance undergoes until it reaches a final status such as `complete`, `canceled`, `error`, or `killed`. HCL Workload Automation also forwards the following scheduling time information for the job instance:

- Estimated duration
- Actual start
- Actual completion

On the basis of this information, and according to the alert configuration you specified in the `Alert Setting and Data Collection` pane, Solution Manager triggers these alerts when any of the thresholds you specified are reached or exceeded. This grants you the means to keep the execution of your workload under control.

To view the alerts for a monitored job, select the `Unified Alert Inbox` view in the `Job Management Administration` panel:

1. Select the monitoring object for the job in the Job Monitoring Standard View.
2. Refresh the alert list table after some monitoring period.

Note: `Alert Inbox` and `Job Monitoring` panels can be also accessed from the `Home` in the `Job Management` section.

Setting the traces on the application server for the major HCL Workload Automation processes

About this task

The application server handles all communications between the HCL Workload Automation processes. The trace for these communications is set to `tws_info` by default (information messages only). The application server can be set to trace `all` communications, either for the whole product or for these specific groups of processes:

- Command line
- Connector
- Database

- Planner
- Utilities
- Dynamic workload broker

Significant impact on performance: Activating traces for the WebSphere Application Server Liberty Base leads to a significant impact on performance, especially if you set the tracing to `all`. Thus you are strongly advised to identify the process group where the problem that you want to trace is occurring, and only set the trace to that group.

To modify the trace level on the WebSphere Application Server Liberty Base, edit the `trace.xml` file as necessary.

Templates for the master domain manager are stored in the following paths:

On UNIX operating systems

TWA_home`/usr/servers/engineServer/configDropins/templates`

On Windows operating systems

TWA_home`\usr\servers\engineServer\configDropins\templates`

Templates for the Dynamic Workload Console are stored in the following paths:

On UNIX operating systems

DWC_home`/usr/servers/dwcServer/configDropins/templates`

On Windows operating systems

DWC_home`\usr\servers\dwcServer\configDropins\templates`

When you edit the file with your customized settings for the master domain manager, move it to the following paths:

On UNIX operating systems

TWA_DATA_DIR`/usr/servers/engineServer/configDropins/overrides`

On Windows operating systems

TWA_home`\usr\servers\engineServer\configDropins\overrides`

When you edit the file with your customized settings for the Dynamic Workload Console, move it to the following paths:

On UNIX operating systems

DWC_DATA_dir`/usr/servers/dwcServer/configDropins/overrides`

On Windows operating systems

DWC_home`\usr\servers\dwcServer\configDropins\overrides`

1. Copy the template file from the `templates` folder to a working folder.
2. Edit the template file in the working folder with the desired configuration.
3. Optionally, create a backup copy of the relevant configuration file present in the `overrides` directory in a different directory. Ensure you do not copy the backup file in the path where the template files are located.

4. Copy the updated template file to the `overrides` folder. Maintaining the original folder structure is not required.
5. Changes are effective immediately.

For example, to modify the trace level on WebSphere Application Server Liberty Base, perform the following steps:

1. Copy the `trace.xml` file from the TWA_home`/usr/servers/engineServer/configDropins/templates` folder to a working folder.
2. Edit the template file in the working folder by changing the following string:

   ```
   <variable name="trace.specification" value="*info"/>
   ```

   into

   ```
   <variable name="trace.specification" value="com.ibm.tws.dao.model.
   =all:com.ibm.tws.dao.rdbms.=all"/>
   ```

3. Copy the updated template file to *TWA_DATA_DIR* `/usr/servers/engineServer/configDropins/overrides`. Changes are effective immediately.

Traces are stored in *DWC_DATA_dir*`/appserver/engineServer/logs`.

The **trace.specification** can be found in `trace.xml` files and can refer to a specific component (tws*_xxx*) or to the whole product, as follows:

tws_all

"com.ibm.tws.**=all:org.apache.wink.server.**=all:com.hcl.tws.*=all"

tws_alldefault

"com.ibm.tws.*=error=enabled"

tws_broker_all

"com.ibm.scheduling.*=all:TWSAgent=all"

tws_rest

" com.ibm.tws.conn.**=all:com.ibm.tws.twsd.rest.**=all:org.apache.wink.server.*=all"

tws_cli

"com.ibm.tws.cli.**=all:com.ibm.tws.objects.**=all"

tws_utils

"com.ibm.tws.util.*=all"

tws_conn

"com.ibm.tws.conn.**=all:com.ibm.tws.objects.**=all:com.ibm.tws.updatemanager.**=all:com.ibm.tws.dao.plan.**=all"

tws_db

"com.ibm.tws.dao.model.**=all:com.ibm.tws.dao.rdbms.**=all"

tws_planner

"com.ibm.tws.planner.**=all:com.tivoli.icalendar.**=all:com.ibm.tws.runcycles.**=all:com.ibm.tws.conn.planner.**=all :com.ibm.tws.cli.planner.*=all"

tws_secjni

"com.ibm.tws.audit.**=all:com.ibm.tws.security.**=all"

tws_engine_broker_all

"com.ibm.tws.**=all:com.ibm.scheduling.**=all:TWSAgent=all"

Editing the logging element above with the **traceSpecifcation** value to `tws_all`, enables "**com.ibm.tws.=all:org.apache.wink.server.=all:com.hcl.tws.*=all**".

Other values are reported in variable tags above. You can also replace the value of the **trace.specification** parameter with a custom string.

Troubleshooting

About this task

The following topics are provided to help troubleshoot and debug extended agent and access method problems.

Job standard list error messages

All output messages from an access method, except those that start with a percent sign (%), are written to the job's standard list (**stdlist**) file. For **GS** and **CF** tasks that are not associated HCL Workload Automation jobs, messages are written to HCL Workload Automation standard list file. For information about a problem of any kind, check these files.

Method not executable

If an access method cannot be run, the following occurs:

- For **LJ** and **MJ** tasks, the job is placed in the FAIL state.
- For the **CF** task, the file dependency is unresolved and the dependent job remains in the HOLD state.
- For the **GS** task, the job dependency is unresolved and the dependent job remains in the HOLD state.

To get more information, review the standard list files (**stdlist**) for the job and for HCL Workload Automation.

Console Manager messages for extended agents only

This error message is displayed if you issue a **start**, **stop**, **link**, or **unlink** command for an extended agent:

Example

```
AWSBHU058E The command issued for workstation: workstation_name,
           cannot be performed, because the workstation is an extended agent,
           where the command is not supported.
```

Composer and compiler messages for extended agents only

The following error messages are generated when **composer** encounters invalid syntax in a workstation definition:

Example

```
AWSDEM045E There is an error in the workstation definition. The ACCESS keyword
           was not followed by a valid method. Valid methods correspond with
           the name of a file in the TWS_home/methods directory
           (the file need not be present when the access method is defined).
```

Example

```
AWSDEM046E There is an error in the workstation definition. The ACCESS keyword
           has been specified more than once.
```

Example

```
AWSDEM047E There is an error in the workstation definition. The ACCESS keyword
           was not followed by a valid method. Valid methods correspond with
           the name of a file in the TWS_home/methods directory
           (the file need not be present when the access method is defined).
```

If an extended agent is defined with an access method but without a host, the following message is displayed:

Example

```
AWSBIA140E For an extended agent you must specify the host and the access method.
```

Jobman messages for extended agents only

For extended agents, error, warning, and information messages are written to **jobman**s **stdlist** file.

A successful job launch generates the following message:

Example

```
AWSBDW019I Launched job job_name, #Jrun_number for user user_ID.
```

Failure to launch a job generates the following message:

Example

```
AWSBDW057E The job  job_name was not launched for this reason:
           error_message
```

Failure of a check file task generates the following message:

Example

```
AWSBDW062E Jobman was unable to invoke the following method file method_name
           for the extended agent. The operating system error is:
           system_error
```

Failure of a manage job task generates the following message:

Example

```
AWSBDW066E Planman has asked jobman to run a task that is not supported on the
           targeted agent. The following method options file was used:
           method_options_file. The job identifier and monitor PID are as
           follows: job, #Jmonitor_pid
```

When a method sends a message to **jobman** that is not recognized, the following message is generated:

Example

```
AWSBDW064E A job that jobman is monitoring has returned the following
           unrecognizable message: incorrect_message. The job identifier,
           monitor PID and method file are as follows: job_name, #Jmonitor_pid
           using method file.
```

Chapter 23. Managing internetwork dependencies

HCL Workload Automation ***internetwork dependencies*** allow jobs and job streams in the local network to use jobs and job streams in a remote network as *follows* dependencies. This chapter describes how to customize your environment to be able to define internetwork dependencies and how to manage the internetwork dependencies.

The chapter is divided into the following sections:

- Internetwork dependencies overview on page 1322
- Configuring a network agent on page 1324
- Defining an internetwork dependency on page 1326
- Managing internetwork dependencies in the plan on page 1327
- Internetwork dependencies in a mixed environment on page 1331

Note: Depending on your needs and requirements, you can choose between internetwork dependencies and cross dependencies to establish a dependency between a job running on the local engine and a job running on a remote HCL Workload Automation engine. See Defining dependencies on page 42 for a description about the differences between these two types of dependencies.

Internetwork dependencies overview

Before you specify an internetwork dependency, you must create a workstation definition for the ***network agent***. A network agent is an HCL Workload Automation workstation that handles follows dependencies between its local network and a remote HCL Workload Automation network.

In the local HCL Workload Automation network there can be more than one network agent, each representing a specific HCL Workload Automation remote network where jobs and job streams referring to locally defined internetwork dependencies are defined. Internetwork dependencies are assigned to jobs and job streams in the same way as local follows dependencies, with the exception that the network agent's name is included to identify the followed job or job stream.

Note: The internetwork dependencies are supported only on ROOT folder.

A special job stream named ***EXTERNAL*** is automatically created by HCL Workload Automation for each network agent in the local network. It contains placeholder jobs to represent each internetwork dependency.

An `EXTERNAL` job is created for each internetwork dependency belonging to job streams planned to start in different days with different schedule dates. This means that an `EXTERNAL` job differs from another one by:

- The script file name, which identifies the remote job or job stream the local job or job stream is dependent on.
- The date the local job stream containing the internetwork dependency is planned to start. If the dependency is defined in a job within the job stream the date the job stream is planned to start is taken into account.

The check of the internetwork dependency check does not start until the job stream matches its time dependency or it is released.

In case of two jobs belonging to different job streams and referring to the same internetwork dependency, as one of their job streams is released and the job starts the internetwork dependency is checked and possibly released. In this case when the second job starts to check its internetwork dependency it finds the dependency already solved but not necessarily on the expected day. If you want to prevent this situation from occurring you must rerun the job representing the internetwork dependency after it is solved the first time.

HCL Workload Automation checks the status of the referred jobs and job streams in the remote network and maps their status in the jobs representing the internetwork dependencies in the `EXTERNAL` job stream. The status of these jobs and job streams is checked over a fixed time interval until the remote job or job stream reaches the `SUCC`, `CANCL`, or `ERROR` state.

Understanding how an internetwork dependency is shown

About this task

This section describes a sample scenario about internetwork dependencies and how to link the job representing the internetwork dependency to the job stream where the dependency is defined. Assume that:

- You defined a job stream named `ELISCHED` running on workstation `TWS206` containing a job named `ELI` with an internetwork dependency from a job stream `TWS207#FINAL.MAKEPLAN` running in a different HCL Workload Automation network.
- `XA_MAST` is the network agent defined in the local network to manage internetwork dependencies from jobs and job streams defined in that remote network.

Use the **conman sj** command to see the internetwork dependency set:

```
                                      (Est) (Est)
CPU    Schedule SchedTime Job State Pr Start Elapse RetCode Deps

TWS206#ELISCHE 0600 03/31 **** HOLD 10 (03/31)
                          ELI  HOLD 10                     XA-MAST::"TWS207#MYJS.JOB1"
```

where `(03/31)` represents the **at** time restriction set in `TWS206#ELISCHE`. Starting from `(03/31)` the status of `TWS207#MYJS.JOB1` is checked in the remote network to see if the internetwork dependency `XA-MAST::"TWS207#MYJS.JOB1"` is satisfied.

If you run the command:

```
%sj XA-MAST#EXTERNAL;info
```

you see the list of jobs representing internetwork dependencies defined in jobs and job streams running in the local network from jobs and job streams defined in the remote network reachable through the network agent `XA-MAST`:

```
CPU      Schedule SchedTime  Job      JobFile               Opt  Job
 Prompt
XA-MAST #EXTERNAL
                            E8802332 TWS207#MYJS.JOB1
```

You can see the details about the job or job stream depending on `TWS207#MYJS.JOB1` in the internetwork dependency represented by job `E8802332` in the `EXTERNAL` job stream, by running the following command:

```
%sj @#EXTERNAL.E8802332;deps
```

The output shows the link between the dependent job and the internetwork dependency:

```
                                        (Est) (Est)
 CPU    Schedule SchedTime Job State Pr Start Elapse RetCode Deps

XA-MAST#EXTERNAL.E8802332 Dependencies are:


 TWS206#ELISCHE 0600 03/31 **** HOLD 10 (03/31)
                           ELI  HOLD 10                     XA-MAST::"TWS207#MYJS.JOB1"
```

The internetwork dependency check does not start until the job stream `TWS206#ELISCHE` matches its time dependency, `(03/31)`, or is released.

If there is another job defined within another job stream in the local network that has a dependency on `TWS207#MYJS.JOB1` and the local job stream is planned to start on the same day, `03/31`, then also the dependency of this other job on `TWS207#MYJS.JOB1` will be listed in the job `E8802332` within the `XA-MAST#EXTERNAL` job stream.

Configuring a network agent

About this task

Network agent workstations are defined as extended agents and require a hosting physical workstation and an access method. The access method for network agents is named **netmth**.

The **batchman** process on the master domain manager queries the **netmth** on the network agent at fixed time intervals to get the status of the remote predecessor job or job stream. You can customize the time interval between two consecutive checks by setting the global option *bm check status* in the `localopts` file on the master domain manager. The HCL Workload Automation continues checking until the remote job or job stream reaches the `SUCC`, `CANCL`, or `ERROR` state.

You must create an options file named **`netmth.opts`** on the workstation where the network agent runs. In this file are defined the user under which the access method runs and the time to wait to get a response from the access method before shutting it down. This options file must have the same path as the access method:

```
TWS_home/methods/netmth.opts
```

The content of the `netmth.opts` file has the following structure:

```
GSuser=login_name
GStimeout=seconds
```

where:

login_name

Is the login used to run the method. If the network agent's host is a Windows® computer, this user must be defined in HCL Workload Automation.

seconds

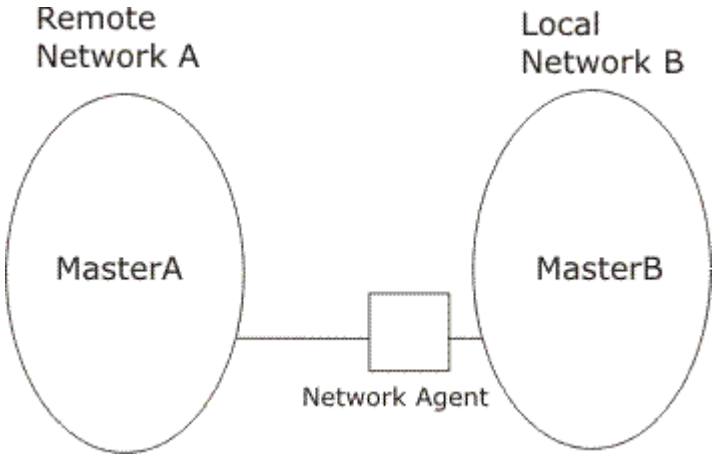
Is the number of seconds, HCL Workload Automation waits for a response before shutting down the access method. The default setting is 300 seconds. The next time **batchman** needs to check the status of the remote predecessor job or job stream, the access method starts automatically.

Changes to this file do not take effect until you stop and restart HCL Workload Automation.

A sample network agent definition

The following example shows how to define a network agent workstation for a remote network, `Network A`, that allows local network, `Network B`, to use jobs and job streams in the remote network as internetwork dependencies.

Figure 47. Local and remote networks



Assuming that:

- `MasterA` is the master domain manager of the remote network, `Network A`, and that:
  - **tws_masterA** is the *TWS_user* defined on `MasterA`.
  - The TCP port number for `MasterA` as 12345.
  - The node where `MasterA` is defined is `MasterA.rome.tivoli.com`.
- `MasterB` is the master domain manager of the local network, `Network B`, and that:
  - **tws_masterB** is the *TWS_user* defined on `MasterB`.
  - The node where `MasterB` is defined is `MasterB.rome.tivoli.com`.

A network agent workstation named `NetAgt`, defined on `MasterB` to manage internetwork dependencies on jobs or job streams defined in `Network A` can be the following:

```
CPUNAME NETAGT
  DESCRIPTION "NETWORK AGENT"
  OS OTHER
  NODE MASTERA.ROME.ITALY.COM
  TCPADDR 12345
  FOR maestro
    HOST MASTERB
    ACCESS netmth
END
```

Important: Write the network access name `netmth` in lowercase on case-sensitive operating systems.

The options file, **`netmth.opts`** defined on `MasterB` can be:

```
GSuser=tws_masterB
GStimeout=600
```

Note: The network agent can be defined on either the master domain manager or a fault-tolerant agent.

Defining an internetwork dependency

About this task

The syntax used to specify an internetwork dependency within a job stream definition is the following:

```
follows  Network_agent_name::remote_workstation#jobstreamname(time [date]).jobname
```

where the `(time [date])` are specific to the time zone used on the workstation of the remote network the network agent is connected to; in our sample the time zone of `MasterA`. If `(time [date])` is not specified in this syntax or if there is more than one job stream with the same `(time [date])`, the first job stream found is taken into account.

Assuming that:

- `schedA` is a job stream defined in the `MasterA` database.
- `jobA` is a job defined in the `MasterA` database.
- `schedB` is a job stream defined in the `MasterB` database.
- `jobB` is a job defined in the `MasterB` database.

you can define internetwork dependencies using the following **follows** statements:

To define an internetwork dependency of `schedB` from the job stream instance `schedA(1100)`

Use the following statement:

```
schedule schedB
      on everyday
      follows NetAgt::MasterA#schedA(1100)
```

```
      :
end
```

To define an internetwork dependency of `jobB` from `jobA` contained in the job stream instance `schedA(1100)`

Use the following statement:

```
schedule schedB
      on everyday
      :
      jobB
           follows NetAgt::MasterA#schedA(1100).jobA
end
```

You can also define internetwork dependencies of a job on a job stream or a job stream on a job.

Managing internetwork dependencies in the plan

About this task

Internetwork dependencies are managed in the plan from the **conman** command line by managing the `EXTERNAL` job stream. Within the `EXTERNAL` job stream the internetwork dependencies are listed as jobs regardless of whether they are defined for jobs or job streams. There is an `EXTERNAL` job stream for every network agent in the plan.

Within the `EXTERNAL` job stream, unique job names representing internetwork dependencies are generated as follows:

```
Ennnmmss
```

where:

nnn

Is a random number.

mm

Is the current minutes.

ss

Is the current seconds.

The actual name of the job or job stream is stored in the script files specification of the job record.

Note: Remote jobs and job streams are defined and run on their local network in the standard manner. Their use as internetwork dependencies has no effect on their local behavior.

States of jobs defined in the EXTERNAL job stream

The status of the jobs defined in the `EXTERNAL` job stream is determined by the access method and listed in the Release Status field of the `EXTERNAL` job stream. The reported status refers to the last time the remote network was checked. Jobs might appear to skip states when states change in between two different checks.

All states for jobs and job streams, except `FENCE`, are listed. In addition to these there are three states that are specific to the `EXTERNAL` jobs, they are:

CANCL

The corresponding job or job stream in the remote network has been cancelled.

ERROR

An error occurred while checking for the remote status.

EXTRN

This is the initial state. If the job is not found in the remote network, it remains in `EXTRN` state.

Note: If you cancel in the local network the instances of jobs or job streams dependent on the same instance of a job or job stream defined in a remote network, make sure you manually cancel also the job, representing that internetwork dependency in the `EXTERNAL` job stream, to prevent the `EXTERNAL` job stream from being continuously carried forward.The same consideration applies when the local job stream dependent on the job or job stream defined in the remote network is not carried forward to the new plan.

Working with jobs defined in the EXTERNAL job stream

About this task

These are the available actions you can perform against jobs in an `EXTERNAL` job stream:

Cancel

Cancels the `EXTERNAL` job, releasing the internetwork dependency for all local jobs and job streams. The status of the dependency is no longer checked.

Rerun

Instructs **conman** to restart checking the state of the `EXTERNAL` job. The job state is set to `EXTRN` immediately after a **rerun** is performed.

Rerun is useful for `EXTERNAL` jobs in the `ERROR` state. For example, if an `EXTERNAL` job cannot be launched because the network access method does not grant execute permission, the job enters the `ERROR` state and its status ceases to be checked. After you correct the permissions, the method can start but **conman** will not start checking the `EXTERNAL` job state until you perform a **rerun**.

Confirm SUCC / ABEND

Sets the status of the `EXTERNAL` job to `SUCC` or `ABEND`, releasing the dependency for all depending local jobs and job streams. The status of the dependency in no longer checked.

Note: None of these commands has any effect on the remote job or job stream in the remote network. They simply manipulate the dependency for the local network.

Sample internetwork dependency management scenarios

About this task

This section provides sample scenarios describing how you can manage internetwork dependency in production using the **conman** command line commands.

Assuming that you have already defined the following:

- A local workstation called `local1`
- A job stream defined for the local workstation `local1` called `sched1`
- A job defined in `local1#sched1` called `job1`
- A network agent called `netagt` defined in the local network to manage internetwork dependency from jobs and job streams defined in the remote network.
- A workstation in the remote network called `remote1`
- A job stream defined for the remote workstation `remote1` called `rcshed`
- A job in defined in `remote1#rsched` called `rjob`

You can perform several actions from the **conman** command line in the local network. The following list contains some examples:

Adding an internetwork dependency from a remote job to a local job.

For example, to add the remote job `rjob` as an internetwork dependency for `job1`, run the following command:

```
adj local1#sched1.job1;follows=netagt::remote1#rsched.rjob
```

Adding an internetwork dependency from a remote job stream to a local job stream.

For example, to add the remote job stream `rsched` as an internetwork dependency for job stream `sched1`, run the following command:

```
ads local1#sched1;follows=netagt::remote1#rsched
```

Cancelling internetwork dependencies managed by a network agent.

For example, to cancel all `EXTERNAL` jobs for a network agent `netagt`, run one of the following two commands:

```
cj netagt#EXTERNAL.@

cj netagt::@
```

Confirming the successful completion of an internetwork dependency.

For example, to confirm that the remote job `remote1#rsched.rjob` completed successfully and so release the corresponding internetwork dependency, run the following command:

```
confirm netagt::remote1#rsched.rjob;succ
```

Deleting an internetwork dependency from a job for a job.

For example, to delete the internetwork dependency from the remote job `remote1#rsched.rjob` for the local job `local1#sched1.job1`, run the following command:

```
ddj local1#sched1.job1;follows=netagt::remote1#rsched.rjob
```

Deleting an internetwork dependency from a job for a job stream.

For example, to delete the internetwork dependency from the remote job `remote1#rsched.rjob` for the local job stream `local1#sched1`, run the following command:

```
dds local1#sched1;follows=netagt::remote1#rsched.rjob
```

Releasing a local job from an internetwork dependency from a remote job.

For example, to release a job from an internetwork dependency from a remote job, run the following command:

```
rj local1#sched1.job1;follows=netagt::remote1#rsched.rjob
```

Releasing a local job stream from an internetwork dependency from a remote job.

For example, to release a job stream from an internetwork dependency from a remote job, run the following command:

```
rs local1#sched1;follows=netagt::remote1#rsched.rjob
```

Rerunning a job in the `EXTERNAL` job stream to restart checking a dependency.

For example, to rerun a job belonging to the `EXTERNAL` job stream to restart checking the internetwork dependency from the remote job `remote1#rsched.rjob`, run one of the following two commands:

```
rr netagt#EXTERNAL.rjob

rr netagt::remote1#rsched.rjob
```

Displaying internetwork dependencies from jobs and job streams defined in a remote network.

For example, to display all the internetwork dependencies defined for a network agent with their original names and their generated job names, run the following command:

```
sj netagt#EXTERNAL.@;info
```

Submitting a job with an internetwork dependency from a job stream defined in a remote network

For example, to submit a `rm` command into the `JOBS` job stream with an internetwork dependency on a remote job stream, run the following command:

```
sbd "rm apfile";follows=netagt::remote1#rsched
```

Internetwork dependencies in a mixed environment

Table 176: Internetwork dependencies in a mixed environment on page 1331 shows the supported configuration for internetwork dependencies defined in a mixed version 8.3 environment. The key to the table is as follows:

Net_A

The network agent defined in the local network.

Wks_B

The workstation in the remote network that the network agent `Net_A` is connected to. `Wks_B` is the workstation that identifies and checks the state of the remote job or job stream specified in the internetwork dependency.

Sym_A

The `Symphony` file processed in the local network.

Sym_B

The `Symphony` file processed in the remote network.

back-level

Version 8.1, 8.2, or 8.2.1

Table 176. Internetwork dependencies in a mixed environment

	Net_A back-level Sym_A back-level	Net_A 8.3 Sym_A back-level	Net_A back-level Sym_A 8.3	Net_A 8.3 Sym_A 8.3
Wks_B back-level **Sym_B back-level**	This is not a mixed version 8.3 environment.	`Net_A` sends the information to `Wks_B` as if it had the same version as `Wks_B`.	`Net_A` sends the information to `Wks_B` in 8.1, 8.2, or 8.2.1 format. The use of the *schedtime* keyword in the job definition is not supported.	`Net_A` sends the information to `Wks_B` as if it had the same version as `Wks_B`. If defined, the *schedtime* keyword in the job definition is automatically removed by `Net_A`.
Wks_B 8.3 **Sym_B back-level**	`Wks_B` works as if it had the same version as `Net_A`.	`Net_A` sends the information to `Wks_B`. If defined, the *schedtime* keyword in the job definition is automatically removed by `Wks_B`.	`Net_A` sends the information to `Wks_B`. If defined, the *schedtime* keyword in the job definition is automatically removed by `Wks_B`.	`Net_A` sends the information to `Wks_B`. If defined, the *schedtime* keyword in the job definition is automatically removed by `Wks_B`.
Wks_B back-level **Sym_B 8.3**	Not supported.	Not supported.	Not supported.	Not supported.

Table 176. Internetwork dependencies in a mixed environment (continued)

	Net_A back-level Sym_A back-level	Net_A 8.3 Sym_A back-level	Net_A back-level Sym_A 8.3	Net_A 8.3 Sym_A 8.3
Wks_B 8.3 **Sym_B 8.3**	Not supported.	Not supported.	`Net_A` sends the information to `Wks_B`. If defined, the *schedtime* keyword is parsed by `Wks_B`.	This is a version 8.3 environment.

Chapter 24. Applying conditional branching logic

A conditional dependency is a relation that uses the status or output of a previous job to determine whether a specific segment of your job flow executes.

With HCL Workload Automation you can define jobs to run as often as necessary. Sometimes some jobs must wait for other jobs to finish successfully before they start. Add flexibility to your job flows by choosing which job to run based on the status or output of a previous job.

You can specify dependencies to define job flows with alternative branches based on *conditional logic*. This behavior duplicates the function of `if-then-else` statements. To determine when a successor job starts, you can use return codes, job status, output variables, and job log content as conditional logic elements.

The Graphical Designer within the Dynamic Workload Console displays a visual representation of these relationships, dependencies, and conditions. You can use the workspace to quickly view and modify your job flows.

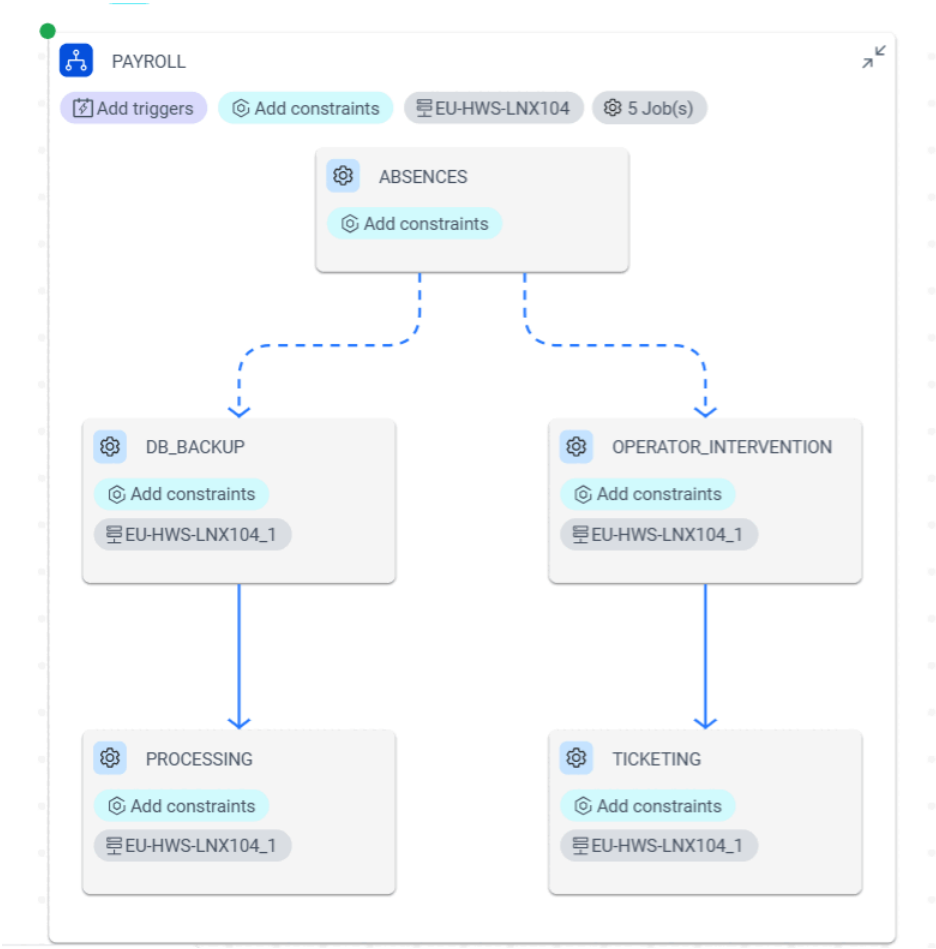
Note:

Conditional dependencies are evaluated after any standard dependencies in the job or job stream are satisfied.

Example:

The `PAYROLL` job stream starts with the `ABSENCES` predecessor job, followed by two conditional branches of jobs that can run. The branch that runs depends on the outcome of the `ABSENCES` job, as defined by its *output conditions*.

If the output conditions are not satisfied, the HCL Workload Automation places the omitted successor jobs in the `SUPPRESSED` state. This behavior differs from regular dependencies, where the system places successor jobs in the `HOLD` state until the predecessor reaches a successful (`SUCC`) state. Predecessors can be either jobs or job streams.



You can define two types of conditions: **status conditions** (based on job status), or **other output conditions** (based on a mapping expression such as a return code, output variables, or output in a job log content). When the predecessor is a job stream, the conditional dependency must be a status condition.

Status conditions

Status conditions evaluate the run state of a job or job stream:

- For jobs, valid states include `FAIL`, `ABEND`, `SUCC`, or `SUPPR`.
- For job streams, valid states include `SUCC`, `SUPPR`, and `ABEND`.

Other output conditions

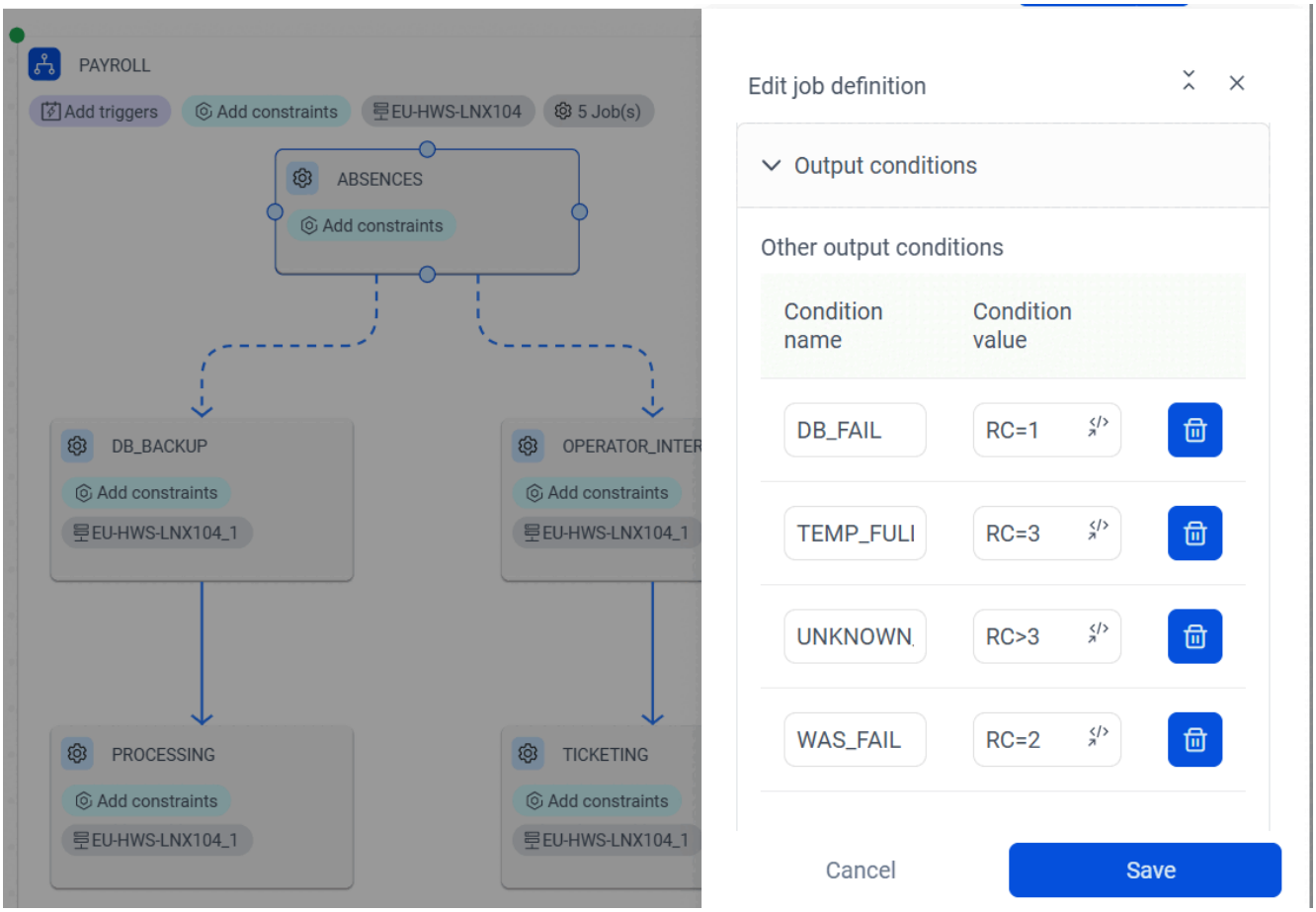
You can evaluate other output conditions by using a mapping expression based on the agent type:

- A return code (available on fault-tolerant agents and dynamic agents)
- Output variables (available on dynamic agents)
- Job log content (available on dynamic agents)

A condition establishes a conditional dependency relationship. You specify output conditions directly within the job definition, and you can define an unlimited number of dependencies. When you add a conditional dependency to a job, you select the exact status and output conditions that the system evaluates during processing.

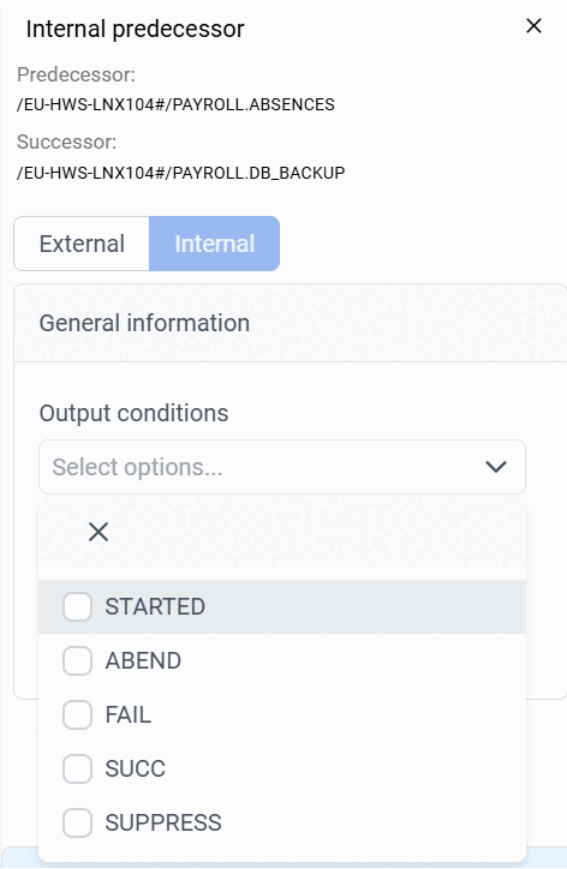
Example:

The following example illustrates other output conditions defined within a job definition:



You can define both successful output conditions (which signify a successful job run) and alternative output conditions (which determine which successor job starts). HCL Workload Automation evaluates these output conditions using **OR** logic.

When you add this job to a job stream as a successor job and apply a conditional dependency to the preceding job (the predecessor), you select the required conditions. The **Properties** panel for the internal or external dependency dynamically updates to include the conditions from the job definition. In addition to these default conditions, you can select conditions based on job status. If the selected conditions are satisfied during job processing, the corresponding successor job runs.



You can aggregate or **join** conditional dependencies from different predecessors. A join contains multiple dependencies, and you specify how many of those dependencies must be met for the system to consider the join satisfied. You can define an unlimited number of conditional dependencies, standard dependencies, or both within a single join.

Conditional dependencies are not supported on internetwork dependencies, nor on Limited Fault-Tolerant Agents for IBM i.

Setting up conditional dependencies

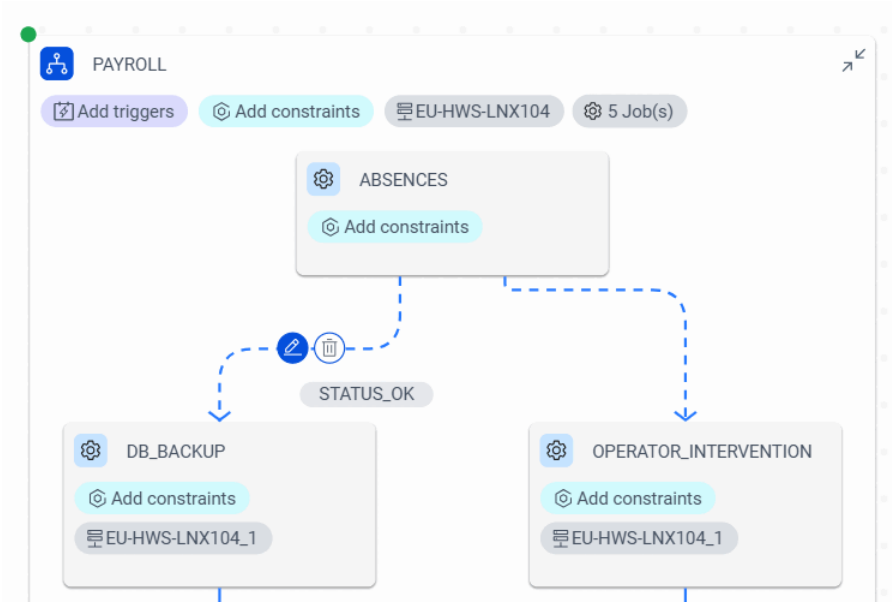
You can set up conditional dependencies to define workflows with alternative branches based on conditions.

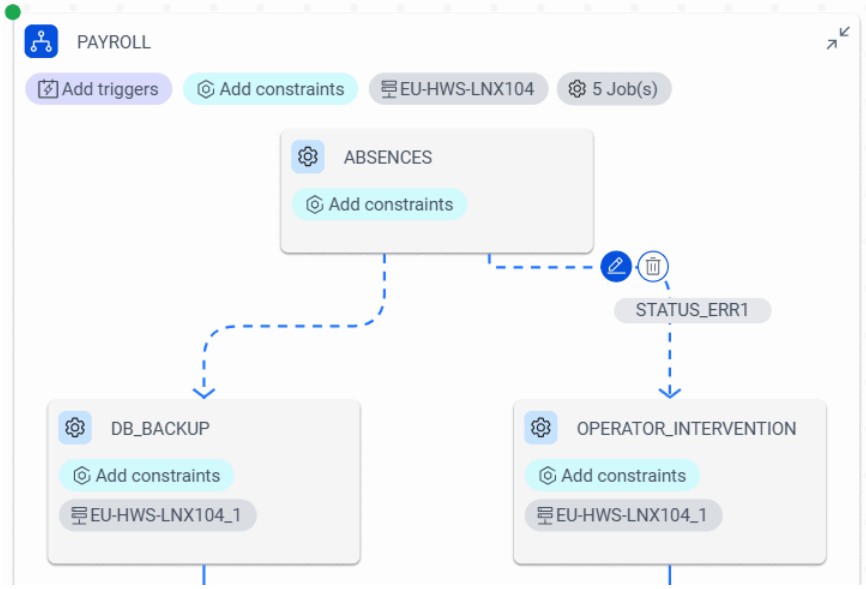
About this task

Using conditional dependencies, you can control when a successor job starts depending on the combination of one or more mapping expressions (for example, return codes) or statuses of a predecessor job.

Example:

The `DB_BACKUP` job runs if the `ABSENCES` job satisfies the condition associated with **STATUS_OK**. The `OPERATOR_INTERVENTION` job runs if the `ABSENCES` job satisfies the condition defined for **STATUS_ERR1**.





To set up this type of job processing, create the `ABSENCES` job definition and define the conditional dependencies.

1. Create the `ABSENCES` job definition:
    a. Open the Dynamic Workload Console.
    b. From the **Design** menu, select **Graphical Designer**.
    c. Create a new workspace.
    d. From the canvas, select the **Assets** tab and click the **Add** button (**+**).
    e. From the drop-down menu, select **Job definition**.
    f. In the search bar, select a job type and click **Next**.
    g. Define information about the job, and name it `ABSENCES`.
2. Define the conditional dependencies:

a. In the **Output conditions** section, under **Successful output conditions**, click the **Add** button (**+**).
b. Enter **STATUS_OK** as condition name, and set `RC=0` as condition value.
c. In **Other output conditions**, click the **Add** button (**+**).
d. Enter **STATUS_ERR1** as condition name, and set `RC=2` as condition value.
e. Click **Add** to add the job definition to the database.

3. Create the `DB_BACKUP` and the `OPERATOR_INTERVENTION` jobs.
4. Create the `PAYROLL` job stream:
   a. From the **Blocks** tab, drag a job stream onto the canvas, name it `PAYROLL` and define information about it.
5. Add the `ABSENCES`, `DB_BACKUP` and `OPERATOR_INTERVENTION` jobs to the `PAYROLL` job stream:
   a. From the **Assets** tab, drag the `ABSENCES`, `DB_BACKUP` and the `OPERATOR_INTERVENTION` jobs onto the `PAYROLL` job stream.
6. Define the internal conditional dependencies:
   a. From the `ABSENCES` job block, drag a connecting arrow to the `DB_BACKUP` connection point.
   b. From the **Internal predecessor** panel, in **General information**, expand the **Output conditions** menu and select **STATUS_OK**.
   c. From the `ABSENCES` job block, drag a connecting arrow to the `OPERATOR_INTERVENTION` connection point.
   d. From the **Internal predecessor** panel, in **General information**, expand the **Output conditions** menu and select **STATUS_ERR1**.

Example

You can define the same scenario by using Orchestration CLI:

Job definition

```
WK1#ABSENCES
  SCRIPTNAME "myscript.sh"
  STREAMLOGON root
  DESCRIPTION "Sample Job Definition"
  TASKTYPE UNIX
  SUCCOUTPUTCOND  STATUS_OK "RC=0”
  OUTPUTCOND STATUS_ERR1 “RC =2”
    RECOVERY CONTINUE
END
```

Job stream

```
SCHEDULE WK1#PAYROLL
ON RUNCYCLE RULE1 "FREQ=DAILY;"
AT 1800
CARRYFORWARD
:
WK1#DB_BACKUP
  FOLLOWS WK1#ABSENCES IF STATUS_OK
WK1#OPERATOR_INTERVENTION
  FOLLOWS WK1#ABSENCES IF STATUS_ERR1
END
```

For more information about defining conditional dependencies using Orchestration CLI see Job definition on page 228, Job stream definition on page 314, and follows on page 344.

Joining or combining conditional dependencies

You can combine a set of dependencies into a single join dependency.

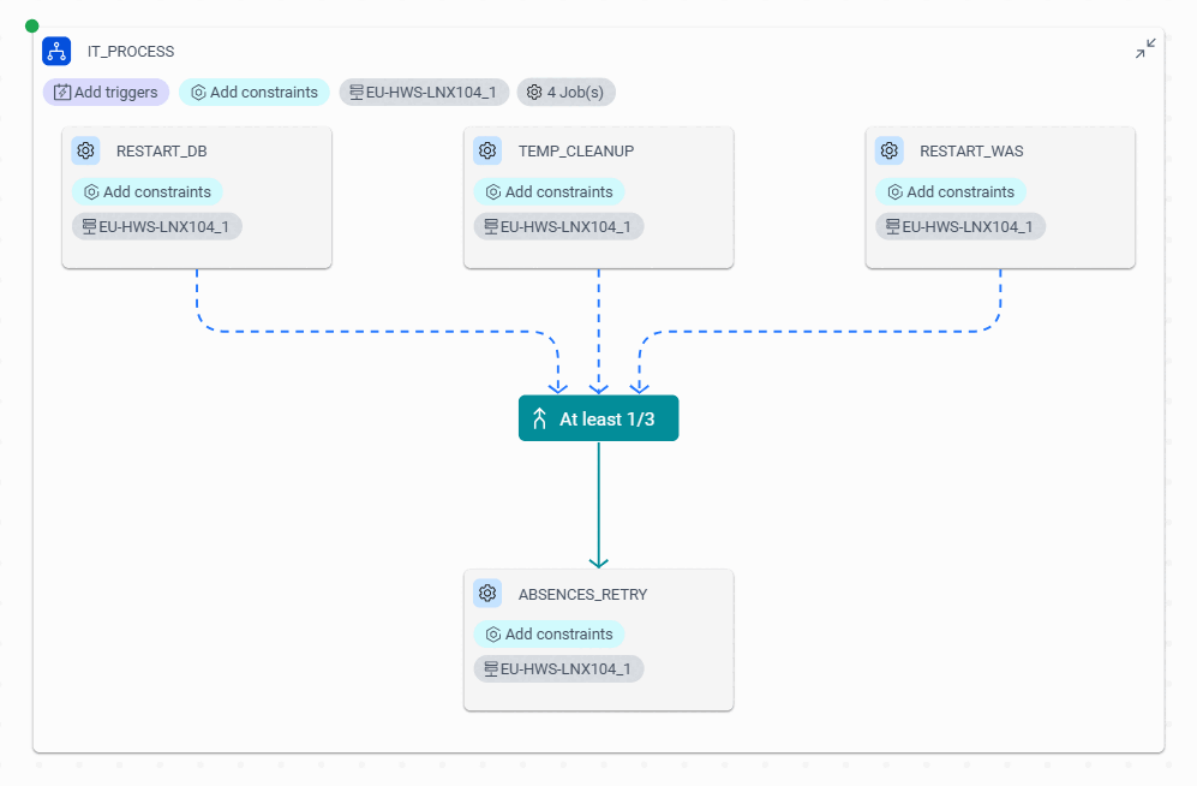
About this task

You can add multiple dependencies from different predecessors to a single join dependency. You then specify how many of those dependencies must be met for HCL Workload Automation to consider the join satisfied. When the conditions satisfy the join, the successor job runs.

Example:

A `PROBLEM_SOLVING` join dependency within a job stream aggregates three conditional dependencies from different predecessor jobs. The rule requires only one conditional dependency to be met to satisfy the `PROBLEM_SOLVING` join.

The `ABSENCES_RETRY` job follows the `PROBLEM_SOLVING` join and runs when at least one of the three predecessor jobs completes successfully. If the conditions do not satisfy the join, HCL Workload Automation places the `ABSENCES_RETRY` job in the `SUPPR` state.



To create the `PROBLEM_SOLVING` join dependency from the Dynamic Workload Console, complete the following steps:

1. From the Graphical Designer, create the following job definitions and configure their properties: `RESTART_DB`, `RESTART_WAS`, `TEMP_CLEANUP`, and `ABSENCES_RETRY`.
2. From the **Blocks** tab, drag a job stream onto the canvas and name it `IT_PROCESS`, then define information about it.

3. From the **Assets** tab, drag `RESTART_DB`, `RESTART_WAS`, `TEMP_CLEANUP`, and `ABSENCES_RETRY` jobs onto the `IT_PROCESS` job stream.
4. From the **Blocks** tab, drag a **Join condition** item onto the `ABSENCES_RETRY` job within the `IT_PROCESS` job stream.
5. Drag the connecting arrow from the **Join condition** to the `RESTART_DB` job, and repeat this action for the `RESTART_WAS` and `TEMP_CLEANUP` jobs.
6. In the **Join condition** properties panel, in the **Name** field, enter `PROBLEM_SOLVING`.
7. Set the **How many** value to 1 to specify that at least one of the dependencies must be satisfied.
8. Select the arrow that connects `RESTART_DB` to the `PROBLEM_SOLVING` join dependency and click the **Edit** icon.
9. In **Status condition**, select **SUCC** from the menu and repeat this action for the `RESTART_WAS` and `TEMP_CLEANUP` jobs.
10. Deploy the workspace.

Results

The `PROBLEM_SOLVING` join condition aggregates the three conditional dependencies on the `ABSENCES_RETRY` job. For the `ABSENCES_RETRY` job to run, at least one predecessor job must satisfy the condition by completing in the `SUCC` state.

Example

You can define the same scenario by using Orchestration CLI:

```
SCHEDULE WK1#PROCESSINFO
ON RUNCYCLE RULE1 "FREQ=DAILY;"
AT 1800
 :
WK1#ABSENCES_RETRY
JOIN PROBLEM_SOLVING 1 OF
 [DESCRIPTION “...”]
  FOLLOWS WK2#RESTART_DB IF SUCC
  FOLLOWS W32#RESTART_WAS IF SUCC
  FOLLOWS W32#TEMP_CLEANUP IF SUCC
ENDJOIN
END
```

For more information about defining a join from Orchestration CLI, see join on page 352.

Scheduling and submitting conditional dependencies

You schedule HCL Workload Automation jobs by defining them in job streams.

You can use the Dynamic Workload Console or the **conman** command line to schedule and submit jobs.

After you define an HCL Workload Automation job, add it to a job stream with all the necessary scheduling arguments and submit it to run.

How to schedule a job or job stream using the Dynamic Workload Console

To schedule a job or job stream based on specific criteria, see the section about designing your workload in *Dynamic Workload Console User's Guide*

How to schedule a job or job stream using the command line

To schedule a job or job stream based on specific criteria, see the section about controlling job and job stream processing in *User's Guide and Reference*.

How to submit a job or job stream using the Dynamic Workload Console

To submit a job or job stream to run according to a defined schedule, see the section about submitting workload on request in production in *Dynamic Workload Console User's Guide*.

How to submit a job or job stream from the conman command line

To submit a job for processing, see the section about the **submit sched** command in *User's Guide and Reference*. To submit a job to be launched, see the section about the **submit job** command in *User's Guide and Reference*.

Evaluating and processing a conditional dependency flow

After you submit your jobs or job streams, HCL Workload Automation adds them to the production plan and evaluates them.

Conditional dependencies increase the flexibility of your workload by choosing which job to run as a result of the job status or of the output of a previous job.

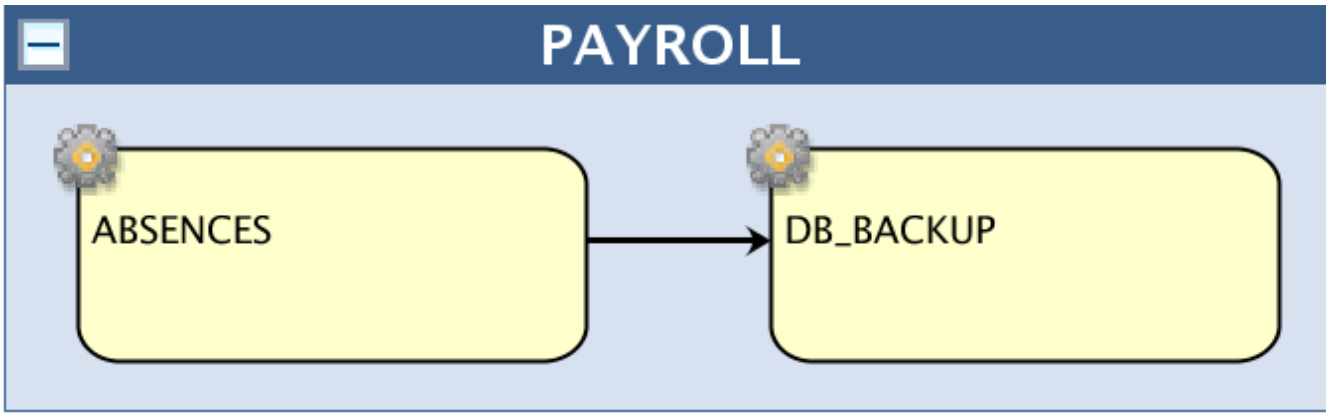
Conditional dependencies are evaluated after any standard dependencies in the job or job stream are satisfied.

If you rerun a job or job stream, the evaluation of the conditional dependency flow is cleared and all dependencies are evaluated again.

If a predecessor job or job stream is cancelled, and the predecessor is in a final state, then the output condition or status condition is evaluated and may or may not be satisfied, or may not be assessable. If the predecessor is not in a final state and has not evaluated any conditions, then the successor remains in HOLD and the job or job stream remains in STUCK.

Follows dependency

A follows dependency is satisfied when the job on which the dependency is defined completes successfully.
Figure 48. A follows dependency on the ABSENCES job

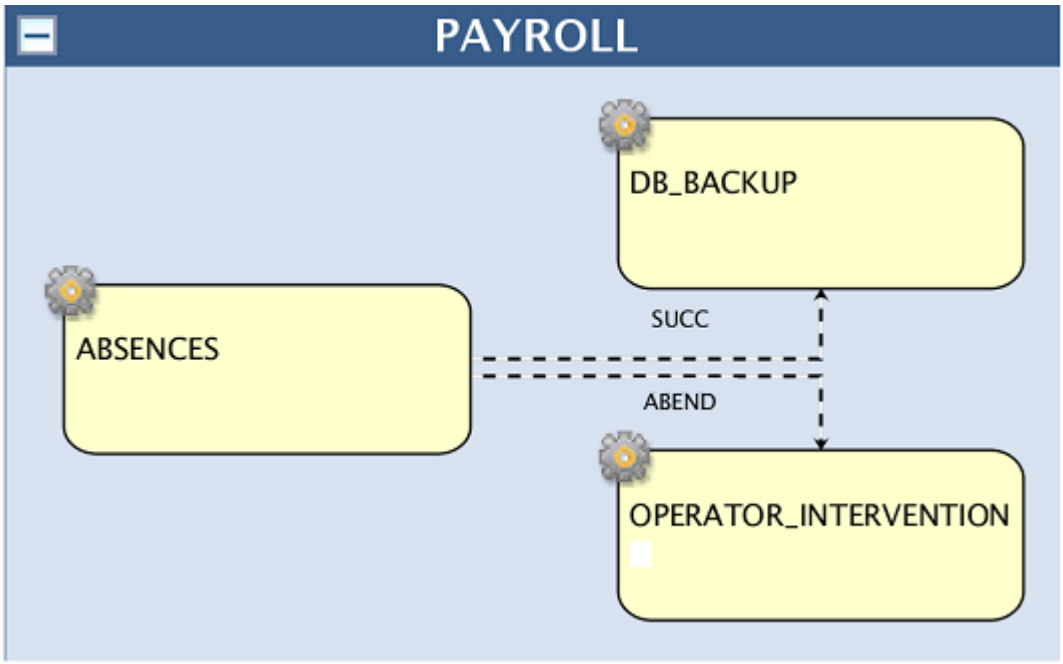


In this example, the DB_BACKUP job can start only after the ABSENCES job completes successfully. If the ABSENCES job does not complete successfully, the DB_BACKUP job remains in HOLD status.

Conditional dependencies on job status

You can use conditional dependencies to make your workload more flexible and automate some recurring tasks. For example, you can decide that a certain job must run when a predecessor completes successfully and a different job must run when the predecessor fails.

Figure 49. Two different conditional dependencies on SUCC and ABEND statuses on the ABSENCES job



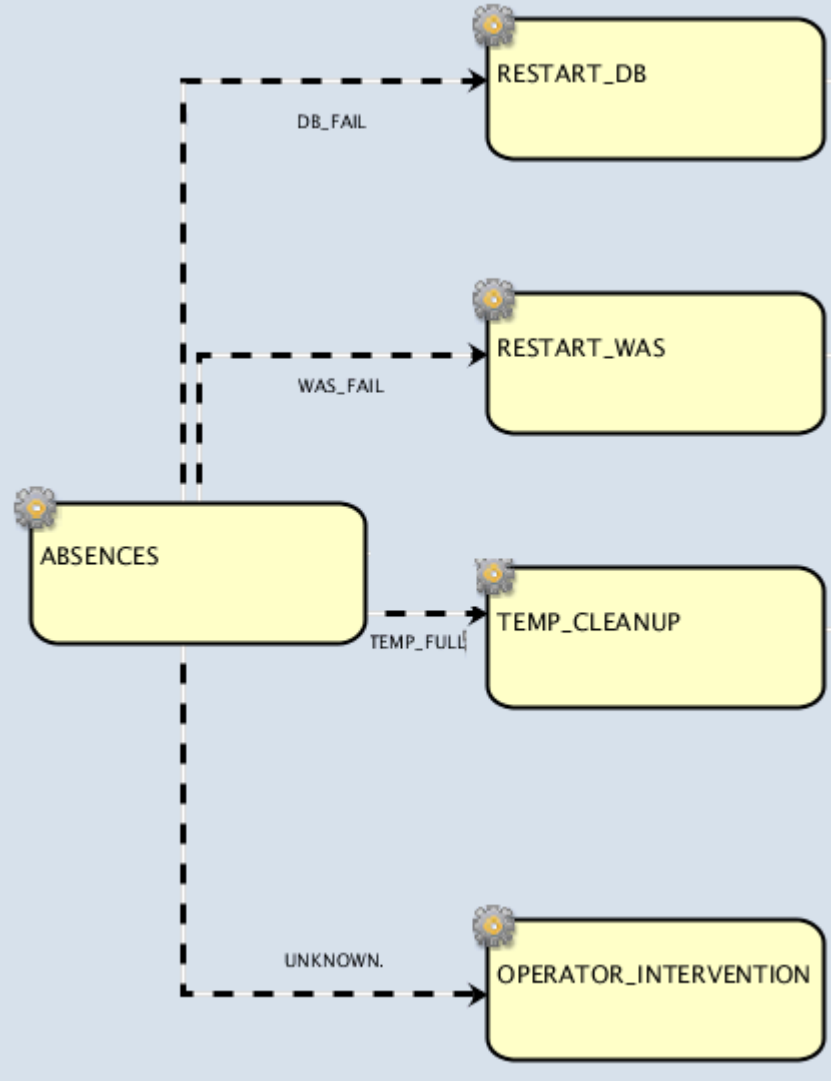
In this example, if the ABSENCES job completes successfully, the DB_BACKUP job runs and the OPERATOR_INTERVENTION job is suppressed. On the contrary, if the ABSENCES job ends in ABEND status, the OPERATOR_INTERVENTION job runs and the DB_BACKUP job is suppressed. If you have a standard follows dependency on the DB_BACKUP job, for example, and the job is suppressed, the follows dependency is released and the successor job can run. If you want to propagate the SUPPR status from the DB_BACKUP job, for example, to a successor job, you define a conditional dependency on the SUPPR status on the DB_BACKUP job. This dependency causes the successor of the DB_BACKUP job to go into SUPPR status when the DB_BACKUP job ends in SUPPR status

If all the conditional dependencies defined in the job stream are satisfied, the job or job stream goes into READY status. If a conditional dependency is not satisfied, the related successor job or job stream goes into SUPPR status.

Conditional dependencies on job output conditions

You can also condition the behavior of your workload based on the job return code and automate a set of responses based on the problems encountered by the predecessor job. There are a number of reasons why the ABSENCES job might fail and some of them can be easily anticipated and solved. The job might fail because the database is down, because WebSphere Application Server Liberty is down, and so on, or an unexpected problem might arise, which requires the intervention of an operator.

Figure 50. Conditional dependencies on output conditions on the ABSENCES job



When defining the ABSENCES job, you associate a specific return code to the problems that might arise, so that a specific job is started to try and solve the problem. For example, a return code of 1 indicates that the database cannot be reached and causes the DB_RESTART job to start, which starts the database; a return code of 2 indicates that WebSphere Application Server Liberty cannot be reached and causes the WAS_RESTART job to start, which starts the WebSphere Application Server Liberty, and so on. Any return code greater than 3 indicates that an unexpected error has occurred and starts the OPERATOR_INTERVENTION job, which alerts the operator.

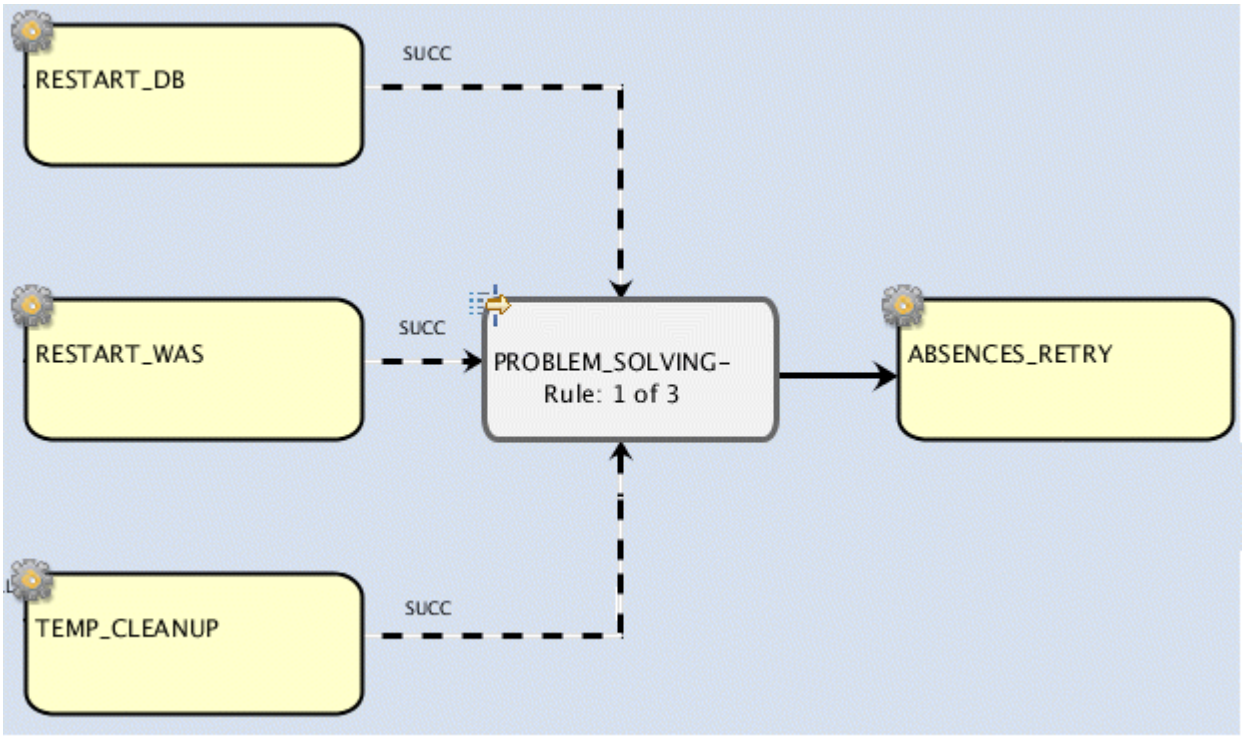
If the ABSENCES job fails with one of the return codes defined in the output conditions, the corresponding job is started, while the remaining jobs are suppressed.

When no output conditions are satisfied, the job or job stream remains in HOLD status.

Join conditional dependencies

You can also combine a set of dependencies into a join dependency and specify how many of them must be met for the join dependency to be satisfied.

For example, consider this portion of the PAYROLL job stream:
Figure 51. A join dependency containing three dependencies on SUCC status



In this case, the PROBLEM_SOLVING join dependency contains three dependencies on SUCC status on three different jobs. This means that when at least one of the RESTART_DB, RESTART_WAS or TEMP_CLEANUP jobs completes successfully, the join dependency is satisfied and the ABSENCES_RETRY job can start.

If none of the predecessor jobs completed successfully, the PROBLEM_SOLVING join dependency is not satisfied and the ABSENCES_RETRY job is suppressed.

If the number of conditional dependencies defined in the join dependency is satisfied, the job or job stream goes into READY status. If the specified number of conditional dependencies in a join dependency is not satisfied, the job or job stream goes into SUPPR status.

Evaluating conditional dependencies in job streams

The evaluation of conditional dependencies in job streams depends on several factors, as described in the following examples.

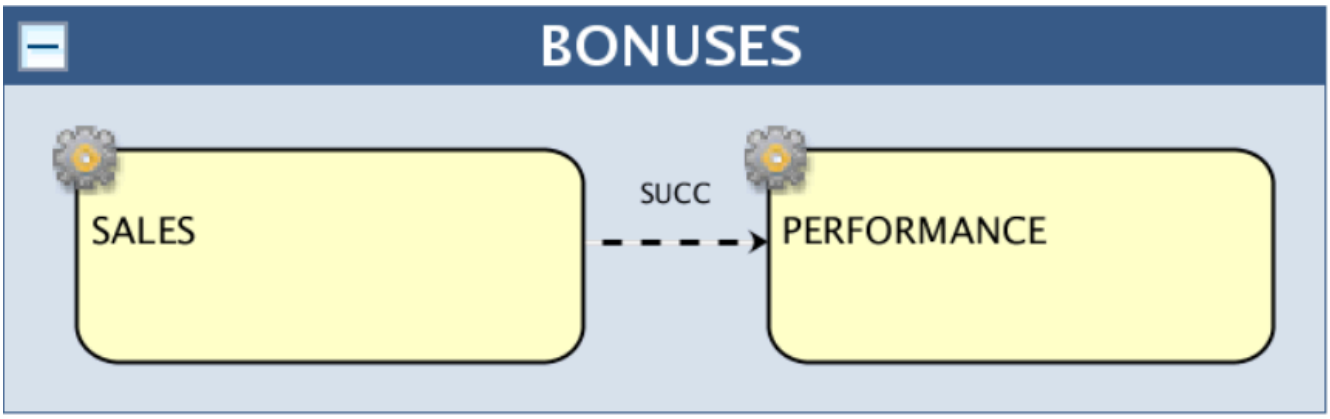
When a job within the job stream is in SUPPR status, its status is evaluated as CANCELLED. If all jobs within the job stream are in SUPPR status, the job stream goes into SUCC status. This is the same behavior that causes a job stream containing only CANCELLED jobs to go into SUCC status.

When you change a job stream status to SUPPR, all the jobs in the job stream that have not reached a final status are changed into SUPPR status. This applies, for example, to jobs that are in READY or HOLD status.

NON-SATISFIED STATUS CONDITION AND RECOVERY STOP SETTING CAUSING THE JOB STREAM TO COMPLETE IN ABEND STATUS

The PERFORMANCE job has a conditional dependency on the SALES job completing in SUCC status. However, the SALES job completes in ABEND status. The conditional dependency on the SALES job is evaluated as unsatisfied and the PERFORMANCE job is suppressed. The job stream completes in ABEND status because the abended SALES job is set to **recovery stop**, and the suppressed status of the PERFORMANCE job is considered as a CANCELLED status.

Figure 52. Status conditional dependency on a job with **recovery stop** setting

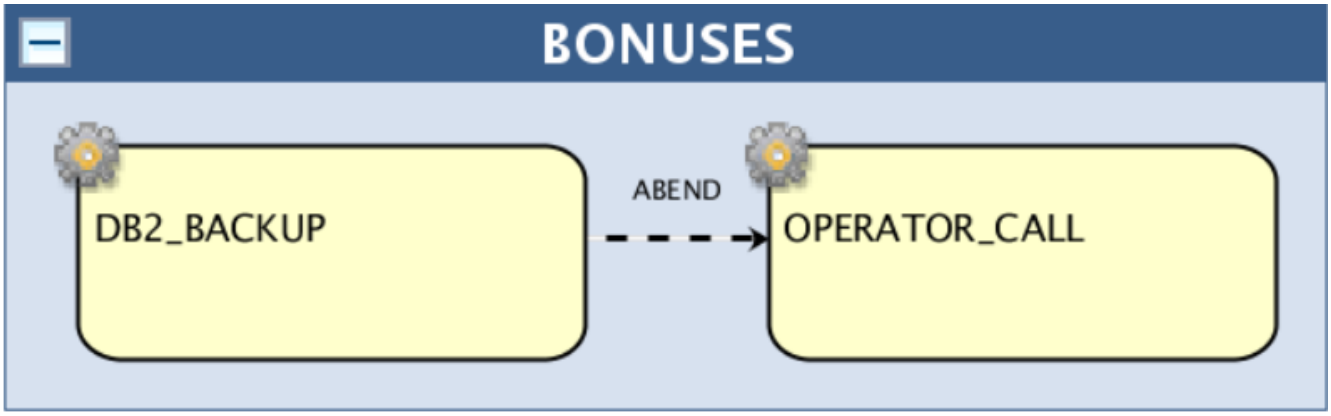


For more information about the **recovery stop** setting, see the section about defining job recovery actions in *User's Guide and Reference*.

NON-SATISFIED STATUS DEPENDENCY CAUSING THE JOB STREAM TO COMPLETE IN SUCC STATUS

The OPERATOR_CALL job has a conditional dependency on the DB2_BACKUP job completing in ABEND status. However, the DB2_BACKUP job completes in SUCC status. The conditional dependency on the DB2_BACKUP job is evaluated as not being satisfied and the OPERATOR_CALL job is suppressed. The job stream status is evaluated in SUCC status because the suppressed status of the OPERATOR_CALL job is considered as a cancelled status.

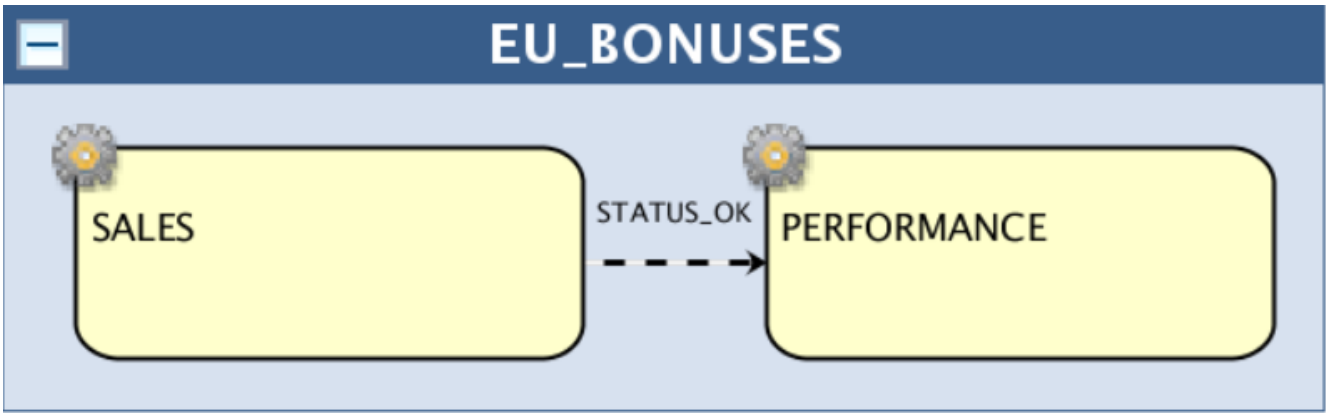
Figure 53. ABEND status conditional dependency



UNSATISFIED STATUS DEPENDENCY CAUSING THE JOB STREAM TO COMPLETE IN STUCK STATUS

The PERFORMANCE job has a conditional dependency on the SALES job completing with the STATUS_OK output condition. However, the SALES job ends in ABEND status and no output condition is satisfied. As a result, the conditional dependency on the SALES job is not evaluated and the job stream completes in STUCK status.

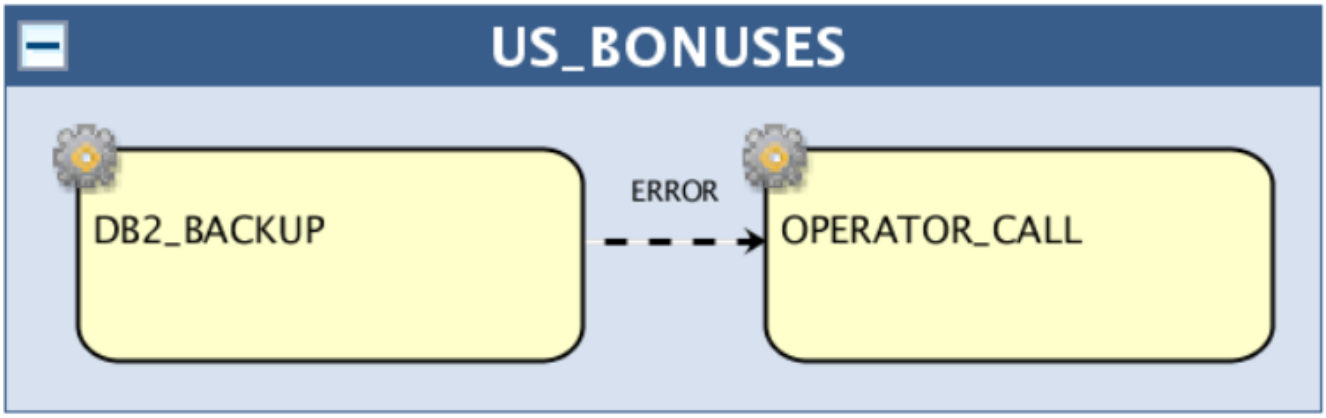
Figure 54. STATUS_OK output condition



SATISFIED OUTPUT CONDITION AND RECOVERY STOP SETTING CAUSING THE JOB STREAM TO COMPLETE IN ABEND STATUS

The OPERATOR_CALL job has a conditional dependency on the DB2_BACKUP job completing with the ERROR output condition. The DB2_BACKUP ends in ABEND and the ERROR output condition is satisfied. As a result, the conditional dependency on the DB2_BACKUP job is evaluated as satisfied. The OPERATOR_CALL job completes in SUCC status. However, the job stream completes in ABEND status because it contains at least one job in ABEND status set to **recovery stop**. To have the job stream complete in SUCC status, set the DB2_BACKUP job to **recovery continue**.

Figure 55. ERROR output condition

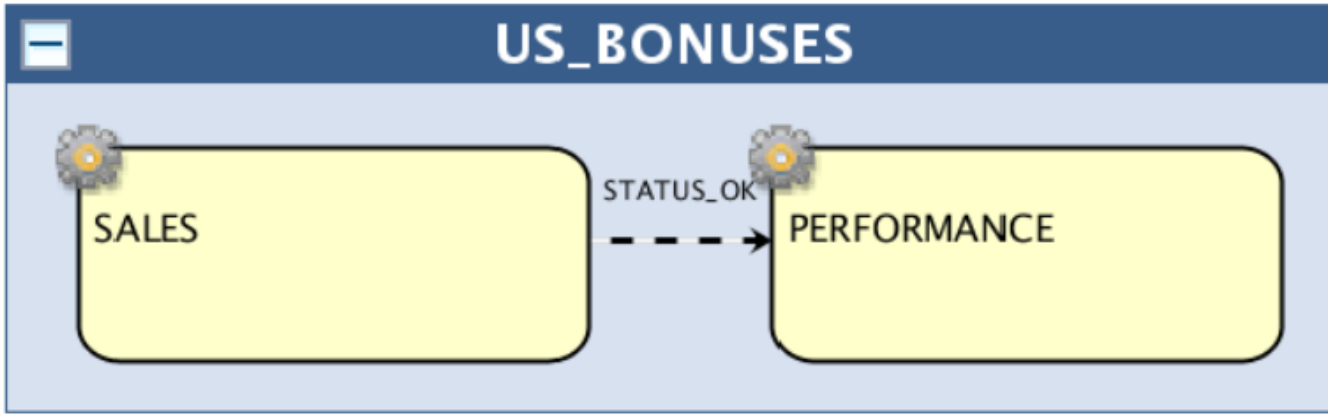


NON-SATISFIED OUTPUT CONDITION AND RECOVERY STOP SETTING CAUSING THE JOB STREAM TO COMPLETE IN ABEND STATUS

The PERFORMANCE job has a conditional dependency on the SALES job completing with the STATUS_OK output condition. However, the SALES job completes in ABEND status and the STATUS_OK output condition is not satisfied. As a result, the

conditional dependency on the SALES job is evaluated as not satisfied and the PERFORMANCE job is suppressed. The job stream completes in ABEND status because the abended SALES job is set to **recovery stop** and the suppressed status of the PERFORMANCE job is considered as a CANCELLED status.

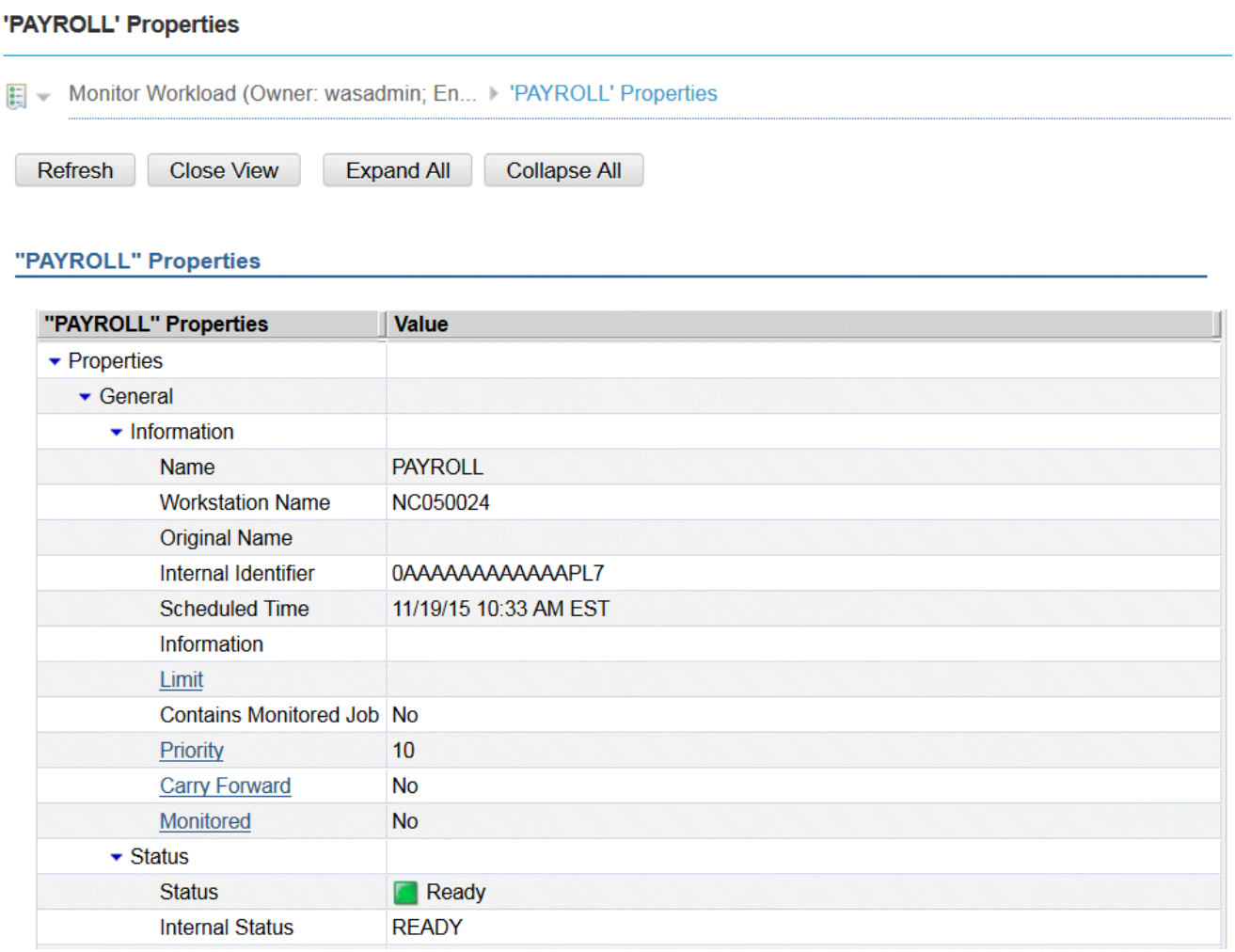
Figure 56. STATUS_OK output condition



Monitoring conditional dependencies

Monitor HCL Workload Automation jobs by using the Dynamic Workload Console or the **conman** command line.

The Dynamic Workload Console displays the following sample information:



You can monitor output conditions using the **conman showjobs** and **showschedules** command line.

In this example, you can see the status of the PAYROLL job stream after submission and before it is run:

```
%ss @#PAYROLL
                                                    (Est)  (Est)    Jobs  Sch
Workstation      Job Stream       SchedTime   State Pr Start  Elapse   #  OK Lim
NC050024        #PAYROLL          0958 11/19  HOLD  10                 7   0
```

This example shows the details about the jobs in the PAYROLL job stream:

```
%sj @#PAYROLL
                                                                                (Est)  (Est)
Workstation      Job Stream       SchedTime  Job                      State Pr Start  Elapse ReturnCode Dependencies

NC050024        #PAYROLL          0958 11/19 ************************ READY 10
                                (NC050024_1_1#)OPERATOR_INTERVENTION      HOLD  10                       ABSENCES IF UNKNOWN_ERR
                                (NC050024_1_1#)DB_BACKUP                  HOLD  10                       JOIN SUCCESS 1 OF
                                                                                                          ABSENCES IF SUCC
                                                                                                       ABSENCES_RETRY IF SUCC
                                (NC050024_1_1#)ABSENCES                    READY 10
                                (NC050024_1_1#)RESTART_DB                  HOLD  10                    ABSENCES IF DB_FAIL
                                (NC050024_1_1#)RESTART_WAS                 HOLD  10                    ABSENCES IF WAS_FAIL
                                (NC050024_1_1#)TEMP_CLEANUP                HOLD  10                    ABSENCES IF TEMP_FULL
                                (NC050024_1_1#)ABSENCES_RETRY              HOLD  10                    JOIN PROBLEM_SOLVING 1 OF
                                                                                                             RESTART_DB
```

```
                                                                                RESTART_WAS
                                                                                TEMP_CLEANUP
```

If you run the **showjob;info** command on one of the jobs, you can obtain some details about the job:

```
%sj NC050024#PAYROLL.ABSENCES;info
-------- Restart ---------
Workstation   Job Stream    SchedTime  Job                                  JobFile                 Opt  Job      Prompt
NC050024      #PAYROLL       0958 11/19
                           (NC050024_1_1#)ABSENCES
.......                                                                    oc: DB_FAIL n/a "RC=1"
                                                                           oc: TEMP_FULL n/a "RC=3"
                                                                           oc: UNKNOWN_ERR n/a "RC>3"
                                                                           oc: WAS_FAIL n/a "RC=2"
```

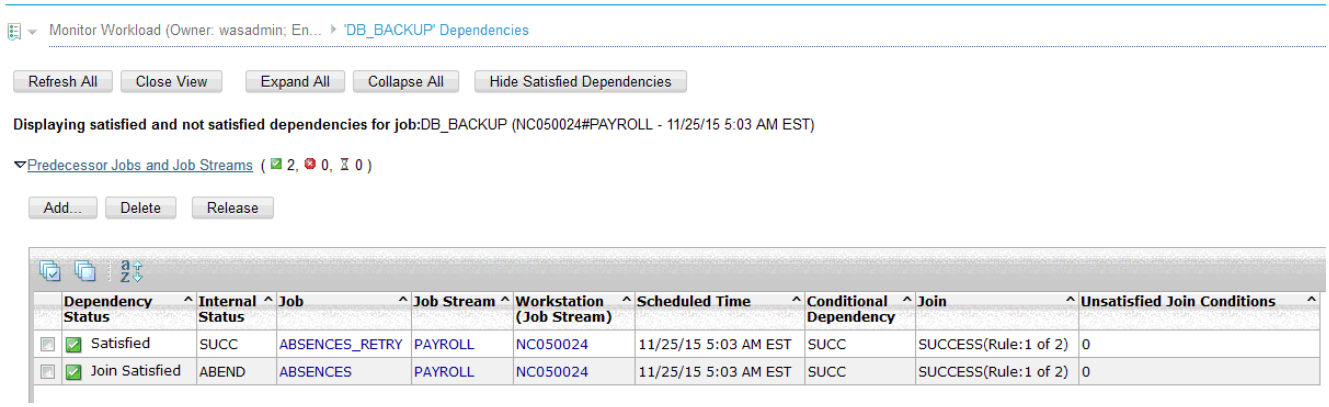
To obtain more details, run the **showjob;props** command. The following sample output is a subset of the information you obtain by running this command:

```
%sj NC050024#PAYROLL.ABSENCES;props
General Information
  Job = ABSENCES
  Workstation = NC050024_1_1
  .....
  Other Output Conditions
    DB_FAIL = n/a "RC=1"
    TEMP_FULL = n/a "RC=3"
    UNKNOWN_ERR = n/a "RC>3"
    WAS_FAIL = n/a "RC=2"

........
```

Monitoring join conditional dependencies

The Dynamic Workload Console displays the following sample information:



Note: From the Dynamic Workload Console, you can retrieve information about dependencies in the join, whether or not they are satisfied, while this is not possible from the command line.

Join dependencies are represented using composer-like syntax. Only unresolved and undecided join dependencies are displayed. If the join dependency is satisfied, no dependencies are displayed.

Here is an example of how the join conditional dependencies are displayed using **conman**. You can see that the PROBLEM_SOLVING join dependency has been satisfied and is no longer displayed. The SUCCESS join dependency will be evaluated next:

```
%sj @#PAYROLL
                                                                                (Est)  (Est)
Workstation      Job Stream       SchedTime  Job                                State Pr Start  Elapse ReturnCode Dependencies

NC050024        #PAYROLL          1007 11/19 ************************************** READY 10 10:08 (00:01)
                              (NC050024_1_1#)OPERATOR_INTERVENTION                 SUPPR 10                         ABSENCES IF UNKNOWN_ERR
                              (NC050024_1_1#)DB_BACKUP                             HOLD  10       (00:01)           JOIN SUCCESS 1 OF
                                                                                                                     ABSENCES IF SUCC
                                                                                                                     ABSENCES_RETRY IF SUCC
                              (NC050024_1_1#)ABSENCES                              ABEND 10 10:08  00:01          1 #J355305499
                              (NC050024_1_1#)RESTART_DB                            SUCC  10 10:08  00:01          0 #J355305500
                              (NC050024_1_1#)RESTART_WAS                           SUPPR 10                         ABSENCES IF WAS_FAIL
                              (NC050024_1_1#)TEMP_CLEANUP                          SUPPR 10                         ABSENCES IF TEMP_FULL
                              (NC050024_1_1#)ABSENCES_RETRY                        SUCC  10 10:09  00:01          0 #J355305501
```

When the job stream completes, the ABSENCES job completed in ABEND status, which caused the RESTART_DB job to start and recover the problem. The REATART_WAS and TEMP_CLEANUP jobs have been suppressed, because they are no longer necessary. The PROBLEM_SOLVING join dependency is now satisfied. The ABSENCES_RETRY job also completes successfully, causing the SUCCESS join dependency to be satisfied, which in turn causes the DB_BACKUP job to start. You can retrieve the following information from the **showjobs** command:

```
%sj @#PAYROLL
                                                                  (Est)  (Est)
Workstation      Job Stream       SchedTime  Job                  State Pr Start  Elapse ReturnCode Dependencies

NC050024        #PAYROLL          1007 11/19 ************************************ ABEND 10 10:08  00:02
   (NC050024_1_1#)OPERATOR_INTERVENTION                 SUPPR 10                         ABSENCES IF UNKNOWN_ERR
   (NC050024_1_1#)DB_BACKUP                             SUCC  10 10:09  00:01          0 #J355305502
   (NC050024_1_1#)ABSENCES                              ABEND 10 10:08  00:01          1 #J355305499
   (NC050024_1_1#)RESTART_DB                            SUCC  10 10:08  00:01          0 #J355305500
   (NC050024_1_1#)RESTART_WAS                           SUPPR 10                         ABSENCES IF WAS_FAIL
   (NC050024_1_1#)TEMP_CLEANUP                          SUPPR 10                         ABSENCES IF TEMP_FULL
   (NC050024_1_1#)ABSENCES_RETRY                        SUCC  10 10:09  00:01          0 #J355305501
```

Note:

There are some differences in the way dependencies are evaluated and resolved when plan replication is enabled. When it is enabled (the default setting), then dependencies are resolved as soon as the conditions necessary to satisfy the dependencies are present. When plan replication is not enabled, then a specific order of evaluation is followed where job streams with dependencies on external job streams or jobs are resolved first, and then dependencies between jobs within the job stream are resolved second.

Similarly, when plan replication is enabled, all dependencies of a job or job stream in suppress state are evaluated. When plan replication is not enabled, the evaluation of dependencies stops as soon as a job or job stream is put in suppress state because of a conditional dependency that was not satisfied. In both of these cases, the end result is

the same, however, when monitoring the progress of the jobs and job streams in these two situations, you might see differing results.

Plan handling of conditional dependencies

Conditional dependencies are managed in the plan from either the conman command line or from the Dynamic Workload Console web user interface.

You can complete the following available actions against jobs where a conditional dependency is defined between the predecessor job and the successor job:

Add a dependency

- Add a dependency to a job: ADDDEP JOB on page 586
- Add a dependency to a job stream: ADDDEP SCHED on page 589

Delete a dependency

- Delete a dependency from a job: DELDEP JOB on page 607
- Delete a dependency from a job stream: DELDEP SCHED on page 609

Release a dependency

- Release a job from its dependencies: RELEASE JOB on page 630
- Release a job stream from its dependencies: RELEASE SCHED on page 632

Confirm the completion

Confirm the completion of job in various status: CONFIRM on page 602

Chapter 25. Defining and managing cross dependencies

HCL Workload Automation ***cross dependencies*** help you to integrate and automate job processing when:

- The workload is spread across different scheduling environments, because some of the activities run at different sites or involve different organizational units or require different skills to be run.
- Even if most of the batch workload is managed locally, none of these environments is completely isolated from the others because they frequently interoperate to exchange or synchronize data and activities.

More specifically, the cross dependency feature is key when you need to synchronize activities between different scheduling environments in an easy way so that you can:

- Define in one scheduling environment dependencies on batch activities that are managed by another scheduling environment.
- Monitor the status of the remote predecessor jobs as if they were running in your local environment.

Additionally, you can control the status of these dependencies by navigating from a unique user interface across the different scheduling environments.

This chapter describes how you define and use cross dependencies.

It contains the following sections:

- An introduction to cross dependencies on page 1352
- Processing flow across the distributed scheduling environment on page 1354
- Defining a cross dependency on page 1357
- How the shadow job status changes until a bind is established on page 1359
- How a z/OS shadow job is bound on page 1362
- How the shadow job status changes after the bind is established on page 1365

Note: Depending on your needs and requirements, you can choose between internetwork dependencies and cross dependencies to establish a dependency between a job running on the local engine and a job running on a remote engine. See Defining dependencies on page 42 for a description of the differences between these two types of dependencies.

An introduction to cross dependencies

A cross dependency is, from a logical point of view, a dependency that a local job has on a job instance that is scheduled to run on a remote engine.

Use cross dependencies to integrate the workload running on different engines, which can be HCL Workload Automation for Z engines (controller), HCL Workload Automation engines (master domain manager and backup master domain manager).

The following objects and terms are used to describe and implement cross dependencies:

Remote engine workstation

A workstation that represents locally a remote HCL Workload Automation engine, either distributed or z/OS. This type of workstation uses a connection based on HTTP or HTTPS protocol to allow the local environment to communicate with the remote environment.

Remote job

A job scheduled to run on a remote HCL Workload Automation engine.

Shadow job

A job defined locally, on a remote engine workstation, which is used to map a remote job. The shadow job definition contains all the information necessary to correctly match, in the remote engine plan, the remote job instance.

Bind

The process to associate a shadow job with a remote job instance scheduled in the remote engine plan.

From a logical point of view in the local environment:

- The remote engine workstation is used to map the remote HCL Workload Automation engine.
- The shadow job, defined on that remote engine workstation, is used to map a remote job instance scheduled to run on that remote HCL Workload Automation engine.

You define a cross dependency when you want that a *local job* (running on your local engine) depends on a *remote job* (running on a remote engine).

To do it, you must do as follows:

1. Create a *shadow* job that runs on your local engine.
2. Define a normal dependency that makes your *local* job dependent on the *shadow* job.

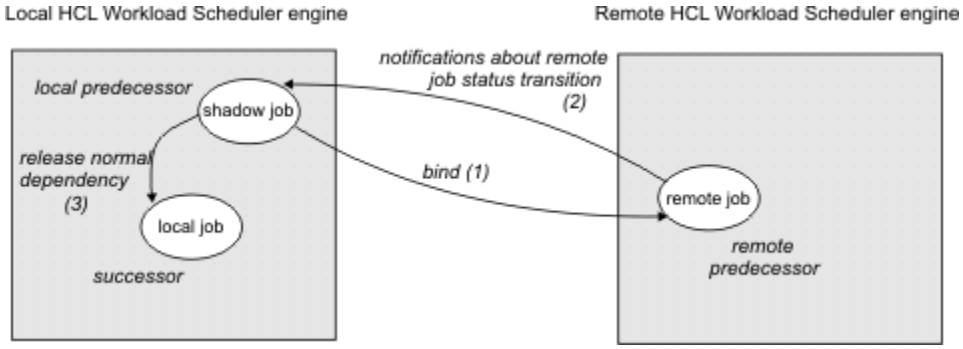
When you create the shadow, consider that

- It must be defined on a workstation of remote engine type, which points to the remote engine (that is the engine where the remote job is scheduled to run).
- You must make it point to the remote job with which you are creating the cross dependency.

Figure 57: Cross dependency logic on page 1354 shows the logical flow implementing cross dependencies:

1. In the bind process, the shadow job is associated to the remote job instance.
2. After the bind is established, the shadow job status is updated according to the remote job status transition.
3. When the shadow job status becomes SUCC the normal dependency of the local job is released, and so also the cross dependency of that local job on the remote job is also released.

Figure 57. Cross dependency logic



Processing flow across the distributed scheduling environment

Depending on whether the local engine emits or receives a bind request, the processing flow and the components involved change. In both cases, the broker workstation in the local environment must be up and running to allow the bind requests management.

Processing flow when the local engine sends a bind request to a remote engine

When you define a shadow job, you specify the information needed to establish a bind with a job in the remote engine plan.

When the shadow job scheduled time arrives, if the shadow job is free from dependencies, it is selected by the local **batchman** for submission and its status is set to INTRO.

The bind request is sent to the remote engine. The shadow job status is set to WAIT.

As soon as the bind processing completes, the remote engine sends back to the local engine a notification with the bind result.

Table 177: Shadow job status transition on page 1354 shows how the shadow job status changes based on:

- Whether the instance to bind exists or not in the remote engine plan.
- The status of the remote job bound.

Table 177. Shadow job status transition

Status of the shadow job in the production plan:	When on the remote engine:
BOUND	**z/OS** The remote job stream instance for the bind was found in the long term plan or in the current plan.

Table 177. Shadow job status transition (continued)

Status of the shadow job in the production plan:	When on the remote engine:
	Distributed The remote job stream instance for the bind was found in the preproduction plan.
ERROR	**z/OS** One of the following situations occurred: • The remote job stream instance for the bind exists neither in the long term plan nor in the current plan. • The remote job stream instance for the bind was found in the long term plan but, when it is included in the current plan, it does not contain the requested job instance. **Distributed** One of the following situations occurred: • The remote job stream instance to bind does not exist in the preproduction plan. • The remote job stream instance for the bind was found in the preproduction plan but, when it is included in the production plan, it does not contain the requested job instance. • The remote bind user is not authorized to access the requested job instance in the production plan.
EXEC	The status of the remote job is EXEC.

Table 177. Shadow job status transition (continued)

Status of the shadow job in the production plan:	When on the remote engine:
SUCC	The status of the remote job is SUCC.
FAIL	The status of the remote job is FAIL.
ABEND	The status of the remote job is ABEND.
SUCC	The status of the remote job is CANCELED.

Note: The status of the shadow job is FAIL also when its submission failed.

For more details about the shadow job status transition, see How the shadow job status changes until a bind is established on page 1359 and How the shadow job status changes after the bind is established on page 1365.

Processing flow when the remote engine receives a bind request from the local engine

When the remote engine receives a bind request from the local engine, the information contained in the request is used to run the bind in the remote preproduction plan.

The bind request also contains an ordered list of URLs that the remote engine uses to send notifications to the local engine. If the local engine is distributed, the list is made up of the URL specified in the JDURL property of the file named *TDWB_HOME*`/config/JobDispatcherConfig.properties`.

Note: By default HCL Workload Automation uses the TWSUser to run the bind in the production plan. If you want to limit and control which jobs can be bound, you can specify a different user using the global option **bindUser**. The user specified does not need to be defined as a user on the operating system, or even have a password, but it must exist as entry in the security file with the following access privileges:

- DISPLAY access to the *job* and *schedule* objects that can be bound
- LIST access to the *job* objects that can be bound. This access is required only if the global option enListSecChk is set to `yes`.

If the required access privileges are not specified, a notification with an error is sent back to the engine that requested the bind. For more information about global options, see the topic about global options - detailed description in *Administration Guide*.

The remote engine sends back to the local engine:

A notification with the status BOUND

If the preproduction plan contains at least one instance of the job stream specified in the bind request and the definition of that job stream contains the job to bind.

A notification with the status of the job instance bound

If the instance of the job to bind is found in the production plan and whenever its status changes.

A notification with an error

If the job instance to bind is not found or if the bind user is not authorized.

The remote **batchman** process writes an entry in the `PlanBox.msg` queue whenever the status of a remote job changes.

Every 30 seconds, the `PlanBox.msg` queue is scanned for new entries that document a change in the status of remote jobs that were bound. Whenever a status change is found, a notification containing the status of the remote job bound is sent back to the engine that requested the bind.

Note: To change the polling interval, specify a value, in seconds, for **com.ibm.tws.planner.monitor.checkPlanboxInterval** in the file `TWSConfig.properties` and then restart the WebSphere Application Server.

Defining a cross dependency

About this task

Perform these steps to define a cross dependency between a job running in your environment and another job running on a different HCL Workload Automation engine:

1. Create a remote engine workstation

   Create a remote engine workstation for a specific remote engine when you need to define dependencies on job instances running on that remote engine. On a remote engine workstation you can run only shadow jobs.

   As a best practice, if the remote HCL Workload Automation engine is distributed, you can define a dynamic pool containing an ordered list of remote engine workstations pointing to the remote master and to its backup masters, to ensure that failover and switch manager capabilities are applied. For more information about workstations pool, see Workstation on page 31.

   Note: It is recommended that:
   - All the distributed environments involved have the timezone feature enabled. For more information, see Enabling time zone management on page 1064.
   - You specify as TIMEZONE property of the remote engine workstations the timezone set on the operating system of the remote master domain managers or backup master domain managers they point to.

   For more information about the specific settings to use when defining a remote engine workstation, see Workstation definition on page 201.

2. Define a shadow job running on the remote engine workstation

Create a shadow job pointing to a specific job instance defined on a remote engine when you want to track in your local environment the status of that remote job and define cross dependencies on that remote job.

On distributed environments, you can use alias for job stream names and job names. If you are defining a distributed shadow job, make sure that:

- The remote job stream name specified, contains the job stream name as it is defined in the database.
- The remote job name specified, contains the alias, if defined, of the remote job to bind.

If you do not follow these guidelines, the bind fails and the status of the shadow job becomes ERROR.

In the shadow job definition set COMPLETE_IF_BIND_FAILS in the *rccondsucc* field to specify if the shadow job status must be forced to SUCC or ERROR if the bind in the remote engine plan fails.

For more information about the specific settings to use when defining a shadow job, see Job definition on page 228.

Depending on whether the remote engine is z/OS or distributed, you can use different matching criteria:

If the remote engine is distributed

You can choose any of the these matching criteria:

Table 178. Matching criteria for distributed shadow jobs

On the Dynamic Workload Console	Corresponding keyword used in composer
Closest preceding	previous
Within a relative interval	relative from=*time_before_scheduled_time* to=*time_after_scheduled_time*
Within an absolute interval	absolute from=*interval_start* to=*interval_end*
Same scheduling date	sameDay

For more information about these matching criteria, see Managing external follows dependencies for jobs and job streams on page 100.

If the remote engine is z/OS based

Closest preceding is the only matching criteria supported by HCL Workload Automation for Z.

The scheduled time of the job stream instance containing the shadow job is used for the matching in the remote engine plan.

The shadow job status transition is mapped to the remote job instance status transition.

3. Add a dependency on the shadow job

You add the cross dependency for a local job on the remote job by defining a dependency for the local job on a shadow job that:

- Points to the remote job instance.
- Is defined on a local workstation that points to the remote engine where the remote job is defined.

The cross dependency on the remote job instance is released when the local dependency on the shadow job is released.

Monitoring a cross dependency resolution in the production plan

Shadow jobs are added to the plan as follows:

- At run time if either of the following situations occurs:
  - A shadow job definition is submitted using the **sbj** command.
  - A job stream containing a shadow job definition is submitted using the **sbs** command.

    Note: When submitting a shadow job, specify a destination job stream with a known scheduled time to better control the remote job instance that will be bound.

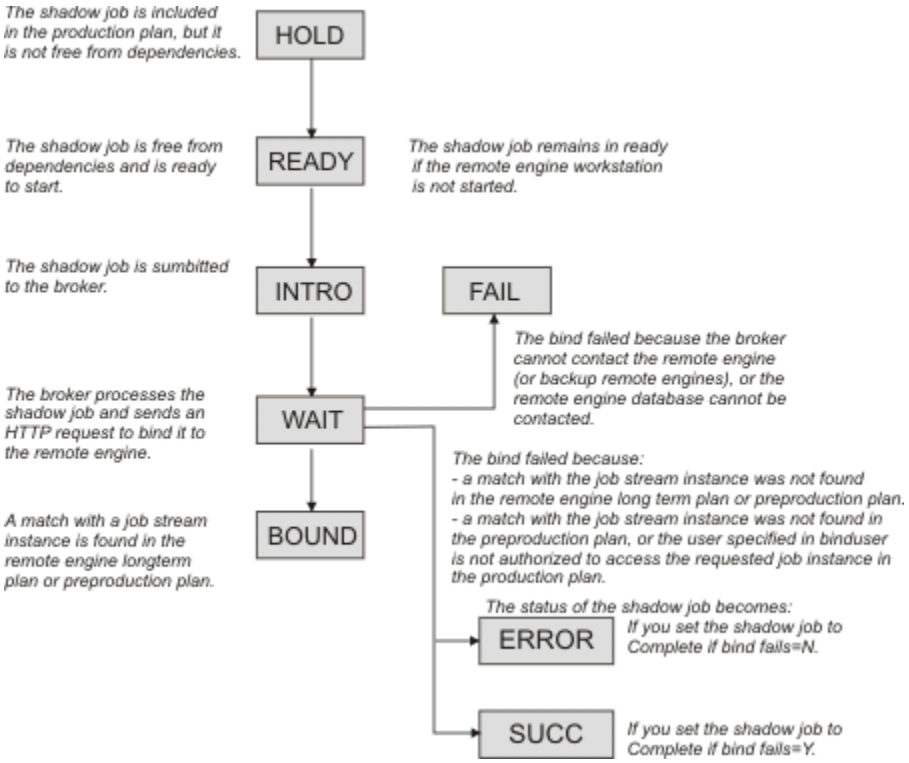
- When the preproduction plan is extended or created and its time frame includes the shadow job scheduled time.

When a shadow job instance is added to the plan, you can start monitoring its status.

How the shadow job status changes until a bind is established

Figure 58: Shadow job status transition until the bind is established on page 1360 summarizes how a shadow job status changes until the bind is established.

Figure 58. Shadow job status transition until the bind is established



As for any other job, the initial status of the shadow job is HOLD and it turns to READY when the job becomes free from dependencies and is ready to start.

The scheduler then sends an HTTP request to the remote engine containing both the information to identify the shadow job in the local production plan and the information to uniquely identify the remote job instance to bind in the remote engine plan, including the matching criteria. The scheduler must also be notified about the status of the remote job instance bound.

The scheduler tries to contact the remote engine, at regular intervals, until a specific timeout expires. If, by then, the remote engine could not be reached, the shadow job status is set to FAIL. To change the timeout and the interval, specify a value, in seconds, for both MaxWaitingTime and StatusCheckInterval in the file *TDWB_HOME*`/config/ResourceAdvisorConfig.properties` and then restart the broker.

If the preproduction plan does not exist on the remote engine when the bind request is received, the distributed shadow job status remains WAIT until the preproduction plan generation is completed and the bind request is processed. This might happen, for example, when the preproduction plan is created again from scratch on the remote engine.

For more information on the reason why the shadow job status is FAIL , see How to see why the shadow job status is FAIL on page 1367.

When the remote engine receives the HTTP request, it tries to identify the job stream instance to use for the bind in its plan; the preproduction plan if the remote engine is distributed or the long term plan if the remote engine is z/OS. The definition of the job stream must contain the definition of the remote job to bind.

For more information about how the match is made in a distributed remote engine plan, see How a distributed shadow job is bound on page 1361.

For more information about how the match is made in a z/OS remote engine plan, see How a z/OS shadow job is bound on page 1362.

How a distributed shadow job is bound

If the remote engine is an HCL Workload Automation master domain manager or backup master domain manager, the search for the remote job instance to bind is done in the preproduction plan. Distributed remote job instances, belonging to the JOBS or USERJOBS job streams, are not involved in the bind process. However, remote jobs that are moved to USERJOBS after binding continue to send status change notifications.

The matching interval, except for the closest preceding matching criteria that does not require interval calculation, is calculated on the remote engine using the settings specified in the distributed shadow job definition.

For example, when the **sameDay** matching criteria is specified, the day that is referred to is the day specified on the remote engine in terms of `[startOfDay, startOfDay+23:59]`.

When using an interval-based matching criteria, the HTTP request sent to the remote engine contains the following information to allow the remote engine to calculate the matching interval:

For absolute interval matching criteria:

The values *hhmm*, *HHMM* and, optionally, *d* and *D*, specified in the clause:

```
<dshadow:matching>
  <dshadow:absolute from="hhmm [+/-d day[s]]" to="HHMM [+/-D day[s]]"/>
  </dshadow:matching>
```

Boundary values are *hhmm* -6 days and *HHMM* +6 days.

The time zone used for the matching criteria is the time zone of the shadow job.

For relative matching criteria:

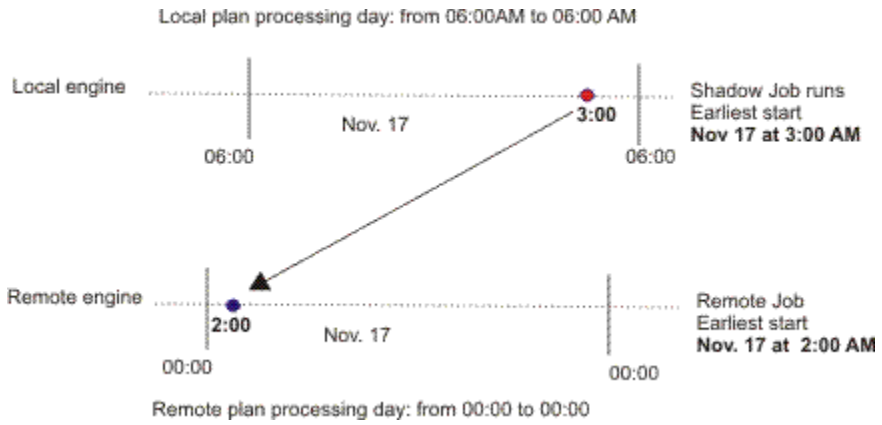
The shadow job scheduled time and the values [*hh*]*hmm* and [*HH*]*HMM* specified in the clause:

```
<dshadow:matching>
  <dshadow:relative from="+/-[hh]hmm" to="+/-[HH]HMM"/>
  </dshadow:matching>
```

Boundary values are +/-167:59 hours.

For example, to create a shadow job that matches a remote job instance whose Earliest start is November, 17 at 2:00 AM, you can specify either of the following matching criteria:

- **Same scheduled date**
- **Within an absolute interval** specifying as an offset: **1 day Before earliest start time.**



The remote job instance to match is identified on the remote engine according to the rules stated for the external follows dependencies. For more details about external follows dependencies resolution criteria, see Managing external follows dependencies for jobs and job streams on page 100.

For more information about defining shadow jobs, see Job definition on page 228.

How a z/OS shadow job is bound

If the remote engine is an HCL Workload Automation for Z controller, the search for the remote instance to bind is done as follows:

- First, the instance is searched in the Long Term Plan (LTP) in the part of the bind interval that follows the Current Plan (CP) end time and precedes the shadow job scheduled time.
- If no instance is found, the instance is searched in the CP in the part of the bind interval that precedes the current plan end.

Note: If the remote controller receives a bind request with a client notify URI that is not defined among the HTTP destinations, the bind request is discarded and the message EQQHT62W is logged in the MLOG.

The following sections describe the scenarios that can occur when binding a z/OS shadow job having:

- Scheduled time: 18:00
- Remote job information:
  - Application ID: **JS2**
  - Operation number: **OP2**

In the figures:

- The white box indicates the time interval covered by the LTP.
- The light grey box indicates the time interval covered by the CP.

- The dark grey box indicates the interval in the remote engine plan during which the job instance to bind must be searched.
- The **JS2** occurrence highlighted in bold is the instance selected for the bind.

Scenario 1: The CP interval contains the shadow job scheduled time and JS2 occurrences exist.

Figure 59. Instance to be bound if the shadow job scheduled time is included in the CP interval

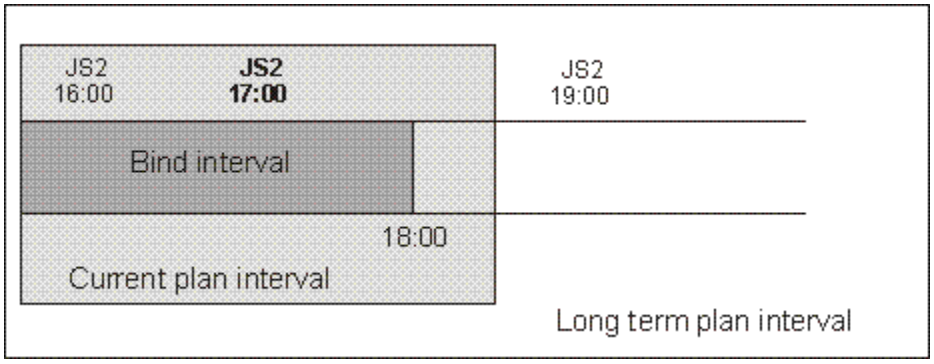


Figure 59: Instance to be bound if the shadow job scheduled time is included in the CP interval on page 1363shows, highlighted in bold, the **JS2** instance that more closely precedes the shadow job scheduled time. This instance is selected for the bind because the scheduled time is contained in the CP. The shadow job and the remote job instance are associated. If, at a later time, a new instance of **JS2** that closest precedes the shadow job scheduled time is submitted ad hoc in the remote engine plan, the match with the **JS2** instance selected for the bind is *not* modified.

At this point, one of the following situations can occur:

The selected JS2 instance contains OP2.

The bind with **OP2** belonging to **JS2** is established and a notification containing:

- The remote job information identifying **OP2** instance in the remote engine plan
- The current status of that **OP2** instance

is sent back, the shadow job instance is updated with the remote job information, and its status is updated accordingly.

The selected JS2 instance no longer contains OP2 because either it was deleted and a daily plan removed it from the CP, or it was never contained in JS2.

The bind fails. A notification informing that the bind failed is sent back, and the shadow job status is updated according to what you set in the **Complete if bind fails** field.

The selected JS2 instance contains OP2 that was deleted but not yet removed from the CP.

The bind is established and a notification informing about the successful execution status is sent back. The shadow job instance is marked as SUCC. Its successors can start.

Scenario 2: The current plan interval contains the shadow job scheduled time, the JS2 instance that most closely precedes the shadow job scheduled time exists in the LTP but was canceled from the CP.

Figure 60. Instance to be bound if the instance that most closely precedes the shadow job scheduled time exists in the LTP but was canceled from the CP

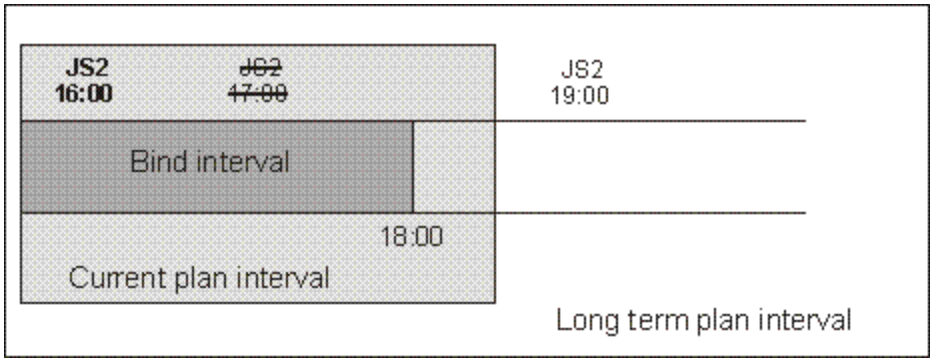


Figure 60: Instance to be bound if the instance that most closely precedes the shadow job scheduled time exists in the LTP but was canceled from the CP on page 1364 shows, highlighted in bold, the **JS2** instance that is selected for the bind, because the occurrence that better matched was deleted.

The bind with **OP2** belonging to **JS2** is established and a notification containing:

- The remote job information identifying the **OP2** instance in the remote engine plan
- The current status of that **OP2** instance

is sent back, the shadow job instance is updated with the remote job information, and its status is updated accordingly.

Scenario 3: The CP interval contains the shadow job scheduled time but no JS2 occurrence exist.

Figure 61. The scheduled time of the shadow job is included in the CP but no instance to bind exists

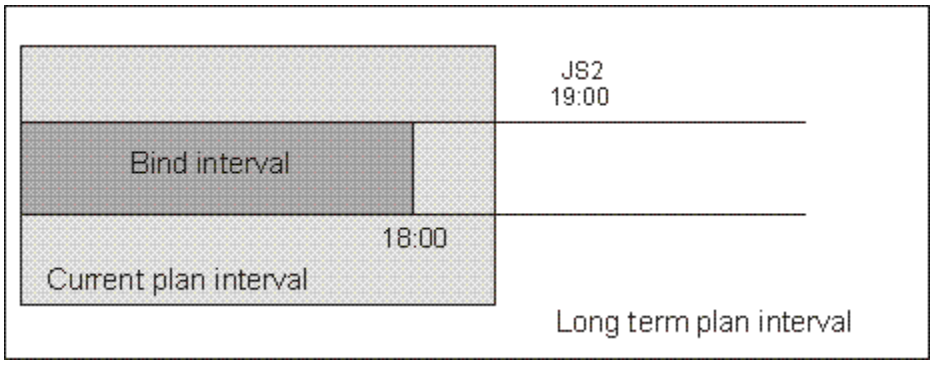


Figure 61: The scheduled time of the shadow job is included in the CP but no instance to bind exists on page 1364 shows that a **JS2** instance that closely precedes the shadow job scheduled time does not exist.

The bind fails. A notification informing that the bind failed is sent back, and the shadow job status is updated according to what you set in the **Complete if bind fails** field.

Scenario 4: The LTP interval contains the shadow job scheduled time and the CP does not yet include the closest preceding JS2 instance.

Figure 62. The instance to be bound exists but it is not yet included in the CP

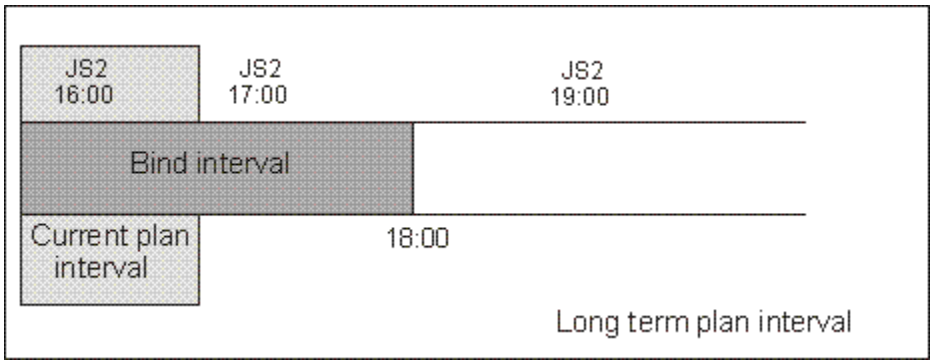


Figure 62: The instance to be bound exists but it is not yet included in the CP on page 1365 shows the **JS2** instance that can be associated with the shadow job, even though the job **JOB2** is not yet in the CP.

A notification informing that the bind is established is sent back and the status of the shadow job is set to BOUND.

Scenario 5: The LTP interval still does not contain the shadow job scheduled time.

Figure 63. The LTP interval still does not contain the shadow job scheduled time

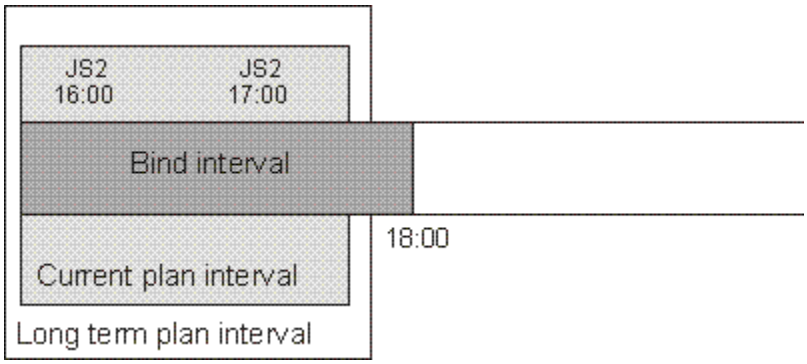


Figure 63: The LTP interval still does not contain the shadow job scheduled time on page 1365 shows that no **JS2** instance can be associated with the shadow job because, until the LTP includes the shadow job scheduled time, closer preceding **JS2** instances can still be added.

In this case, the bind request is put in hold until the LTP is extended to include the shadow job scheduled time. Until then the status of the shadow job remains WAIT.

How the shadow job status changes after the bind is established

When a bind is established, the remote engine sends back an HTTP notification containing the status of the bind and, if the bind was successful, the information to identify the remote job instance bound. This information is shown in the shadow job instance details.

Depending on the type of a remote engine, the following information about the remote job instance is shown in the shadow job properties:

The remote engine type is distributed

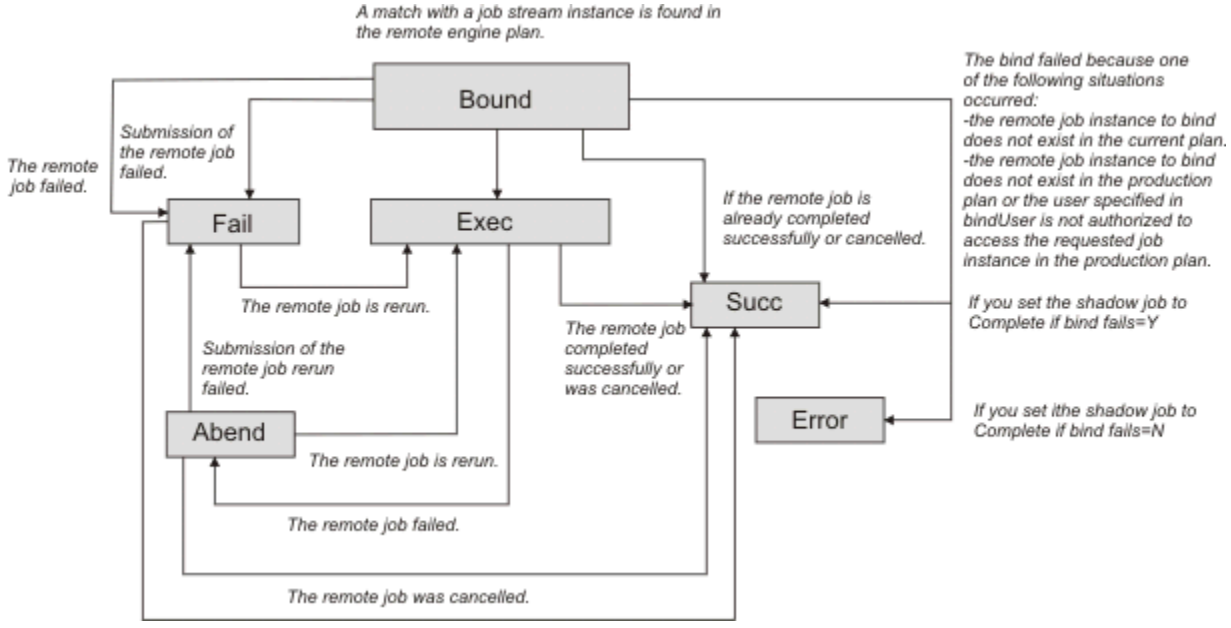
- Job stream name
- Scheduled time
- Job stream workstation
- Job name

The remote engine type is z/OS

- Application ID
- Scheduled time
- Operation number
- Workstation
- Job name, if it was defined on the remote engine.

When the shadow job instance is mapped to an existing remote job instance, notifications about job status changes are sent asynchronously from the remote engine. These notifications are used to map remote job status transition to shadow job status transition. The store and forward mechanism ensures the delivery of the messages and the recovery in case of failures. Figure 64: Shadow job status transition chain after the bind was established on page 1366 shows how the status of a distributed shadow job changes, from when a bind is established until the shadow job status becomes SUCC or ERROR. Only status SUCC and ERROR are considered as the final status for a shadow job.

Figure 64. Shadow job status transition chain after the bind was established



If the remote job instance is already completed when the match is done, the shadow job status becomes SUCC immediately.

For more information on the reason why the shadow job status is FAIL , see How to see why the shadow job status is FAIL on page 1367.

When the shadow job status satisfies the dependency rule, the dependency of the local job on the shadow job is resolved, and the cross dependency for the local job on the remote job is also resolved.

How to see why the shadow job status is FAIL

The shadow job status can be FAIL in one of the following situations:

- The shadow job submission failed.
- The submission of the remote job failed.

To determine why the shadow job status is FAIL, see the log of the shadow job either by running the showjobs command with the ;stdlist option, or by clicking **Job Log...** for the shadow job instance in the **Monitor jobs** view on the Dynamic Workload Console.

Shadow job status during the remote job recovery or rerun

After the bind is established it might happen that the remote job bound is rerun, in this case the status of the shadow job reflects the status of the rerun job. The shadow job status remains EXEC while the remote job recovery is in progress.

The shadow job status is updated only when the remote job reaches one of the following states:

ABEND

When the remote job fails to run.

SUCC

When the remote job succeeds.

FAIL

When the remote job submission fails.

You can see more details about the remote job in the shadow job properties. To see these details:

- Run the **conman** command **showjobs** with the **props** option against the shadow job.
- Access the shadow job properties panel in the Dynamic Workload Console.

How carry forward applies to cross dependencies

Carry forward works the same way with shadow jobs as it does with other types of jobs. Shadow jobs in **WAIT** and **BOUND** status are treated just like jobs in **EXEC** status. Shadow jobs in **ERROR** status are treated like jobs in **FAIL** or **ABEND** status.

The status of a shadow job, bound to a remote job that is not carried forward, is set to ERROR when the remote production plan is extended.

Note: As a best practice, use cross dependencies with carry forward job streams on both local and remote distributed scheduling environments.

For more information about the **carryStates** global option, see the topic about global options - detailed description in *Administration Guide*.

Managing shadow jobs in the production plan

Depending on the status of the shadow job, you can run the following commands:

Kill

You can kill a shadow job with status BOUND, EXEC, or WAIT. The association established through the bind with the remote job is automatically canceled and the status of the shadow job is set to ABEND with return code 0.

Rerun

You can rerun a shadow job with status ABEND, ERROR, SUCC, or FAIL. When you rerun a shadow job a new bind request is triggered.

Chapter 26. Managing an IBM i dynamic environment

About this task

Managing IBM i agents in a dynamic environment and scheduling jobs with advanced options on IBM i agents.

Defining agents on IBM i systems

About this task

To begin scheduling jobs on IBM i agents, the agent must be in the HCL Workload Automation network. At the end of the installation process, the agent is automatically registered in the HCL Workload Automation database.

You can check the existence of the workstation definition of the agent installed on an IBM i system either by using the Dynamic Workload Console or by using the composer command line.

For information about using the console to see the workstation definitions, see the *Dynamic Workload Console User's Guide*, section about Designing your scheduling environment.

For information about using the command line interface to see the workstation definitions, see Workstation definition on page 201.

To include the IBM i agent in the plan, see *HCL Workload Automation Planning and Installation: Part 3. HCL Workload Automation on IBM i - Configuring a dynamic agent*.

Defining jobs on IBM i systems

About this task

On IBM i agents you can define the following types of jobs with advanced options:

Executable jobs

To define executable jobs, see Executable jobs on page 255.

IBM i jobs

To define IBM i jobs that run IBM i operating systems native commands, see IBM i jobs on page 250.

remote command jobs

To define remote command jobs, see Remote command jobs on page 910.

IBM WebSphere MQ jobs

To define IBM WebSphere MQ jobs, see the section about the integration on Automation Hub.

IBM Sterling Connect:Direct jobs

To define IBM Sterling Connect:Direct jobs, see the section about the integration on Automation Hub.

You can define jobs with advanced options on an IBM i agent either by using the Dynamic Workload Console or by using the composer command line.

For more information about the procedure for defining IBM i job definitions, see the sections about the prerequisite steps to create job types with advanced options and about creating job definitions in *HCL Dynamic Workload Console User's Guide*.

Also check the integrations available on Automation Hub for new and updated integrations.

For information about using the command line interface to create job definitions, see Job definition on page 228.

Managing agents on IBM i systems

About this task

You can use HCL Workload Automation on IBM i agents only to start and stop the agent processes. For more information about starting and stopping IBM i agents, see Starting and stopping agents on IBM i systems on page 1370.

To manage the IBM i agent, use the utilities described in Using utility commands for agents on IBM i systems on page 1370.

Starting and stopping agents on IBM i systems

About this task

You can use HCL Workload Automation on IBM i agents only to start and stop the agent processes.

Starting agents on IBM i systems:

Use the utility StartUpLwa.

For information about the utility to start agents on IBM i, see StartUpLwa - Start the agent on page 972.

Stopping agents on IBM i systems:

Use the utility ShutDownLwa.

For information about the utility to stop agents on IBM i, see ShutDownLwa - Stop the agent on page 971.

Using utility commands for agents on IBM i systems

About this task

For agents on IBM i systems, you can use the following utilities:

param

To use param utility, see param on page 990.

twstrace

To use twstrace utility, see TWSTRACE on page 1007.

resource

To use resource utility, see resource on page 993.

cpuinfo

To use cpuinfo utility, see cpuinfo on page 921.

version

To use version utility, see version on page 973.

Scheduling and monitoring jobs on IBM i systems

About this task

When scheduling a job on IBM i systems, the job launches a native command that can be either a system or a user command. For example, the native command might consist of SBMJOB system command, which launches a batch job. The native command can starts one or more IBM i programs. The IBM i programs can be monitored only if they are started by the native command.

You can specify the name of the queue where the monitoring agent component runs by using the **MonitorQueueName** property in the native job launcher section of the `JobManager.ini` file. If you do not specify this property, the default queue (QBATCH) is used.

For more information, see the section about configuring properties of the native job launcher [NativeJobLauncher] in *Administration Guide*.

IBM i programs might generate runtime inquiry messages that require a reply before the program can continue to run. While monitoring IBM i jobs, HCL Workload Automation operators have to check the IBM i console to find inquiry messages waiting for a reply. HCL Workload Automation provides a set of useful features that help operators detecting and replying to inquiry messages.

Check of inquiry messages waiting for a reply

You can use the Dynamic Workload Console and **conman showjobs** command line to check whether an IBM i job is waiting for a reply to a message. An IBM i job that is waiting for a message reply is in the **SUSP** (suspended) status. This status indicates that the job is waiting for input while running. When the input is received, the job status changes to **EXEC** (executing).

For more information about job statuses, see the section about status description and mapping for distributed jobs in *Dynamic Workload Console User's Guide*.

Direct reply to inquiry messages from the Dynamic Workload Console

When an IBM i job is waiting for a reply to a message, you can reply to the message directly from the **Monitor Workload** of the Dynamic Workload Console. The job in SUSP (suspended) status requires your attention on additional information to be displayed. Click on the hyperlink. A pop-up window shows the message that is waiting for your reply. Reply to the message in the pop-up window, then select one of the following actions:

Forward action

To forward your reply. A message in the pop-up window confirms that your reply was sent successfully.

Cancel action

To cancel your reply. The pop-up window is closed.

Note: For a correct display of the pop-up window that shows the message waiting for your reply, your master domain manager must be at version 9.3.0.2.

Monitoring and reply to messages for IBM i child jobs

While you are monitoring a parent job from the Dynamic Workload Console, you can monitor also the child jobs for that parent job. When the parent job is in **SUSP** (suspended) status, you can reply to messages for the parent job and also for the child jobs.

Automated reply to inquiry messages

For the most frequent inquiry messages, you can even define standard rules to automate the reply to the waiting messages. When defining an IBM i job, by using the Workload Designer of the Dynamic Workload Console or the composer command line, specify the list of messages for which you want to set up an automated reply. For each message, specify:

Message Id

The message identifier.

Message Text

The message text.

Message Reply

The automated reply that you want to define.

Message Max Replies

The maximum number of automated replies accepted for this specific message. Valid range is from 0 to 100. Default value is 10. If 0 is specified, the automated reply to the message is disabled.

This parameter optimizes the management of IBM i inquiry messages. For example, when you set a wrong reply to a message in the job definition, IBM i system keeps on sending out the same inquiry message repeatedly while waiting for the a correct reply. To avoid this issue, HCL Workload Automation has the capability to intercept and disable the wrong automatic reply and require, with a prompt, a direct reply from the Dynamic Workload Console. The job remains in SUSP (suspended) status until the correct reply is provided.

For more information, see the section about job definition for IBM i jobs in *User's Guide and Reference*.

Logging of inquiry messages

If an IBM i job generates inquiry messages, the messages and the related replies are written into the correspondent HCL Workload Automation output job log so that the HCL Workload Automation operator can keep track of them.

Reliable monitoring of IBM i job status changes

As an inquiry message receives an automated reply, the IBM i job status changes from SUSP (suspended) to EXEC (executing) and vice versa. All the job status changes are monitored and tracked. This is useful, for example, when you want to create an event rule definition to send an email every time a job status change occurs.

Improved trace facilities

To track an IBM i job, run the following steps:

1. Activate the trace facilities on the IBM i system, by running the following commands:

   ```
   ADDENVVAR   ENVVAR(DMON_TRACE_ENABLED)  VALUE('true')  LEVEL(*SYS)
   ADDENVVAR   ENVVAR(DMON_TRACE_LEVEL)  VALUE('trace_level')  LEVEL(*SYS)
   ```

   where *trace_level* indicates the tracking level and can have one of the following values:
   - **1**: DEBUG MIN
   - **2**: DEBUD MID
   - **3**: DEBUG MAX
2. Customize your IBM i agent, by properly setting the following configuration parameters in the ITA section of JobManager.ini file, for example:

   ```
   DMON_TRACE_ENABLED = true
   DMON_TRACE_LEVEL = trace_level
   ```

   where *trace_level* indicates the tracking level and must have the same value already set on the IBM i system.
3. Analyze the trace file `native.outTR` that you can find in the compressed file named with the job ID in the following path:

   ```
   TWA_home/TWS/stdlist/JM/yyyy.mm.dd/archive
   ```

The agent joblog and TWSASPOOLS environment variable

About this task

By default, all information about the running of jobs is stored in the agent joblog. Most of this information usually consists of spool files. To select the spool file types that you want included in the agent joblog, use the **TWSASPOOLS** system variable, which works at IBM i agent level for any job to be submitted.

The **TWSASPOOLS** system variable forces the IBM i agent to either ignore all spool files or include one or more of them.

On the IBM i agent, create a new system level environment variable named TWSASPOOLS and set it to a list of the spool file types that are to be included. The list must begin with the SPOOLS: token.

For example, to force the IBM i agent to ignore all spool files, create the TWSASPOOLS variable as follows.

```
ADDENVVAR ENVVAR(TWSASPOOLS) VALUE(SPOOLS:) LEVEL(*SYS)
```

where the list after the SPOOL: token is empty. In this case, any agent joblog report for the IBM i agent is limited to the activity report that the Agent Monitor produces to trace its submission and monitoring action, and to the IBM i joblog of the Agent Monitor, which is always added at the end of the agent joblog.

To allow the IBM i agent to include only the QPRINT and the QPJOBLOG spool file types, that is, any spool files produced by printf instructions inside any ILE-C program and any produced joblog, create the TWSASPOOLS as follows:

```
ADDENVVAR ENVVAR(TWSASPOOLS) VALUE('SPOOLS: QPRINT QPJOBLOG') LEVEL(*SYS)
```

If the TWSASPOOLS variable already exists, change it as follows:

```
CHGENVVAR ENVVAR(TWSASPOOLS) VALUE('SPOOLS: QPRINT QPJOBLOG') LEVEL(*SYS)
```

If any VALUE parameter is set to an incorrect string, the IBM i agent ignores the TWSASPOOLS environment variable option. You can create and change the TWSASPOOLS environment variable while with the IBM i agent active, but no workload activity must be running.

Child job monitoring on IBM i agents

About this task

When you submit a command on an IBM i agent, the command might start one or more batch jobs. The IBM i agent monitors these batch jobs, which are referred to as child jobs.

When searching and monitoring any child jobs that are started, the IBM i agent uses a high percentage of its processing time.

If you know that your job scheduling does not start any child jobs or you have no interest in monitoring child jobs, you can instruct the IBM i agent to not search and monitor child jobs, and hence improve the performance of the agent.

You can exclude child job monitoring either at the agent level for all the commands or at the job definition level for a single command. If you want child job monitoring only for some specific submitted commands, you can set this option at the job definition level for a single command.

You can perform one or both of the following procedures to exclude or include child job monitoring:

Exclude child jobs from job monitoring at the agent level

By default child jobs are monitored. You can exclude child jobs from job monitoring for all submitted commands by creating the `TWS_NOCHILDS` system environment variable using the following IBM i system command:

```
ADDENVVAR ENVVAR(TWS_NOCHILDS) LEVEL(*SYS)
```

If the IBM i agent finds the `TWS_NOCHILDS` on the IBM i system, it does not monitor child jobs for any submitted command.

Exclude or include child jobs from job monitoring at the job definition level

You can exclude or include child jobs from job monitoring for a specific job by using `:NOCHILDS` or `:CHILDS` as ending tokens of the command string for the specific command.

- If you add the `:NOCHILDS` end token at the end of the native command you are submitting, the IBM i agent ignores any child jobs that are started by the command.
- If you add the `:CHILDS` end token at the end of the command you are submitting, the IBM i agent finds and monitors all the child jobs that are started by the command.

Note: The setting at job definition level overrides the setting at agent level.

Example

Examples

To monitor any child jobs that are started when the `PAYROLL` program is run, define the following command in the job definition:

- If the `TWS_NOCHILDS` system variable is defined on the IBM i system:

```
CALL PGM(MYLIB/PAYROLL) :CHILDS
```

- If the `TWS_NOCHILDS` system variable is not defined on the IBM i system:

```
CALL PGM(MYLIB/PAYROLL)
```

To not monitor any child jobs that are started when `MYSCHEDULE` program is run, define the following command in the job definition:

- If the `TWS_NOCHILDS` system variable is not defined on the IBM i system:

```
CALL PGM(MYLIB/MYSCHEDULE) :NOCHILDS
```

- If the `TWS_NOCHILDS` system variable is defined on the IBM i system:

```
CALL PGM(MYLIB/MYSCHEDULE)
```

Information about child job monitoring in IBM i agent joblogs

About this task

If you include child job monitoring on IBM i agents as described in Child job monitoring on IBM i agents on page 1374, you can see information related to child job monitoring in the IBM i agent joblog.

Example

Examples

This example shows the information related to child job monitoring included at job level for the CHILDLONG_CHILD job on the NC117025 agent:

```
Job CHILDLONG_CHILD

Workstation (Job) NC117025
```

```
Job Stream AS400ENVSET

Workstation (Job Stream) NC117025


================================================================
= JOB       : NC117025#AS400ENVSET[(1417 10/30/12),(AS400ENVSET)].CHILDLONG_CHILD
= TASK      : <?xml version="1.0" encoding="UTF-8"?>
<jsdl:jobDefinition xmlns:jsdl="http://www.ibm.com/xmlns/prod/scheduling/1.0/jsdl"
xmlns:jsdlibmi="http://www.ibm.com/xmlns/prod/scheduling/1.0/jsdlibmi" name="ibmi">
  <jsdl:variables>
    <jsdl:stringVariable name="tws.jobstream.id">AS400ENVSET</jsdl:stringVariable>
    <jsdl:stringVariable name="tws.job.workstation">NC117025</jsdl:stringVariable>
    <jsdl:stringVariable name="tws.job.iawstz">202310301417</jsdl:stringVariable>
  </jsdl:variables>
  <jsdl:application name="ibmi">
    <jsdlibmi:ibmi>
      <jsdlibmi:IBMIParameters>
         <jsdlibmi:Task>
          <jsdlibmi:command>CALL PGM(MINERMA/SBM5JOBS) :CHILDS</jsdlibmi:command>
         </jsdlibmi:Task>
      </jsdlibmi:IBMIParameters>
     </jsdlibmi:ibmi>
  </jsdl:application>
  <jsdl:resources>
    <jsdl:orderedCandidatedWorkstations>
      <jsdl:workstation>805E5EAC1F5911E2B9DB6F8202778C47</jsdl:workstation>
    </jsdl:orderedCandidatedWorkstations>
  </jsdl:resources>
</jsdl:jobDefinition>
= TWSRCMAP  :
= AGENT     : NC117025
= Job Number: 760232858
= Mon Oct 30 14:16:31 CET 2023
================================================================
The Dynamic Agent submitter-monitor job is qualified as:
   JobName=DYNAMICMON JobUser=CLAUDIO JobNumber=361743
Here follows the user command string
 <CALL PGM(MINERMA/SBM5JOBS) :CHILDS>
2023/10/30 14:16:28.844 - Dynamic Agent job submitted the User Command
   CALL PGM(MINERMA/SBM5JOBS)
The FOLLOWING 5 JOBS STARTED under the submitted User Command
  JobName=CLAUDIO    JobUser=CLAUDIO    JobNumber=361765
  JobName=CLAUDIO    JobUser=CLAUDIO    JobNumber=361756
  JobName=CLAUDIO    JobUser=CLAUDIO    JobNumber=361762
  JobName=CLAUDIO    JobUser=CLAUDIO    JobNumber=361775
  JobName=CLAUDIO    JobUser=CLAUDIO    JobNumber=361774
Message CPF1241 (Success) received on MsgQueue CLAUDIO   QUSRSYS
   for the job CLAUDIO   CLAUDIO   361765
Message CPF1241 (Success) received on MsgQueue CLAUDIO   QUSRSYS
   for the job CLAUDIO   CLAUDIO   361756
Message CPF1241 (Success) received on MsgQueue CLAUDIO   QUSRSYS
   for the job CLAUDIO   CLAUDIO   361762
Message CPF1241 (Success) received on MsgQueue CLAUDIO   QUSRSYS
   for the job CLAUDIO   CLAUDIO   361775
Message CPF1241 (Success) received on MsgQueue CLAUDIO   QUSRSYS
   for the job CLAUDIO   CLAUDIO   361774
```

```
*** END codes gathered by the Monitor job ***
 > END Status Code (Status): 0
 > PROGRAM Return Code (Prc): 0
 > USER Return Code (Urc): 0
   Urc was retrieved through SYSAPI

2023/10/30 14:21:37.890 - Dynamic Agent job ended monitoring the User Command
*** Return Code for submitted Command is 0 ***
*** User Command ended successfully ***
```

This example shows the joblog for the CHILDLING_NC job on the NC117025 agent when child job monitoring is excluded at job level:

```
Job CHILDLING_NC

Workstation (Job) NC117025

Job Stream AS400ENVSET

Workstation (Job Stream) NC117025



================================================================
= JOB       : NC117025#AS400ENVSET[(1417 10/30/12),(AS400ENVSET)].CHILDLING_NC
= TASK      : <?xml version="1.0" encoding="UTF-8"?>
<jsdl:jobDefinition xmlns:jsdl="http://www.ibm.com/xmlns/prod/scheduling/1.0/jsdl"
xmlns:jsdlibmi="http://www.ibm.com/xmlns/prod/scheduling/1.0/jsdlibmi" name="ibmi">
  <jsdl:variables>
    <jsdl:stringVariable name="tws.jobstream.id">AS400ENVSET</jsdl:stringVariable>
    <jsdl:stringVariable name="tws.job.workstation">NC117025</jsdl:stringVariable>
    <jsdl:stringVariable name="tws.job.iawstz">202310301417</jsdl:stringVariable>
  </jsdl:variables>
  <jsdl:application name="ibmi">
    <jsdlibmi:ibmi>
       <jsdlibmi:IBMIParameters>
         <jsdlibmi:Task>
           <jsdlibmi:command>CALL PGM(MINERMA/SBM5JOBS) :NOCHILDS</jsdlibmi:command>
         </jsdlibmi:Task>
       </jsdlibmi:IBMIParameters>
        </jsdlibmi:ibmi>
  </jsdl:application>
  <jsdl:resources>
    <jsdl:orderedCandidatedWorkstations>
      <jsdl:workstation>805E5EAC1F5911E2B9DB6F8202778C47</jsdl:workstation>
    </jsdl:orderedCandidatedWorkstations>
  </jsdl:resources>
</jsdl:jobDefinition>
= TWSRCMAP  :
= AGENT     : NC117025
= Job Number: 760232857
= Mon Oct 30 14:17:01 CET 2023
================================================================
The Dynamic Agent submitter-monitor job is qualified as:
   JobName=DYNAMICMON JobUser=CLAUDIO JobNumber=361817
Here follows the user command string
 <CALL PGM(MINERMA/SBM5JOBS) :NOCHILDS>
2023/10/30 14:16:58.330 - Dynamic Agent job submitted the User Command
```

```
   CALL PGM(MINERMA/SBM5JOBS)
As per user choice, NO job started under the submitted command will be monitored
*** END codes gathered by the Monitor job ***
 > END Status Code (Status): 0
 > PROGRAM Return Code (Prc): 0
 > USER Return Code (Urc): 0
   Urc was retrieved through SYSAPI

2023/10/30 14:17:10.220 - Dynamic Agent job ended monitoring the User Command
*** Return Code for submitted Command is 0 ***
*** User Command ended successfully ***
```

The agent return code retrieval

About this task

The IBM i programming model was originally based on an early object orientation model in which programs communicated through message passing, rather than using return codes. The introduction of the Integrated Language Programming (ILE) model lead to the definitions of common areas to exchange data as return codes in the same job environment: the user return codes and the system end codes.

For information about user return codes, see Controlling the job environment with the user return code on page 1378.

When the IBM i agent verifies that a submitted command or job is completed, it assigns a return code to the job based on the job status of the completed job. The return code is set depending on the completion message of the command or job. If the command or job completes successfully, the return code is set to 0. If the command or job does not complete successfully, the return code is set to the value of the severity of the message related to the exception that caused the abnormal end of the job. The IBM i agent can also set the return code to the value of the user return code when it is returned by the submitted command. If retrieved, the user return code is used as the value to set the return code.

The return code value assigned to the job is included in the IBM i agent joblog for the job and sent back to the scheduler user interface (WEB UI or z/OS ISPF panels) as return code, for compatibility reasons with agents on other operating systems.

Controlling the job environment with the user return code

About this task

With the introduction of the IBM i ILE model, it is possible to retrieve a value returned by a called program inside the same job.

When the Agent Monitor verifies that a submitted command is completed, it retrieves the following end of job codes using an IBM i System API:

End status code or <Status> (0 if successful)

It indicates if the system issued a controlled cancellation of the job. Possible values are:

1

the subsystem or the job itself is canceled.

0

the subsystem or the job itself is not canceled.

blank

the job is not running.

Program return code or <Prc> (0000 if successful)

It specifies the completion code of the last program (such as a data file utility program, or an RPG or COBOL program, invoked by the job).

If the job includes no program, the program return code is 0.

User return code or <Urc> (0000 if successful)

It specifies the user-defined return code set by ILE high-level language constructs. For example, the return code of a program written in C language.

It represents the most recent return code set by any thread within the job.

If the submitted command is a call to a user ILE program returning a value on exiting, this value is found in the Urc end of job code.

You can decide how to control the job environment of your submitted jobs by preparing the commands to be submitted as CALLs to your ILE programs, where the internal flow is controlled and the end status is decided through proper exit values. If a user program ends in error for an incorrect flow control, without returning a value, the Agent Monitor does not set the Return Code as user return code (Urc), but follows the criteria described in The agent return code retrieval on page 1378.

The following example shows an ILE C user program where two batch jobs are launched and a value of 10 is returned to the caller, regardless of the completion status of the batch jobs.

```
============================================================================
   #include <stdio.h>
   #include <stdlib.h>
   #include <string.h>
void main(int argc, char *argv[])
{
   int   EnvVarRC=0;
   printf("issuing SBMJOB CMD(CALL MYLIB/DIVBY0)...\n");
   system("SBMJOB CMD(CALL MYLIB/DIVBY0)");
   printf("issuing SBMJOB CMD(WRKACTJOB OUTPUT(*PRINT))...\n");
   system("SBMJOB CMD(WRKACTJOB OUTPUT(*PRINT)) LOG(4 0 *SECLVL)");
   exit(10);
   return;
}
============================================================================
```

Alternative method to set the user return code

About this task

In some IBM i environments, the system API retrieving the user return code (Urc) from the Agent Monitor code does not retrieve the correct value for Urc. It is therefore not recommended that you use any IBM i system APIs to retrieve the user return code. To receive a value returned by a called program, it is better to provide, instead, a parameter to receive the value.

Even if the Agent Monitor can retrieve the user return code using system API, an alternative user return code retrieval method was implemented in the Agent Monitor code. The alternative retrieval method has the following logic. The `USERRC` job environment variable is created and set to the *INI* value before submitting the user command. When the command ends, the Agent Monitor retrieves its user return code using the system APIs, but it also verifies if the `USERRC` job environment variable was updated at user program level. If a value different from *INI* is found, this is considered as the user return code and the value retrieved using the system APIs is ignored because the user program modified the value of `USERRC` job environment variable.

The change of the `USERRC` variable at user program level requires the `USERRC` value change before exiting from the application user code. In the ILE C case, you can do this using the putenv statement, where the user return code is set to be returned.

The following example shows how the user code returns the user return code using the IBM i agent reserved job environment variable `USERRC`. This code was obtained from the code of the example in Controlling the job environment with the user return code on page 1378 by replacing the exit with the putenv statement.

```
============================================================================
   #include <stdio.h>
   #include <stdlib.h>
   #include <string.h>
void main(int argc, char *argv[])
{
   int   EnvVarRC=0;
   printf("issuing SBMJOB CMD(CALL MYLIB/DIVBY0)...\n");
   system("SBMJOB CMD(CALL MYLIB/DIVBY0)");
   printf("issuing SBMJOB CMD(WRKACTJOB OUTPUT(*PRINT))...\n");
   system("SBMJOB CMD(WRKACTJOB OUTPUT(*PRINT)) LOG(4 0 *SECLVL)");
   EnvVarRC = putenv("USERRC=10");
   return;
}
============================================================================
```

Appendix A. Event-driven workload automation event and action definitions

This appendix documents the event and action providers you can use for event-driven workload automation and gives details on event and action definitions.

Event providers and definitions

One or more events can be added to the event rule.

This section gives details on the event types of the following event providers:

- TWSObjectsMonitor events on page 1381
- FileMonitor events on page 1385
- TWSApplicationMonitor events on page 1395
- DatasetMonitor events on page 1396

TWSObjectsMonitor events

`TWSObjectsMonitor` events are:

- Job Status Changed
- Job Until
- Job Submitted
- Job Cancelled
- Job Restarted
- Job Late
- Job Promoted
- Job Risk Level Changed
- Job Exceeded Maximum Duration
- Job Did not Reach Minimum Duration
- Job Stream Status Changed
- Job Stream Completed
- Job Stream Until
- Job Stream Submitted
- Job Stream Cancelled
- Job Stream Late
- Workstation Status Changed
- Application Server Status Changed
- Child Workstation Link Changed
- Parent Workstation Link Changed
- Prompt Status Changed
- ProductAlert

These events are generated by `batchman` (or `mailman` for the workstations) and written in a mailbox file named `monbox.msg`. The scheduling objects are monitored as follows:

- Jobs are monitored by the workstation where they run
- Job streams are monitored by the master domain manager
- Workstations monitor themselves
- Local prompts are monitored by the workstation running the job or job stream that have a dependency on the prompt
- Global prompts are monitored by the master domain manager

More information is available in the online version of the current topic in the published documentation at HCL Workload Automation welcome page.

Using operator instructions as variables

You can use operator instructions as variables for job and job stream events of `TWSObjectsMonitor` type. The operator instructions are extracted from the **description** and **documentation** fields of jobs and job streams definitions, as these fields can contain important information about the definitions, such as runbooks, recovery processes and support contacts.

Using operator instructions as variables in event definitions enhances the reusability of event rules, and supports automated error recovery by providing relevant operational data at the point of failure.

You can extract either the entire content of the operator instructions, or specific key-value pairs. The syntax varies according to your purpose:

Retrieving the entire field content

Use the following syntax to retrieve the full content of the operator instructions:

```
%{<event>.OpInstr}
```

For example you can add an **Send mail** action to an event rule, and use the following syntax in the email body to retrieve the full runbook for a job:

```
%{jobStatChEvt1.OpInstr}
```

If the job fails, the entire runbook text from the job definition populates the body of the email.

Retrieving specific key-value pairs

To enable the retrieval of specific values from the operator instructions, the description or documentation fields of a job or job stream definition must contain the key-value pair, with a new line separating the pair from other instructions or text. Key-value pairs must contain a *key*, a *separator* (either equal sign "=" or colon sign ":"), and a *value*. The key must be a unique word, and it cannot contain spaces. For example:

```
First step: Check connection
escalation_teams_emails=support.db@acme.it; support.it@acme.it
Last step: Run cleanup script
```

In this case, you can only use `escalation_teams_emails` as variable.

The syntax requires you to specify the exact key defined in the job or job stream description field:

```
%{<event>.OpInstr.<key>}
```

Note: The syntax is limited to one level of key retrieval. Nested retrieval is not permitted.

For example, in the **to** field of a **Send mail** action within an event rule, you can use the following syntax to extract a specific recipient from the description field of a job:

```
%{jobStatChEvt1.OpInstr.escalation_teams_emails}
```

If the job fails, the recipients *support.db@acme.it* and *support.it@acme.it* receive the email.

Only the first matching occurrence is used as key-value pairs. For example, if you use the same syntax as above, and the operator instructions of the job contain the following information:

```
First step: Check connection
escalation_team_email=support.db@acme.it
Last step: Run cleanup script
escalation_team_email=support.it@acme.it
```

In this case, only the `support.db@acme.it` email address receives the email in case of job failure.

To exploit event rule reusability, you can define a generic event rule that triggers on any job or job stream status change or failure. This enables the single rule to perform different actions based on the information provided in the operator instructions of the failing object.

Key-value pairs retrieval hierarchy

Job and job stream information is retrieved according to the item related to the event:

- If the event is related to a job stream, the operator instructions of a job stream are used as key-value pairs.
- If the event is related to a job, the operator instructions of a job instance are used as key-value pairs. If the description or documentation fields of the job instance do not contain a matching key, the operator instructions of the job definition are used as key-value pairs. If nothing matches the key again, the operator instructions of the job stream are used as key-value pairs.

Working with WorkstationStatusChanged events

The event is sent when a workstation is started or stopped. But the following operational differences exist depending on the type of workstation that is monitored:

- For a fault-tolerant agent the event is sent when the workstation is started or stopped.
- For a dynamic workload broker workstation the event is sent also when it is linked or unlinked (as these commands also start or stop the workstation).
- For a dynamic pool workstation the event is never sent (even if the hosting dynamic workload broker is stopped) because there is no monitoring on this type of workstations.

Examples

The rule in the following example submits job stream `RJS_102739750` on workstation `NC125102` as soon as all the jobs of job stream `RCF_307577430` of workstation `NA022502` are in the RUNNING or SUCCESSFUL status.

```
<?xml version="1.0"?>
<eventRuleSet xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
      xmlns="http://www.ibm.com/xmlns/prod/tws/1.0/event-management/rules"
      xsi:schemaLocation="http://www.ibm.com/xmlns/prod/tws/1.0/
                          event-management/rules/EventRules.xsd">
   <eventRule name="TWS_PLAN_EVENTS_JOB_STATUS_CHANGED" ruleType="filter" isDraft="no">
      <description>Event: Job Status Changed; Action: Submit job stream</description>
      <timeZone>Europe/Rome</timeZone>
      <validity from="2023-04-24" to="2024-04-24" />
      <activeTime start="00:00:00" end="12:00:00" />
      <eventCondition name="jobStatChgEvt1"
                      eventProvider="TWSObjectsMonitor"
                      eventType="JobStatusChanged">
      <scope>* # JOBSTREAMVALUE . * [RUNNING, SUCCESSFUL]</scope>
         <filteringPredicate>
          <attributeFilter name="JobStreamWorkstation" operator="eq">
         <value>NA022502</value>
          </attributeFilter>
         <attributeFilter name="JobStreamName" operator="eq">
         <value>RCF_307577430</value>
         </attributeFilter>
         <attributeFilter name="JobName" operator="eq">
         <value>*</value>
         </attributeFilter>
         <attributeFilter name="Priority" operator="ge">
         <value>10</value>
         </attributeFilter>
         <attributeFilter name="Monitored" operator="eq">
         <value>true</value>
         </attributeFilter>
         <attributeFilter name="Status" operator="eq">
         <value>Running</value>
         <value>Successful</value>
         </attributeFilter>
         <attributeFilter name="Login" operator="eq">
         <value>TWS_user</value>
         </attributeFilter>
         </filteringPredicate>
      </eventCondition>
      <action actionProvider="TWSAction" actionType="sbs" responseType="onDetection">
         <description>Launch an existing TWS job stream</description>
         <scope>SBS NC125102#RJS_102739750</scope>
         <parameter name="JobStreamWorkstationName">
            <value>NC125102</value>
         </parameter>
         <parameter name="JobStreamName">
            <value>RJS_102739750</value>
         </parameter>
      </action>
   </eventRule>
</eventRuleSet>
```

The rule in the following example submits job `RJR_30411` on workstation `NC122160` as soon as job stream `RJS_102739750` of workstation `NC125102` is submitted.

```
<?xml version="1.0"?>
<eventRuleSet xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
      xmlns="http://www.ibm.com/xmlns/prod/tws/1.0/event-management/rules"
```

```
      xsi:schemaLocation="http://www.ibm.com/xmlns/prod/tws/1.0/
                          event-management/rules/EventRules.xsd">
   <eventRule name="TWS_PLAN_EVENTS_JOB_STREAM_SUBMITTED" ruleType="filter" isDraft="no">
      <description>Event: Job Stream Submitted; Action: Submit job</description>
      <eventCondition name="jsSubEvt1"
                      <eventProvider="TWSObjectsMonitor"
                      <eventType="JobStreamSubmit">
      <scope>WORKSTATIONVALUE # JOBSTREAMVALUE</scope>
         <filteringPredicate>
          <attributeFilter name="JobStreamWorkstation" operator="eq">
         <value>NC125102</value>
          </attributeFilter>
         <attributeFilter name="JobStreamName" operator="eq">
         <value>RJS_102739750</value>
         </attributeFilter>
         <attributeFilter name="Priority" operator="range">
         <value>15</value>
         <value>30</value>
         </attributeFilter>
         <attributeFilter name="LatestStart" operator="le">
         <value>2024-04-26</value>
         </attributeFilter>
         </filteringPredicate>
      </eventCondition>
      <action actionProvider="TWSAction" actionType="sbj" responseType="onDetection">
         <description>Launch an existing TWS job stream</description>
         <scope>SBJ NC122160#RJR_30411 INTO NC122160#JOBS</scope>
         <parameter name="JobUseUniqueAlias">
            <value>true</value>
         </parameter>
         <parameter name="JobDefinitionName">
            <value>RJR_30411</value>
         </parameter>
         <parameter name="JobDefinitionWorkstationName">
            <value>NC122160</value>
         </parameter>
      </action>
   </eventRule>
</eventRuleSet>
```

FileMonitor events

`FileMonitor` events are:

- `FileCreated`
- `FileDeleted`
- `ModificationCompleted`
- `LogMessageWritten`

When you monitor files by using the `FileCreated`, `FileDeleted`, and `LogMessageWritten` events, the memory consumed by the `ssmagent.bin` and `ssmagent.exe` processes increases linearly with the number of files monitored and with the number of events created. Therefore, keep in mind that the heavier use of wildcards you make within these event types, and the consequent higher number of files monitored, will result in a heavier memory consumption by the `ssmagent.bin` and `ssmagent.exe` processes.

`FileMonitor` events are not supported on:

- Pools, dynamic pools, and remote engine workstations.
- IBM i systems.

Using the MatchExpression property of the LogMessageWritten event rule

The `LogMessageWritten` event plug-in uses the regular expression specified in the `MatchExpression` property to perform substring matches on entries in the log files being monitored. The value of `MatchExpression` must be a valid regular expression in accordance with the regular expression syntax rules of the Netcool/SSM agent that the event plug-in uses.

The following table describes the syntax of the regular expression tokens supported by Netcool/SSM. Note that to write a valid regular expression for the `MatchExpression` property, you must write the `\` (backslash) escape character before each token used in the regular expression syntax (for example, \^ or \$). When the token already specifies a backslash character, you must write two backslash characters (for example, \\< or \\b).

Table 179. Regular expression syntax.

Token	Matches
.	Any character.
^	The start of a line (a zero-length string).
$	The end of a line; a new line or the end of the search buffer.
\<	The start of a word (where a word is a string of alphanumeric characters).
\>	The end of a word (the zero length string between an alphanumeric character and a non-alphanumeric character).
\b	Any word boundary (this is equivalent to (\<¦\>)).
\d	A digit character.
\D	Any non-digit character.
\w	A word character (alphanumeric or underscore).
\W	Any character that is not a word character (alphanumeric or underscore).
\s	A whitespace character.
\S	Any non-whitespace character.
\c	Special characters and escaping. The following characters are interpreted according to the C language conventions: \0, \a, \f, \n, \r, \t, \v. To specify a character in hexadecimal, use the \xNN syntax. For example, \x41 is the ASCII character A.
\	All characters apart from those described above may be escaped using the backslash prefix. For example, to specify a plain left-bracket use \[.

Table 179. Regular expression syntax. (continued)

Token	Matches
[]	Any one of the specified characters in a set. An explicit set of characters may be specified as in [aeiou] as well as character ranges, such as [0-9A-Fa-f], which match any hexadecimal digit. The dash (-) loses its special meaning when escaped, such as in [A\-Z] or when it is the first or last character in a set, such as in [-xyz0-9]. All of the above backslash-escaping rules may be used within []. For example, the expression [\x41-\x45] is equivalent to [A-D] in ASCII. To use a closing bracket in a set, either escape it using [\]] or use it as the first character in the set, such as []xyz]. POSIX-style character classes are also allowed inside a character set. The syntax for character classes is [:class:]. The supported character classes are: • [:alnum:] - alphanumeric characters. • [:alpha:] - alphabetic characters. • [:blank:] - space and TAB characters. • [:cntrl:] - control characters. • [:digit:] - numeric characters. • [:graph:] - characters that are both printable and visible. • [:lower:] - lowercase alphabetic characters. • [:print:] - printable characters (characters that are not control characters). • [:punct:] - punctuation characters (characters that are not letters, digits, control characters, or spaces). • [:space:] - space characters (such as space, TAB and form feed). • [:upper:] - uppercase alphabetic characters. • [:xdigit:] - characters that are hexadecimal digits. Brackets are permitted within the set's brackets. For example, [a-z0-9!] is equivalent to [[:lower:][:digit:]!] in the C locale.
[^]	Inverts the behavior of a character set [] as described above. For example, [^[:alpha:]] matches any character that is not alphabetical. The ^ caret symbol only has this special meaning when it is the first character in a bracket set.
{n}	Exactly n occurrences of the previous expression, where 0 <= n <= 255. For example, a{3} matches aaa.
{n,m}	Between n and m occurrences of the previous expression, where 0 <= n <= m <= 255. For example, a 32-bit hexadecimal number can be described as 0x[[:xdigit:]]{1,8}.
{n,}	At least n or more (up to infinity) occurrences of the previous expression.
*	Zero or more of the previous expression.

Table 179. Regular expression syntax. (continued)

Token	Matches
+	One or more of the previous expression.
?	Zero or one of the previous expression.
(exp)	Grouping; any series of expressions may be grouped in parentheses so as to apply a postfix or bar (¦) operator to a group of successive expressions. For example: • ab+ matches all of abbb • (ab)+ matches all of abababab
¦	Alternate expressions (logical OR). The vertical bar (¦) has the lowest precedence of all tokens in the regular expression language. This means that ab¦cd matches all of cd but does not match abd (in this case use a(b¦c)d).

Tip: When defining regular expressions to match multi-byte characters, enclose each multi-byte character in parentheses ().

Table 180: Regular expression examples. on page 1388 provides a set of regular expression examples, together with sample strings as well as the results of applying the regular expression to those strings.

There are two important cases in matching regular expressions with strings. A regular expression may match an entire string (a case known as a *string match*) or only a part of that string (a case known as a *sub-string match*). For example, the regular expression `\<int\>` will generate a sub-string match for the string `int x` but will not generate a string match. This distinction is important because some subagents do not support sub-string matching. Where applicable, the results listed in the examples differentiate between string and sub-string matches.

Table 180. Regular expression examples.

This expression...	Applied to this string...	Results in...
.	a	String match
	!	String match
	`abcdef`	Sub-string match on `a`
	empty string	No match
`M..COUNT`	`MINCOUNT`	String match
	`MXXCOUNTY`	Sub-string match on `MXXCOUNT`
	`NONCOUNT`	No match
.*	empty string	String match
	`Animal`	String match

Table 180. Regular expression examples. (continued)

This expression...	Applied to this string...	Results in...
.+	Any non-empty string	String match
	empty string	No match
^	empty string	String match
	`hello`	Sub-string match of length 0 at position 0 (position 0 = first character in string)
$	empty string	String match
	`hello`	Sub-string match of length 0 at position 5 (position 0 = first character in string)
^$	empty string	String match
	`hello`	No match
\b`ee`	`tee`	No match
	`Paid fee`	No match
	`feel`	No match
	`eel`	Sub-string match on `ee`
.*`thing`.*	`The thing is in here`	String match
	`there is a thing`	String match
	`it isn't here`	No match
	`thin`xxx	No match
`a`*	empty string	String match
	`aaaaaaaaa`	String match
	`a`	String match
	`aardvark`	Sub-string match on `aa`
	`this string`	Sub-string match
((`ab`)*`c`)*	empty string	String match
	`ccccccccc`	String match
	`ccccabcccabc`	String match
`a`+	empty string	No match
	`aaaaaaaaa`	String match
	`a`	String match

Table 180. Regular expression examples. (continued)

This expression...	Applied to this string...	Results in...
	`aardvark`	Sub-string match on `aa`
	`this string`	No match
(`ab`)+`c`)*	empty string	String match
	`abababababcabc`	String match
(`ab`){2}	`abab`	String match
	`cdababababab`	Sub-string match on `abab`
[0-9]{4,}	`123`	No match
	`a1234`	Sub-string match on `1234`
`a`{0}	empty string	String match
	`a`	No match
	`hello`	Sub-string match of length 0 at position 0 (position 0 = first character in string)
[0-9]{1,8}	`this is not a number`	No match
	`a=4238, b=4392876`	Sub-string match on `4238`
([`aeiou`][^`aeiou`])+	`Hello`	Sub-string match on `el`
	`!!! Supacalafraglistic`	Sub-string match on `upacalaf`
[`+-`]?1	`1`	String match
	`+1`	String match
	`-1`	String match
	`.1`	Sub-string match on `1`
	`value+1`	Sub-string match on `+1`
`a`¦`b`	`a`	String match
	`b`	String match
	`c`	No match
	`Daniel`	Sub-string match on `a`
`abcd`¦`efgh`	`abcd`	String match
	`efgh`	String match
	`abcdfgh`	Sub-string match on `abcd`
[0-9`A-F`]+	`BAADF00D`	String match

Table 180. Regular expression examples. (continued)

This expression...	Applied to this string...	Results in...
	`C`	String match
	`baadF00D`	Sub-string match on `F00D`
	`c`	No match
	`G`	No match
	`g`	No match
`x =` \d+	`x = 1234`	String match
	`x = 0`	String match
	`x = 1234a`	Sub-string match on `x = 1234`
	`x = y`	No match
	`x^=^` where `^` represents a space character	No match
\D\d	`a1`	String match
	`a11`	Sub-string match on `a1`
	`-9`	String match
	`a`	No match
	`8`	No match
	`aa`	No match
	`4t`	No match
\s+	`Hello_w0rld`	No match
	`Hello^^^world` where `^` represents a space character	Sub-string match on `^^^` where `^` represents a space character
	`Widget^` where `^` represents a space character	Sub-string match on `^` where `^` represents a space character
	`^^^^^` where `^` represents a space character	String match
\S+	`Hello_w0rld`	Sub-string match of length 11 on `Hello_w0rld`
	`Hello^^^world` where `^` represents a space character	Sub-string match on `Hello`
	`Widget^` where `^` represents a space character	Sub-string match on `Widget`

Table 180. Regular expression examples. (continued)

This expression...	Applied to this string...	Results in...
	`^^^^^` where `^` represents a space character	No match
\w+	`D4n_v4n Vugt`	Sub-string match on `D4n_v4n`
	`^^^hello` where `^` represents a space character	Sub-string match on `hello`
	`blah`	String match
	`x#1`	No match
	`foo bar`	No match
\W	`Hello there`	Sub-string match of length 1 on separating space character
	`~`	String match
	`aa`	No match
	`a`	No match
	`-`	No match
	`^^^^444 == 5` where `^` represents a space character	Sub-string match of length 1 on first `^` where `^` represents a space character
\w+\s*=\s*\d+	`x = 123`	String match
	`count0=555`	String match
	`my_var=66`	String match
	`0101010=0`	String match
	`xyz = e`	No match
	`delta=`	No match
	`==8`	No match
[[:alnum:]]+	`1234`	String match
	`...D4N13L`	Sub-string match on `D4N13L`
[[:alpha:]]+	`Bubble`	String match
	`...DANI3L`	Sub-string match on `DANI`
	`69`	No match
[:blank:]]+	`alpha^^^^and beta` where `^` represents a space character	Sub-string match on `^^^^` where `^` represents a space character
	`Animal`	No match

Table 180. Regular expression examples. (continued)

This expression...	Applied to this string...	Results in...
	empty string	No match
[[:space:]]+	`alpha^^^^and beta` where `^` represents a space character	Sub-string match on `^^^^` where `^` represents a space character
	`Animal`	No match
	empty string	No match
[[:cntrl:]]+	`...Hello W0rld!`	No match
	empty string	No match
[[:graph:]]+	`hello world`	Sub-string match on `hello`
	`^^^^^` where `^` represents a space character	No match
	`^^^!?` where `^` represents a space character	Sub-string match on `!?`
[:lower:]]+	`Animal`	Sub-string match on `nimal`
	`ABC`	No match
	`0123`	No match
	`foobar`	String match
	`^^^0blaH!` where `^` represents a space character	Sub-string match on `bla`
[_[:lower:]]+	`foo_bar`	String match
	`this_thinG!!!`	Sub-string match on `_thin`
[[:upper:]]+	`YES`	String match
	`#define MAX 100`	Sub-string match on `MAX`
	`f00 b4r`	No match
[[:print:]]+	`hello world`	String match
	`^^^^^` where `^` represents a space character	String match
[[:punct:]]+	`didn't`	Sub-string match on `'`
	`Animal`	No match
[[:xdigit:]]+	`43298742432392187ffe`	String match
	`x = bAAdF00d`	Sub-string match on `bAAdF00d`

Table 180. Regular expression examples. (continued)

This expression...	Applied to this string...	Results in...
	`4327afeffegokpoj`	Sub-string match on `4327afeffe`
c:\\temp	`c:\temp`	String match

Example

The rule in the following example sends an email to a list of recipients as soon as file `/home/book.txt` is created on workstation `editor_wrkstn`.

```
<?xml version="1.0"?>
 <eventRuleSet xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
      xmlns="http://www.ibm.com/xmlns/prod/tws/1.0/event-management/rules"
      xsi:schemaLocation="http://www.ibm.com/xmlns/prod/tws/1.0/
                          event-management/rules/EventRules.xsd">
   <eventRule name="FILE_MONITOR_FILE_CREATED" ruleType="filter" isDraft="no">
      <description>Event: File Created; Action: Send mail</description>
      <validity to="2024-04-22" />
      <eventCondition name="fileCrtEvt1" eventProvider="FileMonitor" eventType="FileCreated">
      <scope>/HOME/BOOK.TXT ON EDITOR_WRKSTN</scope>
         <filteringPredicate>
          <attributeFilter name="FileName" operator="eq">
         <value>/home/book.txt</value>
          </attributeFilter>
         <attributeFilter name="SampleInterval" operator="eq">
         <value>60</value>
         </attributeFilter>
         <attributeFilter name="Workstation" operator="eq">
         <value>editor_wrkstn</value>
         </attributeFilter>
         <attributeFilter name="Hostname" operator="eq">
         <value>ceditor</value>
         </attributeFilter>
                  </filteringPredicate>
      </eventCondition>
      <action actionProvider="MailSender" actionType="SendMail" responseType="onDetection">
         <description>Send an eMail</description>
         <scope>SAUL.FELLOW@US.IBM.COM, ISAAC.LINGER@US.IBM.COM : THE EXPECTED FILE
                HAS BEEN CREATED!</scope>
         <parameter name="Cc">
            <value>william.waulkner@us.ibm.com</value>
         </parameter>
         <parameter name="Bcc">
            <value>ernest.demingway@us.ibm.com</value>
         </parameter>
         <parameter name="Body">
            <value>The expected file was created!
                   The book is ready to be published.</value>
         </parameter>
         <parameter name="To">
            <value>saul.fellow@us.ibm.com, isaac.linger@us.ibm.com</value>
         </parameter>
         <parameter name="Subject">
            <value>The expected file was created!</value>
         </parameter>
      </action>
   </eventRule>
</eventRuleSet>
```

TWSApplicationMonitor events

`TWSApplicationMonitor` events concern HCL Workload Automation processes, file system, and message box. They are:

- `MessageQueuesFilling`
- `TivoliWorkloadSchedulerFileSystemFilling`
- `TivoliWorkloadSchedulerProcessNotRunning`

`TWSApplicationMonitor` events are not supported on IBM i systems.

`TWSApplicationMonitor` events are supported on fault-tolerant agents only.

For a detailed example about how to set up an event rule that monitors the used disk space, see the section about maintaining the file system in the *Administration Guide*.

Example

The rule in the following example logs warning message `LOGMSG01W` as soon as either `intercom` or `mailbox` message queue files on workstation `NC122160` reach 70 percent of their size.

```
<?xml version="1.0"?>
<eventRuleSet xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
      xmlns="http://www.ibm.com/xmlns/prod/tws/1.0/event-management/rules"
      xsi:schemaLocation="http://www.ibm.com/xmlns/prod/tws/1.0/
                          event-management/rules/EventRules.xsd">
   <eventRule name="TWS_APPL_MONITOR_MESSAGE_QUEUES_FILLING" ruleType="filter" isDraft="no">
      <description>Event: Message queues filling; Action: Message logger</description>
      <timeZone>America/Los_Angeles</timeZone>
      <validity from="2011-04-25"/>
      <activeTime end="17:00:00"/>
      <eventCondition name="twsMesQueEvt1" eventProvider="TWSApplicationMonitor"
 eventType="TWSMessageQueues">
      <scope>INTERCOM, MAILBOX FILLED UP 70% ON NC122160</scope>
         <filteringPredicate>
          <attributeFilter name="MailboxName" operator="eq">
         <value>intercom</value>
         <value>mailbox</value>
          </attributeFilter>
         <attributeFilter name="FillingPercentage" operator="ge">
         <value>70</value>
         </attributeFilter>
         <attributeFilter name="Workstation" operator="eq">
         <value>NC122160</value>
         </attributeFilter>
         <attributeFilter name="SampleInterval" operator="eq">
         <value>60</value>
         </attributeFilter>
                  </filteringPredicate>
      </eventCondition>
      <action actionProvider="MessageLogger" actionType="MSGLOG" responseType="onDetection">
         <description>Write a warning message log</description>
         <scope>OBJECT=LOGMSG01W MESSAGE=MAILBOX AND/OR INTERCOM QUEUE
                HAS REACHED 70% OF FILLING</scope>
         <parameter name="ObjectKey">
            <value>LOGMSG01W</value>
         </parameter>
         <parameter name="Message">
            <value>Mailbox and/or Intercom queue has reached 70% of filling</value>
         <parameter name="Severity">
```

```
            <value>Warning</value>
        </action>
    </eventRule>
</eventRuleSet>
```

DatasetMonitor events

Use this function to create event rules and trigger events

`DatasetMonitor` events are:

- `ReadCompleted`
- `ModificationCompleted`

Table 181. `SMF` events

Event type	Event trigger
ReadCompleted	A data set is closed after it was opened in read mode.
ModifcationCompleted	A data set is closed after it was opened in write mode. This event is sent also when you create an empty data set.

Table 182. Parameters of ReadCompleted and ModificationCompleted event types

attributeFilter name	Type	Required	Wilcard allowed	Length (min-max)		Default value
FileName	string	✓	✓	1	44	

Note: For parameters with wildcard allowed, you can use the following wildcards:

*****

To match any sequence of characters.

?

To match any single character. For example, if you specify AB?, ABC is a match, AB or ABCD are not a match.

%

For compatibility with earlier versions, it is supported for the same function as ?.

The following list provides a detailed description of the parameters:

FileName

Specifies the data set name to be monitored for actions on special resources. For details about how the Agent requests to change the resource availability, based on the specified `FileName` value.

Examples

The following examples show how to combine language elements and use wildcards:

```
<?xml version="1.0"?>
<eventRuleSet xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
      xmlns="http://www.ibm.com/xmlns/prod/tws/1.0/event-management/rules"
      xsi:schemaLocation="http://www.ibm.com/xmlns/prod/tws/1.0/
                          event-management/rules/EventRules.xsd">
   <eventRule name="FILE_MONITOR_FILE_CREATED" ruleType="filter" isDraft="no">
      description>Event: File Created; Action: Send mail/description>
      validity to="2024-04-22" />
      <eventCondition name="fileCrtEvt1" eventProvider="FileMonitor" eventType="FileCreated">
      <scope>/HOME/BOOK.TXT ON EDITOR_WRKSTN</scope>
         <filteringPredicate>
          <attributeFilter name="FileName" operator="eq">
         <value>/home/book.txt</value>
          </attributeFilter>
         <attributeFilter name="SampleInterval" operator="eq">
         <value>60</value>
         </attributeFilter>
         <attributeFilter name="Workstation" operator="eq">
         <value>editor_wrkstn</value>
         </attributeFilter>
         <attributeFilter name="Hostname" operator="eq">
         <value>ceditor</value>
         </attributeFilter>
                  </filteringPredicate>
      </eventCondition>
      <action actionProvider="MailSender" actionType="SendMail" responseType="onDetection">
         <description>Send an eMail</description>
         <scope>SAUL.FELLOW@US.IBM.COM, ISAAC.LINGER@US.IBM.COM : THE EXPECTED FILE
                HAS BEEN CREATED!</scope>
         <parameter name="Cc">
            <value>william.waulkner@us.ibm.com</value>
         </parameter>
         <parameter name="Bcc">
            <value>ernest.demingway@us.ibm.com</value>
         </parameter>
         <parameter name="Body">
            <value>The expected file was created!
                   The book is ready to be published.</value>
         </parameter>
         <parameter name="To">
            <value>saul.fellow@us.ibm.com, isaac.linger@us.ibm.com</value>
         </parameter>
         <parameter name="Subject">
            <value>The expected file was created!</value>
         </parameter>
      </action>
   </eventRule>
</eventRuleSet>
```

Action providers and definitions

This section gives details on the action types of the following action providers:

- GenericAction on page 1398
- MailSender on page 1399
- MessageLogger on page 1399
- TWSAction on page 1400
- TWSForZosAction on page 1400

GenericAction actions

This provider implements a single action named `RunCommand` that runs non-HCL Workload Automation commands. Commands are run on the same computer where the event processor runs.

Only *TWS_user* is authorized to run the command.

Important: When the command includes output redirection (through the use of one or two `>` signs), insert the command in an executable file, and set the file name as the argument of the `Command` property.

More information is available in the online version of the current topic in the published documentation at HCL Workload Automation welcome page.

Example

The rule in the following example runs the ps -ef command to list all the currently running processes on a UNIX workstation when an invalid parameter is found on that workstation. Note that the rule is based on a custom event developed using the `GenericEventPlugIn` event provider. For more information on developing custom event types, see Defining custom events on page 195.

```
<?xml version="1.0"?>
<eventRuleSet xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
      xmlns="http://www.ibm.com/xmlns/prod/tws/1.0/event-management/rules"
      xsi:schemaLocation="http://www.ibm.com/xmlns/prod/tws/1.0/
                          event-management/rules/EventRules.xsd">
   <eventRule name="CUSTOM_EVENT_GENERIC_EVENT" ruleType="filter" isDraft="yes">
      <description>Event: Generic Event; Action: Run Command</description>
      <activeTime start="08:30:00" end="17:30:00"/>
      <eventCondition name="genericEvt3" eventProvider="GenericEventPlugIn"
      eventType="Event1">
      <scope>INVALID PARAMETER ON WORKSTATIONVALUE</scope>
         <filteringPredicate>
           <attributeFilter name="Param1" operator="ne">
            <value>Invalid Parameter</value>
           </attributeFilter>
           <attributeFilter name="Workstation" operator="eq">
            <value>WorkstationValue</value>
           </attributeFilter>
        </filteringPredicate>
      </eventCondition>
      <action actionProvider="GenericActionPlugin" actionType="RunCommand"
      responseType="onDetection">
         <description>Run a command</description>
         <scope>PS -EF</scope>
         <parameter name="Command">
```

```
            <value>ps -ef</value>
         </parameter>
         <parameter name="WorkingDir">
            <value>/home</value>
         </parameter>
      </action>
   </eventRule>
</eventRuleSet>
```

MailSender actions

This provider implements a single action named `SendMail` that connects to an SMTP server to send an email. Use `optman` to customize the following related attributes (for detailed information about `optman`, see the *Administration Guide*):

- Mail sender
- SMTP server
- SMTP port number
- Mail user name
- Mail user password
- SSL

MessageLogger actions

This provider implements a single action named `MSGLOG` that logs the occurrence of a situation in an internal auditing database. The number of entries within the auditing database is configurable. There is an automatic cleanup based on a FIFO policy.

More information is available in the online version of the current topic in the published documentation at HCL Workload Automation welcome page.

ServiceNow actions

This provider implements two actions:

`Open Incident`

This action opens an incident in ServiceNow when a job that matches a defined policy ends in error.

`Send Event`

This action sends events directly to a ServiceNow instance when a specific rule is triggered.

Depending on your business needs, you can choose to either open an incident or trigger a custom event, or even do both.

Use `optman` to specify the ServiceNow server by setting the `servicenowUrl`, `servicenowUserName`, and `servicenowUserPassword` global options.

For detailed information about `optman` and these global options, see the topic with a detailed description of all global options in the *Administration Guide*.

When defining actions in the **Workload Designer** or `composer`, you can specify parameters that override the default values set in `optman`. The local override ensures that specific event rules can be tailored for unique ServiceNow instances without requiring a global configuration change on the primary node.

Note: Action properties might require specific formats, as detailed in the parameter information notes. Ensure that all input adheres to these format requirements.

For further information about the integration, browse to Automation Hub.

TWSAction actions

`TWSAction` actions are:

- SubmitJobStream
- SubmitJob
- SubmitAdHocJob
- ReplyPrompt

Using the `SchedTimeResolutionCriteria` property of the `SubmitJob` action

You use this property to match the job in question with a specific instance of the job stream that contains it (defined with the `JobStreamName` property) based on the job stream scheduled time. The possible values that you can set are:

Previous

The job is submitted with the closest previous job stream instance in plan.

Next

The job is submitted with the closest next job stream instance in plan.

Any

The job is submitted with any of the closest previous or closest next job stream instance in plan.

TWSForZosAction

This provider implements a single action named `AddJobStream` that adds an application occurrence (job stream) to the current plan on HCL Workload Automation for Z. This provider is for use in HCL Workload Automation end-to-end scheduling configurations.

The application description of the occurrence to be added must exist in the AD database of HCL Workload Automation for Z.

Example

In this example, a pharmaceutical company uses rule `ZOSRULE031` to produce a distribution schedule of the merchandise under the control of department `DISTR07`. As soon as the list of ordered merchandise that is up for delivery in the upcoming month is ready and placed in file `MONTHLYORDERS.TXT` on agent `RU192298` in a branch office, the centralized system adds

application (job stream) `ADFIRST` to the current plan. `ADFIRST` contains the operations (jobs) that produce an optimized delivery schedule for the next month.

```
<?xml version="1.0"?>
<eventRuleSet xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
      xmlns="http://www.ibm.com/xmlns/prod/tws/1.0/event-management/rules"
      xsi:schemaLocation="http://www.ibm.com/xmlns/prod/tws/1.0/
                          event-management/rules/EventRules.xsd">
   <eventRule name="ZOSRULE031" ruleType="filter" isDraft="no">
      <eventCondition name="fileCrtEvt19" eventProvider="FileMonitor"
      eventType="FileCreated">
      <scope>/PRODORDER/MONTHLYORDERS.TXT ON RU192298</scope>
         <filteringPredicate>
           <attributeFilter name="Param1" operator="ne">
            <value>/prodorder/monthlyorders.txt</value>
           </attributeFilter>
           <attributeFilter name="SampleInterval" operator="eq">
            <value>60</value>
           </attributeFilter>
           <attributeFilter name="Workstation" operator="eq">
           <value>RU192298</value>
           </attributeFilter>
        </filteringPredicate>
      </eventCondition>
      <action actionProvider="TWSForZosAction" actionType="AddJobStream"
      responseType="onDetection">
        <scope>
          ADD JOBSTREAM ADFIRST[DEADLINE OFFSET: 0001] WITH OWNER DISTR07 IN PLAN
        </scope>
         <parameter name="HoldAll">
            <value>false</value>
         </parameter>
         <parameter name="Priority">
            <value>5</value>
         </parameter>
         <parameter name="JobStreamDeadlineOffset">
            <value>0001</value>
         </parameter>
         <parameter name="JobStreamName">
            <value>ADFIRST</value>
         </parameter>
         <parameter name="OwnerDescription">
            <value>Owner description</value>
         </parameter>
         <parameter name="Owner">
            <value>distr07</value>
         </parameter>
         <parameter name="DependenciesResolution">
            <value>All</value>
         </parameter>
         <parameter name="AuthorityGroup">
            <value>AuthGrpBase</value>
         </parameter>
         <parameter name="Parm_1">
            <value>var1=value1</value>
         </parameter>
         <parameter name="Parm_2">
            <value>var2=value2</value>
```

```
         </parameter>
         <parameter name="JCLVariableTable">
            <value>VarTableZos01</value>
         </parameter>
         <parameter name="JobStreamDescription">
            <value>This job stream contains jobs that process orders for
                        owner DISTR07.</value>
         </parameter>
         <parameter name="Group">
            <value>GroupBase</value>
         </parameter>
      </action>
   </eventRule>
</eventRuleSet>
```

Appendix B. Quick reference for commands

This appendix is divided into four sections:

- Managing the plan on page 1403
- Managing objects in the database on page 1405
- Managing objects in the plan on page 1420
- Utility commands on page 1428
- Report commands on page 1432

Managing the plan

This section describes the operations you can perform against the plan using the **JnextPlan** script and the **planman** command line:

Table 183. Commands used against the plan

Command or script syntax	Action performed
JnextPlan [**-from** *mm/dd/[yy]yy[hh[:]mm[**tz** | **timezone** tzname]]*] {**-to** *mm/dd/[yy]yy[hh[:]mm[**tz** | **timezone** tzname]]* | **-for** *[h]hh[:]mm* [**-days** *n*] | **-days** *n*}	Creates or extends the production plan.
planman [*connection_parameters*] **crt** [**-from** *mm/dd/[yy]yy* [*hh[:]mm* [**tz** | **timezone** *tzname*]]] {**-to** *mm/dd/[yy]yy[hh[:]mm[**tz** | **timezone** tzname]]* | **-for** *[h]hh[:]mm* [**-days** *n*] | **-days** *n*}	Creates an intermediate production plan.
planman [*connection_parameters*] **deploy** [**-scratch**]	Deploys all rules that are not in draft state.
planman [*connection_parameters*] **ext** {**-to** *mm/dd/[yy]yy[hh[:]mm[**tz** | **timezone** tzname]]* | **-for** *[h]hh[:]mm* [**-days** *n*] | **-days** *n*}	Creates an intermediate plan for a plan extension.
planman [*connection_parameters*] **showinfo**	Retrieves the production plan information.
planman [*connection_parameters*] **crttrial** *file_name*	Creates a trial plan.

Table 183. Commands used against the plan (continued)

Command or script syntax	Action performed
[**-from** *mm/dd/[yy]yy* [*hh[:]mm* [**tz** | **timezone** *tzname*]]] {**-to** *mm/dd/[yy]yy[hh[:]mm[**tz** | **timezone** tzname]]* | **-for** *[h]hh[:]mm* [**-days** *n*] | **-days** *n*}	
planman [*connection_parameters*] **exttrial** *file_name* {**-to** *mm/dd/[yy]yy[hh[:]mm[**tz** | **timezone** tzname]]* | **-for** *[h]hh[:]mm* [**-days** *n*] | **-days** *n*}	Creates a trial plan of a production plan extension.
planman [*connection_parameters*] **crtfc** *file_name* [**-from** *mm/dd/[yy]yy* [*hhmm* [**tz** | **timezone** *tzname*]]] {**-to** *mm/dd/[yy]yy[hh[:]mm[**tz** | **timezone** tzname]]* | **-for** *[h]hh[:]mm* [**-days** *n*] | **-days** *n*}	Creates a forecast plan.
planman [*connection_parameters*] **unlock**	Unlocks the production plan.
ResetPlan [*connection_parameters*] [**-scratch**]	Resets the production plan.
planman reset -scratch	Removes the preproduction plan while maintaining the Symphony file.
planman [*connection_parameters*] **resync**	Replicates plan data from the Symphony file to the database.
planman [*connection_parameters*] **checksync**	Monitors the progress and outcome of the process of replicating plan data in the database.

where *connection_parameters* are the following:

```
[-file filename]
[-host hostname]
[-port port_name]
[-protocol protocol_name] [-proxy proxy_name]
[-proxyport proxy_port_number]
[-password user_password]
[-timeout seconds]
[-username user_name]
```

For more information, see Creating and extending the production plan on page 125.

Managing objects in the database

The section is divided into the following subsections:

- General purpose commands on page 1405
- Scheduling objects on page 1405
- Composer commands on page 1413

General purpose commands

This section describes the names, the syntax of general purpose commands that are run from the **composer** program, and the user authorization, when needed, that is necessary to run them.

Table 184. General purpose commands

Command	Syntax	User Authorization
continue	**continue**&*command argument*&*command argument*	Authorization for using composer
edit	**edit** *filename*	Authorization for using composer
exit	**exit**	Authorization for using composer
help	**help** *commandname*	Authorization for using composer
redo	**redo** *directives*	Authorization for using composer
validate	**validate** *filename* [;syntax]	Authorization for using composer
version	**version**	Authorization for using composer

Scheduling objects

This section contains all scheduling objects definition syntax.

In the table displaying the list of commands that can be used against the scheduling object, *filename* indicates an existing file when used in the syntax for the **add** and **replace** commands, it indicates a not existing file when used in the syntax for the **create/extract** command.

Calendar

File definition syntax:

$calendar
[folder/]calendarname [**"***description"*]
date [...]

Domain

File definition syntax:

domain *domainname*[**description** *"description"*]
*** manager** *workstation*
[**parent** *domainname* | **ismaster**]
end

Event rule

XML definition syntax:

- eventRule name=" " ruleType=" " isDraft=" " (1, 1)
  - description (0, 1)
  - timeZone (0, 1)
  - validity from=" " to=" " (0, 1)
  - activeTime start=" " end=" " (0, 1)
  - timeInterval amount=" " unit=" " (0, 1)
  - eventCondition eventProvider=" " eventType=" " (1, n)
    - scope (0, 1)
    - filteringPredicate (0, 1)
      - attributeFilter name=" " operator="eq" (0, n)
        - value (1, n)
      - attributeFilter name=" " operator="ne" (0, n)
        - value (1, n)
      - attributeFilter name=" " operator="le" (0, n)
        - value (1, 1)
      - attributeFilter name=" " operator="ge" (0, n)
        - value (1, 1)
      - attributeFilter name=" " operator="range" (0, 1)
        - value (1, 2)
  - correlationAttributes (0, 1)
    - attribute name=" " (1, n)
  - action actionProvider=" " actionType=" " responseType=" " (0, n)
    - description (0, 1)
    - scope (0, 1)

- parameter name=" "(1, n)
- value (1, 1)

Folder

Folder definition syntax:

folder *foldername*
end
folder *foldername*/*foldername*
end

Job

File definition syntax:

$jobs
[*[folder/]workstation***#**][*folder/*]*jobname*
{**scriptname** *filename* **streamlogon** *username* |
docommand "*command*" **streamlogon** *username* |
task *job_definition* }
[**description** "*description*"]
[**tasktype** *tasktype*]
[**interactive**]

[**succoutputcond** *Condition_Name* "*Condition_Value*"]
[**outputcond** *Condition_Name* "*Condition_Value*"]

[**recovery**
{**stop**
[**after** [*[folder/]workstation#*][*folder/*]*jobname*]
[**abendprompt** "*text*"]]
|**continue**
[**after** [*[folder/]workstation#*][*folder/*]*jobname*]
[**abendprompt** "*text*"]]
|**rerun [same_workstation]**
[[**repeatevery** *hhmm*] [**for** *number* **attempts**]]
[**after** [*[folder/]workstation#*][*folder/*]*jobname*]
|[**after** [*[folder/]workstation#*][*folder/*]*jobname*]
[**abendprompt** "*text*"]}

Note:

1. This keyword is available on Windows platforms only.

Job stream

File definition syntax:

schedule [*[folder/]workstation#*][*folder/]jobstreamname*
 # comment
[**validfrom** *date*]
[**timezone** | **tz** *tzname*]
[**description** *"text"*]
[**draft**]
[**isservice**
servicename ***service_name***
servicedescription ***service_description***
servicetags " | ***service_tag*** |***service_tag***|"
servicevariables "{***var***: \"***value***\"}"]
[**vartable** *[folder/]table_name*]
[**freedays** *[folder/]calendarname* [**-sa**] [**-su**]]
[**on** [**runcycle** *name*
[**validfrom** *date*] [**validto** *date*]
[**description** *"text"*]
[**vartable** *[folder/]table_name*]]
{*date* | *day*|*[folder/]calendar* | *request* | *"icalendar"* | **$RCG** *[folder/]runcyclegroup*} [**,...**]
[**fdignore** | **fdnext** | **fdprev**]
[({**at** *time* [**+***n* **day**[**s**]] |
schedtime *time* [**+***n* **day**[**s**]]}
[**until** | **jsuntil** *time* [**+***n* **day**[**s**]] [**onuntil** *action*]]
[**every** *rate* {**everyendtime** *time*[**+***n* **day**[**s**]]}
[**deadline** *time* [**+***n* **day**[**s**]]])]]
[**,...**]]

[**except** [**runcycle** *name*]
[**validfrom** *date*] [**validto** *date*]
[**description** *"text"*]
{*date* | *day*|*[folder/]calendar* | *request* | *"icalendar"* | **$RCG** *[folder/]runcyclegroup*} [**,...**]
[**fdignore** | **fdnext** | **fdprev**]
[{(**at** *time* [**+***n* **day**[**s**]])] |
(**schedtime** *time* [**+***n* **day**[**s**]])}]
[**,...**]
[**startcond filecreated** | **filemodified** *[folder/]workstation_name#file_name*
user *username*

```
        interval seconds
        [(alias startcond_jobname
        rerun batch outfile outputfilename
        params "filemonitor additional parameters")] |
    startcond job  [folder/]workstation_name#[folder/]job_name
        outcond  joboutputcondition
        interval seconds
        [(alias startcond_jobname rerun)]]
   [{at time [timezone | tz tzname] [+n day[s]] |
   schedtime time [timezone | tz tzname] [+n day[s]]}]
   [until | jsuntil time [timezone | tz tzname] [+n day[s]] [onuntil action]]
   [deadline time [timezone | tz tzname] [+n day[s]]]
   [carryforward]
   [matching {previous | sameday | relative from [+ | -] time to [+ | -] time|
     from time [+ | -n day[s]] to time [+ n day[s]] [,...]}]
   [follows {[netagent::][workstation#]jobstreamname[.jobname |
     @] [previous|
     sameday|relative from [+|-] time to [+|-] time|
     from time [+|-n day[s]] to time [+|-n day[s]]
   ][if <condition> [| <condition>...]]
      } ] [,...]] [...]
   [join condition_name [number | numconditions | all] of
     description "..."]
     .....
     endjoin

        [keysched]
   [limit joblimit]
   [needs { [n] [[folder/]workstation#][folder/]resourcename } [,...] ] [...]
   [opens { [[folder/]workstation#]"filename" [ (qualifier) ] [,...] }]  [...]
   [priority number | hi | go]
   [prompt {[folder/]promptname|"[:|!]text"} [,...] ] [...]
   [onoverlap {parallel|enqueue|donotstart}]
  :
job-statement
   # comment
job_name [job_alias]
[outcond joboutputcondition interval seconds]
   [{at time [timezone | tz tzname] [+n day[s]] |
   schedtime time [timezone | tz tzname] [+n day[s]]}][,...]
   [until  time [timezone | tz tzname] [+n day[s]] [onuntil action]
   [deadline time [timezone | tz tzname] [+n day[s]] [onlate action] ]
```

[**maxdur** *time* | *percentage* % **onmaxdur** *action*]
[**mindur** *time* | *percentage* % **onmindur** *action*]
[**every** *rate*]
[**follows** {[*netagent::*][*workstation#*]*jobstreamname*{.*jobname* **@**} [**previous**|
sameday|**relative from [+|-]** *time* **to [+|-]** *time* |
from *time* [+|-*n* day[s]] **to** *time* [+|-*n* day[s]]
]}][**if** <*condition*> [| <condition>...]] [**,**...]] [...]
[**join** *condition_name* [*number* | *numconditions* | all] of
description "..."]
.....
endjoin

[**confirmed**]
[**critical**]
[**keyjob**]
[**needs** { [*n*] [*[folder/]workstation#*][*folder/]resourcename* } [**,**...]] [...]
[**opens** { [*[folder/]workstation#*]"*filename*" [**(***qualifier***)**] [**,**...] }] [...]
[**priority** *number* | **hi** | **go**]
[**prompt** {*[folder/]promptname*|"[**:**|**!**]*text*"} [**,**...]] [...]
[**nop**]
[**statistictype custom**]

[*job-statement***...**]
end

Parameter

File definition syntax:

$parm
[*tablename.*][*folder/]variablename "variablevalue"*
...

Prompt

File definition syntax:

$prompt
[folder/]promptname "[**:** | **!**]*text*?

Resource

File definition syntax:

$resource
[folder/]workstation#*[folder/]resourcename units* ["*description*?]

Run cycle group

File definition syntax:

runcyclegroup
[folder/]runcyclegroupname [**description***"text"*]
vartable *[folder/]tablename*
[freedays *[folder/]calendarname* [**-sa**] [**-su**]]
[**on** [**runcycle** *[folder/]name*
[**validfrom** *date*] [**validto** *date*]
[**description** *"text"*]
[**vartable** *[folder/]table_name*]]
{*date*|*day*|*[folder/]calendar*|*request*|*"icalendar"*} [**,...**]
[**fdignore**|**fdnext**|**fdprev**][**subset** *subsetname* **AND**|**OR**]
[({**at** *time* [**+***n* **day**[**s**]] |
schedtime *time* [**+***n* **day**[**s**]]}
[**until** | **jsuntil** *time* [**timezone**|**tz** *tzname*][**+***n* **day**[**s**]]
[**onuntil***action*]]
[**every** *rate* {**everyendtime** *time*[**+***n* **day**[**s**]]}
[**deadline** *time* [**+***n* **day**[**s**]]])]]
[**,...**]]
[**except** [**runcycle** *[folder/]name*]
[**validfrom** *date*] [**validto** *date*]
[**description** *"text"*]
{*date*|*day*|*[folder/]calendar*|*request*|*"icalendar"*} [**,...**]
[**fdignore**|**fdnext**|**fdprev**][**subset** *subsetname* **AND**|**OR**]
[{(**at** *time* [**+***n* **day**[**s**]])] |
(**schedtime** *time* [**+***n* **day**[**s**]])}]
[**,...**]
[{**at** *time* [***timezone*|*tz*** *tzname*] [**+***n* **day**[**s**]] |
schedtime *time* [***timezone*|*tz*** *tzname*] [**+***n* **day**[**s**]]}]
[**until** | **jsuntil** *time* [**timezone**|**tz** *tzname*][**+***n* **day**[**s**]] [**onuntil***action*]]
[**deadline** *time* [**timezone**|**tz** *tzname*] [**+***n* **day**[**s**]]]
end

Variable table

File definition syntax:

vartable *[folder/]table_name*

[**description** "*description*"]
[**isdefault**]
members
[*variablename* "*variablevalue*"]
...
[*variablename* "*variablevalue*"]
end

For more information, see Customizing your workload using variable tables on page 164.

Workload application

File definition syntax:

wat *[folder/]wat_name*
[**description** "description"]
[**vendor** "vendor"]
jstreams
[*[folder/]workstation#[folder/]jobstream* [*[folder/]workstation#[folder/]jobstream*]...]
end

Workstation

File definition syntax:

cpuname *[folder/]workstation* [**description** "*description*"]
[**licensetype** *type*]
[**vartable** *table_name*]
os *os-type*
[**node** *hostname*] [**tcpaddr** *port*]
[**secureaddr** *port*] [**timezone**|**tz** *tzname*]
[**domain** *domainname*]
[**for maestro** [**host** *[folder/]workstation* [**access** *method* | **agentID** *agentID*]]
[**type fta** | **s-agent** | **x-agent** | **manager** | **broker** | **agent** | **rem-eng** |
pool | **d-pool**]
[**ignore**]
[**autolink on** | **off**]
[**behindfirewall on** | **off**]
[**securitylevel enabled** | **on** | **force** | **force_enabled**]
[**fullstatus on** | **off**]

[**server** *serverid*]]
[**protocol** `http | https`]
[**members** [*[folder/]workstation*] [...]]
[**requirements** *jsdl_definition*]]
end

Workstation class

File definition syntax:

cpuclass *[folder/]workstationclass*
[**description** "*description*"]
[**ignore**]
members [*[folder/]workstation* | **@**] [...]
end
[**cpuname** ...]
[**cpuclass** ...]
[**domain** ...]

User definition

File definition syntax:

```
username[workstation#][domain\]username[@internet_domain]
password "password"
end
```

Composer commands

This section describes the operations you can perform in the database using the **composer** command line interface program with syntax:

```
composer [connection_parameters] [-defaultws twscpu]
        ["command[&[command]][...]"]
```

where *connection_parameters*, if they are not supplied in the `localopts` or `useropts` files, are the following:

```
[-file filename] |
[-host hostname]
[-port port_name]
[-protocol protocol_name]
[-proxy proxy_name]
[-proxyport proxy_port_number]
[-password user_password]
[-timeout seconds]
[-username user_name]
```

See Setting up options for using the user interfaces on page 93 for more details.

These operations can only be run from any composer client command line installed.

In Table 185: Composer commands on page 1414 displaying the list of commands that can be used against the scheduling object, *filename* indicates an existing file when used in the syntax for the **add** and **replace** commands, it indicates a not existing file when used in the syntax for the **create/extract** command.

Table 185. Composer commands

Command	Syntax	User Authorization
add	{**add** | **a**} *filename* [**;unlock**]	*add* or *modify*
authenticate	{**authenticate** | **au**} [**username=***username* **password=***password*]	
chgfolder	{**chfolder** | **cf**} *foldername*	***display***
continue	{**continue** | **co**}	
create extract	{**create** | **cr** | **extract** | **ext**} *filename* **from** {[**calendars** | **calendar** | **cal**=*[folder/]calname*] | [**eventrule** | **erule** | **er**=*[folder/]eventrulename*] | [**parms** | **parm** | **vb**=[*[folder/]tablename*.]*variablename*] | [**vartable** | **vt**=*[folder/]tablename*] | [**prompts** | **prom**=*[folder/]promptname*] | [**resources** | **resource** | **res**=[*[folder/]workstationname#*][*folder/]resourcename*] | [**runcyclegroup** | **rcg**=*[folder/]runcyclegroupname*] | [**cpu**={*[folder/]workstationname* | *[folder/]workstationclassname* | *domainame*}] | [**workstation** | **ws**=*[folder/]workstationname*] | [**workstationclass** | **wscl**=*[folder/]workstationclassname*] | [**domain** | **dom**=*domainame*] | [**jobs** | **jobdefinition** | **jd**=[*[folder/]workstationname#*][*folder/]jobname*] | [**sched** | **jobstream** | **js**= [*[folder/]workstationname#*][*folder/]jstreamname *[**valid from** date|**valid to** date |**valid in** date date]* *[**;full**]*] | [**users** | **user**=[*[folder/]workstationname#*]*username* [;password]] | [**accesscontrollist** | **acl for** *securitydomainname*] | [**securitydomain** | **sdom**=*securitydomainname*] | [**securityrole** | **srol**=*securityrolename*]} [**;lock**]	*display*
delete	{**delete** | **de**} {[**calendars** | **calendar** | **cal**=*[folder/]calname*] | [**domain** | **dom**]=*domainame*] | [**eventrule** | **erule** | **er**=*[folder/]eventrulename*] | [**folder** | **fol**=*foldername*] | [**parms** | **parm** | **vb**=[*[folder/]tablename*.]*variablename*] |	*delete*

Table 185. Composer commands (continued)

Command	Syntax	User Authorization
	[**prompts** | **prom**=*[folder/]promptname*] |	
	[**resources** | **resource** | **res**=[*[folder/]workstationname#*][*folder/]resourcename*] |	
	[**runcyclegroup** | **rcg**=*[folder/]runcyclegroupname*] |	
	[**vartable** | **vt**=*[folder/]tablename*] |	
	[**wat**=*[folder/]workloadapplicationtemplatename*]	
	[**cpu**={*[folder/]workstationname* [**;force**] | *[folder/]workstationclassname* [**;force**]| *domainame*}]	
	[**workstation** | **ws**=*[folder/]workstationname*] [**;force**] |	
	[**workstationclass** | **wscl**]=*[folder/]workstationclassname* [**;force**] |	
	[**jobs** | **jobdefinition** | **jd**]=[*[folder/]workstationname#*][*folder/*]*jobname* |	
	[**sched** | **jobstream** | **js**]= [*[folder/]workstationname#*][*folder/*]*jstreamname*	
	*[**valid from** date*|***valid to** date* |***valid in** date date]*] |	
	[**users** | **user**=[*[folder/]workstationname#*]*username*] |	
	[**accesscontrollist** | **acl for** *securitydomainname*] |	
	[**securitydomain** | **sdom**=*securitydomainname*] |	
	[**securityrole** | **srol**=*securityrolename*]}	
	[**;noask**]	
display	{**display** | **di**}	*display*
	{[**calendars** | **calendar** | **cal**=*[folder/]calname*] |	
	[**eventrule** | **erule** | **er**=*[folder/]eventrulename*] |	
	[**folder**|**fol**=*foldername* |]	
	[**parms** | **parm** | **vb**=*variablename.*]*variablename*] |	
	[**vartable** | **vt**=*[folder/]tablename*] |	
	[**prompts** | **prom**=*[folder/]promptname*] |	
	[**resources** | **resource** | **res**=[*[folder/]workstationname#*][*folder/]resourcename*] |	
	[**runcyclegroup** | **rcg**=*[folder/]runcyclegroupname*] |	
	[**cpu**={*[folder/]workstationname* | *[folder/]workstationclassname* | *domainame*}]	
	[**wat**=*[folder/]workloadapplicationtemplatename*]	
	[**workstation** | **ws**=*[folder/]workstationname*] |	
	[**workstationclass** | **wscl**=*[folder/]workstationclassname*] |	
	[**domain** | **dom**=*domainame*] |	
	[**jobs** | **jobdefinition** | **jd**=[*[folder/]workstationname#*][*folder/*]*jobname*] |	
	[**sched** | **jobstream** | **js**= [*[folder/]workstationname#*][*folder/*]*jstreamname*	
	[**valid from** *date*|**valid to** *date* |**valid in** *date date*]	
	[**;full**]] |	
	[**users** | **user**=[*[folder/]workstationname#*]*username*] |	
	[**accesscontrollist** | **acl for** *securitydomainname*] |	
	[**securitydomain** | **sdom**=*securitydomainname*] |	

Table 185. Composer commands (continued)

Command	Syntax	User Authorization
	[**securityrole** | **srol**=*securityrolename*]}	
	[**;offline**]	
edit	{**edit** | **ed**} *filename*	
exit	{**exit** | **e**}	
list print	{**list** | **l**}	*display*
	{[**calendars** | **calendar** | **cal**=*[folder/]calname*] |	
	[**eventrule** | **erule** | **er**=*[folder/]eventrulename*] |	
	folder|**fol**=*foldername* |	
	[**parms** | **parm** | **vb**=[*[folder/]tablename*.]*variablename*] |	
	[**vartable** | **vt**=*[folder/]tablename*] |	
	[**prompts** | **prom**=*[folder/]promptname*] |	
	[**resources** | **resource** | **res**=[*[folder/]workstationname#*]*resourcename*] |	
	[**runcyclegroup** | **rcg**=*[folder/]runcyclegroupname*] |	
	[**cpu**={*[folder/]workstationname* | *[folder/]workstationclassname* | *domainame*}]	
	[**wat**=*workloadapplicationtemplatename*]	
	[**workstation** | **ws**=*[folder/]workstationame*] |	
	[**workstationclass** | **wscl**=*[folder/]workstationclassname*] |	
	[**domain** | **dom**=*domainame*] |	
	[**jobs** | **jobdefinition** | **jd**=[*[folder/]workstationame#*][*folder/*]*jobname*] |	
	[**sched** |**jobstream** | **js**= [*[folder/]workstationame#*][*folder/*]*jstreamname*	
	*[**valid from** date*|	
	valid to** date* |valid in** date date]* |	
	[**users** | **user**=[*[folder/]workstationame#*]*username*] |	
	[**accesscontrollist** | **acl for** *securitydomainname*] |	
	[**securitydomain** | **sdom**=*securitydomainname*] |	
	[**securityrole** | **srol**=*securityrolename*]}	
	[**;offline**]	
	[**;showid**]	
listfolder	{**listfolder** | **lf**} *foldername*	***list***, or ***list*** and ***display***
lock	{**lock** | **lo**}	*modify*
	{[**calendars** | **calendar** | **cal**=*[folder/]calname*] |	
	[**eventrule** | **erule** | **er**=*[folder/]eventrulename*] |	
	[**folder** | **fol**=*foldername*] |	
	[**parms** | **parm** | **vb**=[*[folder/]tablename*.]*variablename*] |	
	[**vartable** | **vt**=*[folder/]tablename*] |	

Table 185. Composer commands (continued)

Command	Syntax	User Authorization
	[**prompts** | **prom**=*[folder/]promptname*] |	
	[**resources** | **resource** | **res**=[*[folder/]workstationname#*][*folder/]resourcename*] |	
	[**runcyclegroup** | **rcg**=*[folder/]runcyclegroupname*] |	
	[**cpu**={*[folder/]workstationname* | *[folder/]workstationclassname* | *domainame*}]	
	[**workstation** | **ws**=*[folder/]workstationname*] |	
	[**workstationclass** | **wscl**=*[folder/]workstationclassname*] |	
	[**domain** | **dom**=*domainame*] |	
	[**jobs** | **jobdefinition** | **jd**=[*[folder/]workstationname#*][*folder/*]*jobname*] |	
	[**sched**|**jobstream**|**js**= [*[folder/]workstationname#*][*folder/*]*jstreamname*	
	*[**valid from** date*|***valid to** date* |***valid in** date date]*] |	
	[**users** | **user**=[*[folder/]workstationname#*]*username*] |	
	[**accesscontrollist** | **acl for** *securitydomainname*] |	
	[**securitydomain** | **sdom**=*securitydomainname*] |	
	[**securityrole** | **srol**=*securityrolename*]}	
mkfolder	{**mkfolder** | **mf**} *foldername*	***add***, if new, *modify* if existing
modify	{**modify** | **m**}	*modify* or *add*
	{[**calendars** | **calendar** | **cal**=*[folder/]calname*] |	
	[**eventrule** | **erule** | **er**=*[folder/]eventrulename*] |	
	folder|**fol** |	
	[**parms** | **parm** | **vb**=[*[folder/]tablename.*]*variablename*] |	
	[**vartable** | **vt**=*[folder/]tablename*] |	
	[**prompts** | **prom**=*[folder/]promptname*] |	
	[**resources** | **resource** | **res**=[*[folder/]workstationname#*][*folder/]resourcename*] |	
	[**runcyclegroup** | **rcg**=*[folder/]runcyclegroupname*] |	
	[**cpu**={*[folder/]workstationname* | *[folder/]workstationclassname* | *domainame*}]	
	[**wat**=*workloadapplicationtemplatename*]	
	[**workstation** | **ws**=*[folder/]workstationname*] |	
	[**workstationclass** | **wscl**=*[folder/]workstationclassname*] |	
	[**domain** | **dom**=*domainame*] |	
	[**jobs** | **jobdefinition** | **jd**=[*[folder/]workstationname#*][*folder/*]*jobname*] |	
	[**sched**|**jobstream**|**js**= [*[folder/]workstationname#*][*folder/*]*jstreamname*	
	*[**valid from** date*|***valid to** date* |***valid in** date date]*	
	*[**;full**]* |	
	[**users** | **user**=[*[folder/]workstationname#*]*username*] |	
	[**accesscontrollist** | **acl for** *securitydomainname*] |	
	[**securitydomain** | **sdom**=*securitydomainname*] |	
	[**securityrole** | **srol**=*securityrolename*]}	

Table 185. Composer commands (continued)

Command	Syntax	User Authorization
new	**new** [**calendar** | **domain** | **eventrule** | **folder**|**fol** | **job** | **jobstream** | **parameter** | **prompt** | **resource** | **runcyclegroup** | **user** | **vartable** | **wat** | **workstation** | **workstationclass** | **accesscontrollist** | **securitydomain** | **securityrole**]	*add* or *modify*
rename	{**rename** | **rn**} {**calendars**|**calendar**|**cal** | **parms**|**parm**|**vb** | **vartable**|**vt** | **prompts**|**prom** | **resorces**|**resource**|**res** | **runcyclegroup**|**rcg** | **workstation**|**ws** | **workstationclass**|**wscl** | **domain**|**dom** | **jobs**|**jobdefinition**|**jd** | **jobsched**|**jb** | **eventrule**|**erule**|**er** **sched**|**jobstream**|**js** | **users**|**user** } *old_object_identifier new_object_identifier*	*add* and *delete*
renamefolder	{**renamefolder** | **rn**} *previousname newname*	***delete*** and ***add***

Table 185. Composer commands (continued)

Command	Syntax	User Authorization
replace	{**replace** | **rep**} *filename* [**;unlock**]	*modify* or *add*
rmfolder	{**rmfolder** | **rf**} *foldername*	***delete***
unlock	{**unlock** | **un**} {[**calendars** | **calendar** | **cal**=*[folder/]calname*] | [**eventrule** | **erule** | **er**=*[folder/]eventrulename*] | [**folder** | **fol**=*foldername*] | [**parms** | **parm** | **vb**=[*[folder/]tablename.*]*variablename*] | [**vartable** | **vt**=*[folder/]tablename*] | [**prompts** | **prom**=*[folder/]promptname*] | [**resources** | **resource** | **res**=[*[folder/]workstationname#*][*[folder/]resourcename*] | [**runcyclegroup** | **rcg**=*[folder/]runcyclegroupname*] | [**cpu**={*[folder/]workstationname* | *[folder/]workstationclassname* | *domainame*}] [**workstation** | **ws**=*[folder/]workstationame*] | [**workstationclass** | **wscl**=*[folder/]workstationclassname*] | [**domain** | **dom**=*domainame*] | [**jobs** | **jobdefinition** | **jd**=[*[folder/]workstationame#*][*folder/*]*jobname*] | [**sched**|**jobstream**|**js**= [*[folder/]workstationame#*][*folder/*]*jstreamname* *[**valid from** date*|***valid to** date* |***valid in** date date]*] | [**users** | **user**=[*[folder/]workstationame#*]*username*] | [**accesscontrollist** | **acl for** *securitydomainname*] | [**securitydomain** | **sdom**=*securitydomainname*] | [**securityrole** | **srol**=*securityrolename*]} [**;forced**]	*modify* and *display*
update	{**update** | **up**} {[**cpu**={*[folder/]workstationame* | *[folder/]workstationclassname*}] | [**workstation** | **ws**=*[folder/]workstationame*] | [**workstationclass** | **wscl**=*[folder/]workstationclassname*]; [**filter** *workstation_filter_criteria*= selection [...]]; **set** [**ignore**= *on* | *off*]]} [;noask]	
validate	{**validate** | **val**} *filename* [**;syntax**]	
version	{**version** | **v**}	

Managing objects in the plan

This section describes the operations you can perform against the plan using the **conman** command line interface program with syntax:

```
conman ["command[&[command]...] [&]"]
```

Conman commands

This section lists the commands you can run from the **conman** program.

This is how you access to the **conman** command line:

```
conman [connection_parameters] ["command[&[command]...] [&]"]
```

where *connection_parameters*, if they are not supplied in the `localopts` or `useropts` files, are the following:

```
[-file filename]
[-host hostname]
[-port port_name]
[-protocol protocol_name]
[-proxy proxy_name]
[-proxyport proxy_port_number]
[-password user_password]
[-timeout seconds]
[-username user_name]
```

For more details, see Setting up options for using the user interfaces on page 93.

This is how you select jobs in commands:

```
[workstation#]
[folder/]{jobstreamname(hhmm[ date]) job|jobnumber}
[{+|~}jobqualifier[...]]
```

or:

```
[workstation#]
jobstream_id.
job
[{+|~]jobqualifier[...]]
;schedid
```

This is how you select job streams in commands:

```
[workstation#]
[folder/]jobstreamname(hhmm[ date])
[{+|~}jobstreamqualifier[...]]
```

or:

```
[workstation#]
jobstream_id
 ;schedid
```

You can run these commands from different types of workstations. In this table:

F

stands for domain managers and fault-tolerant agents.

S

stands for standard agents.

For each command you find the name, the syntax, the type of workstations from where you can issue the command, and the needed authorization, if any.

Table 186. Commands that can be run from conman

Command	Syntax	Workstation types	User Authorization
adddep job	{**adddep job** | **adj**} **=** *jobselect* **;***dependency*[**;**...] [**;noask**]	F	*adddep* - (*use* when using prompts and needs)
adddep sched	{**adddep sched** | **ads**} **=** *jstreamselect* **;***dependency*[**;**...] [**;noask**]	F	*adddep* - (*use* when using prompts and needs)
altpass	**altpass** [*[folder/]workstation***#**] *username* [**;**"*password*"]	F	*altpass*
altpri	{**altpri** | **ap**} *jobselect* | *jstreamselect* [**;***pri*] [**;noask**]	F	*altpri*
cancel job	{**cancel job** | **cj**} *jobselect* [**;pend**] [**;noask**]	F	*cancel*
cancel sched	{**cancel sched** | **cs**} *jstreamselect* [**;pend**] [**;noask**]	F	*cancel*
checkhealthstatus	{**checkhealthstatus** | **chs**} [*[folder/]workstation*]	M,F,S	
chfolder	{**chfolder** | **cf**} *foldername*	F	display

Table 186. Commands that can be run from conman (continued)

Command	Syntax	Workstation types	User Authorization
confirm	{**confirm** | **conf**} *jobselect* **;**{**succ** | **abend**} [**;IF** '*output_condition_name*[**,** *output_condition_name*] [**,** ...]'] [**;noask**]	F	*confirm*
console	{**console** | **cons**} [**sess** | **sys**] [**;level=***msglevel*]	F-S	*console*
continue	{**continue** | **cont**}	F-S	
deldep job	{**deldep job** | **ddj**} *jobselect* **;***dependency*[**;**...] [**;noask**]	F	*deldep*
deldep sched	{**deldep sched** | **dds**} *jstreamselect* **;***dependency*[**;**...] [**;noask**]	F	*deldep*
deployconf	{**deployconf** | **deploy**} [*domain***!**][*folder/*]*workstation*	F,S	Permission to *start* actions on *cpu* objects
display	{**display file** | **df**} *filename* [**;offline**] {**display job** | **dj**} *jobselect* [**;offline**] {**display sched** | **ds**} *jstreamselect* [**valid** {**at** *date* | **in** *date date*} [**;offline**]	F-S[1]	*display*
exit	{**exit** | **e**}	F-S	
fence	{**fence** | **f**} *workstation* **;***pri* [**;noask**]	F	*fence*

Table 186. Commands that can be run from conman (continued)

Command	Syntax	Workstation types	User Authorization
help (UNIX® only)	{**help** | **h**} {*command*|*keyword*}	F-S	
kill	{**kill** | **k**} *jobselect* [**;noask**]	F	*kill*
limit cpu	{**limit cpu** | **lc** } *[folder/]workstation* **;***limit* [**;noask**]	F	*limit*
limit sched	{**limit sched** | **ls** } *jstreamselect* **;***limit* [**;noask**]	F	*limit*
link	{**link** | **lk**} [*domain***!**][*folder/]workstation* [**;noask**]	F-S	*link*
listfolder	{**listfolder** | **lf**} *foldername*	F	If ***enListSecChk*** global option is set to **yes** on the MDM, then, you must have either ***list*** access, or ***list*** and ***display*** access.
listsym	{**listsym** | **lis**} [**trial** | **forecast**] [**;offline**]	F	
recall	{**recall** | **rc**} [*[folder/]workstation*] [**;offline**]	F	*display*
redo	{**redo** | **red**}	F-S	
release job	{**release job** | **rj**} *jobselect* [**;***dependency*[**;**...]] [**;noask**]	F	*release*

Table 186. Commands that can be run from conman (continued)

Command	Syntax	Workstation types	User Authorization
release sched	{**release sched** | **rs**} *jstreamselect* [**;***dependency*[**;**...]] [**;noask**]	F	*release*
reply	{**reply** | **rep**} { *promptname* | [*[folder/]workstation***#**]*msgnum*} **;***reply* [**;noask**]	F	*reply*
rerun	{**rerun** | **rr**} *jobselect* [[**;from=**[*[folder/]wkstat***#**]*job* [**;at=***time*] [**;pri=***pri*]] | [[**;streamlogon**|**logon=***new_logon*] [**;docommand=***"new_command"***;script=***"new_script"*]] | [**;step=***step*]] [**;sameworkstation=**] [**;noask**]	F	*rerun*
resource	{**resource** | **reso**} [*[folder/]workstation***#**] *[folder/]resource***;***num* [**;noask**]	F	*resource*
setsym	{**setsym** | **set**} [**trial** | **forecast**] [*filenum*]	F	
showcpus	{**showcpus** | **sc**} [[*domain***!**]*[folder/]workstation*] [**;info**|**;link**] [**;offline**] [**;showid**]	F-S	*list*[2]
showdomain	{**showdomain** | **showdom** | **sd**} [*domain*] [**;info**] [**;offline**]	F-S	*list*[2]
showfiles	{**showfiles** | **sf**} [[*[folder/]workstation#*]*file*] [**;***state*[**;**...]] [**;keys**] [**;offline**]	F	

Table 186. Commands that can be run from conman (continued)

Command	Syntax	Workstation types	User Authorization
	{**showfiles** | **sf**} [[*[folder/]workstation#*]*file*] [**;***state*[**;**...]] [**;deps**[**;keys** | **info** | **logon**]] [**;offline**]		
showjobs	{**showjobs** | **sj**} [*jobselect*] [**;deps**[**;keys** | **info** | **logon**]] [**;short** | **single**] [**;offline**] [**;showid**] [**;props**] {**showjobs** | **sj**} [*jobselect* | [*[folder/]workstation#*]*jobnumber.hhmm*] [**;stdlist**[**;keys**]] [**;short** | **single**] [**;offline**] [**;showid**] [**;props**]	F	*list*[2]
showprompts	{**showprompts** | **sp**} [*[folder/]promptselect*] [**;keys**] [**;offline**] [**;showid**] {**showprompts** | **sp**} [*[folder/]promptselect*] [**;deps**[**;keys** | **info** | **logon**]] [**;offline**] [**;showid**]	F	*list*[2]
showresources	{**showresources** | **sr**} [[*[folder/]workstation#*][*folder/]resourcename*] [**;keys**] [**;offline**] [**;showid**] {**showresources** | **sr**} [[*[folder/]workstation#*][*folder/]resourcename*] [**;deps**[**;keys** | **info** | **logon**]]	F	*list*[2]

Table 186. Commands that can be run from conman (continued)

Command	Syntax	Workstation types	User Authorization
	[;offline] **[;showid]**		
showschedules	{**showscheds** | **ss**} [*jstreamselect*] **[;keys]** **[;offline]** **[;showid]** {**showscheds** | **ss**} [*jstreamselect*] **[;deps[;keys | info | logon]]** **[;offline]** **[;showid]**	F	*list*[2]
shutdown	{**shutdown** | **shut**} **[;wait]**	F-S	*shutdown*
start	**start** [*domain***!**]*[folder/]workstation* **[;mgr]** **[;noask]** **[;demgr]**	F-S	*start*
startappserver	**startappserver** [*domain***!**]*[folder/]workstation* **[;wait]**	F-S	Permission to *start* actions on *cpu* objects
startevtp	{**starteventprocessor** | **startevtp**} [*domain***!**]*workstation*	M[4]	Permission to *start* actions on *cpu* objects
startmon	{**startmon** | **startm**} [*domain***!**]*[folder/]workstation* **[;noask]**	F-S	Permission to *start* actions on *cpu* objects
status	{**status** | **stat**}	F-S	appserver
stop	**stop** [*domain***!**]*[folder/]workstation* **[;wait]** **[;noask]**	F-S	*stop*
stop ;progressive	**stop ;progressive**		*stop*

Table 186. Commands that can be run from conman (continued)

Command	Syntax	Workstation types	User Authorization
stopappserver	{**stopappserver** | **stopapps**} [*domain***!**][*folder/*]*workstation* [**;wait**]	F-S	Permission to *stop* actions on *cpu* objects
stopevtp	{**stopeventprocessor** | **stopevtp**} [*domain***!**][*[folder/]workstation*]	M[4]	Permission to *stop* actions on *cpu* objects
stopmon	{**stopmon** | **stopm**} [*domain***!**][*folder/]workstation [**;wait**] [**;noask**]	F-S	Permission to *stop* actions on *cpu* objects
submit docommand	{**submit docommand** = | **sbd**} [*[folder/]workstation#*]"*cmd*" [**;alias**[**=***name*]] [**;into=**[*[folder/]workstation***#**] {*jobstream_id***;schedid** |[*folder/*]*jobstreamname* ([*hhmm*[*date*]])}] [**;***joboption*[**;**...]]	F-S	*submit* - (*use* when using prompts and needs)
submit file	{**submit file** = | **sbf**} "*filename*" [**;alias**[**=***name*]] [**;into=**[*[folder/]workstation***#**]{*jobstream_id* **;schedid** |[*folder/*]*jobstreamname*([*hhmm*[*date*]])}] [**;***joboption*[**;**...]] [**;noask**]	F-S	*submit* - (*use* when using prompts and needs)
submit job	{**submit job** = | **sbj** = } [*workstation#*][*folder/*]*jobname* [**;alias**[**=***name*]] [**;into=**[*[folder/]workstation***#**]{*jobstream_id* **;schedid** | [*folder/*]*jobstreamname*([*hhmm*[*date*]])}] [**;***joboption*[**;**...]] [**;vartable=***tablename*] [**;noask**]	F-S[3]	*submit* - (*use* when using prompts and needs)
submit sched	{**submit sched** = | **sbs** = } [*[folder/]workstation#*][*folder/*]*jstreamname* [**;alias**[**=***name*]] [**;***jstreamoption*[**;**...]]	F-S[3]	*submit* - (*use* when using prompts and needs)

Table 186. Commands that can be run from conman (continued)

Command	Syntax	Workstation types	User Authorization
	[**;vartable=***tablename*] [**;noask**]		
switchevtp	{**switcheventprocessor** | **switchevtp**} *[folder/]workstation*	M[4]	Permission to *start* and *stop* actions on *cpu* objects
switchmgr	{**switchmgr** | **switchm**} *domain***;***newmgr*	F	*start stop*
system	[**:** | **!**] *system-command*	F-S	
tellop	{**tellop** | **to**} [*text*]	F-S	
unlink	**unlink** [*domain***!**]*[folder/]workstation* [**;noask**]	F-S	*unlink*
version	{**version** | **v**}	F-S	

where:

(1)

Indicates that you can only display files on a standard agent.

(2)

You must have **list** access to the object being shown if the ***enListSecChk*** option was set to **yes** on the master domain manager when the production plan was created or extended.

(3)

Indicates that you can use submit job (**sbj**) and submit sched (**sbs**) on a standard agent by using the connection parameters or specifying the settings in the `useropts` file when invoking the **conman** command line.

(4)

You can use this command on master domain managers and backup masters as well as on workstations installed as backup masters but used as ordinary fault-tolerant agents.

Utility commands

This section contains the list of the utility commands that you can run from the operating system command prompt. The utility commands are divided into three groups, those you can run on both UNIX® and Windows® operating systems, those you can run only on UNIX®, and those you can run only on Windows®.

Utility commands available for both UNIX® and Windows® operating systems

Table 187. Utility commands available for both UNIX® and Windows®

Command	Syntax
cpuinfo	**cpuinfo -V | -U** **cpuinfo** *workstation* [*infotype*] [...]
datecalc	**datecalc -V | -U** **datecalc** *base-date* [*offset*] [**pic** *format*][**freedays** *[folder/]Calendar_Name* [**-sa**] [**-su**]] **datecalc -t** *time* [*base-date*] [*offset*] [**pic** *format*] **datecalc** *yyyymmddhhtt* [*offset*] [**pic** *format*]
delete	**delete -V | -U** **delete** *filename*
evtdef	**evtdef -U | -V** **evtdef** [*connection parameters*] **dumpdef** *file-path* **evtdef** [*connection parameters*] **loaddef** *file-path*
evtsize	**evtsize -V | -U** **evtsize** *filename size* **evtsize -compact** *filename* [*size*] **evtsize -info** *filename* **evtsize -show** *filename* **evtsize -info | -show pobox**
filemonitor	**filemonitor -V | -U** **filemonitor -path** *path_to_monitor* -**event** *event_to_monitor* {**fileCreated** | **fileModified** [-**modificationCompletedTime** *seconds*]} [**-repositoryName** *repository_name*]

Table 187. Utility commands available for both UNIX® and Windows® (continued)

Command	Syntax
	[**-repositoryPath** *repository_path*] [**-recursive**] [**-outputFile** *output_filename*] [**-scanInterval** *scan_interval*] [**-maxEventsThreshold** *max_events*] [**-minFileSize** *min_file_size*] [**-timeout** *seconds* **filemonitor -reset**
jobinfo	**jobinfo -V** | **-U** **jobinfo** *job-option* [...]
jobstdl	**jobstdl -V** | **-U** **jobstdl** [**-day** *num*] [{**-first** | **-last** | **-num** *n* | **-all**}] **[-twslog]** [{**-name** ["*[folder/]jobstreamname* [(*hhmm date*),(*jobstream_id*)].]*jobname*"|*jobnum* | **-schedid** *jobstream_id.[folder/]jobname*}]
maestro	**maestro** [**-V** | **-U**]
makecal	**makecal** [**-c** *name*] **-d** *n* | **-e** | {**-f 1** | **2** | **3 -s** *date*} | **-l** | **-m** | **-p** *n* | {**-r** *n* **-s** *date*} | **-w** *n* [**-i** *n*] [**-x** | **-z**][**-freedays** *[folder/]Calendar_Name* [**-sa**] [**-su**]]
morestdl	**morestdl -V** | **-U** **morestdl** [**-day** *num*] [**-first** | **-last** | **-num** *n* | **-all**] **[-twslog]** [{**-name** ["*[folder/]jobstreamname* [(*hhmm date*),(*jobstream_id*)].]*jobname*"|*jobnum* | **-schedid** *jobstream_id.jobname*}]
param	**param -u** | **-V** **param {-c** | **-ec}** [*file.section.*|*file.*|*section.*] *variable* [*value*] **param** [*file.section.*|*file.*|*section.*] *variable* **param {-d** | **-fd}** [*file.section.*|*file.*|*section.*] *variable*
parms	**parms** {[**-V** | **-U**] | **-build**}

Table 187. Utility commands available for both UNIX® and Windows® (continued)

Command	Syntax
	parms {-replace | -extract} *filename* **parms** [**-d**]*[folder/]parametername***parms -c** *parametername value*
release	**release -V | -U** **release** [**-s**] [*[folder/]workstation***#**]*[folder/]resourcename* [*count*]
rmstdlist	**rmstdlist -V | -U** **rmstdlist** [**-p**] [*age*]
sendevent	**sendevent -V | ? | -help | -U | -usage** **sendevent** [**-hostname** *hostname*] [{**-port | -sslport}** *port*] *eventType* *source* [[*attribute*=*value*]...] In dynamic environments: **sendevent** [**-hostname** *hostname*] [**-port** *port*] *eventType* *source* [[*attribute*=*value*]...]
showexec	**showexec** [**-V | -U | INFO**]
ShutDownLwa	**ShutDownLwa**
StartUp	**StartUp** [**-V | -U**]
StartUpLwa	**StartUpLwa**
wa_pull_info	**wa_pull_info -help** **wa_pull_info** **-component *DWC|TWS*** [**-date** *yyyy-mm-dd*] [**-isroot**=*true|false*] [**-output** *output_path*]

Table 187. Utility commands available for both UNIX® and Windows® (continued)

Command	Syntax
	-user *userid* [**-workdir** *working_directory*]

Utility commands available for UNIX® operating system only

Table 188. Utility commands available for UNIX® only

Command	Syntax
at	**at -V** | **-U** **at -s***jstream* | **-q***queuetime-spec*
batch	**batch -V** | **-U** **batch** [**-s** *jstream*]
showexec	**showexec** [**-V** | **-U** | **-info**]
version	**version -V** | **-U** | **-h** **version** [**-a**] [**-f** *vfile*] [*file* [...]]

Utility commands available for Windows® operating system only

Table 189. Utility commands available for Windows® only

Command	Syntax
listproc **(UNSUPPORTED)**	**listproc**
killproc **(UNSUPPORTED)**	**killproc** *pid*
shutdown	**shutdown [-V | -U] [-appsrv]**

Report commands

This section contains a list and syntax of report commands and report extract programs. These commands are run from the operating system command prompt.

Report commands

Table 190. Report commands

Name	Output produced	Syntax
rep1	Reports 01 - Job Details Listing	**rep**[*x*] [**-V**|**-U**]
rep2	Report 02 - Prompt Listing	**rep**[*x*] [**-V**|**-U**]
rep3	Report 03 - Calendar Listing	**rep**[*x*] [**-V**|**-U**]
rep4a	Report 04A - Parameter Listing	**rep**[*x*] [**-V**|**-U**]
rep4b	Report 04B - Resource Listing	**rep**[*x*] [**-V**|**-U**]
rep7	Report 07 - Job History Listing	**rep7 -V**|**-U** **rep7** [**-c** *wkstat*] [**-s** *jstream_name*] [**-j** *job*] [**-f** *date* **-t** *date*] [**-l**]
rep8	Report 08 - Job Histogram	**rep8 -V**|**-U** **rep8** [**-f** *date* **-b** *time* **-t** *date* **-e** *time*] [**-i** *file*] [**-p**] **rep8** [**-b** *time* **-e** *time*] [**-i** *file*] [**-p**]
rep11	Report 11 - Planned Production Schedule	**rep11 -V**|**-U** **rep11** [**-m** *mm*[*yy*] [...]] [**-c** *wkstat* [...]] [**-s** *jstream_name*] [**-o** *output*]
reptr	Report 09A - Planned Production Summary Report 09B - Planned Production Detail Report 10A - Actual Production Summary	**reptr** [**-V**|**-U**] **reptr -pre** [-{**summary** | **detail**}] [*symfile*] **reptr -post** [-{**summary** | **detail**}] [*logfile*]

Table 190. Report commands (continued)

Name	Output produced	Syntax
	Report 10B - Actual Production Detail	
xref	Report 12 - Cross Reference Report	**xref** [**-V**|**-U**]
		xref [**-cpu** *wkstat*] [**-s** *jstream_name*] [**-depends**|**-files**|**-jobs**|**-prompts**|**-resource**|**-schedules**|**-when** […]]

Report extract programs

Table 191. Report extract programs

Extract Program	Used to generate	Syntax
jbxtract	Report 01	**jbxtract** [**-V** | **-U**] [**-j** *job*] [**-c** *wkstat*] [**-o** *output*]
	Report 07	
prxtract	Report 02	**prxtract** [**-V** | **-U**] [**-o** *output*]
		prxtract [**-V** | **-U**] [**-m** *mm*[*yyyy*]] [**-c** *wkstat*] [**-o** *output*]
caxtract	Report 03	**caxtract** [**-V** | **-U**] [**-o** *output*]
paxtract	Report 04A	**paxtract** [**-V** | **-U**] [**-o** *output*]
rextract	Report 04B	**rextract** [**-V** | **-U**] [**-o** *output*]
r11xtr	Report 11	**r11xtr** [**-V** | **-U**] [**-m** *mm*[*yyyy*]] [**-c** *wkstat*] [**-o** *output*] [**-s** *jstream_name*]
xrxtrct	Report 12	**xrxtrct** [**-V** | **-U**]

Appendix C. Accessibility

Accessibility features help users with physical disabilities, such as restricted mobility or limited vision, to use software products successfully. The major accessibility features in this product enable users to do the following:

- Use assistive technologies, such as screen-reader software and digital speech synthesizer, to hear what is displayed on the screen. Consult the product documentation of the assistive technology for details on using those technologies with this product.
- Operate specific or equivalent features using only the keyboard.
- Magnify what is displayed on the screen.

In addition, the product documentation was modified to include features to aid accessibility:

- All documentation is available in both HTML and convertible PDF formats to give the maximum opportunity for users to apply screen-reader software.
- All images in the documentation are provided with alternative text so that users with vision impairments can understand the contents of the images.

Navigating the interface using the keyboard

Standard shortcut and accelerator keys are used by the product and are documented by the operating system. Refer to the documentation provided by your operating system for more information.

The Event Rule Editor panel is the only one that does not allow keyboard-only operations and CSS cannot be disabled. However, as an alternative, you can perform all the operations available in this panel by launching the composer on page 434 command from the command line interface.

Magnifying what is displayed on the screen

You can enlarge information on the product windows using facilities provided by the operating systems on which the product is run. For example, in a Microsoft Windows environment, you can lower the resolution of the screen to enlarge the font sizes of the text on the screen. Refer to the documentation provided by your operating system for more information.

Notices

This document provides information about copyright, trademarks, terms and conditions for product documentation.

© Copyright IBM Corporation 1993, 2016 / © Copyright HCL Technologies Limited 2016, 2026

This information was developed for products and services offered in the US. This material might be available from HCL in other languages. However, you may be required to own a copy of the product or product version in that language in order to access it.

HCL may not offer the products, services, or features discussed in this document in other countries. Consult your local HCL representative for information on the products and services currently available in your area. Any reference to an HCL product, program, or service is not intended to state or imply that only that HCL product, program, or service may be used. Any functionally equivalent product, program, or service that does not infringe any HCL intellectual property right may be used instead. However, it is the user's responsibility to evaluate and verify the operation of any non-HCL product, program, or service.

HCL may have patents or pending patent applications covering subject matter described in this document. The furnishing of this document does not grant you any license to these patents. You can send license inquiries, in writing, to:

HCL
330 Potrero Ave.
Sunnyvale, CA 94085
USA
Attention: Office of the General Counsel

For license inquiries regarding double-byte character set (DBCS) information, contact the HCL Intellectual Property Department in your country or send inquiries, in writing, to:

HCL
330 Potrero Ave.
Sunnyvale, CA 94085
USA
Attention: Office of the General Counsel

HCL TECHNOLOGIES LTD. PROVIDES THIS PUBLICATION "AS IS" WITHOUT WARRANTY OF ANY KIND, EITHER EXPRESS OR IMPLIED, INCLUDING, BUT NOT LIMITED TO, THE IMPLIED WARRANTIES OF NON-INFRINGEMENT, MERCHANTABILITY OR FITNESS FOR A PARTICULAR PURPOSE. Some jurisdictions do not allow disclaimer of express or implied warranties in certain transactions, therefore, this statement may not apply to you.

This information could include technical inaccuracies or typographical errors. Changes are periodically made to the information herein; these changes will be incorporated in new editions of the publication. HCL may make improvements and/or changes in the product(s) and/or the program(s) described in this publication at any time without notice.

Any references in this information to non-HCL websites are provided for convenience only and do not in any manner serve as an endorsement of those websites. The materials at those websites are not part of the materials for this HCL product and use of those websites is at your own risk.

HCL may use or distribute any of the information you provide in any way it believes appropriate without incurring any obligation to you.

Licensees of this program who wish to have information about it for the purpose of enabling: (i) the exchange of information between independently created programs and other programs (including this one) and (ii) the mutual use of the information which has been exchanged, should contact:

HCL
330 Potrero Ave.
Sunnyvale, CA 94085
USA
Attention: Office of the General Counsel

Such information may be available, subject to appropriate terms and conditions, including in some cases, payment of a fee.

The licensed program described in this document and all licensed material available for it are provided by HCL under terms of the HCL Customer Agreement, HCL International Program License Agreement or any equivalent agreement between us.

The performance data discussed herein is presented as derived under specific operating conditions. Actual results may vary.

Information concerning non-HCL products was obtained from the suppliers of those products, their published announcements or other publicly available sources. HCL has not tested those products and cannot confirm the accuracy of performance, compatibility or any other claims related to non-HCL products. Questions on the capabilities of non-HCL products should be addressed to the suppliers of those products.

This information is for planning purposes only. The information herein is subject to change before the products described become available.

This information contains examples of data and reports used in daily business operations. To illustrate them as completely as possible, the examples include the names of individuals, companies, brands, and products. All of these names are fictitious and any similarity to actual people or business enterprises is entirely coincidental.

COPYRIGHT LICENSE:

This information contains sample application programs in source language, which illustrate programming techniques on various operating platforms. You may copy, modify, and distribute these sample programs in any form without payment to HCL, for the purposes of developing, using, marketing or distributing application programs conforming to the application programming interface for the operating platform for which the sample programs are written. These examples have not been thoroughly tested under all conditions. HCL, therefore, cannot guarantee or imply reliability, serviceability, or function of these programs. The sample programs are provided "AS IS", without warranty of any kind. HCL shall not be liable for any damages arising out of your use of the sample programs.

© (HCL Technologies Limited) (2026).
Portions of this code are derived from Sample Programs.
© Copyright 2016

Trademarks

HCL®, and other HCL graphics, logos, and service names including "hcltech.com" are trademarks of HCL. Except as specifically permitted herein, these Trademarks may not be used without the prior written permission from HCL. All other trademarks not owned by HCL that appear on this website are the property of their respective owners, who may or may not be affiliated with, connected to, or sponsored by HCL.

Adobe™, the Adobe™ logo, PostScript™, and the PostScript™ logo are either registered trademarks or trademarks of Adobe™ Systems Incorporated in the United States, and/or other countries.

IT Infrastructure Library™ is a Registered Trade Mark of AXELOS Limited.

Linear Tape-Open™, LTO™, the LTO™ Logo, Ultrium™, and the Ultrium™ logo are trademarks of HP, IBM® Corp. and Quantum in the U.S. and other countries.

Intel™, Intel™ logo, Intel Inside™, Intel Inside™ logo, Intel Centrino™, Intel Centrino™ logo, Celeron™, Intel Xeon™, Intel SpeedStep™, Itanium™, and Pentium™ are trademarks or registered trademarks of Intel™ Corporation or its subsidiaries in the United States and other countries.

Linux™ is a registered trademark of Linus Torvalds in the United States, other countries, or both.

Microsoft™, Windows™, Windows NT™, and the Windows™ logo are trademarks of Microsoft™ Corporation in the United States, other countries, or both.



Java™ and all Java-based trademarks and logos are trademarks or registered trademarks of Oracle and/or its affiliates.

Cell Broadband Engine™ is a trademark of Sony Computer Entertainment, Inc. in the United States, other countries, or both and is used under license therefrom.

ITIL™ is a Registered Trade Mark of AXELOS Limited.

UNIX™ is a registered trademark of The Open Group in the United States and other countries.

Terms and conditions for product documentation

Permissions for the use of these publications are granted subject to the following terms and conditions.

Applicability

These terms and conditions are in addition to any terms of use for the HCL website.

Personal use

You may reproduce these publications for your personal, noncommercial use provided that all proprietary notices are preserved. You may not distribute, display or make derivative work of these publications, or any portion thereof, without the express consent of HCL.

Commercial use

You may reproduce, distribute and display these publications solely within your enterprise provided that all proprietary notices are preserved. You may not make derivative works of these publications, or reproduce, distribute or display these publications or any portion thereof outside your enterprise, without the express consent of HCL.

Rights

Except as expressly granted in this permission, no other permissions, licenses or rights are granted, either express or implied, to the publications or any information, data, software or other intellectual property contained therein.

HCL reserves the right to withdraw the permissions granted herein whenever, in its discretion, the use of the publications is detrimental to its interest or, as determined by HCL, the above instructions are not being properly followed.

You may not download, export or re-export this information except in full compliance with all applicable laws and regulations, including all United States export laws and regulations.

HCL MAKES NO GUARANTEE ABOUT THE CONTENT OF THESE PUBLICATIONS. THE PUBLICATIONS ARE PROVIDED "AS-IS" AND WITHOUT WARRANTY OF ANY KIND, EITHER EXPRESSED OR IMPLIED, INCLUDING BUT NOT LIMITED TO IMPLIED WARRANTIES OF MERCHANTABILITY, NON-INFRINGEMENT, AND FITNESS FOR A PARTICULAR PURPOSE.

Index

Special Characters

.jobmanrc configuration script 86
$MANAGER keyword 212
$MASTER keyword 212

A

ABAP step definition
  attribute 1210
ABAP/4 modules
  SAP importing 1137
abend
  job state 567
  job stream state 579
abend prompt 31
abenp
  job state 567
access
  extended and network agents 212
  workstation definition 212
access control list definition
  security access control list 415
access method 1071
  agent
    syntax 1071
  dynamic agent
    option file 1074
  extended agent
    option file 1074
  HCL Workload Automation Agent
    option file 1074
  interface 1071
  PeopleSoft options 1099
  SAP 1128
  SAP common options 1154
  SAP global configuration options 1147
  SAP local configuration options 1150
  task options 1071
access method for dynamic agent
  overview 1070
access method for extended agent
  overview 1070
accessibility 1435
actions on security objects
  specifying actions on security objects 423
activating
  criteria profile 1222
  job interception 1232
ad-hoc prompt 31
add
  job state 567, 567
  job stream state 579
add command 454
adddep job command 586
adddep sched command 589
advanced rerun options 47
agent 31, 34
  access method
    syntax 1071
  defining Windows user 284
  starting 63
  stopping 63
  workstation definition 215
aggregate
  dependencies 1339
altjob command 592
altpass command 593
altpri command 595
anomaly detection
  metrics 891
APARs
  IY92806 1164, 1236, 1295
  IY97424 1230, 1231
  IZ03505 1148, 1148, 1148, 1148, 1148, 1152, 1152, 1152, 1152, 1152
  IZ12321 1124
  IZ26839 1297
  IZ33555 1161
  IZ37273 1187, 1207
  IZ42262 1190, 1203, 1209, 1214
API key authentication 437, 538, 548
application job plug-ins 233
  scheduling 34, 910
application server
  SAP R/3 1168
  stopping 704
  trace settings 1316
application servers
  multiple, PeopleSoft 1102
Appserverbox.msg 66
appservman
  stopping 696
architecture 57
archiving
  job instances 988
at command 918
  ATSCRIPT variable 919
at keyword 317, 325
auditbox.msg 67
authenticate command 457
authenticating with API key 437, 538, 548
authorization profile
  SAP 1132
  transaction PFCG 1134
  transaction su02 1133
autolink
  workstation definition 217
automating plan processing
  final job stream 150
automating processing
  production plan 150
autostart monman 706
AWSBEH002E error message
  The target host computer is not defined... 57, 441, 551
AWSBHU611W warning message
  Conman could not initialize the HTTP or HTTPS connection. 57, 441, 551

B

backup master domain manager 31
balancing SAP R/3 workload using server groups 1192
batch command 918
batch processing ID
  PeopleSoft 1103
batch reports
  logs 1063
  sample scenario 1057
  traces 1063
batchman process 59
BDC wait
  R/3 1222
behindfirewall
  workstation definition 217
bind
  definition 1352
bind process
  for distributed shadow job 1361
broker
  workstation definition 214
broker jobs promotion 162
broker promotion variables 162
building
  criteria hierarchy 1221
Business Information Warehouse support
  SAP 1233
Business Warehouse components
  InfoPackage 1234
  process chain 1234
  SAP 1234
Business Warehouse InfoPackage
  displaying details 1241
Business Warehouse InfoPackage and process chain
  managing 1235

C

calendar
  freedays 30, 30
  holidays 30, 30
  run cycle 30, 30, 304, 338, 366
calendar definition 286
cancel command 596
cancel sched command 598
carry forward
  remote job 1367
  shadow job 1367
carry forward settings
  conditional dependencies 115
carryforward
  customizing 115
  job stream keywords 115
  stageman 115
  variable 144
carryforward keyword 326
carryStates
  variable 115, 120
caxtract command 1037
CCMS
  sending data from job throttler 1255
CCMS event
  committing MTE alert 1292
  defining event rule, business scenario 1281, 1281
  getting CCMS alert status 1290
changing password, RFC user
  SAP R/3 1141
check file task
  extended agent
    syntax 1079
checkhealthstatus command 599
checking mailbox health 599
CHECKINTERVAL, PeopleSoft option 1099
chfolder command 457, 600
child job on i5/OS 1374
child job on IBM i 1374
child job settings on AS/400 for performances 1374
child job settings on i5/OS for performance 1374
child job settings on IBM i for performances 1374

child jobs settings for performances on AS/400 1374
class
  workstation 36
cleanuserjobs command 983
CLIConfig.properties file
  command-line configuration 979
closest preceding
  follows 344
  follows previous 101
  join 352
  matching criteria 101
collecting job interception 1225, 1226
combine
  dependencies 1339
command
  Cpuinfo 1081
  logman 146
  stageman 143
command line
  committing SAP event 1265, 1292
  composer 52
  conman 52
  defining extended agent job 1093
  defining extended agent workstation 1088
  getting CCMS alert status 1290
  monitoring SAP event 1171
  optman 52
command line interface
  setting 93
command line reporting
  setting up 1058
command-line authentication 437, 538, 548
command-line configuration
  CLIConfig.properties file 979
commands 762
  adddep job 586
  adddep sched 589
  altjob 592
  altpass 593
  altpri 595
  at 918
  batch 918
  cancel job 596
  cancel sched 598
  caxtract 1037
  checkhealthstatus 599
  cleanuserjobs 983
  confirm 602
  console 605
  continue (composer) 459
  continue (conman) 607
  cpuinfo 921
  da_test_connection 932
  dataexport 924
  dataimport 925
  datecalc 926
  deldep job 607
  deldep sched 609
  delete 933
  deployconf 611
  display 612
  evtdef 934
  evtsize 938
  exit 614
  exportserverdata 984
  fence 615
  getmon 645
  help 616
  importserverdata 986
  jbxtract 1034
  jobinfo 947
  jobprop 987
  jobstdl 949
  kill 618
  limit cpu 619, 619
  limit sched 620
  link 621
  listsucc 626
  listsym 623
  maestro 952
  makecal 953
  metronome 955, 974
  morestdl 955
  movehistorydata 988
  param 990
  parms 958
  paxtract 1038
  prxtract 1036
  r11xtr 1040
  recall 627
  redo 628
  release 961
  release job 630
  release sched 632
  rep1 1012
  rep11 1017
  rep2 1012
  rep3 1012
  rep4a 1012
  rep4b1 1012
  rep7 1014
  rep8 1015
  reply 634
  reptr 1018
  rerun 635
  rerunsucc 639
  resource 643, 994
  rextract 1039
  rmstdlist 965
  sendevent 967, 1005
  setsym 644
  showcpus 645
  showdomain 654
  showexec 969
  showfiles 656
  showjobs 659
  showprompts 683
  showresources 686
  showschedules 690
  shutdown 696
  start 697
  startappserver 699
  starteventprocessor 700
  startmon 700
  StartUp 971
  StartUpLwa 972
  status 701
  stop 702
  stop ;progressive 703
  stopappserver 704
  stopeventprocessor 705
  stopmon 706
  submit docommand 707
  submit file 711
  submit job 715
  submit sched 720
  switcheventprocessor 725
  switchmgr 726
  tellop 728
  unlink 729
  version 730
  version utility command 973
  xref 1020
  xrxtrct 1042
comment keyword 327
common options, SAP 1154
compiler
  messages 1320
composer
  command line 52
  messages 1320
composer program 434
  command line return codes 447
  connection parameters 437
  control characters 441
  defining extended agent job 1093
  defining extended agent workstation 1088
  delimiters 446
  editor 436
  filters 442
  list of commands 447
  offline output 435
  prompt 436
  running commands 437
  setup 435
    variables 1010
  special characters 446
  terminal output 435
  wildcards 442
  XML editor 436
composer reference 196
computer association
  retrieving 994
COMPUTERNAME variable 75
computers associated to a resource
  retrieving 994
condition-based automation 169
condition-based workload automation
  defining 172
conditional
  dependencies 42
  dependency 42
conditional automation 169
conditional dependencies 1333, 1336
  carry forward behavior 115
  defining 1336
  definition 1333, 1336
  join 1339
  plan replication 1350
conditional logic 1333, 1336
CONFIG FILE 757
configuration file
  example, R/3 1124
configuration options
  PeopleSoft 1099
  SAP, common 1154
  SAP, global 1148
  SAP, local 1150
  SAP, usage 1166
configuration scripts
  .jobmanrc 86
  djobmanrc.cmd 91
  jobmanrc 83
  jobmanrc.cmd 88
configuring
  job class inheritance for job throttling 1253
  job interception for job throttling 1252
  local properties 74
  parent-child for job throttling 1233
  SAP 1131
  SAP access method 1145
  SAP environment 1132

tracing utility 1120
confirm command 602
confirmed keyword 328
conman
command line 52
filters 553
wildcards 553
conman program 545
control characters 551
delimiters 554
filters 553
list of commands 582
offline output 547
processing 555
prompt 547
running commands 548, 552
selecting job 556
arguments 556
jobstream_ID 557
jobstreamname 557
schedid 557
using at 558
using confirm 560
using critical 560
using critnet 560
using deadline 560
using every 561
using finished 562
using follows 562
using logon 563
using needs 564
using opens 564
using priority 565
using prompt 565
using recovery 565
using scriptname 565
using started 566
using state 567
using until 570
selecting job streams 571
arguments 571
jobstream_id 572
jobstreamname 571
schedid 572
using at 573
using carriedforward 574
using carryforward 574
using finished 574
using follows 575
using limit 576
using needs 576
using opens 577
using priority 577
using prompt 578
using started 578
using state 579
using until 580
setup 546
set variables 546
special characters 554
terminal output 546
user prompting 552
wildcards 553
Conman program
Conman command line return codes 581
conman reference 545
conman startappserver
JnextPlan 125
connecting
SAP 1167
connecting to SAP 1174
connection parameters
setting 93
connection to SAP
troubleshooting 1296
considerations about return code mapping 1115
console command 605
Console Manager
messages 1319
context 762
continue command (composer) 459
continue command (conman) 607
control file
SAP 1136
controlling
job processing
edit job definition 49
conventions, typeface xvi
correction and transport files
SAP 1136
Courier.msg 67
cpuclass
workstation class definition 225
Cpuinfo
command 1081
cpuinfo command 921
cpuname
workstation definition 208
CPUREC statement
creating 1090
create
folder 493
create command 472
CreatePostReports
JnextPlan 126
creating
action parameters for CCMS event rules 1288
correlation rules for CCMS events 1288
CPUREC statement 1090
criteria profile 1220
DOMREC statement 1090
event rule based on CCMS alerts 1283
event rule based on CCMS alerts, business scenario 1281, 1281
event rule based on IDocs 1271
event rule based on IDocs, business scenario 1271
event rule based on SAP event 1266
internetwork dependency based on SAP event 1262, 1265
job containing InfoPackage 1235
job containing process chains 1235
PeopleSoft job 1107
RFC user, SAP 1132
SAP jobs 1172
creating forecast
planman command line 136
creating trial
planman command line 134
criteria hierarchy
building 1221
description 1218
Criteria Manager 1218
criteria profile
activating 1222
creating 1220
description 1218
critical dynamic job promotion 162
critical HCL Workload Automation jobs 247
critical job
prioritization 247
promotion 247
critical job priority
enhancing 247
critical jobs
global options 154
local options 156
security file 157
critical keyword 328
critical path
job promotion 247
cross
dependencies 42
dependency 42
cross dependencies
defining 1352
managing 1352
cross dependency
as dependency on a shadow job 1357
definition 1357
how to add to the plan 1359
information flow 1354, 1362
introduction 1352
logic 1352
monitoring resolution in plan 1359
production plan 1359
remote engine workstation 1352
remote job 1352
shadow job 1352
steps to define 1357
crucial dynamic jobs 162
custom events
defining 195, 934
sending 195, 967
sending from dynamic agents 1006
CUSTOM keyword, to filter SAP events 1270
customization procedure
SAP 1131
customizing
properties file 1121, 1121, 1122
customizing the workload
using variable table 164
CyberArk integration 261
job definition example 263
CyberArk query format 262

D

d-pool
workstation 216
da_test_connection command 932
daily run cycle 22
data file
SAP 1136
data integrity
variable table 165
database
replicating plan 50
database objects 521
access method jobs 256
add command 454
authenticate command 457
calendars 286
continue command 459
create command 472
delete command 460
display command 465
displaying composer banner 521
domains 227
edit command 471
event rules 395
executable jobs 255

exit command 471
extract command 472, 472
folders 287
help command 478
IBM i jobs 250
job stream 314
jobs 228
list command 479
lock command 488
modify command 495
new command 501
print command 479
prompts 31, 298
redo command 504
rename command 506
replace command 511
resources 300
run cycle group 301
Security access control list definition 414
Security domain definition 416
Security role definition 419
twstrace command 1007
unlock command 513
update command 517
users 281
validate command 520
variable table 295
variables 289
wappman command 538
workstation classes 224
workstations 201
database performance
improving 988
dataexport command 924
datagather tool 974
replaced by wa_pull_info 974
dataimport command 925
date
run cycle 304, 337, 365
datecalc command 926
day
run cycle 304, 338, 365
DB tables maintenance
movehistorydata command 988
deadline keyword 329
default variable table
using 164
defining
ABAP step attribute 1210
action parameters for CCMS event rules 1288
condition-based workload automation 172
conditional dependencies 1336
correlation rules for CCMS events 1288
database objects
access method jobs 256
AS/400 jobs 250
calendars 286
domains 227
event rules 395
executable jobs 255
folders 287
i5/OS jobs 250
IBM i jobs 250
job stream 314
jobs 228
prompts 31, 298
resources 300
run cycle group 301
Security access control list 414
Security domain 416
Security role 419
variables 289
windows users 281
workstation classes 224
workstations 201
defining security objects in the database 414
dependencies
follows 344
join 352
needs 362
opens 374
prompts 378
event rule based on CCMS alerts 1283
event rule based on CCMS alerts, business scenario 1281, 1281
event rule based on IDocs 1271
event rule based on IDocs, business scenario 1271
event rule based on SAP event 1266
external command step attribute 1214
external program step attribute 1214
global options file 1081
local options file 1081
PeopleSoft job 1107
remote command jobs 910
SAP event as internetwork dependency 1262, 1265
SAP job 1180, 1237
SAP job dynamically 1199
SAP jobs 1172
SAP variant 1175
security objects in the database 414
supported agents job 1092
supported agents workstation 1086
defining a dependency 42
Defining agents on AS/400 systems 1369
Defining agents on i5/OS systems 1369
Defining agents on IBM i systems 1369, 1369
defining dependencies 42
Defining IBMi jobs 1369
defining job
SAP 1180
SAP, dynamically 1199
defining job streams 314
Defining jobs on AS/400 1369
Defining jobs on i5/OS 1369
Defining jobs on IBM i 1369
defining non-operational jobs in job streams 364
Defining objects
in the database 196
definition
variable table 295
deldep job command 607
deldep sched command 609
delete
folders 505
delete command 460, 933
deleting
ITWS for Apps 1257
SAP job 1192
SAP variant 1175
dependencies 42, 1333, 1336
conditional 1333, 1336
join, combine 1339
orphaned 106
dependency 42
based on SAP event, defining 1262, 1265
based on SAP event, limitation with XBP 2.0 1261
committing SAP event by external task 1265
cross 1362
internetwork 1322, 1327
mapping between definition and resolution, SAP 1264
deployconf command 611
Deploying rules
planman command line 138
description keyword 330
directory names, notation xvii
display command 465, 612
displaying details
Business Warehouse InfoPackage 1241
Business Warehouse InfoPackage job 1241
process chain job 1241
SAP job 1190
djobmanrc configuration script 91
docommand
job definition 233
domain 37
workstation definition 211
domain definition 227
ismaster 228
manager 227
parent 228
domain manager 31
DOMREC statement
creating 1090
done
job state 567
draft keyword 331
dynamic agent
access method
option file 1074
gateway 34
overview 1070
workstation definition 215
dynamic agents 910
dynamic capabilities 910
dynamic database jobs 228
dynamic file transfer jobs 228
dynamic java jobs 228
dynamic job creation 228
dynamic job definition parameter description
SAP 1201
dynamic job definition syntax
ABAP step definition attribute 1210
external command step definition attribute 1214
external program step definition attribute 1214
SAP 1200
dynamic job promotion 162
dynamic jobs 201, 228, 233
access method jobs 256
AS/400 jobs 250
executable jobs 255
exporting properties 270
i5/OS jobs 250
IBM i jobs 250
remote command job 910
dynamic pool 31, 35
defining 201
defining Windows user 284
workstation 216
dynamic pools
scheduling job types with advanced options 910
dynamic scheduling 31, 34, 34, 35, 910
job definition 228

job types with advanced options 910
task job definition 233
workstation definition 201
dynamic web service jobs 228
dynamic workload broker instance URI 984, 986
Dynamic Workload Console
defining supported agents job 1093
defining workstation for agent with access method 1086
dynamic workstations 31, 201
dynamically
defining SAP jobs 1200
dynamically, defining SAP job 1199

E

edit command 471
enabling
SSL communication 218
time zone 1064
enCarryForward
variable 115, 120
enCFInterNetworkDeps
variable 121
enCFResourceQuantity
variable 121
encrypting passwords 1102, 1141, 1166
encrypting user password
PeopleSoft 1102
SAP 1166
end keyword 331
end-to-end scheduling
defining extended agent job 1094
defining supported agent workstation 1089
enigma program
encrypting user password 1166
enLegacyStartOfDayEvaluation
variable 124, 1065
enPreventStart
variable 122
enTimeZone
variable 124, 1064
environment variables
job promotion 162
environment variables, notation xvii
error
job state 568
event rule 39
action parameters for CCMS event rules 1288
based on CCMS alerts 1282
based on CCMS alerts, business scenario 1281, 1281
based on IDocs 1271
based on IDocs, business scenario 1271
based on IDocs, matching criteria 1272
correlation rules for CCMS alerts 1288
definition 1266
monitoring SAP event 1170
SAP, defining 1266
SAP, filtering events 1270
SAP, prerequisite to define 1169
event rule definition 394
eventRules.xsd 395
keywords
actionProvider 405
actionType 406
activeTime 399
correlationAttributes 405
daylight saving time 399
description 398, 408
eventCondition 400
eventProvider 400
eventRule 397
eventType 401
filteringPredicate 404
isDraft 398
name 397
onDetection 407
onTimeOut 407
operator 404
responseType 407
ruleType 397
scope 404, 408
timeInterval 399
timeZone 398
validity 399
event rules
instances 194
sample scenarios 175, 185
timeout option 183
variable substitution 184
events, logging 1220
events, raising SAP 1195
every keyword 332
used in job definitions 333
used in job stream definitions 332
evtdef command 934
evtsize command 938
example
dynamic SAP job definition 1216
return code mapping 1114
except keyword 337
exclusive run cycle 22
exec
job state 568
job stream state 580, 580
existing job types
definition 20
exit command 471, 614
exporting SAP calendars
business scenario 1257
exporting SAP R/3 calendars
r3batch export function 1258
exportserverdata command 984
extended agent
access method
option file 1074
response messages 1073
running 1077
troubleshooting 1319
check file task
syntax 1079
get status task
syntax 1079
overview 1070
workstation definition 213
extended agent job
SAP 1171
extended agent job defining with
command line 1093
extended agent workstation, defining with
command line 1088
ISPF 1091
extended and dynamic agent workstation defining with
end-to-end scheduling 1089
EXTERNAL
job stream 1327
jobs 1328
external command step definition
attribute 1214
external program step definition
attribute 1214
extract command 472
extrn
job state 568

F

fail
job state 568
fault-tolerant agent 31
fdignore
except 342
on 308, 369
fdnext
except 342
on 308, 369
fdprev
except 342
on 308, 369
feature
job interception, setting SAP 1227, 1227
features
job interception and parent-child, SAP 1223
job interception, activating SAP 1232
job interception, collecting SAP 1225, 1226
job interception, implementing SAP 1223
job interception, SAP 1223
job interception, setting SAP 1227
job throttling, SAP R/3 1250
parent-child R/3 1233
PeopleSoft 1096
return code mapping 1112
SAP R/3 1126
fence
job state 568
fence command 615
file 1121
configuration for R/3 1124
dependencies 42
dependency 42
psagent.properties 1121
r3batch.properties 1121
return code mapping 1113
file name
return code mapping 1118
files
Appserverbox.msg 66
at.allow 920
at.deny 920
auditbox.msg 66
Courier.msg 67
Intercom.msg 67
Mailbox.msg 67
mirrorbox.msg 67
Monbox.msg 67
Moncmd.msg 68
NetReq.msg 68
PlanBox.msg 68
Server.msg 68
filtering SAP events in security file 1270
filters
composer 442
final job stream
automating plan processing 150
FNCJSI
preproduction plan 99
Folder
Folder definition syntax 1407
folder definition 287
folder keyword 343
folders

chfolder command 457, 600
create 493
delete 505
list command 488, 623
mkfolder command 493
remove command 505
renamefolder command 510
follows 42
matching criteria 344
follows absolute to
matching criteria 103
within an absolute interval 103
follows keyword 344
follows previous
closest preceding 101
matching criteria 101
follows relative to
matching criteria 102
within a relative interval 102
follows sameday
matching criteria 100
same day 100
forecast plan
creating 136
description 118
earliest start time calculation 118
loop detection 151
forecast start time enablement 118
freedays keyword 348
freedays run cycle 22
fta
workstation definition 213
fullstatus
workstation definition 218

G

get status task
extended agent
syntax 1079
global options
carryforward variable 144
carryStates 115
carryStates variable 120
enCarryForward variable 115, 120
enCFInterNetworkDeps variable 121
enCFResourceQuantity variable 121
enLegacyStartOfDayEvaluation variable 124, 1065
enPreventStart variable 122
enTimeZone variable 124, 1064
logmanMinMaxPolicy variable 123
logmanSmoothPolicy variable 123
maxLen variable 119
minLen variable 119
startOfDay variable 119, 1065
untilDays 120
global options file
defining 1081
mvsjes.opts 1082, 1084
mvsopc.opts 1082, 1084
name 1075, 1082, 1084
psagent.opts 1082, 1084
r3batch.opts 1082, 1084
global options, SAP 1147
global parameter
definition 289
variable table 164
global prompt 31

H

HCL Workload Automation
architecture 57
basic concepts 19
controlling job processing 42
critical jobs 247
critical path 247
defining activities 42
issuing commands on Windows 57
managing production 50
network 40
object 19
overview 19
processes 57
quick start 54
running event management 51
runtime environment 41
user interfaces 52, 52
HCL Workload Automation Agent
access method
option file 1074
HCL Workload Automation jobs
prioritizing 247
help command 478, 616
historical data
alerts 891
metrics 891
hold
job state 569
job stream state 580
hold status 630
HOME variable 75, 79, 89
HOMEDRIVE variable 75
HOMEPATH variable 75
host
extended agents 211
workstation definition 211

I

icalendar
run cycle 305, 339, 367
identifying job stream instances
in the plan 99
at 99
scheddateandtime 99
preproduction plan 99
IDoc
defining event rule 1271
defining event rule, business scenario 1271
defining event rule, matching criteria 1272
ignore
workstation class definition 226
implementing
job interception 1223
important dynamic jobs 162
importserverdata command 986
improving database performance 988
inclusive run cycle 22
InfoPackage
managing 1235
user authorization 1234
InfoPackage schedule options
SAP 1235
inheritance r3batch
definition 1086
installing
ABAP modules, SAP 1132
instana 905
integration type
name property 270
Integration with IBM IBM Z Monitoring
BmEvents.conf
FILE parameter 899
integrity of data
variable table 165
interactive
job definition 235
intercepted job
return code mapping 1120
interception criteria
setting, SAP feature 1227
Intercom.msg 67
interface 1071
intermediate plan
extending with 132
generating 130
internetwork
dependencies 42
dependency 42
internetwork dependency
based on SAP event, defining 1262, 1265
based on SAP event, limitation with XBP 2.0 1261
based on SAP event, limitation with XBP 3.0 1262
based on SAP R/3 event, prerequisites 1261
committing SAP event by external task 1265
creating 1326
managing using conman 1327
mapping between definition and resolution, SAP 1264
placeholder SAP job 1261
intro
job state 569
introduction
HCL Workload Automation for SAP 1125
PeopleSoft 1096
SAP 1128
ismaster
domain definition 228
ISPF
defining extended agent workstation 1091
isservice keyword 351
ITWS_PSXA project
PeopleSoft 1103

J

Java API 53
jbxtract command 1034
JnextPlan
conman startappserver 125
CreatePostReports 126
MakePlan 125
SwitchPlan 126
UpdateStats 126
job 20
assigning a server group 1193
controlling process
edit job definition 49
PeopleSoft 1107
SAP job state 1193
submitting for supported agent 1096
job creation 228
job definition 228
access method jobs 256
docommand 233
executable jobs 255
IBM i jobs 250
interactive 235
PeopleSoft 1107
recovery option 238
remote command jobs 910
SAP 1180

SAP, dynamically 1199
scriptname 232
streamlogon 233
task 233
tasktype 234
using variables and parameters 243
job definition parameter description
SAP dynamic 1201
job environment on AS/400 1378
job environment on i5/OS 1378
job environment on IBM i 1378
job HCL Workload Automation
creating job containing InfoPackage 1235
creating job containing process chain 1235
job in job stream
onlate 372
job instances
archiving 988
job interception
activating, SAP feature 1232
collecting, SAP feature 1225, 1226
enabling and configuring for job
throttling 1252
implementing, SAP feature 1223
SAP feature 1223
setting placeholders in template file 1232
setting, SAP feature 1227, 1227
job loop detection 151
job processing
configuring 91
job promotion 154, 247
environment variables 162
job promotion on dynamic pools 162
job properties
cross-job metadata 270
passing to successor job 270
job state
SAP 1193
job statement
in job streams 349
job states
abend 567
abenp 567
add 567, 567
done 567
error 568
exec 568
extrn 568
fail 568
fence 568
hold 568
intro 569
pend 569
ready 569
sched 569
succ 569
succp 569
wait 570
job status
status property 270
job status mapping
PeopleSoft 1110
job stream 20, 314
EXTERNAL 1327
job stream definition 314, 314
job stream keywords
at 325
carryforward 115, 326
comment 327
confirmed 328
critical 328
deadline 329
description 330
draft 331
end 331
every 332
used in job definitions 333
used in job stream definitions 332
except 337
folder 343
follows 344
freedays 348
isservice 351
job statement 349
join 352
jsuntil 354
keyjob 356
keysched 357
limit 357
matching 358
maxdur 360
mindur 361
needs 362
nop 364
on 364, 372
onoverlap 373
opens 374
priority 377
prompt 378
schedtime 379
schedule 381
servicedescription 388
servicename 387
servicetags 388
servicevariables 389
startcond 383
statisticstype custom 382
timezone 389
until 390
validfrom 393
vartable 394
job stream loop detection 151
job stream states
abend 579
add 579
exec 579, 580
hold 580
ready 580
stuck 580
succ 580
job streams 314
move 522
rename 522
job streams keywords
at 317
jsuntil 354
onmaxdur 360
onmindur 361
onuntil 310, 311, 355, 390, 390
startcond 383
job throttling
business scenario 1250
configuring logging properties 1253
deleting ITWS for Apps 1256
enabling and configuring job
interception 1252
enabling job class inheritance 1253
options in options file 1251
SAP R/3 feature 1250
sending data to CCMS 1255
starting 1254
stopping 1255
throttling_send_ccms_data 1255
throttling_send_ccms_rate 1255
job tracking
PeopleSoft 1098
job types with advanced options 910
definition 20
dynamic scheduling 20
scheduling 34, 233
scheduling dynamically 910
scheduling statically 910
static scheduling 20
jobinfo command 947
jobman
environment variables 75
Jobman
messages 1320
jobman process 60
limit cpu 60
jobmanrc configuration script 83, 88
jobprop command 987
jobs
move 522
new SAP 1172
rename 522
jobs dynamic definition
SAP 1200
jobstdl command 949, 955
jobthrottling.bat, usage parameter 1254
jobthrottling.sh, usage parameter 1254
join
dependencies 1339
matching criteria 352
join keyword 352
jsuntil keyword 354, 354

K

keyjob keyword 356
keysched keyword 357
keywords
at 317, 325
carryforward 326
comment 327
confirmed 328
critical 328
deadline 329
description 330
draft 331
end 331
every 332
used in job definitions 333
used in job stream definitions 332
except 337
folder 343
follows 344
freedays 348
isservice 351
join 352
jsuntil 354
keyjob 356
keysched 357
limit 357
matching 358
maxdur 360
mindur 361
needs 362
nop job in job stream 364
on 364
onlate 372
onoverlap 373
opens 374
priority 377

prompt 378
schedtime 379
schedule 381
servicedescription 388
servicename 387
servicetags 388
servicevariables 389
startcond 383
statisticstype custom 382
timezone 389
until 390
validfrom 393
vartable 394
kill command 618
killing SAP job 1195

L

LANG variable 75, 79
Language setting 757
late status 329
LD_LIBRARY_PATH variable 79
LD_RUN_PATH variable 79
licensetype
licensing 208
licensing
licensetype 208
licensing model
actual workstation 676
pricing model 676
limit cpu
jobman process 60
limit cpu command 619
limit keyword 357
limit sched command 620
link command 621
list command 479
listfolder command 488, 623
listsucc command 626
listsym command 623
LJUSER, PeopleSoft option 1099
local option
mm retry link variable 149
local options file
defining 1081
name 1075, 1082, 1084
local options, SAP R/3 1150
local parameter
database 958
definition 289
exporting 958
importing 958
managing 958
local password
managing on dynamic agents 990
local prompt 31
local properties 74
local variable
managing on dynamic agents 990
LOCAL_RC_OK variable 84, 89
lock command 488
lock mechanism
variable table 165
log level on AS/400 1371, 1373
log level on i5/OS 1371, 1373
log level on IBM i 1371, 1373
log settings on AS/400 1371, 1373
log settings on i5/OS 1371, 1373
log settings on IBM i 1371, 1373
logging job statistics 146
logging raised events 1220
logical resource 39
logical resource association
retrieving 994
logical resource information
retrieving 994
logmanMinMaxPolicy
variable 123
logmanSmoothPolicy
variable 123
LOGNAME variable 75, 79
logon group
SAP R/3 1169
logs observability
OpenTelemetry 903
long term plan
preproduction plan 98
loop detection
forecast plan 151
trial plan 151

M

maestro command 952
MAESTRO_OUTPUT_STYLE variable 75, 80
MAIL_ON_ABEND variable 84, 85, 89
on a Windows workstation 91
mailbox files
Appserverbox.msg 66
auditbox.msg 66
Courier.msg 67
Intercom.msg 67
Mailbox.msg 67
mirrorbox.msg 67
Monbox.msg 67
Moncmd.msg 68
NetReq.msg 68
PlanBox.msg 68
Server.msg 68
setting size 939
Mailbox.msg 67
mailman process 59
ServerID 59
makecal command 953
MakePlan
JnextPlan 125
manager
workstation definition 214
managing
Business Warehouse InfoPackage and process chain 1235
external follows dependencies 100
matching criteria 100
objects in the database 196
production cycle 96
SAP extended agent job running 1171
shadow job in the plan 1368
workload applications 412
Managing agents on AS/400 systems 1370
Managing agents on i5/OS systems 1370
Managing agents on IBM i systems 1370, 1370
managing events
starting the event processing server 700
starting the monitoring engine 700
stopping the event processing server 705
stopping the monitoring engine 706
switching the event processing server 725
managing jobs and agents on AS/400 1369
managing jobs and agents on i5/OS 1369
managing jobs and agents on IBM i 1369
managing jobs and agents on IBM i dynamic environment 1369
managing objects
command line 434
in plan 545
managing plan
adding dependency to job streams 589
adding dependency to jobs 586
altering priority 595
altering user password 593
assigning console 605
cancelling job streams 598
cancelling jobs 596
confirming job completion 602
deleting dependency to job streams 609
deleting dependency to jobs 607
displaying conman banner 730
displaying help information 616
displaying jobs or job streams 612
displaying production plan status 701
displaying workstation information 645
exiting conman 614
get active monitors 645
ignoring command 607
limiting jobs running in job stream 620
linking workstations 621
listing job successors 626
listing processed plans 623
listing unresolved prompts 627
modifying job fence 615
modifying jobs 592
modifying jobs running on workstation 619
modifying resource units 643
releasing job streams from dependency 632
releasing jobs from dependency 630
replying to prompts 634
rerunning commands 628
rerunning job successors 639
rerunning jobs 635
selecting processed plan 644
sending messages to operator 728
setting message level 605
showing domain information 654
showing file dependencies 656
showing job information 659
showing job streams information 690
showing prompts information 683
showing resource information 686
shutting down workstation processes 696
starting the application server 699
starting workstation processes 697
stopping behindfirewall workstation processes 703
stopping jobs 618
stopping the application server 704
stopping workstation processes 702
submitting commands as jobs 707
submitting file as jobs 711
submitting job streams 720
submitting jobs 715
switching domain management 726
unlinking workstations 729
updating the monitoring configuration file 611
managing time zone 1064
time zone name
with variable length 1064
managing workload applications 527
mapping
HCL Workload Automation and SAP job states 1193
mapping file, regular expressions 535
mapping file, workload applications 529

mapping job status
PeopleSoft 1110
master domain manager 31
matching criteria
closest preceding 101, 358
follows 344
follows absolute to 103
follows previous 101
follows relative to 102
follows sameday 100
join 352
pending predecessor 105
predecessor 104
same day 100, 358
successor 104
within a relative interval 102, 358
within an absolute interval 103, 358
matching keyword 358
max_jobs_to_release_for_user 1148
maxdur keyword 360
maxLen
variable 119
mechanism of lock
variable table 165
members
workstation class definition 226, 417, 419
workstation definition 221
messages
compiler 1320
composer 1320
Console Manager 1319
Jobman 1320
metrics
AI Data Advisor 891
AIDA 891
authorizing access 891
exposing 891
Grafana 891
list 891
OpenMetrics standard 891
Prometheus 891
visualizing 891
metrics observability
OpenTelemetry 903
metronome command 955, 974
mindur keyword 361
minLen
variable 119
mirrorbox.msg 67
mirroring 50
mkfolder command 493
model 762, 775, 776, 779, 785, 785, 788, 793, 796, 796, 799, 800, 802, 804, 806, 881, 881
modify command 495
Monbox.msg 67
Moncmd.msg 68
monitoring
MTEs 1282
SAP event defined as event rule 1170
SAP event defined as internetwork dependency, XBP 2.0 1262
monman process 58
move
job streams 521
jobs 521
resources 524
variable table 523
workstation 523
movehistorydata command 988
MTE alert
action parameters for CCMS event rules 1288
committing by external task 1292
correlation rules 1288
defining event rule 1283
defining event rule, business scenario 1281, 1281
getting CCMS alert status 1290
mapping
attribute MTE name and HCL Workload Automation fields 1287
context MTE name and HCL Workload Automation fields 1286
object MTE name and HCL Workload Automation fields 1287
multiple application servers, PeopleSoft 1102
mvsjes.opts file
definition 1082, 1084
mvsopc.opts file
definition 1082, 1084

N

name
global options file 1075, 1082, 1084
local options file 1075, 1082, 1084
name property 270
named prompt 31
National Language support
SAP 1294
needs keyword 362
netman process 58
netmth access method 1324
NetReq.msg 68
network agent
access method
options file 1324
access method netmth 1324
definition 1324
EXTERNAL 1327
EXTERNAL state
ERROR 1328, 1328
EXTRN 1328
internetwork dependency 1322
creating 1326
managing using conman 1327
overview 1322
reference 1322
sample scenario 1325
network communication 70
job processing 70
start of day 70
new command 501
new copy
re-running jobs SAP 1199
new executors 233
access method jobs 256
AS/400 jobs 250
executable jobs 255
i5/OS jobs 250
IBM i jobs 250
remote command job 910
scheduling 34, 910
new plug-ins
access method jobs 256
AS/400 jobs 250
executable jobs 255
i5/OS jobs 250
IBM i jobs 250
remote command job 910
node loop detection 151
nop keyword 364
notation
environment variables xvii
path names xvii
typeface xvii

O

object attribute values
specifying object attribute values 429
object attributes
attributes for object types 428
observability 905
ocli 762
ocli command 775, 776, 779, 785, 785, 788, 793, 796, 796, 799, 800, 802, 804, 806, 881, 881
offset-based run cycle 22
old copy
re-running jobs SAP 1199
on
run cycle 364
on keyword 364
run cycle 364
onlate
job in job stream 372
onlate keyword 372
run cycle 372
onmaxdur keyword 360
onmindur keyword 361
onoverlap keyword 373
onuntil keyword 310, 311, 355, 390, 390
opens keyword 374
opentelemetry 905
OpenTelemetry
bootstrap.properties 903
configuration 903
enabling after upgrade 903
logs observability 903
metrics observability 903
traces observability 903
operator password
encrypting on PeopleSoft 1102
option file
dynamic agent
access method 1074
extended agent
access method 1074
HCL Workload Automation Agent
access method 1074
option inheritance r3batch
definition 1086
options
R/3 National Language support 1295
untilDays 120
options file 1098
global 1081
local 1081
PeopleSoft 1098, 1101
SAP 1128, 1145
SAP example 1166
setting job throttling options, SAP R/3 1251, 1253
optman
command line 52
OQL 732
Orchestration CLI 762, 775, 776, 779, 785, 785, 788, 793, 796, 796, 799, 800, 802, 804, 806, 881, 881
Orchestration CLI program
Orchestration CLI command line return codes 884
orphaned

dependencies 106
os type
workstation definition 209
overview
access method for dynamic agent 1070
access method for extended agent 1070
dynamic agent 1070
extended agent 1070
HCL Workload Automation for SAP 1125
PeopleSoft 1096
SAP 1128

P

param command 990
parameter 39
enable trace utility 1123
max trace files 1123
properties file 1121, 1122
return code mapping 1113
SAP dynamic job definition 1201
SAP job definition 1180
trace file path 1122
trace file size 1124
trace level 1121
parameter definition 289
parameters
in job definitions 243
parent
domain definition 228
parent-child feature
R/3 1233
parms command 958
password
defining on dynamic agent 258
job types with advanced options 258
resolving on dynamic agent 258
password management 261
job definition example 263
password resolver 261
job definition example 263
password retrieval
using queries 262
password solver 261
job definition example 263
password storage 261
job definition example 263
password vault 261
job definition example 263
path names, notation xvii
PATH variable 80
paxtract command 1038
pend
job state 569
pending predecessor
matching criteria 105
orphaned dependencies 106
successor 105
PeopleSoft
access method options 1099
batch processing ID 1103
configuration tasks 1098
connecting to multiple application servers 1102
creating job 1107
defining job 1107
encrypting operator password 1102
functional overview 1097
introduction 1096
ITWS_PSXA project 1103
job definition 1107
job status mapping 1110
job tracking 1098
options file 1098, 1101
overview 1097
parameters to define job 1108
return code mapping 1115
roles and responsibilities 1097
security 1098
task string parameters 1108
PeopleSoft operator password
changing 1102
encrypting 1102
physical resource 39
placeholder
for job interception in template file 1232
SAP job 1262
plan 762
quick start 54
plan data 50
plan management
basic concepts 96
customizing 115, 119
logman 146
stageman 143
plan replication
conditional dependencies 1350
PlanBox.msg 68
planman command line
connection parameters 129
creating forecast 136
creating trial 134
Deploying rules 138
intermediate plan 130, 132
monitor replication 142
removing plan 140
replicating plan data 140
resetting plan 139
retrieving plan info 133
trial extension 135
unlocking plan 139
planman deploy 183
plug-ins 53
pool 31, 34
defining 201
defining Windows user 284
workstation 216
pools
scheduling job types with advanced options 910
POSIXHOME variable 89
predecessor
matching criteria 104
successor 98, 105
preproduction plan
description 98
FNCJSI 98
long term plan 98
removing plan 140
print command 479
priority keyword 377
problem prevention
metrics 891
process chain
creating HCL Workload Automation job 1235
managing 1235
rerunning job 1247
restarting 1247
schedule options 1235
user authorization 1234
process chain job
displaying details 1241
rerunning 1243
processes
batchman 59
jobman 60
mailman 59
monman 58
netman 58
ssmagent 58
writer 59
production cycle 96
identifying job stream instances 99
managing 96
planman command line 129
production plan
automating processing 150
description 114
generating 125, 127
JnextPlan 96, 125, 127
monitor replication 142
replicating data 140
resetting plan 139
retrieving info 133
starting processing 149
Symphony file 96, 125, 127
unlocking plan 139
program
composer 1088, 1093
project
ITWS_PSXA PeopleSoft 1103
promoting a job 154
prompt
abend 31
ad-hoc 31
dependencies 42
dependency 42
global 31
local 31
named 31
recovery 31
prompt definition 31, 298
prompt keyword 378
prompts
unique ID 684
properties file
customizing 1121
DEBUG_MAX 1122
DEBUG_MID 1122
DEBUG_MIN 1121
parameter 1121, 1122
trace file path 1122
trace level 1121
protocol
workstation definition 220
Proxy user 757
prxtract command 1036
PS_DISTSTATUS, PeopleSoft option 1100
psagent 1098, 1102
psagent.opts file
definition 1082, 1084
psagent.properties
file 1121
PSFT_DOMAIN_PWD,PeopleSoft option 1100
PSFT_OPERATOR_ID,PeopleSoft option 1100
PSFT_OPERATOR_PWD, PeopleSoft option 1100
PSJOAPATH, PeopleSoft option 1100
pwdcrypt program
encrypting user password 1102

Q

queries for password retrieval 262

Querying with the OQL syntax 732

R

R/3
BDC wait 1222
configuration file 1124
parent-child feature 1233
return code mapping 1116
Unicode support 1143
r11xtract command 1040
r3batch
export function 1258
option inheritance 1086
r3batch.opts
definition 1082, 1084
options file 1145
SAP 1145
r3batch.properties 1121
r3evman command 1171
r3evmon event configuration file 1300
raised events 1220
raising SAP event 1195
re-running job
new copy, SAP 1199
old copy, SAP 1199
ready
job state 569
job stream state 580
recall command 627, 627
recovery
job definition 238
recovery options
continue 47
recovery jobs 47
rerun 47
stop 47
recovery prompt 31
redo command 504, 628
referential integrity check 449
refreshing
SAP variant 1175
release command 961
release job command 630, 634
release sched command 632
release_all_intercepted_jobs_for_request 1148
rem-eng
workstation 215
remote engine
cross dependency 1362
how it is bound 1361
workstation 201, 215
remote engine workstation 31, 36, 40
defining 1352, 1357
Remote Function Call
RFC 1141
remote job
carry forward 1367
defining 1352
failed 1367
status transition during recovery 1367
removing plan
planman command line 140
rename
job streams 522
jobs 521
resources 524
variable table 523
workstation 523
rename command 506
renamefolder command 510
rep1 command 1012
rep11 command 1017
rep2 command 1012
rep3 command 1012
rep4a command 1012
rep4b command 1012
rep7 command 1014
rep8 command 1015
replace command 511
replicate plan 50
report commands 1010
Actual Production Detail
sample output 1028
Actual Production Details 1018
Actual Production Summary 1018
Calendar Listing 1012
sample output 1024
changing date format 1011
Cross Reference 1020
sample output 1031
extract programs 1033
caxtract 1037
jbxtract 1034
paxtract 1038
prxtract 1036
r11xtr 1040
rextract 1039
xrxtract 1042
Job Details Listing 1012
sample output 1021
Job Histogram 1015
sample output 1027
Job History Listing
sample output 1026
Parameter Listing
sample output 1025
Parameters Listing 1012
Planned Production Detail
sample output 1027
Planned Production Details 1018
Planned Production Schedule 1017
sample output 1029
Planned Production Summary 1018
Prompt Listing 1012
sample output 1024
Resource Listing 1012
sample output 1025
sample outputs 1021
setup 1010
reports commands
Job History Listing 1014
list of commands 1011
reptr command 1018
requirements
workstation definition 221
rerun command 635
rerun counter
reruncount property 270
rerun with successors 47
reruncount property 270
rerunning job
process chain 1243, 1247
SAP 1197, 1243
rerunsucc command 639
reserved keywords
for job streams 198
for user definitions 200
for workstations 199
reserved words
for job streams 198
for user definitions 200
for workstations 199
resetFTA command 642
resetting plan
planman command line 139
resolution
variable 166
resource
dependencies 42
dependency 42
logical 39
physical 39
scheduling 39
resource command 643, 994
running from agent
CLIConfig.properties setup 1004
requirement 1004
resource definition 300
resources
unique ID 687
restarting
process chain 1247
return code mapping 1115
considerations 1115
example 1114
feature 1112
file name 1118
file, creating 1113
intercepted job 1120
parameter 1113
PeopleSoft 1115
R/3 1116
syntax 1114
return code on AS/400 1378
return code on i5/OS 1378
return code on IBM i 1378
Reusing a workload in another environment 528, 528
rextract command 1039
RFC
Remote Function Call 1141
RFC profile
SAP 1132
RFC user
SAP 1297
RFC user ID password
changing 1141
encrypting 1141
RFC user password
SAP R/3 1141
rial plan
loop detection 151
rmfolder command 505
rmstdlist command 965
roles and tasks
PeopleSoft 1097
SAP 1129
rule 39
rule-based run cycle 22
run cycle 25
calendar 304, 338, 366
daily 22
date 304, 337, 365
day 304, 338, 365
exclusive 22, 30, 30
freedays 22
icalendar 305, 339, 366
inclusive 22, 30, 30
offset-based 22
on 364
rule-based 22
run cycle group 308, 341, 369
simple 22

weekly 22
yearly 22
run cycle group 25, 301
file definition syntax 1411
run cycle 308, 342, 369
run cycle group definition 301
run cycle groups 25
RUNLOCATION, PeopleSoft option 1100
Running commands 762
running system commands
from composer 512
from conman 552, 727

S

same day
follows 344
follows sameday 100
join 352
matching criteria 100
SAP
ABAP/4 modules, importing 1137
access method 1128
authorization profile 1132
Business Information Warehouse support 1233
Business Warehouse components 1234
CCMS alerts used in event rule 1282
configuration 1131
configuration options 1147
connection 1296
control file 1136
correction and transport files 1136
creating job containing InfoPackage 1235
creating job containing process chain 1235
creating jobs 1172
creating RFC user 1132
customization procedure 1131
data file 1136
defining event as event rule 1266
defining event as internetwork dependency 1265
defining event rule based on CCMS alerts 1283
defining event rule based on IDocs 1271
defining job 1180
defining job dynamically 1199
defining jobs 1172
defining variant 1175
deleting variant 1175
dynamic job definition example 1216
dynamic job definition parameter description 1201
dynamic job definition syntax 1200
encrypting user password 1166
event defined as event rule 1266
event defined as internetwork dependency 1265
event rule based on CCMS alerts, business scenario 1281, 1281
event rule based on IDocs, business scenario 1271
event rule prerequisite 1169
exporting calendars, business scenario 1257
filtering events in security file 1270
IDoc record used in event rule 1271
InfoPackage schedule options 1235
installing ABAP modules 1132
introduction 1125, 1128
job interception 1223, 1225, 1226, 1232
job interception and parent-child 1223
jobs dynamic definition 1200
National Language support 1294
options file 1128, 1145, 1166
options National Language support 1295
parameters to define job 1180
prerequisite to define event rule 1169
process chain schedule options 1235
re-running jobs 1199
refreshing variant 1175
rerunning job 1197, 1243
security file, filtering events 1270
setting a filter in security file 1270
setting interception criteria 1227, 1227
setting variant 1175
supported code pages 1296
task string 1180
task string parameter 1237
task string parameters 1180
updating variant 1175
viewing variant 1175
SAP data connection 1174
SAP event
committing by external task 1265
defining as event rule 1266
defining as internetwork dependency 1262, 1265
filtering in security file 1270
monitoring 1170
placeholder job for internetwork dependencies 1261
prerequisite to define a rule 1169
prerequisites for defining as internetwork dependency 1261
r3evman command 1170
r3evmon configuration file 1300
raising 1195
SAP job
defining dynamically 1199
deleting 1192
displaying details 1190
editing 1179
example of defining dynamically 1216
killing 1195
placeholder 1262
task string 1180
variable substitution 1216
verifying status 1191
SAP R/3
ABAP step definition attribute 1210
application server 1168
calendars, exporting 1258
changing password, RFC user 1141
committing MTE alert by external task 1291
committing SAP event by external task 1264
common options 1154
configuration options usage 1166
connecting 1167
defining event as internetwork dependency 1262
event defined as internetwork dependency 1262
exporting factory calendars, command 1258
extended agent job 1171
external command step definition attribute 1214
external program step definition attribute 1214
features 1125
getting CCMS alert status by external task 1290
global options 1147
job throttling 1250
local options 1150
logon group 1169
mapping between definition and resolution of internetwork dependency 1264
monitoring event 1170
placeholder job for internetwork dependencies 1261
r3batch export function 1258
r3evman command 1170
r3evmon configuration file 1300
RFC user password 1141
roles and responsibilities 1129
transaction PFCG 1134
transaction se38 1252
transaction su02 1133
variable substitution 1216
SAP table criteria
setting template file 1230
setting using the Dynamic Workload Console, BC-XBP 2.0 1227
setting using the Dynamic Workload Console, BC-XBP 3.0 1229
SAP user password
changing 1166
encrypting 1166
sched
job state 569
schedtime keyword 379
schedule keyword 381
scheduling job types with advanced options 910
scheduling language 314
scheduling objects
moving to folders in batch mode 521
scheduling resource 39
scriptname
job definition 232
secure network communications 1142
secureaddr
workstation definition 210
security
PeopleSoft 1098
variable tables 165
security access control list
security access control list definition 415
Security access control list definition 414
security domain
security domain definition 417
security domain definition
security domain 417
Security domain definition 416
security file, filtering SAP events 1270
security role
security role definition 419
security role definition
security role 419
Security role definition 418
security SAP
SNC 1142
securitylevel 218
workstation definition 218
sendevent command 967, 1005
server group
balancing SAP R/3 workload 1193
SERVER_NAME_LIST, PeopleSoft option 1100
Server.msg 68
ServerID

mailman process 59
workstation definition 220
servicedescription keyword 388
servicename keyword 387
ServiceNow
event actions 1399
servicetags keyword 388
servicevariables keyword 389
setsym command 644
setting
connection parameters 93
job interception 1227
job interception using interception criteria and template files 1227
job throttling 1251
SAP table criteria on the workstation 1227
SAP table criteria using the Dynamic Workload Console 1229
SAP variant 1175
template file 1230
setting trace levels for application server 1316
SETTING UP OCLI 757
setup
command line reporting 1058
shadow job 20
carry forward 1367
defining 1352, 1357
during remote job recovery 1367
failed 1367
managing in the current plan 1368
status fail 1367
status transition after bind 1365
SHELL_TYPE variable 85
showcpus command 645
showdomain command 654
showexec command 969
showfiles command 656
showjobs command 659
showprompts command 683
showresources command 686
showschedules command 690
shutdown
utility command 970
shutdown command 696
ShutDownLwa
utility command 971
simple run cycle 22
skipping objects when importing 528, 542
slow database access 988
SNC 1142
Solution Manager
job scheduling 1309, 1309, 1313
direct 1314
from job documentation 1315
monitoring jobs 1315
registering 1309
SMSE adapter 1309, 1309, 1313, 1314, 1315, 1315
spool data
browsing 1194
introduction 1194
SSL communication
enabling 218
stageman
carryforward 115
SwitchPlan 143, 143
standard agent
workstation definition 213
start command 697
start condition 169
start of day
establishing communication 71
startappserver command 699
startcond keyword 383, 383
starteventprocessor command 700
starting
WebSphere Application Server Liberty 63
workstation processes 63
Starting and stopping agents on AS/400 systems 1370
Starting and stopping agents on i5/OS systems 1370
Starting and stopping agents on IBM i 1370
Starting and stopping agents on IBM i systems 1370
starting processing
production plan 149
startmon command 700
startOfDay
variable 119, 1065
StartUp command 971
StartUpLwa command 972
state
SAP job 1193
statisticstype custom, keyword 382
status
late 329
status command 701
status mapping
PeopleSoft job 1110
status property 270
stop command 702
stop; progressive command 703
stopappserver command 704
stopeventprocessor command 705
stopmon command 706
stopping
WebSphere Application Server Liberty 63
workstation processes 63
streamlogon
job definition 233
windows user definition 281
stuck
job stream state 580
submit docommand command 707
submit file command 711
submit job
return code 581, 884
submit job command 715
submit sched command 720
submitting
supported agent job 1096
succ
job state 569
job stream state 580
successor
matching criteria 104
pending predecessor 105
predecessor 98, 105
successor job
retrieving properties 270
succp
job state 569
supported
code pages, SAP 1296
supported agent job
submitting 1096
supported agent job defining with end-to-end scheduling 1094
supported agents job defining with Dynamic Workload Console 1093
switcheventprocessor command 725
switching extended agents
$MANAGER keyword 212
$MASTER keyword 212
switchmgr command 726
SwitchPlan
JnextPlan 126
Symphony corruption
resetFTA command 642
Symphony file
JnextPlan 125, 127
production plan 114, 125, 127
syntax 732
agent
access method 1071
defining SAP jobs dynamically 1200
extended agent
check file task 1079
get status task 1079
return code mapping 1114
SystemDrive variable 76
SystemRoot variable 76

T

table criteria SAP
setting template file 1230
setting using the Dynamic Workload Console 1227, 1229
table of variables
using 164
task
job definition 233
task options
access method 1071
task string parameters
PeopleSoft job 1108
SAP job 1180, 1180, 1237
tasktype
job definition 234
tcpaddr
workstation definition 210
tellop command 728
TEMP variable 76
template file
creating 1230
description 1230
setting placeholders for job interception 1232
templates
for scheduling object definitions 200
temporary variants, examples 1216
throttling, job 1250
time zone
enabling 1064
timezone
in job streams 389
workstation definition 211
timezone keyword 389
TIVOLI_JOB_DATE variable 76, 80
TMPDIR variable 76
TMPTEMP variable 76
trace file
trace-psagent.log 1122
trace-r3batch.log 1123
trace levels
application server
setting 1316
traces observability
OpenTelemetry 903
tracing utility
configuring 1120
tracking

PeopleSoft job 1098
transaction PFCG
SAP R/3 1134
transactions, SAP
se16 1227
transactions, SAP R/3
PFCG 1134
sa38 1303
se16 1252
se37 1262
se38 1252, 1303
sm69 1215, 1215
su02 1133
trial extension
planman command line 135
trial plan
creating 134
description 117
extension 135
trialsked
forecast plan 118
trial plan 117
trigger action 39
troubleshooting
EEO0778E 1303
extended agent
job log 1308
PeopleSoft, job submission fails 1308
PeopleSoft, local options file rights 1308
extendend agent
Oracle, job submission fails 1308
job throttling
alerts are not generated according to threshold 1307
does not start 1306, 1306
does not stop 1306
saving MTE properties generates message 1307
longlink file 1305
permission denied
job log 1308
r3batch 1298
does not start 1305
environment variables 1305
modifying job step error 1304, 1304
monitoring IDoc events 1301, 1302, 1303
monitoring SAP events 1298, 1300, 1300, 1300
output with unreadable characters 1297
scheduling SAP jobs 1304
system cannot intercept jobs 1307
r3event
output with unreadable characters 1297
r3evmon
monitoring events 1299
monitoring events increases memory consumption 1302
restarting process of subchain 1299
SAP
connection 1296
SAP R/3
error defining internetwork dependency 1303
troubleshooting tool 974
tws_inst_pull_info
deprecated 974
replaced by wa_pull_info 974
wa_pull_info 974
tws_inst_pull_info command 974
TWS_MAX_WAIT_TIME, PeopleSoft option 1101
TWS_MIN_WAIT_TIME, PeopleSoft option 1101
TWS_PROMOTED_JOB 1078
TWS_PROMOTED_JOB variable 76, 80
TWS_RETRY, PeopleSoft option 1101
TWS_TISDIR variable 80
twstrace command 1007
TWSXA_INLINE_CI, PeopleSoft option 1101
TWSXA_SCHED_METH, PeopleSoft option 1101
type
workstation definition 213
typeface conventions xvi
TZ variable 76, 80

U

u
jobthrottling parameter 1254
Unicode support
R/3 1143
UNISON_CPU variable 76, 80
UNISON_DATE variable 78
UNISON_DATE_FORMAT variable 82
UNISON_DIR variable 77, 80
UNISON_EXEC_PATH variable 77, 81
UNISON_EXIT variable 84
UNISON_HOST variable 77, 81
UNISON_JCL variable 84
UNISON_JOB variable 77, 81
UNISON_JOBNUM variable 77, 81
UNISON_MASTER variable 77, 81
UNISON_RUN variable 77, 81
UNISON_SCHED variable 77, 81
UNISON_SCHED_DATE variable 82
UNISON_SCHED_EPOCH variable 78, 82
UNISON_SCHED_IA variable 78, 82
UNISON_SCHED_ID variable 78, 82
UNISON_SHELL variable 78, 82
UNISON_STDLIST variable 78, 82, 84
UNISON_SYM variable 78, 82
UNISONHOME variable 77, 81
UNIXTASK 234
UNKNOWN 234
unlink command 729
unlock command 513
unlocking plan
planman command line 139
until keyword 390
untilDays
option 120
update command 517
UpdateStats
JnextPlan 126
updating
SAP variant 1175
USE_EXEC variable 85
user 39
user authorizations
Business Warehouse InfoPackage 1234
Business Warehouse process chain 1234
user definition 281
trusted domain 285
using on job types with advanced options 285
user interfaces
composer 53
conman 53
Dynamic Workload Console 52
Java API 53
optman 53
planman 54
plug-ins 53
user password
encrypting on PeopleSoft 1102
encrypting on SAP 1166
user return code on AS/400 1378, 1379
user return code on i5/OS 1378, 1379
user return code on IBM i 1378, 1379
user security
setting 1130
USERDOMAIN variable 78
USERNAME variable 79
USERPROFILE variable 79
using
default variable table 164
variable table 164
Using utility commands on agents on AS/400 systems 1370
Using utility commands on agents on i5/OS systems 1370
Using utility commands on agents on IBM i systems 1370, 1370
utility commands 915
agents 978
at 918
at.allow file 920
at.deny file 920
ATSCRIPT variable 919
batch 918
changing date format 926
checking dynamic agent connection 932
creating and managing variables and passwords locally on dynamic agents 990
creating calendars 953
defining custom events 934
deleting files 933
displaying content of standard list files 955
displaying product version 973
displaying running jobs 969
displaying standard list files 965
dynamic 978
dynamic domain manager 978
exporting data 924
getting HTML reports 955, 974
getting job information 947
getting TWS_home path 952
getting workstation information 921
importing data 925
list of commands 915
listing standard list files 949
managing parameters locally 958
releasing resource units 961
removing standard list files 965
sending custom events 967
setting mailbox file size 939
setting variables locally on dynamic agents 987
shutdown 970
ShutDownLwa 971
starting up netman 971, 972
utility commands for dynamic agents
sending custom events 1005

V

validate command 520
validfrom keyword 393
variable 40
defining on dynamic agent 258
definition 289
in job definitions 243

job types with advanced options 258
resolution 166
resolving on dynamic agent 258
variable substitution
cross-job properties 270
SAP R/3 1216
variable table 40
data integrity 165
default 164
definition 295
lock mechanism 165
using 164
variable table definition
vartable 209
variable tables
security 165
variables
ATSCRIPT 919
carryforward 144
carryStates 115, 120
COMPUTERNAME 75
enCarryForward 115, 120
enCFInterNetworkDeps 121
enCFResourceQuantity 121
enLegacyStartOfDayEvaluation 124, 1065
enPreventStart 122
enTimeZone 124, 1064
exported locally by .jobmanrc 83, 86, 88, 91
exported on UNIX 79
HOME 75, 79, 89
HOMEDRIVE 75
HOMEPATH 75
LANG 75, 79
LD_LIBRARY_PATH 79
LD_RUN_PATH 79
local variables 88, 91
LOCAL_RC_OK 84, 89
logmanMinMaxPolicy 123
logmanSmoothPolicy 123
LOGNAME 75, 79
MAESTRO_OUTPUT_STYLE 75, 80
MAIL_ON_ABEND 84, 85, 89, 91
maxLen 119
minLen 119
PATH 80
POSIXHOME 89
SHELL_TYPE 85
startOfDay 120, 1065
SystemDrive 75
SystemRoot 76
TEMP 76
TIVOLI_JOB_DATE 76, 80
TMPDIR 76
TMPTEMP 76
TWS_PROMOTED_JOB 76, 80
TWS_TISDIR 80
TZ 76, 80
UNISON_CPU 76, 80
UNISON_DATE 78
UNISON_DATE_FORMAT 82
UNISON_DIR 77, 80
UNISON_EXEC_PATH 77, 81
UNISON_EXIT 84
UNISON_HOST 77, 81
UNISON_JCL 84
UNISON_JOB 77, 81
UNISON_JOBNUM 77, 81
UNISON_MASTER 77, 81
UNISON_RUN 77, 81
UNISON_SCHED 77, 81
UNISON_SCHED_DATE 81
UNISON_SCHED_EPOCH 78, 82
UNISON_SCHED_IA 78, 82
UNISON_SCHED_ID 78, 82
UNISON_SHELL 78, 82
UNISON_STDLIST 78, 82, 84
UNISON_SYM 78, 82
UNISONHOME 77, 81
untilDays 120
USE_EXEC 85
USERDOMAIN 78
USERNAME 78
USERPROFILE 79
variables, environment, notation xvii
variant SAP
defining 1175
deleting 1175
refreshing 1175
setting 1175
updating 1175
viewing 1175
variants
temporary 1216
vartable
variable table definition 209
vartable keyword 394
verifying status
SAP job 1191
version
utility command 973
version command 521, 730
displaying composer banner 521
viewing
SAP variant 1175
spool data 1194

W

wa_pull_info command 974
wait
job state 570
wappman command 538
logs and traces 538
WAS
stopping 704
WebSphere Application Server Liberty
infrastructure 57
starting 63
stopping 63, 704
weekly run cycle 22
wildcards
composer 442
Windows command prompt
privilege level to issue HCL Workload Automation commands 57
Windows operating systems
privilege level to issue HCL Workload Automation commands 57
Windows OS
special characters, handling 147
Windows user
defining 284
definition 284
running jobs on a dynamic pool 284
running jobs on a pool 284
running jobs on an agent 284
scheduling on a dynamic pool 284
scheduling on a pool 284
scheduling on an agent 284
WINDOWSTASK 234
within a relative interval
follows 344
follows relative to 102
join 352
matching criteria 102
within an absolute interval
follows 344
follows absolute to 103
join 352
matching criteria 103
workload analysis
alerts 891
metrics 891
workload application
command line 538
definition 21
skipping objects 528, 542
workload applications
defining 412
importing 537
logs and traces 538
mapping 537
mapping file 529
regular expressions 535
skipping objects 538
workload applications managing 527
workload automation 169
workload customizing
using variable table 164
workload monitoring
alerts 891
metrics 891
workload service assurance 247, 247
calculating job start times 118
forecast plan 118
workload, balancing SAP R/3 1193
workstation
backup master domain manager 31
class 36
creating 201
d-pool type 216
defining 201
domain manager 31
dynamic pool 31
dynamic pool type 216
fault-tolerant agent 31
mailbox files
NetReq.msg 66
master domain manager 31
pool 31
pool type 216
processes 57
remote engine type 201, 215
remote engine workstation 31
workstation class 36
workstation class definition 224
cpuclass 225
ignore 226
members 226, 417, 419
workstation definition 201, 218
access 212
agent 215
autolink 217
behindfirewall 217
broker 214
cpuname 208
domain 211
dynamic agent 215
extended agent 213
fta 213
fullstatus 218
host 211
manager 214
members 221

os type 209
protocol 220
requirements 221
secureaddr 210
ServerID 220
standard agent 213
tcpaddr 210
timezone 211
type 213
workstation for extended agent or dynamic agent, defining with
Dynamic Workload Console 1086
workstation for extended agent, defining with
ISPF 1091
workstation for supported agent, defining with
command line 1088
workstation links status 650
workstation process status 650
workstation processes 58, 58
batchman 59
inter-process communication 66
jobman 60
mailman 59
ServerID 59
managing change of job states 72
monman 58
netman 58
processes tree on UNIX 61
processes tree on Windows 62
start of day
establishing communication 71
starting 63
stopping 63
writer 59
workstation properties 654
workstation status 650, 654
workstation, defining with
Dynamic Workload Console 1086
workstations
unique ID 484, 646
writer process 59

X

XATASK 234
xref command 1020
xrxtrct command 1042

Y

yearly run cycle 22