

HCL Workload Automation
User's Guide and Reference
Version 10.2.5



Note

Before using this information and the product it supports, read the information in [Notices on page mcxl](#).

This edition applies to version 10, release 2, modification level 5 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	Customizing job processing on a Windows workstation - djobmanrc.cmd.....	79
List of Tables.....	xi	Setting up options for using the user interfaces.....	81
About this publication.....	xv	Chapter 4. Managing the production cycle.....	83
What is new in this release.....	xv	Plan management basic concepts.....	83
What is new in this publication.....	xv	Preproduction plan.....	85
Accessibility	xv	Identifying job stream instances in the plan.....	86
Conventions used in this publication.....	xv	Managing external follows dependencies for jobs and job streams.....	87
Typeface conventions.....	xv	Production plan.....	99
Operating system-dependent variables and paths.....	xvi	Understanding carry forward options.....	100
Command syntax.....	xvi	Trial plan.....	102
Chapter 1. HCL Workload Automation.....	19	Forecast plan.....	103
Understanding basic concepts.....	19	Customizing plan management using global options.....	104
HCL Workload Automation database objects.....	19	Creating and extending the production plan.....	108
The HCL Workload Automation network.....	38	JnextPlan.....	109
Configuring your runtime environment.....	39	Planman command line.....	111
Defining scheduling activities.....	40	Creating an intermediate production plan.....	113
Controlling job and job stream processing.....	40	Creating an intermediate plan for a plan extension.....	114
Managing production scheduling activities.....	47	Retrieving the production plan information.....	115
Automating workload using event rules.....	48	Creating a trial plan.....	116
HCL Workload Automation user interfaces.....	49	Creating a trial plan of a production plan extension.....	118
Starting production.....	50	Creating a forecast plan.....	119
Chapter 2. Understanding basic processes and commands.....	54	Deploying rules.....	120
Issuing commands on Windows operating systems.....	54	Unlocking the production plan.....	121
HCL Workload Automation workstation processes.....	54	Resetting the production plan.....	121
Starting and stopping processes on a workstation.....	58	Removing the preproduction plan.....	122
Workstation inter-process communication.....	61	Replicating plan data in the database.....	123
HCL Workload Automation network communication.....	64	Monitoring the replication of plan data in the database.....	124
Support for Internet Protocol version 6.....	66	The stageman command.....	125
Chapter 3. Configuring the job environment.....	68	Managing concurrent accesses to the Symphony file.....	127
Job environment overview.....	68	Scenario 1: Access to Symphony file locked by other HCL Workload Automation processes.....	127
Environment variables exported by jobman.....	69	Scenario 2: Access to Symphony file locked by stageman.....	127
Customizing date formatting in the stdlist.....	72	Managing follows dependencies using carry forward prompt.....	128
Customizing job processing on a UNIX workstation - jobmanrc.....	72	The logman command.....	128
Customizing the MAIL_ON_ABEND section of jobmanrc.....	74	Starting production plan processing.....	131
Customizing job processing for a user on UNIX workstations- jobmanrc.....	75	Automating production plan processing.....	132
Customizing job processing on a Windows workstation.....	77	Detecting loops between jobs and job streams in current and trial plans.....	133
Customizing the MAIL_ON_ABEND section of jobmanrc.cmd.....	78	Chapter 5. Using workload service assurance.....	135

Enabling and configuring workload service assurance.....	135
Planning critical jobs.....	139
Processing and monitoring critical jobs.....	141
Workload service assurance scenario.....	143
Chapter 6. Customizing your workload using variable tables.....	145
Migrating global parameters from previous versions.....	145
The default variable table.....	146
Data integrity for variable tables.....	147
Locking mechanism for variable tables.....	147
Variable table security.....	147
Variable resolution.....	148
Chapter 7. Condition-based workload automation.....	150
A business scenario.....	153
Chapter 8. Running event-driven workload automation.....	155
The event rule management process.....	158
Using the involved interfaces and commands...	160
Defining event rules.....	163
Security checks on event rules.....	165
Event rule examples.....	166
Rule operation notes.....	173
Triggered rule elements.....	175
Defining custom events.....	176
Chapter 9. Defining objects in the database.....	177
Defining scheduling objects.....	177
Workstation definition.....	181
Workstation class definition.....	201
Domain definition.....	203
Job definition.....	205
User definition.....	222
Calendar definition.....	227
Folder definition.....	228
Variable and parameter definition.....	230
Variable table definition.....	235
Prompt definition.....	237
Resource definition.....	239
Run cycle group definition.....	240
Job stream definition.....	252
Job stream definition keyword details.....	263
Variable table.....	330
Event rule definition.....	330
Workload application definition.....	346
Security object definition.....	348
Chapter 10. Managing objects in the database - composer.....	367
Setting up the composer command-line program.....	367
Setting up the composer environment.....	368
Running the composer program.....	370
Running commands from composer.....	374
Filters and wildcards.....	374
Delimiters and special characters.....	378
Composer return codes.....	379
COMPOSER COMMANDS.....	380
Referential integrity check.....	382
Add.....	387
Authenticate.....	389
Chfolder.....	390
Continue.....	392
Delete.....	393
Display.....	398
Edit.....	404
Exit.....	405
Extract.....	405
Help.....	411
List.....	412
Listfolder.....	421
Lock.....	422
Mkfolder.....	427
Modify.....	428
New.....	434
Print.....	437
Redo.....	437
Rmfolder.....	439
Rename.....	440
Renamefolder.....	444
Replace.....	445
System command.....	446
Unlock.....	447
Update.....	451
Validate.....	454
Version.....	455
Organizing scheduling objects into folders.....	455
Chapter 11. Managing workload applications.....	460
Reusing a workload in another environment.....	461
Resolving the mapping file.....	462
Using regular expressions to modify the mapping file.....	468
Deploying a workload application.....	470
Running the wappman program.....	471
Chapter 12. Managing objects in the plan - conman.....	478
Setting up the conman command-line program.....	478
Setting up the conman environment.....	479
Running the conman program.....	481
Running commands from conman.....	485

Wildcards.....	486	Showcpus.....	572
Delimiters and special characters.....	487	Showdomain.....	581
Conman commands processing.....	488	Showfiles.....	583
Selecting jobs in commands.....	489	Showjobs.....	586
Syntax.....	489	Showprompts.....	610
Arguments.....	489	Showresources.....	613
Selecting job streams in commands.....	500	Showschedules.....	617
Syntax.....	501	Shutdown.....	623
Arguments.....	501	Start.....	624
Conman return codes.....	509	Startappserver.....	626
Conman commands.....	509	Starteventprocessor.....	626
Adddep job.....	513	Startmon.....	627
Addep Sched.....	516	Status.....	628
Altjob.....	519	Stop.....	629
Altpass.....	520	Stop ,progressive.....	630
Altpri.....	521	Stopappserver.....	630
Cancel job.....	522	Stopeventprocessor.....	632
Cancel sched.....	524	Stopmon.....	632
Checkhealthstatus.....	526	Submit docommand.....	633
Chfolder.....	526	Submit file.....	638
Confirm.....	528	Submit job.....	642
Console.....	532	Submit sched.....	646
Continue.....	533	Switcheventprocessor.....	651
Deldep job.....	534	Switchmgr.....	653
Deldep sched.....	536	System command.....	654
Deployconf.....	538	Tellop.....	654
Display.....	538	Unlink.....	655
Exit.....	541	Version.....	657
Fence.....	542	Chapter 13. Using Orchestration Query Language.....	658
Help.....	543	Chapter 14. Orchestration CLI.....	682
Kill.....	544	Configuring Orchestration CLI.....	682
Limit cpu.....	545	Running commands from Orchestration CLI.....	687
Limit sched.....	547	Authenticating Orchestration CLI using API Keys.....	688
Link.....	548	Orchestration CLI commands.....	689
Listfolder.....	549	Access required to run Orchestration CLI	
Listsym.....	550	commands.....	690
Listsucc.....	552	Context commands.....	695
Recall.....	554	Model commands.....	699
Redo.....	555	Plan commands.....	730
Release job.....	557	help.....	782
Release sched.....	559	version.....	783
Reply.....	561	Special characters as wildcards and delimiters.....	783
Rerun.....	562	Orchestration CLI return codes.....	785
Rerunsucc.....	566	Managing multiple config.yaml files.....	788
ResetFTA.....	569	Managing multiple formats.....	789
Resource.....	570	Chapter 15. Modifying context parameter temporarily...	791
Setsym.....	571	Chapter 16. Monitoring HCL Workload Automation	792
		Exposing metrics to monitor your workload.....	792

The BmEvents configuration file.....	795	da_test_connection.....	881
Logged events.....	799	DELETE.....	882
Enabling observability with OpenTelemetry.....	802	Evtdef.....	883
Chapter 17. Extending HCL Workload Automation capabilities.....	805	Evtsize.....	888
Prerequisite steps to create job types with advanced options.....	805	Filemonitor.....	890
Creating advanced job definitions.....	807	Jobinfo.....	896
Job definition - z/OS jobs.....	807	Jobstdl.....	898
Remote command jobs.....	808	Maestro.....	901
IBM i jobs.....	811	Makecal.....	901
Executable jobs.....	816	Metronome.....	904
Access method jobs.....	817	Morestdl.....	904
Shadow jobs.....	819	Parms.....	906
Variable Table jobs.....	821	Release.....	909
Job Management jobs.....	824	Rmstdlist.....	914
Job Stream Submission jobs.....	829	Sendevent.....	915
Return Codes for job types with advanced options.....	833	Showexec.....	917
Defining variables and passwords for local resolution on dynamic agents.....	835	Shutdown.....	919
Specifying local variables and passwords in the job definitions.....	835	ShutDownLwa.....	919
Obtaining passwords from password vaults.....	838	StartUp.....	920
Defining variables in dynamic workload broker jobs.....	844	StartUpLwa.....	921
Passing variables between jobs	846	version.....	921
Passing job properties from one job to another in the same job stream instance.....	846	wa_pull_info.....	923
Passing job standard output from one job to another in the same job stream instance.....	852	Unsupported commands.....	925
Passing job standard output from one job to another as standard input in the same job stream instance.....	853		
Passing variables set by using jobprop in one job to another in the same job stream instance.....	854	Chapter 20. Using utility commands in the dynamic environment.....	926
Passing variables from one job to another in the same job stream or in a different job stream by using variable tables	856	Command-line configuration file.....	927
Running a script when a job completes.....	856	cleanuserjobs.....	931
Chapter 18. Managing dynamic scheduling.....	858	exportserverdata.....	932
Dynamic capability: a business scenario.....	859	importserverdata.....	933
Defining file dependencies in dynamic scheduling....	860	jobprop.....	935
Promoting jobs scheduled on dynamic pools.....	863	movehistorydata.....	936
Limitations in dynamic scheduling.....	863	param.....	938
Chapter 19. Using utility commands.....	865	resource.....	941
Command descriptions.....	865	Using the resource command from an agent....	951
at and batch.....	868	sendevent.....	953
Cpuinfo.....	871	twstrace.....	955
Dataexport.....	874	Chapter 21. Getting reports and statistics.....	958
Dataimport.....	875	Setup for using report commands.....	958
Datecalc.....	876	Changing the date format.....	959
		Command descriptions.....	959
		Rep1 - rep4b.....	960
		Rep7.....	962
		Rep8.....	963
		Rep11.....	965
		Reptr.....	966
		Xref.....	968
		Sample report outputs.....	969
		Report 01 - Job Details Listing.....	969

Report 02 - Prompt Listing.....	972	Chapter 24. Managing internetwork dependencies.....	1030
Report 03 - Calendar Listing.....	972	Internetwork dependencies overview.....	1030
Report 04A - Parameter Listing.....	973	Understanding how an internetwork dependency is shown.....	1031
Report 04B - Resource Listing.....	973	Configuring a network agent.....	1032
Report 07 - Job History Listing.....	974	A sample network agent definition.....	1033
Report 08 - Job Histogram.....	975	Defining an internetwork dependency.....	1034
Report 9B - Planned Production Detail.....	975	Managing internetwork dependencies in the plan...	1035
Report 10B - Actual Production Detail.....	976	States of jobs defined in the EXTERNAL job stream.....	1035
Report 11 - Planned Production Schedule.....	977	Working with jobs defined in the EXTERNAL job stream.....	1036
Report 12 - Cross Reference Report.....	979	Sample internetwork dependency management scenarios.....	1036
Report extract programs.....	981	Internetwork dependencies in a mixed environment.....	1038
Jbextract.....	982	Chapter 25. Applying conditional branching logic	1040
Prxtract.....	984	Setting up conditional dependencies.....	1042
caxtract.....	985	Joining or combining conditional dependencies.....	1044
Paxtract.....	985	Scheduling and submitting conditional dependencies.....	1046
Rextract.....	986	Evaluating and processing a conditional dependency flow.....	1046
R11xtr.....	987	Monitoring conditional dependencies.....	1053
Xrxtrct.....	989	Plan handling of conditional dependencies.....	1056
Running reports and batch reports.....	995	Chapter 26. Defining and managing cross dependencies.....	1058
Historical reports.....	998	An introduction to cross dependencies.....	1058
Production reports.....	1003	Processing flow across the distributed scheduling environment.....	1060
Running batch reports from the command line interface.....	1004	Defining a cross dependency.....	1063
Chapter 22. Managing time zones.....	1011	Monitoring a cross dependency resolution in the production plan.....	1065
Enabling time zone management.....	1011	How the shadow job status changes until a bind is established.....	1065
How HCL Workload Automation manages time zones.....	1012	How the shadow job status changes after the bind is established.....	1071
Moving to daylight saving time on.....	1014	How to see why the shadow job status is FAIL.....	1073
Moving to daylight saving time off.....	1015	Shadow job status during the remote job recovery or rerun.....	1073
General rules.....	1015	How carry forward applies to shadow jobs.....	1073
Chapter 23. Defining access methods for agents.....	1017	Managing shadow jobs in the production plan.....	1074
Access method interface.....	1018	Chapter 27. Managing an IBM i dynamic environment. 1075	
Method command line syntax.....	1018	Defining agents on IBM i systems.....	1075
Method response messages.....	1020	Defining IBM i jobs.....	1075
Method options file.....	1021	Managing agents on IBM i systems.....	1076
Running methods.....	1024	Starting and stopping agents on IBM i systems.....	1076
Launch job task (LJ).....	1024	Using utility commands on agents on IBM i systems.....	1076
Manage job task (MJ).....	1025	Scheduling jobs on IBM i systems.....	1077
Check file task (CF).....	1026		
Get status task (GS).....	1026		
Cpuinfo command for extended agents only...	1027		
Troubleshooting.....	1027		
Job standard list error messages.....	1028		
Method not executable.....	1028		
Console Manager messages for extended agents only.....	1028		
Composer and compiler messages for extended agents only.....	1028		
Jobman messages for extended agents only..	1029		

The agent joblog and TWSASPOOLS environment variable.....	1079
Child job monitoring on IBM i agents.....	1080
The agent return code retrieval.....	1084
Controlling the job environment with the user return code.....	1084
Alternative method to set the user return code.....	1085
Appendix A. Event-driven workload automation event and action definitions.....	1087
Event providers and definitions.....	1087
TWSObjectsMonitor events.....	1087
FileMonitor events.....	1090
TWSApplicationMonitor events.....	1099
DatasetMonitor events.....	1100
Action providers and definitions.....	1102
GenericAction actions.....	1103
MailSender actions.....	1104
MessageLogger actions.....	1104
ServiceNow actions.....	1104
TWSAction actions.....	1104
TWSForZosAction actions.....	1105
Appendix B. Quick reference for commands.....	1107
Managing the plan.....	1107
Managing objects in the database.....	1109
General purpose commands.....	1109
Scheduling objects.....	1109
Composer commands.....	1117
Managing objects in the plan.....	1124
Conman commands.....	1124
Utility commands.....	1132
Report commands.....	1136
Appendix C. Accessibility.....	1139
Notices.....	mcxl
Index.....	1144

List of Figures

Figure 1: Single-domain network.....	35	Figure 25: Example when start of day conversion is not applied.....	1013
Figure 2: Multiple-domain network.....	36	Figure 26: Example when start of day conversion is applied.....	1014
Figure 3: Process tree in UNIX®.....	57	Figure 27: Local and remote networks.....	1033
Figure 4: Process tree in Windows®.....	58	Figure 28: A follows dependency on the ABSENCES job..	1047
Figure 5: Inter-process communication on the master domain manager.....	63	Figure 29: Two different conditional dependencies on SUCC and ABEND statuses on the ABSENCES job.....	1048
Figure 6: Inter-process communication on the master domain manager and fault-tolerant agent.....	64	Figure 30: Conditional dependencies on output conditions on the ABSENCES job.....	1049
Figure 7: Sameday matching criteria.....	87	Figure 31: A join dependency containing three dependencies on SUCC status.....	1050
Figure 8: Closest preceding matching criteria.....	88	Figure 32: Status conditional dependency on a job with recovery stop setting.....	1051
Figure 9: Within a relative interval matching criteria.....	88	Figure 33: ABEND status conditional dependency.....	1051
Figure 10: Within an absolute interval matching criteria.....	89	Figure 34: STATUS_OK output condition.....	1052
Figure 11: Closest preceding predecessor job	90	Figure 35: ERROR output condition.....	1052
Figure 12: Pending predecessor instance.....	91	Figure 36: STATUS_OK output condition.....	1053
Figure 13: Sameday matching criteria - Step 1: at Start of Day (SOD) on a Thursday.....	92	Figure 37: Cross dependency logic.....	1060
Figure 14: Sameday matching criteria - Step 2: at 9:00	93	Figure 38: Shadow job status transition until the bind is established.....	1066
Figure 15: Sameday matching criteria - Step 3: at 15:00.....	93	Figure 39: Instance to be bound if the shadow job scheduled time is included in the CP interval.....	1069
Figure 16: Closest preceding matching criteria - Step 1: before 08:00.....	94	Figure 40: 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.....	1070
Figure 17: Closest preceding matching criteria - Step 2: at 08:00 on weekdays except Thursdays and Fridays.....	95	Figure 41: The scheduled time of the shadow job is included in the CP but no instance to bind exists.....	1070
Figure 18: Closest preceding matching criteria - Step 3: at 09:00 on Thursdays and Fridays.....	95	Figure 42: The instance to be bound exists but it is not yet included in the CP.....	1071
Figure 19: Closest preceding matching criteria - Step 4: at 15:00 on every day	96	Figure 43: The LTP interval still does not contain the shadow job scheduled time.....	1071
Figure 20: Relative Interval matching criteria - at start of day on Thursday.....	98	Figure 44: Shadow job status transition chain after the bind was established.....	1072
Figure 21: Absolute interval matching criteria - at start of day on Thursday.....	99		
Figure 22: Critical path.....	142		
Figure 23: Condition-based workload automation.....	152		
Figure 24: User definition.....	226		

List of Tables

Table 1: Command syntax.....	xvii	Table 22: List of reserved words when defining workstations.....	180
Table 2: Scenario 1. No time restriction in the run cycle group.....	26	Table 23: List of reserved words when defining users.....	181
Table 3: Scenario 2. Time restriction in the run cycle group without offset.....	27	Table 24: Attribute settings for management workstation types.....	182
Table 4: Scenario 3. Time restriction in the run cycle group with offset (+1 12:00).....	27	Table 25: Attribute settings for target workstation types..	184
Table 5: Starting and stopping HCL Workload Automation on a workstation.....	59	Table 26: Type of communication depending on the security level value.....	197
Table 6: Starting and stopping the agent	61	Table 27: Examples: renaming the job definition.....	207
Table 7: Job environment variables for Windows®.....	69	Table 28: Comparison operators.....	211
Table 8: Job environment variables for UNIX®.....	70	Table 29: Logical operators.....	212
Table 9: Variables defined by default in the jobmanrc file...	73	Table 30: Recovery options and actions.....	215
Table 10: Variables defined by default in the jobmanrc.cmd file.....	78	Table 31: Supported HCL Workload Automation variables in JSDL definitions.....	221
Table 11: Carry forward global options settings.....	100	Table 32: Folder commands.....	229
Table 12: Resulting carry forward settings.....	101	Table 33: How to handle a backslash in variable substitution.....	232
Table 13: Workload service assurance global options.....	136	Table 34: Keywords that can take local parameters in submit commands.....	232
Table 14: Workload service assurance local options.....	138	Table 35: Required access keyword on variable table in Security file (varable object) and allowed actions.....	237
Table 15: The relationship between variable tables and their enclosed variables in the HCL Workload Automation security file.....	148	Table 36: List of scheduling keywords.....	255
Table 16: conman commands for managing monitoring engines.....	158	Table 37: Explanation of the notation defining the number of occurrences for a language element.....	331
Table 17: conman commands for managing the event processing server.....	160	Table 38: <code>TWSObjectsMonitor</code> events.....	336
Table 18: Interfaces and commands for managing event-driven workload automation.....	161	Table 39: <code>TWSApplicationMonitor</code> events.....	337
Table 19: List of supported scheduling object keywords..	178	Table 40: <code>FileMonitor</code> events.....	338
Table 20: List of supported security object keywords.....	179	Table 41: <code>DatasetMonitor</code> events.....	338
Table 21: List of reserved words when defining jobs and job streams.....	179	Table 42: Action types by action provider.....	341
		Table 43: Security object types.....	353
		Table 44: Actions that users or groups can perform on the different objects.....	354

Table 45: Actions that users or groups can perform when designing and monitoring the workload.....	356	Table 68: OQL syntax fields for workstation.....	660
Table 46: Actions that users or groups can perform when modifying current plan.....	356	Table 69: OQL syntax fields for job stream.....	663
Table 47: Actions that users or groups can perform when submitting workload	357	Table 70: OQL syntax fields for jobs.....	671
Table 48: Actions that users or groups can perform when managing the workload environment.....	358	Table 71: OQL syntax fields for resources.....	678
Table 49: Actions that users or groups can perform when managing event rules.....	359	Table 72: Mandatory fields.....	683
Table 50: Administrative tasks that users or groups can perform	360	Table 73: Optional fields.....	683
Table 51: Actions that users or groups can perform on workload reports.....	360	Table 74: Scheduling items.....	701
Table 52: Actions that users or groups can perform on folders.....	361	Table 75: Output format for displaying items.....	704
Table 53: Attributes for security object types.....	362	Table 76: Scheduling items.....	706
Table 54: Scheduling objects filtering criteria	376	Table 77: Scheduling items.....	709
Table 55: Delimiters and special characters for composer.....	378	Table 78: Output format for displaying items.....	712
Table 56: List of composer commands.....	380	Table 79: Scheduling items.....	714
Table 57: Object identifiers for each type of object defined in the database.....	382	Table 80: Scheduling items.....	717
Table 58: Object definition update upon deletion of referenced object.....	383	Table 81: Scheduling items.....	720
Table 59: Referential integrity check when deleting an object from the database.....	383	Table 82: Scheduling items.....	724
Table 60: Objects extracted during the export process....	463	Table 83: Scheduling items.....	728
Table 61: Resolving the mapping file.....	467	Table 84: Change in status after confirm command.....	742
Table 62: Delimiters and special characters for conman..	487	Table 85:	753
Table 63: List of conman commands.....	510	Table 86: Special characters as delimiters.....	785
Table 64: State change after confirm command.....	529	Table 87:	786
Table 65: Recovery options retrieval criteria.....	563	Table 88: Workload Automation exposed metrics.....	792
Table 66: Successors status.....	567	Table 89: Events filtered by CHSCHED.....	797
Table 67: OQL syntax keywords.....	658	Table 90: Logged events.....	799
		Table 91: Required and optional attributes for the definition of a z/OS job.....	807
		Table 92: Required and optional attributes for the definition of a remote command job.....	809
		Table 93: Required and optional attributes for the definition of an IBM i job.....	812
		Table 94: Required and optional attributes for the definition of an executable job.....	816
		Table 95: Required and optional attributes for the definition of an access method job.....	817

Table 96: Required and optional attributes for the definition of a Variable Table job.....	822	Table 122: Xdep_job output fields (continued).....	990
Table 97: Required and optional attributes for the definition of a Job Management job.....	825	Table 123: Xdep_sched output fields.....	991
Table 98: Required and optional attributes for the definition of a Job Stream Submission job.....	830	Table 124: Xfile output fields.....	992
Table 99: Status mapping.....	832	Table 125: Xjob output fields.....	992
Table 100: Supported HCL Workload Automation variables in JSDL definitions.....	845	Table 126: Xprompts output fields.....	993
Table 101: Properties for shadow jobs.....	849	Table 127: Xresource output fields.....	993
Table 102: Properties for IBM Sterling Connect:Direct jobs.....	849	Table 128: Xsched output fields.....	994
Table 103: Properties for file transfer jobs.....	850	Table 129: Xwhen output fields.....	995
Table 104: Properties for JSR 352 Java Batch jobs	850	Table 130: Supported report output formats.....	997
Table 105: Properties for Job Management jobs.....	851	Table 131: Summary of historical reports.....	998
Table 106: Properties for Job Stream Submission jobs....	851	Table 132:	1004
Table 107: Properties for database jobs.....	852	Table 133: Method command task options.....	1019
Table 108: Features partially or not supported for dynamic scheduling.....	863	Table 134: Launch job task (LJ) messages.....	1025
Table 109: List of utility commands.....	865	Table 135: Check file task (CF) messages.....	1026
Table 110: Additional properties that can be used for defining custom events.....	886	Table 136: Get status task (GS) messages.....	1027
Table 111: List of utility commands for dynamic workstations.....	926	Table 137: Internetwork dependencies in a mixed environment.....	1039
Table 112: Date formats.....	959	Table 138: Shadow job status transition.....	1060
Table 113: List of report commands.....	959	Table 139: Matching criteria for distributed shadow jobs.....	1064
Table 114: Report extract programs.....	981	Table 140: Regular expression syntax.....	1091
Table 115: Jbextract output fields.....	983	Table 141: Regular expression examples.....	1093
Table 116: Prxtract output fields.....	984	Table 142: SMF events.....	1101
Table 117: Caxtract output fields.....	985	Table 143: Parameters of ReadCompleted and ModificationCompleted event types.....	1101
Table 118: Paxtract output fields.....	986	Table 144: Commands used against the plan.....	1107
Table 119: Rextract output fields.....	987	Table 145: General purpose commands.....	1109
Table 120: R11xtr output fields.....	988	Table 146: Composer commands.....	1118
Table 121: Xdep_job output fields.....	990	Table 147: Commands that can be run from conman....	1125
		Table 148: Utility commands available for both UNIX® and Windows®.....	1133
		Table 149: Utility commands available for UNIX® only...	1136

Table 150: Utility commands available for Windows® only..... 1136

Table 151: Report commands..... 1137

Table 152: Report extract programs..... 1138

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.

Accessibility

Accessibility features help users with a physical disability, such as restricted mobility or limited vision, to use software products successfully.

With this product, you can use assistive technologies to hear and navigate the interface. You can also use the keyboard instead of the mouse to operate all features of the graphical user interface.

For detailed information, see the appendix about accessibility in the *HCL Workload Automation User's Guide and Reference*.

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.	<i>prompt_name</i>
Vertical bar ()	Mutually exclusive options are separated by a vertical bar (). 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.	{-prompts -prompt prompt_name }
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
<i>Italic</i>	<i>Italic</i> 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.	<i>file_name</i>
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]...

Table 1. Command syntax (continued)

Syntax convention	Description	Example
		An ellipsis used with this syntax indicates that you must specify -x <i>file_name</i> 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 49](#)
- [Starting production on page 50](#)

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 38](#)
- [Configuring your HCL Workload Automation runtime environment on page 39](#)
- [Defining scheduling activities using HCL Workload Automation on page 40](#)
- [Managing production scheduling activities with HCL Workload Automation on page 47](#)

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 23](#)
- Calendar, see [Calendar on page 28](#)
- Prompt, see [Prompt on page 28](#)
- Workstation, see [Workstation on page 29](#)
- Workstation class, see [Workstation class on page 33](#)
- Domain, see [Domain on page 34](#)
- Event rule, see [Event rule on page 37](#)
- Resource, see [Resource on page 37](#)
- Parameter, see [Parameter on page 37](#)
- Variable table, see [Variable table on page 38](#)

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 205](#).

For information about how to define workstations, see [Workstation definition on page 181](#).

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 252](#).

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 Workload Automation on Cloud 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. You can specify the following types of run cycle:

simple

A specific set of user-defined days a job stream is run. A simple run cycle is defined for a specific job stream and cannot be used by other job streams.

daily

A run cycle that specifies that the job stream runs according to a day frequency and type that you set. For example, it might run daily, every three days, or just on working days.

weekly

A run cycle that specifies the days of the week that a job stream is run. For example, a job stream can be specified to run every Monday, Wednesday, and Friday using a weekly run cycle.

monthly

A run cycle that specifies that the job stream runs according to a monthly day or date that you set. For example, it might run every 1st and 2nd day of the month, every two months, or every 1st Monday and 2nd Tuesday of the month, every three months.

yearly

A run cycle that specifies that a job stream runs, for example, yearly or every three years.

offset-based

A run cycle that uses a combination of user-defined periods and offsets. For example, an offset of 3 in a period of 15 days is the third day from the beginning of the period. It is more practical to use offset-based run cycles when the cycle is based on cyclic periods. This term is only used as such in HCL Workload Automation for Z, but the concept applies also to the distributed product.

rule-based

A run cycle that uses rules based on lists of ordinal numbers, types of days, and common calendar intervals (or period names in HCL Workload Automation for Z. For example, the last Thursday 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. In HCL Workload Automation for Z, run cycles can also be based on periods that you define, such as a semester. This term is only used as such in HCL Workload Automation for Z, but the concept applies also to the distributed product. You can also specify a rule to establish when a job stream runs if it falls on a free day.

Any of these run cycle types 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 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 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 2. 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 3. 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 4. 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 4. 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.

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 228](#).

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 227](#).

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.

There are three types of prompts:

global or named

A prompt that is defined in the database as a scheduling object. It is identified by a unique name and can be used by any job or job stream.

local or ad-hoc

A prompt that is defined within a job or job stream definition. It does not have a name, and it is not defined as a scheduling object in the database, therefore it cannot be used by other jobs or job streams.

recovery or abend

A special type of prompt that you define to be used when a job ends abnormally. The response to this prompt determines the outcome of the job or job stream to which the job belongs. A recovery prompt can also be associated to an action and to a special type of job called a *recovery job*.

For information about how to define prompts, see [Prompt definition on page 237](#)

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 33](#), and for domains, see [Domain on page 34](#).

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 workstation and registered in the HCL Workload Automation database as **d-pool**.

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 181](#).

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 34](#)

For more information about how to define workstation classes, see [Workstation class definition on page 201](#).

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.

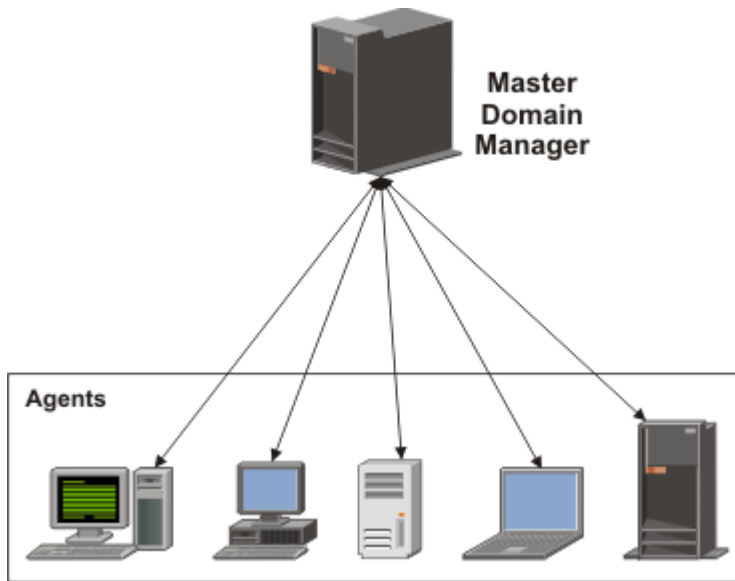
All networks have a master domain where the domain manager is the master domain manager. It maintains the database of all scheduling objects in the domain and the central configuration files. The master domain manager generates the plan and creates and distributes the Symphony file. In addition, logs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

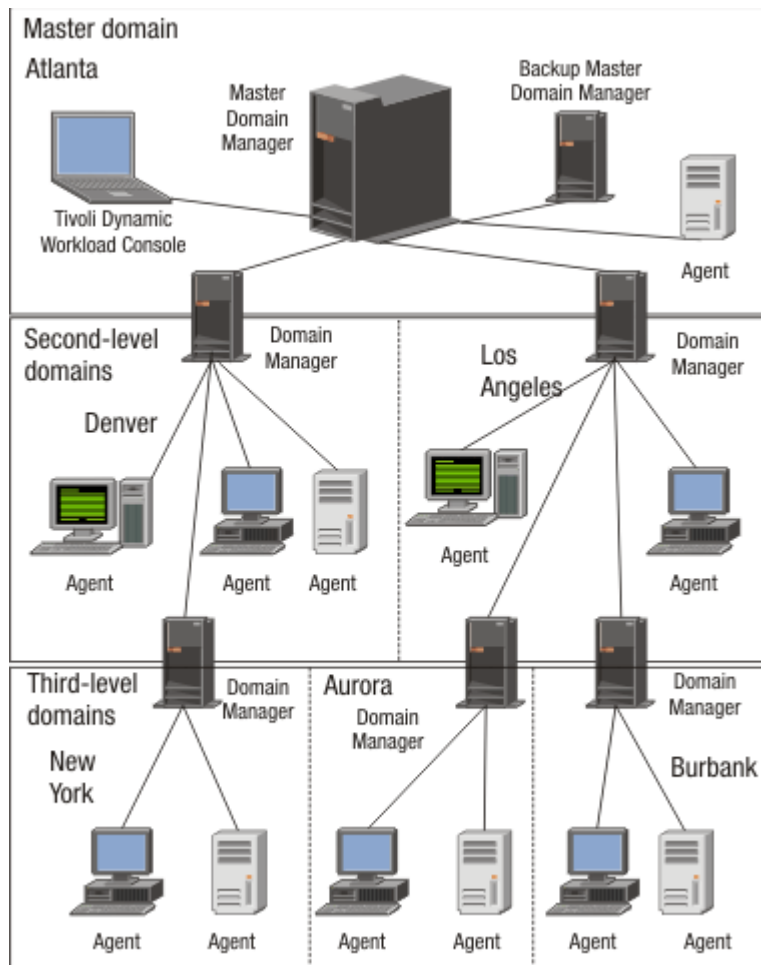
A single domain network consists of a master domain manager and any number of agents. The following shows an example of a single domain network. A single domain network is well suited to companies that have few locations and business functions. All communication in the network is routed through the master 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 33](#)

For information about how to define domains, see [Domain definition on page 203](#).

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 163](#).

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 239](#).

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 230](#).

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 235](#).

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 1058](#).

For information about workstations, see [Workstation definition on page 181](#).

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 83](#).

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 81](#).

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 177](#).

For more information about report utility commands, refer to [Getting reports and statistics on page 958](#).

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 jobs and job streams are processed by setting one or more rules from the following:

Defining dependencies

About this task

A **dependency** is a prerequisite that must be satisfied before processing can proceed. You can define dependencies for both jobs and job streams to ensure the correct order of processing. 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 281](#).
- **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. Ensure that all the components in the IBM Workload Scheduler environment are at version 9.3 Fix Pack 1, or later. 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 299](#).
- **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 310](#).
- **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 237](#) and [prompt on page 314](#).

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 1044](#) and [join on page 289](#).

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 for Z engines (controller) or another HCL Workload Automation engines (master domain manager). Two types of dependencies implement such requirement:

Internetwork dependency

It is a simple and distributed based implementation. 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 1030](#).

Cross dependency

It is a more comprehensive and complete implementation. 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 1058](#).

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 542](#) and [priority on page 313](#).

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 545](#), and [LIMIT SCHED on page 547](#).

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 239](#).

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 528](#).

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 552](#) command to identify the job's successors and the [RERUNSUCCE on page 566](#) 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 552](#) and [RERUNSUCCE on page 566](#) 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 [RERUNSUCCE on page 566](#) and [LISTSUCC on page 552](#).

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 562](#) 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 205](#).

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.

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 155](#).

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 177](#) and [Managing objects in the database - composer on page 367](#).

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 478](#).

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. For more information and to learn how to access the documentation of the API and plug-ins, refer to *Developer's Guide: Extending HCL Workload Automation*.

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 111](#).

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 81](#).

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 203](#).

◦ 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 181](#). 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 222](#).

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 227](#).

6. Optionally define parameters, prompts, and resources

For additional information refer to [Variable and parameter definition on page 230](#), [Prompt definition on page 237](#), and [Resource definition on page 239](#).

7. Define jobs and job streams

For additional information refer to [Job on page 1111](#), and to [Job stream definition on page 252](#).

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 1044](#) and [join on page 289](#). 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 132](#).

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 545](#) 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 478](#) 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.

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 58](#). 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 548](#)) and are stopped by unlink requests (see [UNLINK on page 655](#)) 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 181](#).

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 68](#). 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 545](#)).

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 1111](#).

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 57](#) shows the process tree on HCL Workload Automation workstations, other than standard agents, installed on UNIX®:

Figure 3. Process tree in UNIX®

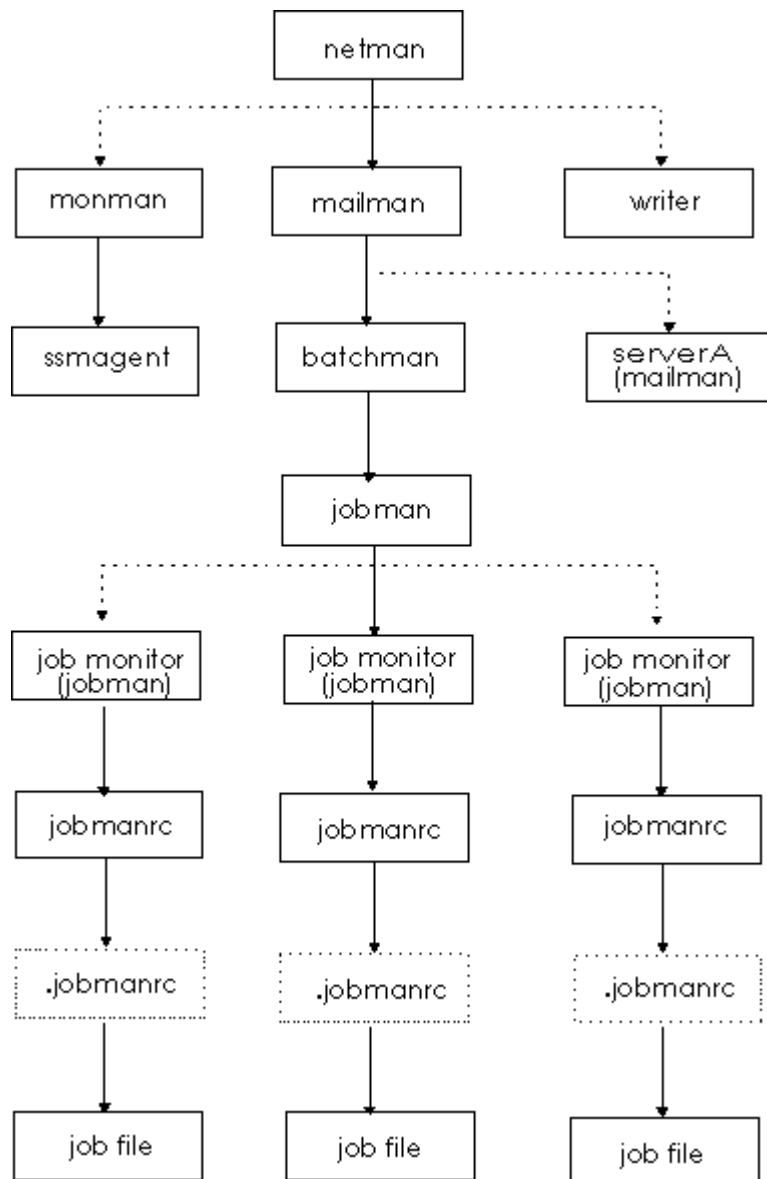
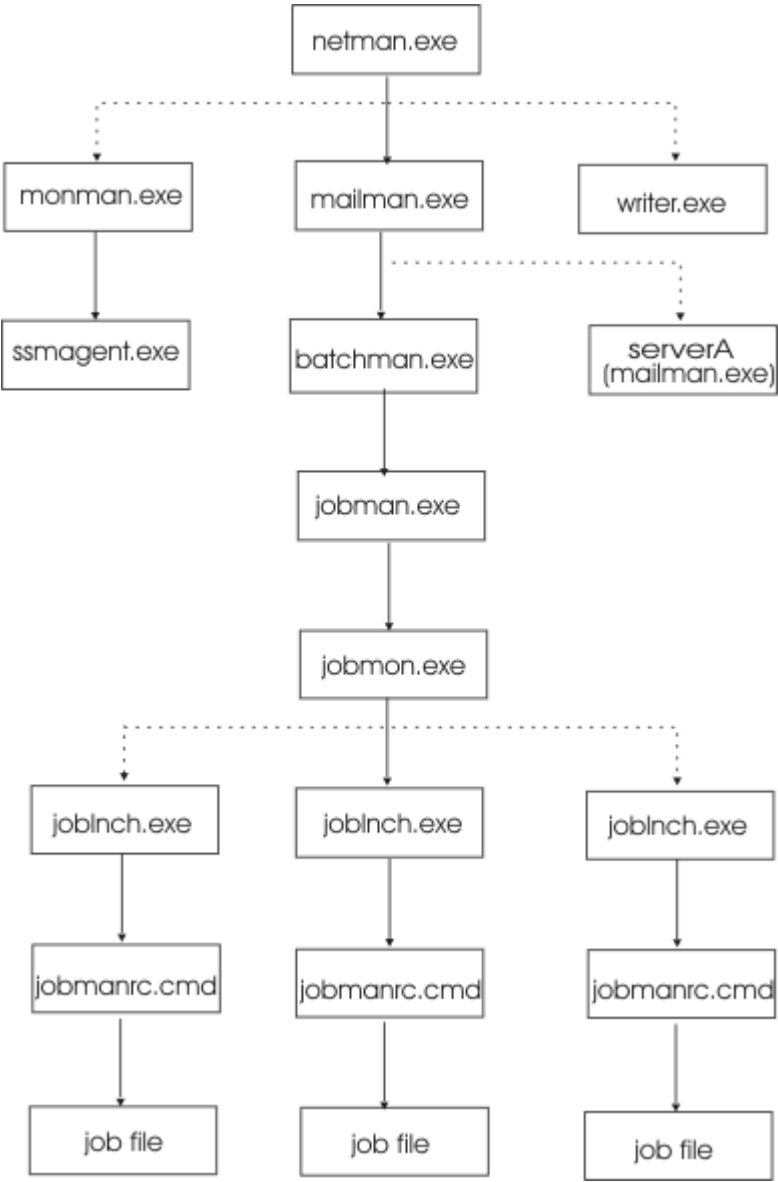


Figure 4: [Process tree in Windows on page 58](#) shows the process tree on HCL Workload Automation workstations, other than standard agents, installed on Windows®:

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 5: Starting and stopping HCL Workload Automation on a workstation on page 59](#) 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 5. 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 5. 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 <i>TWS_user</i> or root user only.	
Stop the agent locally	<code>./ShutDownLwa.sh</code>	<code>shutdownlwa</code>
	 Note: can be run by <i>TWS_user</i> 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 920](#) for more information about the **Startup** utility command.

Refer to [shutdown on page 919](#) 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 624](#) for more information about the **conman start** command.

Refer to [STOP on page 629](#) for more information about the **conman stop** command.

Refer to [SHUTDOWN on page 623](#) for more information about the **conman shutdown** command.

Refer to [STARTAPPSERVER on page 626](#) for more information about the **conman startappserver** command.

Refer to [STOPAPPSERVER on page 630](#) for more information about the **conman stopappserver** command.

Refer to [STARTMON on page 627](#) for more information about the **conman startmon** command.

Refer to [STOPMON on page 632](#) 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 6. Starting and stopping the agent

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

For more information about stopping and starting the agent, see [ShutDownLwa - Stop the agent on page 919](#) and [StartUpLwa - Start the agent on page 921](#).

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

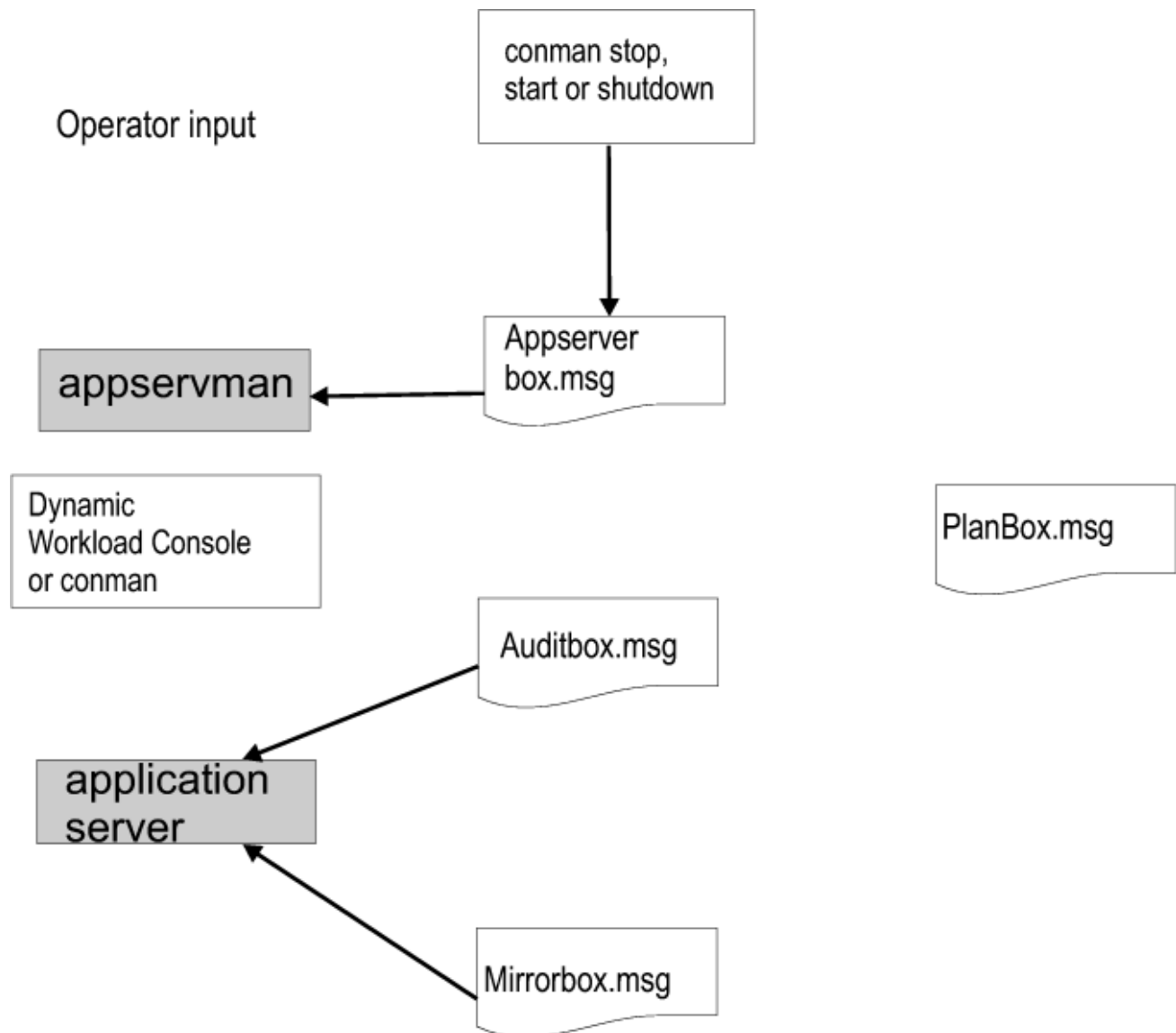
This message file is written by the **batchman** process and read by the engine.

Server.msg

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

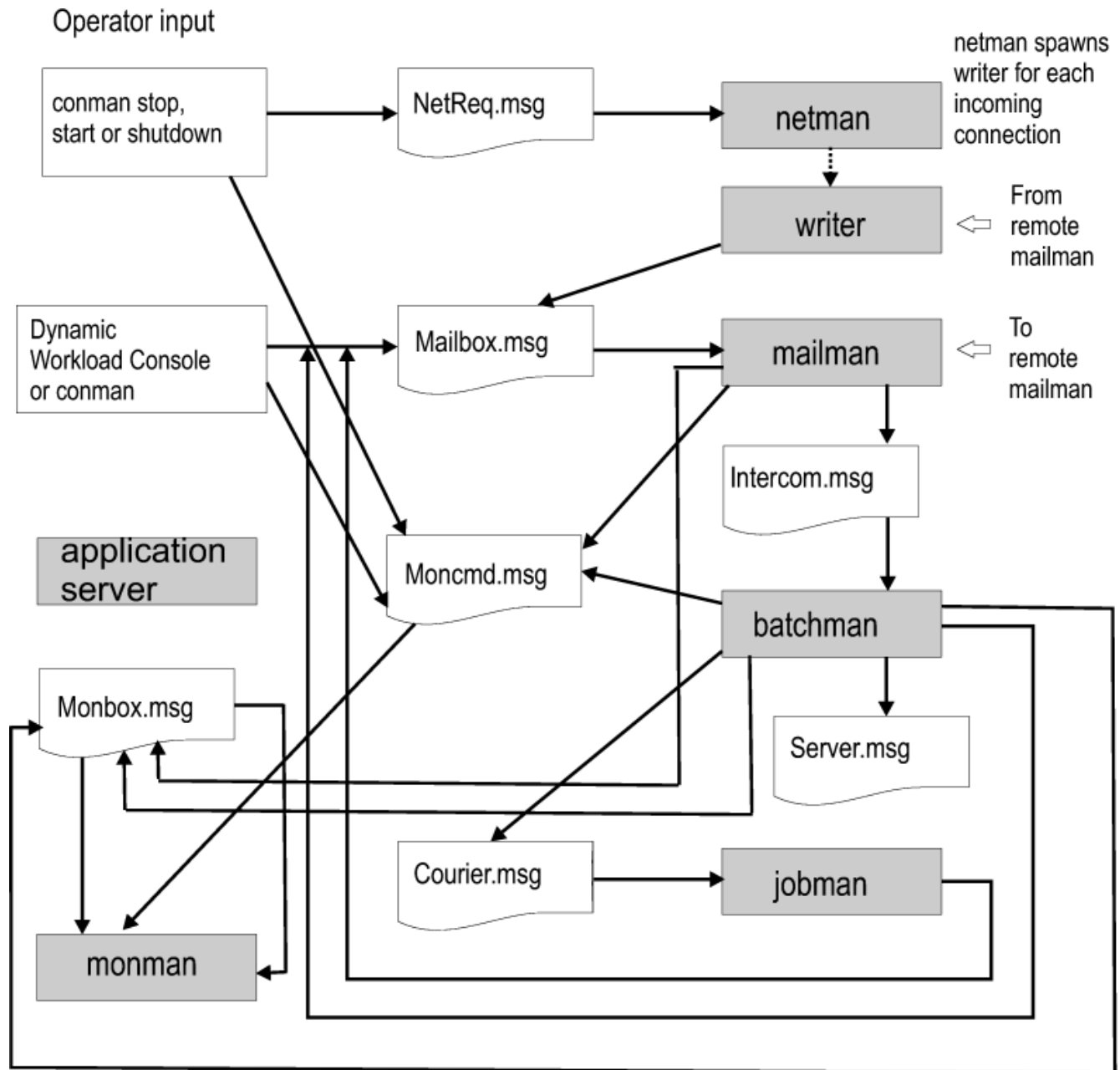
This figure illustrates the inter-process communication on the master domain manager.

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 **evtsiz** utility (see [evtsiz](#) on page 888).

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 181 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 181](#), and by the [LINK on page 548](#) and [UNLINK on page 655](#) 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 68](#)
- [Environment variables exported by jobman on page 69](#)
- [Customizing job processing on a UNIX workstation - jobmanrc on page 72](#)
- [Customizing job processing for a user on UNIX workstations - .jobmanrc on page 75](#)
- [Customizing job processing on a Windows workstation - jobmanrc.cmd on page 77](#)
- [Customizing job processing on a Windows workstation - djobmanrc.cmd on page 79](#)

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 7: Job environment variables for Windows on page 69](#) are set locally on the workstation and exported by **jobman** on Windows® operating systems:

Table 7. Job environment variables for Windows®

Variable Name	Value
COMPUTERNAME	The value of the <i>COMPUTERNAME</i> set in the user environment.
HOME	The path where the HCL Workload Automation instance was installed.
HOMEDRIVE	The value of the <i>HOMEDRIVE</i> set in the user environment.
HOMEPATH	The value of the <i>HOMEPATH</i> set in the user environment.
LANG	The value of the <i>LANG</i> 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 <i>SYSTEMDRIVE</i> set in the user environment.
SystemRoot	The value of the <i>SYSTEMROOT</i> set in the user environment.
TEMP	The value of the <i>TEMP</i> 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 <i>TMP</i> set in the user environment. If not specified, its value is set to c:\temp.
TMPDIR	The value of the <i>TMPDIR</i> 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 <i>UNISON_DIR</i> set in the user environment.
UNISON_EXEC_PATH	The <code>jobmanrc</code> fully qualified path.
UNISONHOME	The path where the HCL Workload Automation instance was installed.
UNISON_HOST	The name of the host CPU.

Table 7. Job environment variables for Windows® (continued)

Variable Name	Value
UNISON_JOB	The absolute job identifier: <code>workstation#sched_id.job</code> .
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.
UNISON_SCHED_DATE	The HCL Workload Automation production date (yymmdd) reported in the header of the <code>Symphony</code> file.
UNISON_SCHED_ID	The <i>jobstreamID</i> 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 <code>Symphony</code> record number of the launched job.
USERDOMAIN	The value of the <i>USERDOMAIN</i> set in the user environment.
USERNAME	The value of the <i>USERNAME</i> set in the user environment.
USERPROFILE	The value of the <i>USERPROFILE</i> set in the user environment.

The variables listed in [Table 8: Job environment variables for UNIX on page 70](#) are set locally on the workstation and exported by **jobman** on UNIX® operating systems:

Table 8. Job environment variables for UNIX®

Variable Name	Value
HOME	The home directory of the user.
LANG	The value of the <i>LANG</i> set in the user environment.
LD_LIBRARY_PATH	The value of the <i>LD_LIBRARY_PATH</i> set in the user environment.
LD_RUN_PATH	The value of the <i>LD_RUN_PATH</i> set in the user environment.
LOGNAME	The login user name.
MAESTRO_OUT	The setting for output style for long object names.
PUT_STYLE	

Table 8. Job environment variables for UNIX® (continued)

Variable Name	Value
PATH	/bin:/usr/bin
TIVOLI_JOB_DA TE	The scheduled date for a job.
TWS_PROMOTE D_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 <i>TWS_TISDIR</i> set in the user environment.
TZ	The time zone set.
UNISON_CPU	The name of this workstation.
UNISON_DIR	The value of the <i>UNISON_DIR</i> set in the user environment.
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: workstation#sched_id.job.
UNISON_JOBN UM	The job number.
UNISON_MAST ER	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.
UNISON_SCHED _ID	The <i>jobstreamID</i> 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_STDLI ST	The path name of the standard list file of the job.

Table 8. Job environment variables for UNIX® (continued)

Variable Name	Value
UNISON_SYM	The <code>symphony</code> 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 9: Variables defined by default in the jobmanrc file on page 73](#) describes the *jobmanrc* variables.

Table 9. Variables defined by default in the jobmanrc file

Variable Name	Value
UNISON_JCL	The path name of the job's script file.
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 <i>\$UNISON_JCL</i> 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 75 for more information. If set to no, the presence of a local configuration script is ignored, and <i>\$UNISON_JCL</i> 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: <pre>cpu#sched.job jcl-file failed with exit-code Please review standard-list-filename</pre> You can change the wording of the message or translate the message into another language. For an explanation of how

Table 9. Variables defined by default in the jobmanrc file (continued)

Variable Name	Value
	to do this, see Customizing the MAIL_ON_ABEND section of jobmanrc on page 74 .
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 <code>/bin/sh</code> is used to run the local configuration script or <code>\$UNISON_JCL</code>. Commands are echoed to the job's standard list file. If set to user, the local configuration script or <code>\$UNISON_JCL</code> is run by the user's login shell (<code>\$UNISON_SHELL</code>). Commands are echoed to the job's standard list file. If set to script, the local configuration script or <code>\$UNISON_JCL</code> is run directly, and commands are not echoed unless the local configuration script or <code>\$UNISON_JCL</code> 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 <code>MAIL_ON_ABEND</code> 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 9: Variables defined by default in the jobmanrc file on page 73](#)).
- 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 9: Variables defined by default in the jobmanrc file on page 73](#)).
- `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 9: Variables defined by default in the jobmanrc file on page 73](#)), 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 "tellog This is a database error - page the dba"
elif [ $RETVAL -ge 100 ]
then
conman "tellog 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 10: Variables defined by default in the jobmanrc.cmd file on page 78](#) describes the `jobmanrc.cmd` variables.

Table 10. Variables defined by default in the jobmanrc.cmd file

Variable Name	Value
HOME	The path to the <i>TWS_home</i> directory
POSIXHOME	The path to the <i>TWS_home</i> directory in a POSIX compliant 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 <i>email_ID</i> 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 78.

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
set RETVAL=%ERRORLEVEL%
: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 it based 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 83](#)
- [Customizing plan management using global options on page 104](#)
- [Creating and extending the production plan on page 108](#)
- [Planman command line on page 111](#)
- [Starting production plan processing on page 131](#)
- [Automating production plan processing on page 132](#)

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 108](#).

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 54](#).



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 301](#), [at on page 263](#) and [schedtime on page 315](#).

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 586](#).

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 500](#).

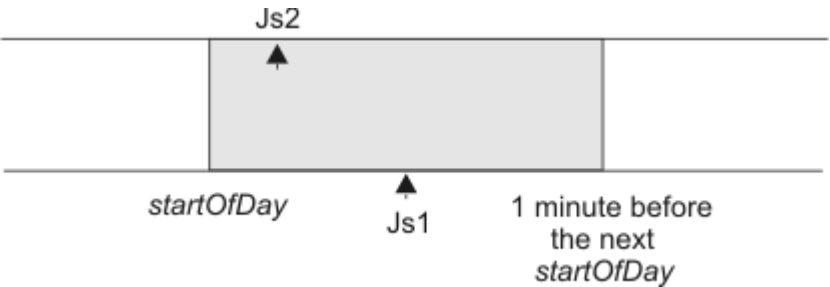
Managing external follows dependencies for jobs and job streams

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 87](#) 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

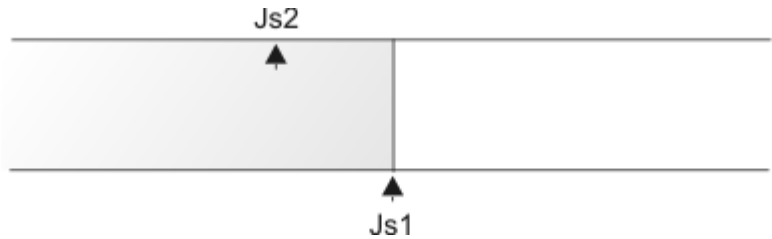
The job stream **Js1** is not launched until the job stream instance of **Js2** on the workstation **wk1** completes successfully.

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 88](#) 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.

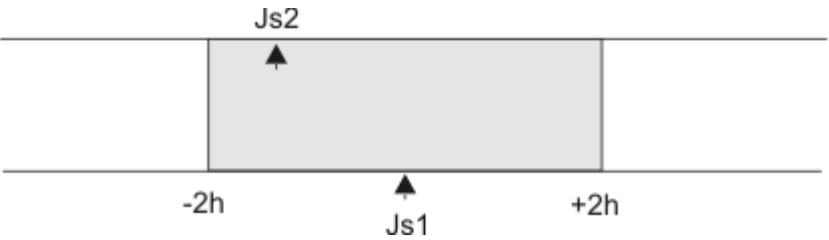
schedule Js2	schedule Js1
on Th	on Fr
at 0700	at 1000
:job2	follows wk1#Js2 previous
end	:job1
	end

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

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 88](#) 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

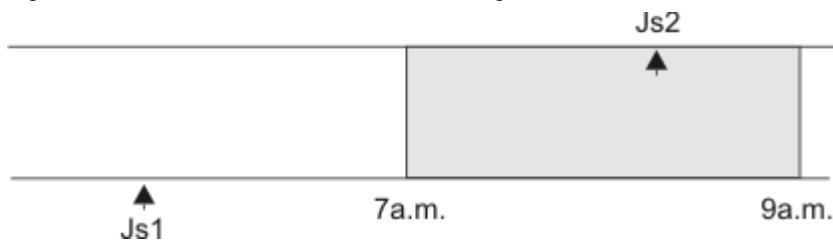
The job stream `Js1` is not launched until the job stream instance of `Js2` on the workstation `wk1` that runs in the 08:00 to 12:00 time frame completes successfully.

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 89 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` is 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 90](#) shows in bold the instances of `job1` the successor job or job stream is dependent on.

Figure 11. Closest preceding predecessor job



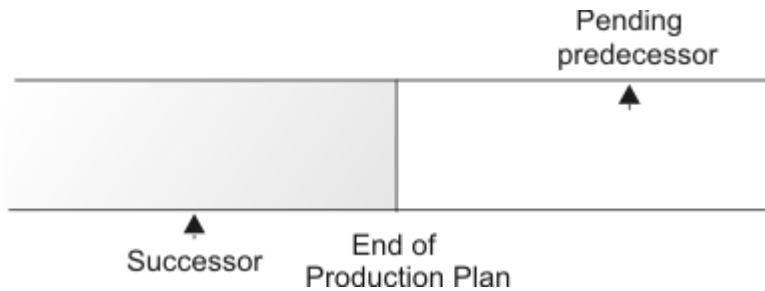
External follows dependencies are identified between jobs and job streams stored in the database whose instances are added to the preproduction plan when the preproduction plan is automatically created or extended. Job and job stream instances submitted in production from the **conman** command line 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 91](#) 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 586 and `conman showschedules` on page 617 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 [0] in the `dependencies` column in the output of the `conman showjobs` on page 586 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

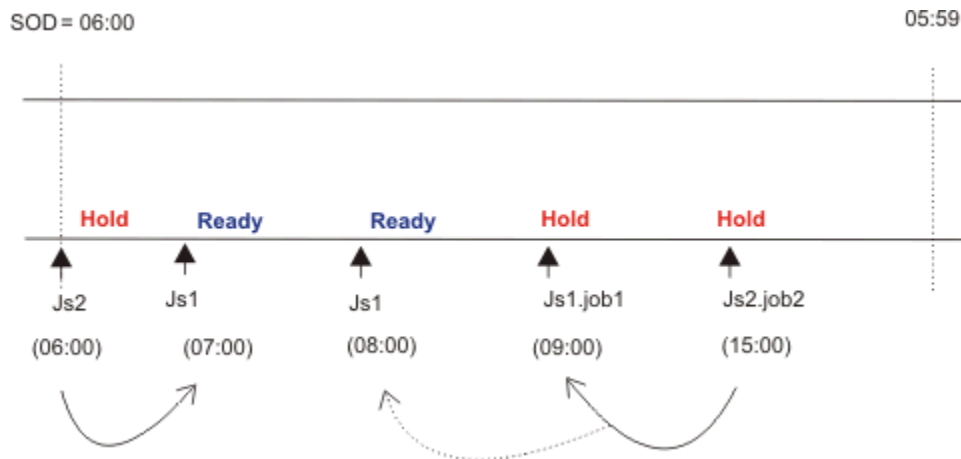
SCHEDULE MY_MASTER#JS2
ON RUNCYCLE RULE2 "FREQ=DAILY;"
FOLLOWS MY_MASTER#JS1.@ SAMEDAY
:
MY_MASTER#JOB2
AT 1500
END

```

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

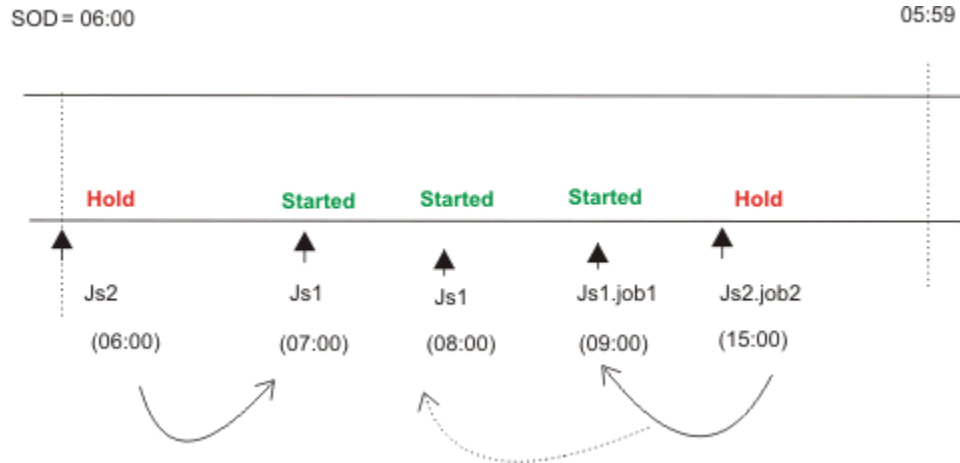
1. On Thursdays, the instance of `Js2` scheduled at 06:00 depends on the instance of `Js1` scheduled to run at 07:00. On any other day of the week, `Js2` has a dependency on the instance of `Js1` scheduled at 08:00. [Figure 13: Sameday matching criteria - Step 1: at Start of Day \(SOD\) on a Thursday on page 92](#) 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



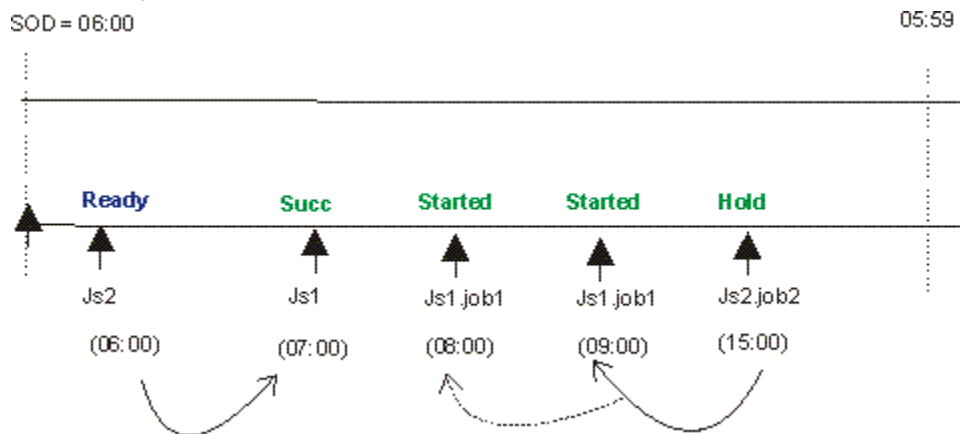
2. At 09:00, `Js1.job1` starts and `Js1` changes status. `Js2.job2` is held until its scheduled time. [Figure 14: Sameday matching criteria - Step 2: at 9:00 on page 93](#) shows the status of the job streams in the plan at 09:00.

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 93](#) 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

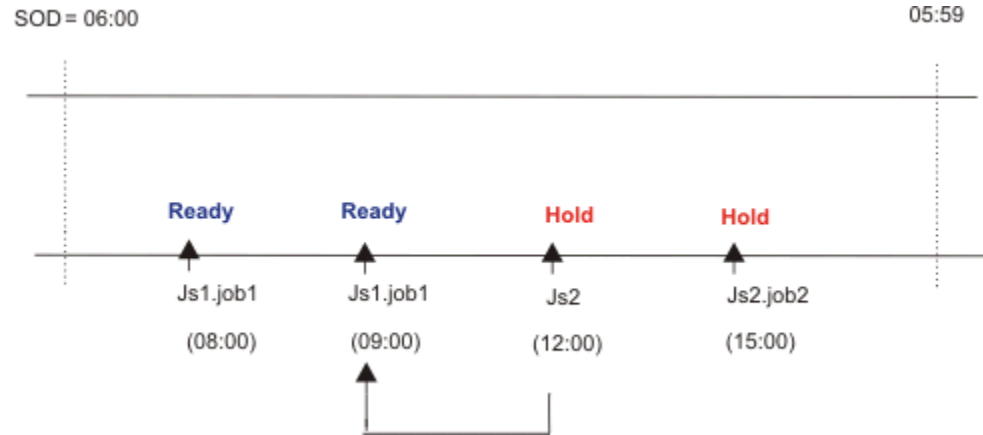
SCHEDULE MY_MASTER#JS2
ON RUNCYCLE RULE1 "FREQ=DAILY;"
(AT 1200)
```

```
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:

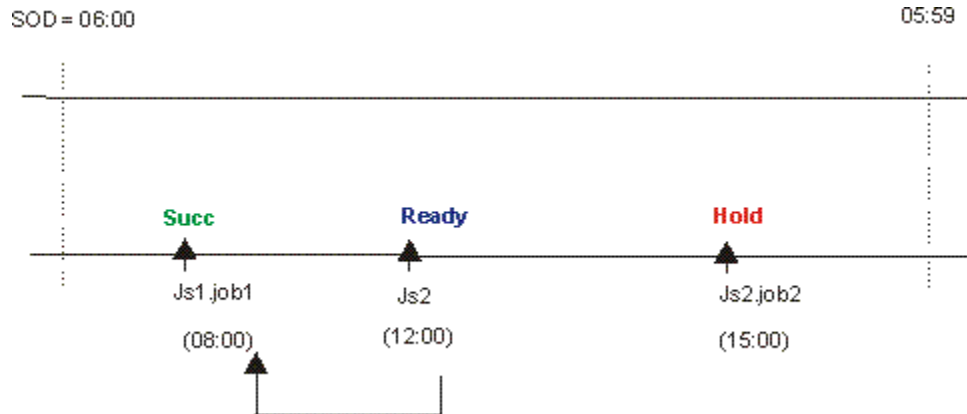
- 1. Before 12:00 on Thursdays and Fridays, there are two instances of `Js1.Job1`. Job stream `Js2` has a dependency on the instance of `Js1.Job1` that is scheduled to run at 09:00, because it is the closest preceding in terms of time. [Figure 16: Closest preceding matching criteria - Step 1: before 08:00 on page 94](#) 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



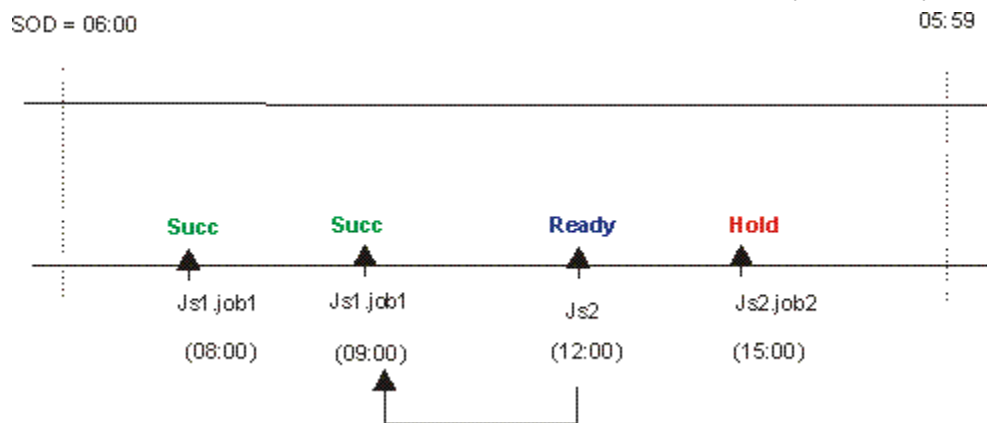
- 2. On any other day of the week, the only instance of `Js1.Job1` in plan, is the one scheduled to run at 08:00. In this case, `Js2` has a dependency on this instance. When `Job1` completes successfully, the status of `Js2` becomes **Ready**. [Figure 17: Closest preceding matching criteria - Step 2: at 08:00 on weekdays except Thursdays and Fridays on page 95](#) 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



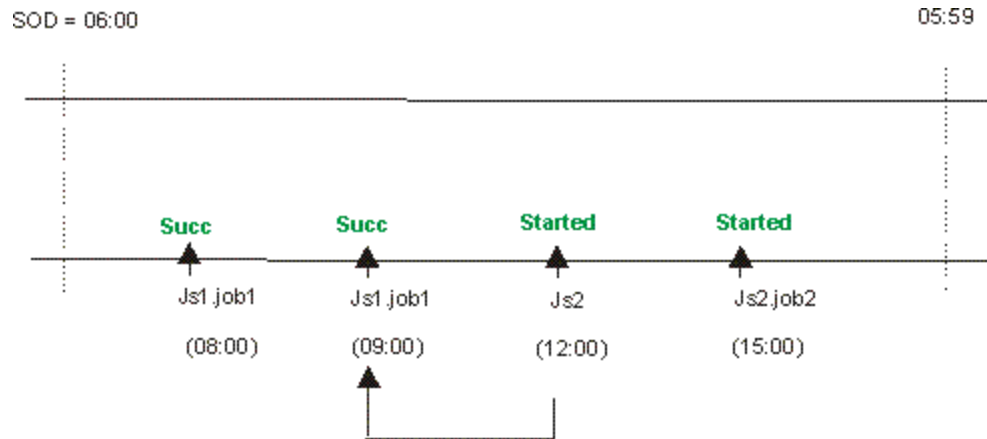
3. On Thursdays and Fridays at 09:00, the second instance of Js1.job1 completes successfully. Job stream Js2 changes to **Ready**. Js2.job2 is held until its scheduled start time. [Figure 18: Closest preceding matching criteria - Step 3: at 09:00 on Thursdays and Fridays on page 95](#) shows the status 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 96 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

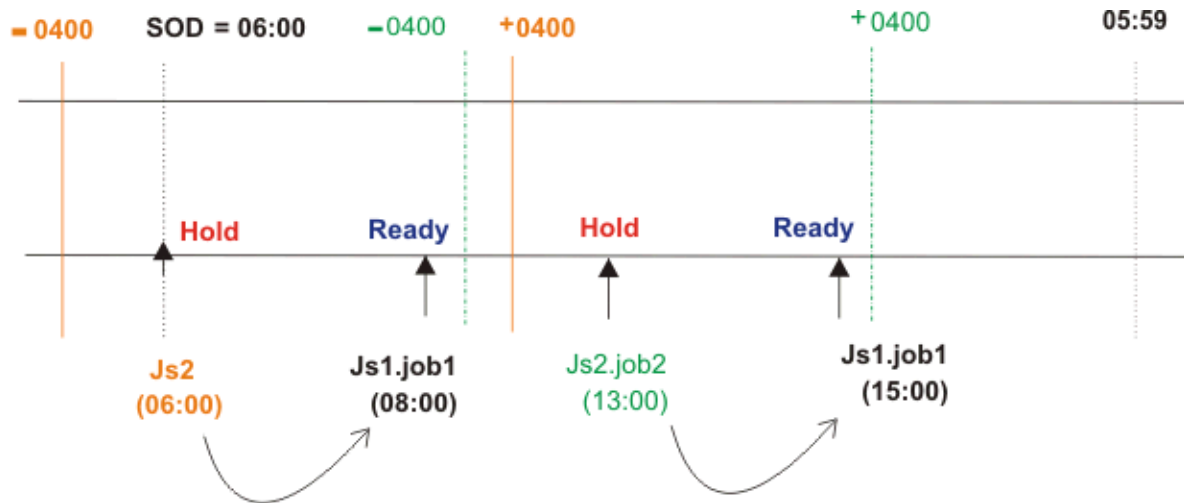
SCHEDULE MY_MASTER#JS2
ON RUNCYCLE RULE3 "FREQ=WEEKLY;BYDAY=TH"
ON RUNCYCLE RULE2 "FREQ=DAILY;"
(AT 1300)
FOLLOWS MY_MASTER#JS1.@
RELATIVE FROM -0400 TO 0400
:
MY_MASTER#JOB2
AT 1300
END
```

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 98 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
```

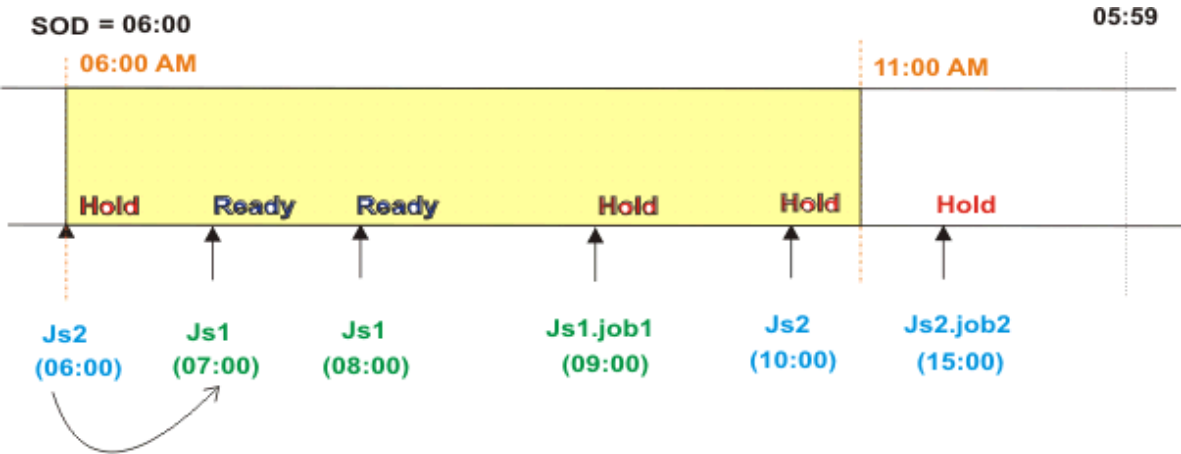
```
AT 1300
END
```

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 11/13/09).@
              JOB2    HOLD 10(15:00)
```

Figure 21: Absolute interval matching criteria - at start of day on Thursday on page 99 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 108](#).

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 265](#).
- 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 125](#).
- 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 11: Carry forward global options settings on page 100](#) 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 11. 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 11. 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 12: Resulting carry forward settings on page 101 shows the result of the carry forward setting based on how the **enCarryForward** global option and the **stageman -carryforward** keywords are set.

Table 12. 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 104](#) and [maxLen on page 104](#) 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 111](#).

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 111](#).

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 the *startOfDay* of the day specified in its scheduled time 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 586](#). 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 streams, 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 1012](#) 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 54](#).

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 81](#)). 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 111](#).

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 125](#). **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 128](#). **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 109](#).



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 `hhmm`, 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 incompleting 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 81](#). 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 113](#)
- [Creating an intermediate plan for a plan extension on page 114](#)
- [Retrieving the production plan information on page 115](#)
- [Creating a trial plan on page 116](#)
- [Creating a trial plan of a production plan extension on page 118](#)
- [Creating a forecast plan on page 119](#)
- [Unlocking the production plan on page 121](#)
- [Removing the preproduction plan on page 122](#)
- [Resetting the production plan on page 121](#)
- [Replicating plan data in the database on page 123](#)
- [Monitoring the replication of plan data in the database on page 124](#)

You can also use **planman** to deploy [scheduling event rules on page 155](#). The command is explained in: [Deploying rules on page 120](#). 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 111](#).

-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 111](#).

-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 *hhmm*, 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 111](#).



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 10.2.5
Licensed Materials - Property of IBM* and HCL**
5698-WSH
(c) Copyright IBM Corp. 1998, 2016 All rights reserved.
(c) Copyright HCL Technologies Ltd. 2016, 2025 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*] **crtrial** *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 111](#).

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 **Tfilename**. 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 `hh:mm`, where `hh` 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 111](#).

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 **Tfilename**. 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 *hh:mm*, where *hh* 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 111](#).

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 **Ffilename**. 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 155](#). 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 111](#).

-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 is 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 111](#).



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 111](#).

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 **dbbrunstats** before the **JnextPlan** script. See the *Administration Guide* for details on **dbbrunstats**.

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 111](#).



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 132](#) 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 111](#).

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 100](#) 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 550](#) and [SETSYM on page 571](#). If neither **-log** nor **-nolog** keywords are specified, HCL Workload Automation archives the old production plan using the following naming convention:

```
Myyyyymmddhhtt
```

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='datecal 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 500](#).

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 `twsl1"twsl`, write it as `"twsl1\"twsl"`.

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 81](#) 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 108](#) 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` file 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>/twc_env.csh`
- UNIX®: on Korn shells launch `<TWS_home>/twc_env.sh`
- From a Windows® command line: launch `TWS_home\twc_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 135](#)
- [Planning critical jobs on page 139](#)
- [Processing and monitoring critical jobs on page 141](#)
- [Workload service assurance scenario on page 143](#)

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 13. Workload service assurance global options (continued)

Option	Description
<code>approachingLateOffset al</code>	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 <code>approachingLateOffset</code> 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.
<code>deadlineOffset do</code>	In general, a deadline should be specified for a job flagged as <code>critical</code>. If it is not, the scheduler uses the deadline defined for the job stream. The <code>deadlineOffset</code> 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 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 14: Workload service assurance local options on page 138](#) 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 14. Workload service assurance local options

Option	Description
<code>jm promoted nice</code>	Sets the <code>nice</code> 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 <code>jm nice</code> 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 <code>jm promoted nice</code> is set to -4 and <code>jm nice</code> to -2, the critical job submitted by user root would have a priority of -6.
<code>jm promoted priority</code>	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

Table 14. Workload service assurance local options (continued)

Option	Description
	• 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.

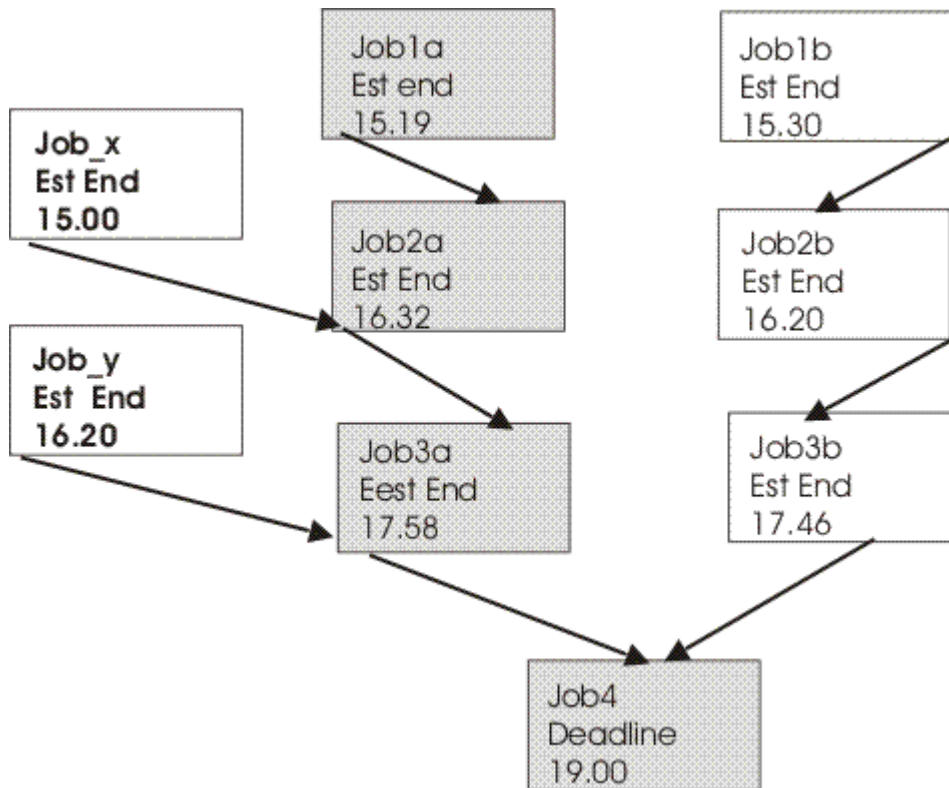
The timing of promotions is controlled by the global option `promotionoffset`. Promoted jobs are selected for submission after jobs that have priorities of "high" and "go", but before all other jobs. Prioritizing of operating system resources is controlled by the local options `jm promoted nice` (UNIX 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 142](#) 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.

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.

Starting from IBM Workload Scheduler version 8.5, what were called global parameters in previous versions, are now called variables. 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.

Migrating global parameters from previous versions

Considerations when migrating global parameters from previous versions

When you upgrade from versions earlier than 8.5, the global parameter definitions, now called variable definitions, that you have in the database, are automatically migrated to the default variable table called MAIN_TABLE. After the upgrade:

- All variables are preceded by the default table name. For example, after you migrate, the *REP_PATH* variable acquires the following name:

```
MAIN_TABLE.REP_PATH
```

When you request a list of variables you get *variabletable.variablename* pairs to easily identify to which variable table the variable belongs.

- Your workload is resolved in the same way as before the migration because any HCL Workload Automation object containing variables refers to the MAIN_TABLE for variable resolution.
- For every user section that includes the `parameter` keyword, the following row is added in the security file:

```
vartable name=@ access=add,delete,display,modify,list,use,unlock
```

For details about the upgrade process, see *HCL Workload Automation: Planning and Installation*.

When upgrading from version 8.3 or later, do not modify variables until you migrate the master domain manager and all its backup masters because in this transition phase you have two different versions of the database. If you must add or modify variables during this transition phase, make sure you make the change in both version 8.3 or 8.4 and version 8.5 of the master domain managers.

Local parameters that were created and managed with the **parms** utility command in the local parameter database on the workstations, work in the same way as before.

In V8.3 and V8.4 the parameters are stored in the MDL.VAR_VARIABLES table of the database. After you upgrade to V8.5 or later, the same parameters are stored in the MAIN_TABLE contained in the MDL.VAR_VARIABLES2 database table. If you started to migrate your environment, but did not yet complete the migration, and the master domain manager is V8.3 or V8.4 and the backup master domain manager is V8.5 or later, or viceversa, the V8.3 or V8.4 master domain manager does not recognize the table MDL.VAR_VARIABLES2 so if you change a parameter in the V8.3 or V8.4 master domain manager this change is stored in the old table (MDL.VAR_VARIABLES). If you change the parameter in the V8.5 master domain manager the change is stored in the MDL.VAR_VARIABLES2 table.

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	05/07/2021	-
VT_1		05/07/2021	-
VT_2		05/07/2021	-
VARIABLE3		05/07/2021	-
V4	05/07/2021	-	
VT5		05/07/2021	-
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:

```
variable 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 15. The relationship between variable tables and their enclosed variables in the HCL Workload Automation security file

Defined access to variable	Allowed action on enclosed variables
	add
modify	delete
	modify
display	display
unlock	unlock

Starting with version 8.5, the `parameter` keyword in the security file applies to local parameters only.

See the *HCL Workload Automation Administration Guide* for details about the security file.

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 230](#) 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 149](#).
4. In the default variable table, but only when the variables are specified in the `^variablename^` on page 231 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 149](#).
4. In the default variable table, but only when the variables are specified in the `^variablename^` on page 231 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 231 format in the job definition. Variables specified in the `${variablename}` format are not resolved.
2. In the workstation. See [Workstation considered for variable resolution on page 149](#).
3. In the default variable table, but only when the variables are specified in the `^variablename^` on page 231 format in the job definition. Variables specified in the `${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** 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** 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 153](#) 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*.

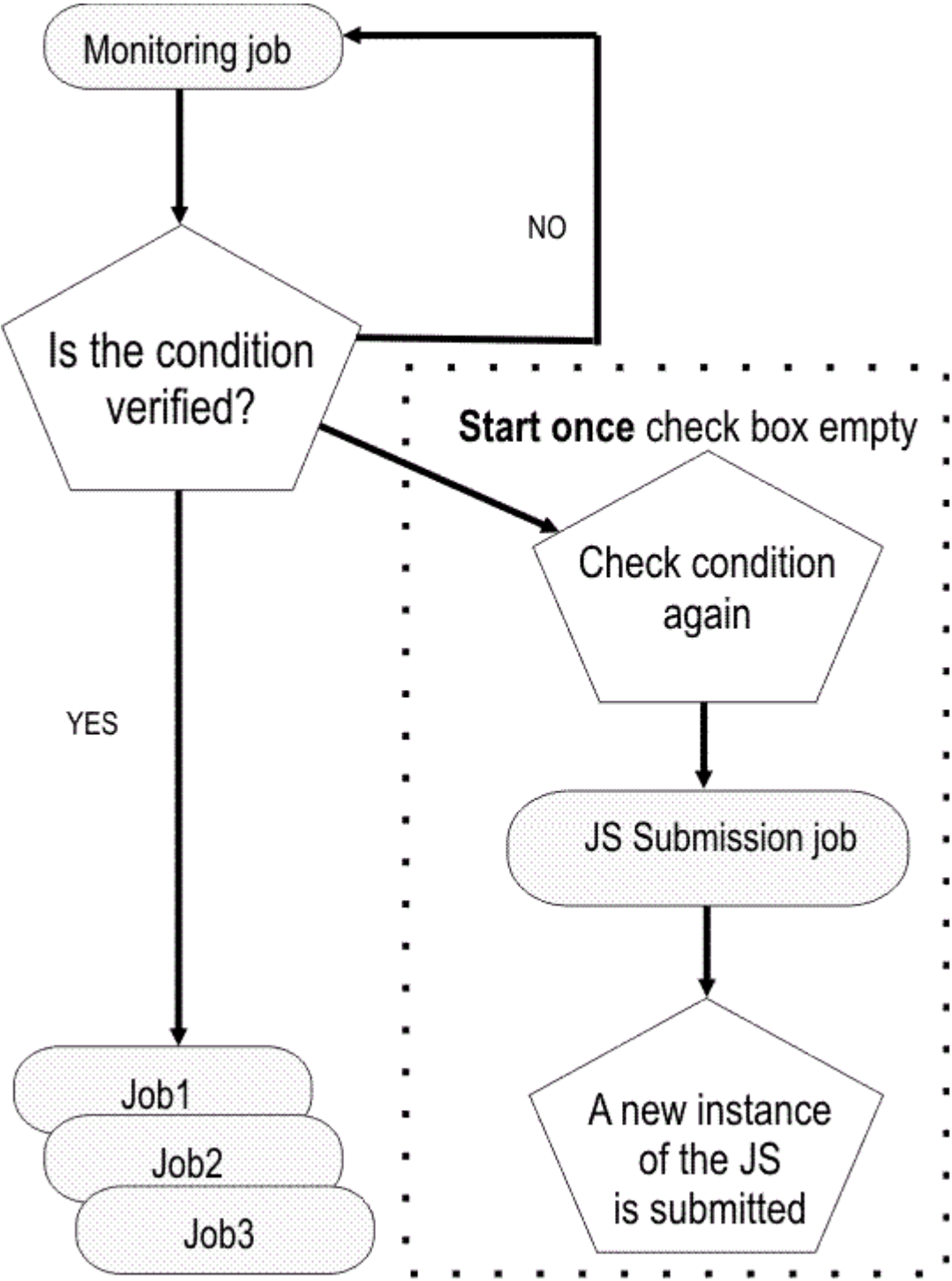


Note: The default name for the pool workstation, MASTERAGENTS, can be modified using the optman global option `resubmitJobName`. See the detailed description of the global options in the *Administration Guide* for details about this option.

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 152](#) 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. For more information about iterative workflows, see the section about automation of iterative workflows in *Overview*.

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 252](#).

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 166](#).

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 163](#) or [non-draft on page 163](#).
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 16. 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.

Table 16. conman commands for managing monitoring engines (continued)

Command	Purpose
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 <code>monman</code> on an agent. Can be issued from a different agent.
stopmon	Stops <code>monman</code> 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 17. 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 651` command to make it the active event processing server.



Note: If you set the ignore keyword on the [workstation definition on page 181](#) 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 18: Interfaces and commands for managing event-driven workload automation on page 161](#) summarizes the ones you need:

Table 18. 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 (<code>deploymentFrequency</code>). Modified definitions are deployed in the HCL Workload Automation domain • The EIF port number where the event processing server receives events (<code>eventProcessorEIFPort</code>, or <code>eventProcessorEIFSSLPort</code> when SSL-protected). • Management of the cleanup policies of rule instance, action run, and message log data (<code>logCleanupFrequency</code>). • SMTP server properties if you deploy rules implementing actions that send emails via an SMTP server (<code>smtpServerName</code>, <code>smtpServerPort</code>, <code>smtpUseAuthentication</code>, <code>smtpUserName</code>, <code>smtpUserPassword</code>, <code>smtpUseSSL</code>, <code>smtpUseTLS</code>). • The possibility to disable the event rule management mechanism (<code>enEventDrivenWorkloadAutomation</code>) which is installed by default with the product. See the <i>Administration Guide</i> 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 330 to learn how to define event rules. See Managing objects in the database - composer on page 367 for command reference.
Dynamic Workload Console	Have a graphical user interface to:

Table 18. 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 16: conman commands for managing monitoring engines on page 158 and Table 17: conman commands for managing the event processing server on page 160 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 16: conman commands for managing monitoring engines on page 158 and Table 17: conman commands for managing the event processing server on page 160. See Managing objects in the plan - conman on page 478 for command reference.
utility commands	Create custom event definitions and manually send custom events to the event processing server. See evtdef on page 883 and sendevent on page 915 for details on these commands.
planman	Manually deploy new and changed rules. See Deploying rules on page 120 for details.
Security file	Set security authorizations to manage event rules, events, actions, and their instances. See the <i>HCL Workload Automation Administration Guide</i> 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/`. In the Swagger, you can find the REST API V2s by entering `WA_API3_v2.json` in the search bar. 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 330](#), 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 173](#) 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 156](#).

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="SCENARIO02_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>
    </action>
  </eventRule>
</eventRuleSet>
```

```

</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 `twsooper@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>twsooper@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 RUNCMD.FM.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="TWSPProcessMonitor">
      <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>
      RUNCMD.FM.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 Message 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=%{PRODUCTALERT1.WORKSTATION} MESSAGE=%{PRODUCTALERT1.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

```
twc_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

```
twc_install_dir\TWS\ssm\Config\init.cfg
```

where `twc_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 333 with shorter than 24 hours activity time on page 334 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 1087](#), 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 883](#) 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 915](#) 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 883](#) 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 915](#) 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 422](#) and [UNLOCK on page 447](#).

You can use short and long keywords when issuing commands from **composer**, as shown in [Table 19: List of supported scheduling object keywords on page 178](#) and [Table 20: List of supported security object keywords on page 179](#). 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 19. 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 20. 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 21: List of reserved words when defining jobs and job streams on page 179](#) when defining jobs and job streams:

Table 21. List of reserved words when defining jobs and job streams

[illegible]

**Table 21. List of reserved words when defining jobs and job streams
(continued)**

List of reserved words:				
sa	sameday	schedtime	schedule	scriptname
stop	streamlo gon	su	succoutputcond	tasktype
timezone	to	token_in	until	validfrom
validto	vartable	vt	weekday(s)	workday(s)

Avoid using the keywords listed in [Table 22: List of reserved words when defining workstations on page 180](#) when defining workstations, workstation classes, and domains:

Table 22. List of reserved words when defining workstations

List of reserved words:				
access	AIX	agent_type	autolink	behindfirew all
command	cpuclass	cpuname	description	domain
enabled	end	extrane ous	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 23: List of reserved words when defining users on page 181](#) when defining users:

Table 23. List of reserved words when defining users

List of reserved words:		
username	passwd	end
	ord	

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]
[variable 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

```

Arguments

Table 24. 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.		
variable	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:		

Table 24. Attribute settings for management workstation types (continued)

Attributes	Master domain manager	Domain manager	Backup domain manager
	UNIX WNT OTHER IBM_i		
node	The system host name or IP address.		
tcpaddr	The value to be assigned to <i>nm port</i> in the <code>localopts</code> file. For multiple workstations on a system, enter an unused port number. The default value is 31111.		
secureaddr	The value to be assigned to <i>nm ssl port</i> in the <code>localopts</code> file. Specify it if securitylevel is set to <code>on</code> , <code>force</code> , <code>enabled</code> , or <code>force_enabled</code> .		
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		

Table 24. Attribute settings for management workstation types (continued)

Attributes	Master domain manager	Domain manager	Backup domain manager
	<code>force</code> <code>force_enabled</code>		
fullstatus	<code>ON</code>		
server	Not applicable		This setting is ignored.
protocol	Not applicable		
members	Not applicable		
requirements	Not applicable		

Table 25: Attribute settings for target workstation types on page 184 describes the values that you set for each attribute for target workstation types. Following the table you find additional details about each attribute.

Table 25. 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						
variable	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: <code>UNIX</code> <code>WNT</code> <code>OTHER</code> <code>IBM_i</code> Specify <code>OTHER</code> for IBM i systems running as	<code>OTHER</code>	The operating system installed on the machine. Specify one of the following values: <code>UNIX</code> <code>WNT</code> <code>OTHER</code> <code>IBM_i</code>	This value setting is discovered on the system.	The operating system installed on the machine. Specify one of the following values: <code>UNIX</code> <code>WNT</code> <code>ZOS</code>	The operating system installed on the machine. Specify one of the following values: <code>UNIX</code> <code>WNT</code> <code>OTHER</code> <code>IBM_i</code>	

Table 25. 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
	limited fault-tolerant agents.						
node	The system host name or IP address.		The system host name or IP address. Specify <code>NULL</code> when host is set to <code>\$MASTER</code> , or when defining an extended agent for PeopleSoft, SAP or Oracle.	Agent host name or IP address.	Remote engine host name or IP address.	Not applicable	
tcpaddr	The value to be assigned to <i>nm port</i> in the <code>localopts</code> 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 <i>nm port</i> in the <code>localopts</code> 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 <code>http</code> .	The port number to communicate with the remote engine when the protocol is <code>http</code> .	Not applicable	
secureaddr	The value to be assigned to <i>nm ssl port</i> in the <code>localopts</code> file. Specify it if securitylevel is set to <code>on</code> , <code>force</code> , <code>enabled</code> . or <code>force_enabled</code> . 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 <code>https</code> .	The port number to communicate with the remote engine when the protocol is <code>https</code> .	Not applicable	

Table 25. 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
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 value for fault-tolerant agents is <code>MASTERDM</code> . This setting is mandatory for standard agents.	Specify an existing domain. This setting is mandatory.	This setting is needed only if the value assigned to host is <code>\$MANAGER</code> .	Not applicable			
host	Not Applicable		The host workstation. It can be set to <code>\$MASTER</code> or <code>\$MANAGER</code> .	The broker workstation.			
access	Not Applicable			Select the appropriate access method file name.	Not Applicable		
agentID				The unique identifier of the agent			
type	<code>fta</code> <code>s-agent</code> The default value is <code>fta</code> . Specify <code>fta</code> for IBM i systems running as limited fault-tolerant agents.	<code>broker</code>	<code>x-agent</code>	<code>agent</code>	<code>rem-eng</code>	<code>pool</code>	<code>d-pool</code>
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:		<code>OFF</code>	Not applicable			

Table 25. 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 This is an optional attribute. The default value is ON.						
behindfirewall	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.		OFF	Not applicable			
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:		Not applicable	

Table 25. 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
				http https This attribute is optional. When not specified, it is automatically determined from the settings specified for tcpaddr and secureaddr .			
members	Not applicable					Required value	Not applicable
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 21: List of reserved words when defining jobs and job streams on page 179](#).



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.

variable *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 workstations.

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 1011](#) 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 i systems 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 *Scheduling Job Integrations with HCL Workload Automation*.

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 203](#). 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
- Variable



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 1058](#).

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
- Variable
- 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 26. 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
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
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
end

cpuname distr-a2
  os wnt
  node quatro.ibm.com
  tz America/New_York
  domain distr-a
  for maestro type s-agent
  end
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
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 itdwbst11.ibm.com TCPADDR 41114
    timezone Europe/Rome
    domain MASTERDM
    for MAESTRO
        type BROKER
        autolink OFF
        behindfirewall OFF
        fullstatus OFF
    end
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
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 DPOOLINUX
    DESCRIPTION "Sample Dynamic Pool Workstation"
    VARIABLE 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
VARIABLE 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 181](#).

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.

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 27. 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 : 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 252](#) 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 370](#).

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 390](#) for more information about changing folder paths. See [Folder definition on page 228](#) 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.

If special characters are included, other than slashes (/) and backslashes (\), the entire string must be enclosed in quotes (").

The job `fails` if the script specified in the **scriptname** option is not found or does not have execute permission. It `abends` if the script that is not found or does not have execute permission includes 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 **recondsucc** 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 219](#) 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.

For a complete list of supported job types, see [Creating advanced job definitions on page 807](#).

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.

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 28. Comparison operators

Example	Operator	Description
RC<value	<	Less than
RC<=value	<=	Less than or equal to
RC>value	>	Greater than
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 29. 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 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.

- 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 consider a job successful by analyzing the content of the job log.

You can set a success or unsuccessful condition for the job by analyzing the job output. To analyze the job output, 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 214](#) and [repeatevery hhmm on page 214](#) options, or with the [after \[folder/\] \[workstation#\]\[folder/\]jobname on page 214](#) and [abendprompt "text" on page 215](#) 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 219](#) for more information.

[Table 30: Recovery options and actions on page 215](#) 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 30. Recovery options and actions

	Stop	Continue	Rerun
Recovery prompt: No Recovery job: No	Intervention is required.	Run <code>job2</code> .	Rerun <code>job1</code> . If <code>job1</code> ends abnormally, issue a prompt. If reply is yes, repeat above. If <code>job1</code> is successful, run <code>job2</code> .
Recovery prompt: Yes Recovery job: No	Issue recovery prompt. Intervention is required.	Issue recovery prompt. If reply is yes, run <code>job2</code> .	Issue recovery prompt. If reply is yes, rerun <code>job1</code> . If <code>job1</code> ends abnormally, repeat above. If <code>job1</code> is successful, run <code>job2</code> .

Table 30. Recovery options and actions (continued)

	Stop	Continue	Rerun
Recovery prompt: No Recovery job: Yes	Run <code>jobr</code> . If it ends abnormally, intervention is required. If it is successful, run <code>job2</code> .	Run <code>jobr</code> . Run <code>job2</code> .	Run <code>jobr</code> . If <code>jobr</code> ends abnormally, intervention is required. If <code>jobr</code> is successful, rerun <code>job1</code> . If <code>job1</code> ends abnormally, issue a prompt. If reply is yes, repeat above. If <code>job1</code> is successful, run <code>job2</code> .
Recovery prompt: Yes Recovery job: Yes	Issue recovery prompt. If reply is yes, run <code>jobr</code> . If it ends abnormally, intervention is required. If it is successful, run <code>job2</code> .	Issue recovery prompt. If reply is yes, run <code>jobr</code> . Run <code>job2</code> .	Issue recovery prompt. If reply is yes, run <code>jobr</code> . If <code>jobr</code> ends abnormally, intervention is required. If <code>jobr</code> is successful, rerun <code>job1</code> . If <code>job1</code> ends abnormally, repeat above. If <code>job1</code> is successful, run <code>job2</code> .

**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"?>
  <jsdsl: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">
    <jsdsl: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"?>
  <jsdsl: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">
    <jsdsl: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"?>
  <jsdsl:jobDefinition
    xmlns:jsdl="http://www.ibm.com/xmlns/prod/scheduling/1.0/jsdl"
```

```

        xmlns:jsdl="http://www.ibm.com/xmlns/prod/scheduling/1.0/jsdl">
    <jsdl:application name="executable">
    <jsdl:executable interactive="false">
    jsdl:script>dir</jsdl:script>
    </jsdl: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.
- Refer to [Variable and parameter definition on page 230](#) for additional information and examples.

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 230](#).

For information about how to pass variables between jobs in the same job stream instance, see [Passing variables between jobs on page 846](#).

Using variables in Workload Broker jobs

This section explains how to define and use variables in jobs for additional flexibility.

Dynamic workload broker supports the use of variables in jobs for additional flexibility. You can assign values to the variables or leave them blank, so that you can define the value when the job is submitted.

When you define jobs that will be processed through dynamic scheduling, you can include variables that can be used at run time to assign a value or override the variables defined in the JSDL job definition.

You define the variables in the Task String section of the HCL Workload Automation job, as described in the following example:

```
jobName -var var1Name=var1Value,...,varNName=varNValue
```

To define variables in the HCL Workload Automation job, perform the following steps:

1. Create a JSDL job definition using the Job Brokering Definition Console.
2. Define the variables for the job. For example, you can define the **memory** variable to specify the amount of memory required for the job to run.
3. Move to the **Resources** tab, **Hardware Requirements** section and type the name of the variable in the **Exact** value field in the **Physical Memory** section. When the job is submitted, the value assigned to the **memory** variable defines the amount of physical memory.
4. Save the job definition in the **Job Repository** database.
5. Create a job to be submitted in HCL Workload Automation. This job contains the reference to the job in HCL Workload Automation created in step 1. Define the HCL Workload Automation job as follows:
 - a. In the Dynamic Workload Console, from the navigation bar, click **Administration > Workload Design > Manage Workload Definitions**.
 - b. Select **New > Job Definition > Cloud > Workload Broker**.
 - c. In the **General** tab, in the **Workstation** field specify the workload broker workstation.
 - d. In the **Task** tab, in the **Workload Broker job name** field specify the name of the JSDL job definition you created in step 1.
6. Add the HCL Workload Automation job to a job stream.
7. Submit or schedule the HCL Workload Automation job using either the Dynamic Workload Console or **conman**.
8. After any existing dependencies are resolved, the master domain manager submits the HCL Workload Automation job to HCL Workload Automation via the workload broker workstation.
9. The workload broker workstation identifies the job definition to be submitted based on the information on the **Task String** section of the HCL Workload Automation job. It also creates an alias which contains the association with the job.
10. The job definition is submitted to HCL Workload Automation with the value specified for the **memory** variable.
11. Dynamic workload broker manages and monitors the whole job lifecycle.
12. Dynamic workload broker returns status information on the job to the workload broker workstation, which communicates it to the Master Domain Manager. The job status is mapped to the HCL Workload Automation status.

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 31. Supported HCL Workload Automation variables in JSDL definitions.

Variables that can be inserted in the HCL Workload Automation job definition	Description
tw.s.host.workstation	Name of the host workstation
tw.s.job.date	Date of the submitted job.
tw.s.job.fqname	Fully qualified name of the job (UNISON_JOB)
tw.s.job.ia	Input arrival time of the job
tw.s.job.interactive	Job is interactive. Values can be <code>true</code> or <code>false</code> . Applies only to backward-compatible jobs.
tw.s.job.logon	Credentials of the user who runs the job (LOGIN). Applies only to backward-compatible jobs.
tw.s.job.name	Name of the submitted job
tw.s.job.num	UNISON_JOBNUM
tw.s.job.priority	Priority of the submitted job
tw.s.job.promoted	Job is promoted. Values can be <code>YES</code> or <code>NO</code> . For more information about promotion for dynamic jobs, see the section about promoting jobs scheduled on dynamic pools in <i>HCL Workload Automation: User's Guide and Reference</i> .
tw.s.job.recnum	Record number of the job.
tw.s.job.resourcesFor Promoted	Quantity of the required logical resources assigned on a dynamic pool to a promoted job. Values can be <code>1</code> if the job is promoted or <code>10</code> if the job is not promoted. For more information about promotion for dynamic jobs, see the section about promoting jobs scheduled on dynamic pools in <i>HCL Workload Automation: User's Guide and Reference</i> .
tw.s.job.taskstring	Task string of the submitted job. Applies only to backward-compatible jobs.
tw.s.job.workstation	Name of the workstation on which the job is defined
tw.s.jobstream.id	ID of the job stream that includes the job (UNISON_SCHED_ID)
tw.s.jobstream.name	Name of the job stream that includes the job (UNISON_SCHED)
tw.s.jobstream.workst ation	Name of the workstation on which the job stream that includes the job is defined
tw.s.master.workstat ion	Name of the master domain manager (UNISON_MASTER)
tw.s.plan.date	Start date of the production plan (UNISON_SCHED_DATE)
tw.s.plan.date.epoch	Start date of the production plan, in epoch format (UNISON_SCHED_EPOCH)

Table 31. Supported HCL Workload Automation variables in JSDL definitions. (continued)

Variables that can be inserted in the HCL Workload Automation job definition	Description
tw.s.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="tw.s.jobstream.name">none</jSDL:stringVariable>
    <jSDL:stringVariable name="tw.s.job.fqname">none</jSDL:stringVariable>
    <jSDL:stringVariable name="tw.s.master.workstation">none</jSDL:stringVariable>
    <jSDL:stringVariable name="tw.s.plan.runnumber">none</jSDL:stringVariable>
    <jSDL:stringVariable name="tw.s.plan.date">none</jSDL:stringVariable>
    <jSDL:stringVariable name="tw.s.plan.date.epoch">none</jSDL:stringVariable>
    <jSDL:stringVariable name="tw.s.job.logon">none</jSDL:stringVariable>
  </jSDL:variables>
  <jSDL:application name="executable">
    <jSDLe:executable output="{tw.s.plan.runnumber}">
      <jSDLe:environment>
        <jSDLe:variable name="UNISON_SCHED">{tw.s.jobstream.name}</jSDLe:variable>
        <jSDLe:variable name="UNISON_JOB">{tw.s.job.fqname}</jSDLe:variable>
        <jSDLe:variable name="UNISON_MASTER">{tw.s.master.workstation}</jSDLe:variable>
        <jSDLe:variable name="UNISON_RUN">{tw.s.plan.runnumber}</jSDLe:variable>
        <jSDLe:variable name="UNISON_SCHED_DATE">{tw.s.plan.date}</jSDLe:variable>
        <jSDLe:variable name="UNISON_SCHED_EPOCH">{tw.s.plan.date.epoch}</jSDLe:variable>
        <jSDLe:variable name="LOGIN">{tw.s.job.logon}</jSDLe:variable>
      </jSDLe:environment>
    </jSDLe:executable>
  </jSDL:application>
</jSDL:jobDefinition>
```

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 226](#).



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@twsvbt.com
    password "internetpwd"
end
#
username serverA#dom1\jack
    password "righto"
end
#
username serverB#user1@twsvbt.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 405` 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:

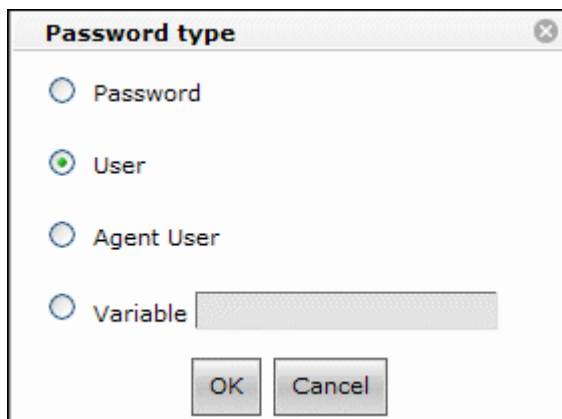
- There must be mutual trust between `Domain1` and `Domain2`.
- In `Domain1` on the computers where jobs are launched, `sue` must have the right to **Log on as batch**.
- In `Domain1`, `maestro` must be a domain user.
- On the domain controllers in `Domain2`, `maestro` must have the right to **Access 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 16 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.

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 32. Folder commands

Command		Description
Composer	Conman	
CHFOLDER on page 390	CHFOLDER on page 526	Changes the working directory or the current folder.
LISTFOLDER on page 421	LISTFOLDER on page 549	Lists all folders in the root or in a folder.
MKFOLDER on page 427	-	Creates a new folder in the database.
RMFOLDER on page 439	-	Deletes a folder defined in the database.
RENAMEFOLDER on page 444	-	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.

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.

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.

However, the recommended way to define variables is to use a [Variable table definition on page 235](#). 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.

\${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 233](#).



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 148](#).

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 33. How to handle a backslash in variable substitution

Wrong way	Right way
1. Define the MYDIR variable as: <pre>\$PARM MYDIR "scripts"</pre> 2. Use it in this way: <pre>job01 scriptname "c:\operid\^MYDIR^\test.cmd"</pre> 3. Use it in this way: <pre>job01 scriptname "c:\operid\\${MYDIR}\test.cmd"</pre>	1. Define the MYDIR variable as: <pre>\$PARM MYDIR "\scripts"</pre> 2. Use it in this way: <pre>job01 scriptname "c:\operid^MYDIR^\test.cmd"</pre> 3. Use it in this way: <pre>job01 scriptname "c:\operid\${MYDIR}\test.cmd"</pre>

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 906](#) 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 34: Keywords that can take local parameters in submit commands on page 232](#) summarizes in which submit command keyword you can use parameters.

Table 34. 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	✓	✓	✓	

Keyword	submit docommand (sbd command)	submit file (sbf command)	submit job (sbj command)	submit job stream (sbs command)
scriptname		✓		
docommand	✓			
logon	✓	✓		
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 478](#).

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: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:on<jsdle: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 235](#)
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 145](#).

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 230** 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.

For more information, see [Customizing your workload using variable tables on page 145](#).

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

```
variable [folder/]tablename
[description "description"]
```

[isdefault]

members

[*variablename* "*variablevalue*"]

...

[*variablename* "*variablevalue*"]

end

Arguments

variable [*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.

```
VARIABLE 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/Business_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 35. Required access keyword on variable table in Security file (variable object) and allowed actions.

Required security file access keyword on enclosing variable table	Allowed action on listed variable entries
	Add
	Delete
Modify	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.

global or named prompts

A global prompt is defined in the database as a scheduling object, it is identified by a unique name and it can be used by any job or job stream. 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.

This section describes global prompts. For more information on local prompts refer to [Job on page 1111](#) and [Job stream definition on page 252](#).

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: 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 230](#) 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.

Each run cycle group definition has the following format and arguments:

Syntax

runcyclegroup

```
[folder/]runcyclegroupname [description "text"]
    variable [folder/]tablename
    [freedays [folder/]calendarname [-sa] [-su]]
[on [runcycle [folder/]name
    [validfrom date] [validto date]
    [description "text"]
    [variable [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]]
    [onuntilaction]]
    [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]] [onuntilaction]]
[every rate {everyendtime time[+n day[s]]}]
[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 (&).

variable [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 285](#) 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.

variable

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 1011](#) 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 1011](#) 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 1011](#) 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]]] [onuntilaction]

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 1011](#) 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"
VARIABLE TABLE1
FREEDAYS CAL1 -SA -SU
ON      RUNCYCLE RUN_CYCLE1 VALIDFROM 03/31/2024 VALIDTO 04/12/2024 DESCRIPTION
        "Inclusive Run Cycle" VARIABLE 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
```

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

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\"}"]
    [variable [folder/table_name]
    [freedays [folder/calendarname [-sa] [-su]]
    [on [runcycle name
        [validfrom date] [validto date]
        [description "text"]
        [variable [folder/table_name]]
        {date|day|folder/calendar|request|calendar|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|calendar|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 36: List of scheduling keywords on page 255 contains a brief description of the job stream definition keywords. A detailed description of each scheduling keyword is provided in the next subsections.

Table 36. List of scheduling keywords (continued)

Keyword	Description	Page
confirmed	Specifies that the completion of this job requires confirmation.	confirmed on page 266
critical	Specifies that the job is mission critical and must therefore be managed preferentially.	critical on page 267
deadline	Specifies the time within which a job or job stream should complete. When defined in a run cycle specifies the time within which a job or a job stream must complete in that specific run cycle.	deadline on page 267
description	Contains a description of the job stream. The maximum length of this field is 120 characters.	description on page 268
draft	Specifies that the plan generation process must ignore this job stream.	draft on page 269
end	Marks the end of a job stream.	end on page 269
every	Launches a job stream or a job repeatedly at a specified rate.	every on page 270
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 275
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 275
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 390 for more information about changing folder paths.	folder on page 281
follows	Specifies jobs or job streams that must complete successfully or must satisfy one or more output	follows on page 281

Table 36. List of scheduling keywords (continued)

Keyword	Description	Page
	conditions before the job or the job stream that is being defined is launched.	
freedays	Specifies a freeday calendar for calculating <i>workdays</i> for the job stream. It can also set Saturdays and Sundays as <i>workdays</i> .	freedays on page 285
interval	How often HCL Workload Automation checks whether the condition is met.	startcond on page 319
job statement	Defines a job and its dependencies.	job statement on page 287
isservice	Specifies that a job stream can be submitted as a service to Self-Service Catalog. When creating and editing SSC-ready job streams, it is recommended you use the Dynamic Workload Console.	isservice on page 288
join	Defines a set of conditional dependencies on a job or job stream.	join on page 289
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 326 .	jsuntil on page 291
keyjob	Marks a job as key in both the database and in the plan .	keyjob on page 293
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 294
limit	Sets a limit on the number of jobs that can be launched concurrently from the job stream.	limit on page 294
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 295

Table 36. List of scheduling keywords (continued)

Keyword	Description	Page
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 296
mindur	Specifies the shortest amount of time within which a job normally runs and completes.	mindur on page 298
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 299
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 <code>Cancel Pending</code> status and is not executed.	nop on page 300
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 301
onlate	Defines the action to be taken on a job in the job stream when the job's deadline expires.	onlate on page 308
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 309
opens	Defines files that must be accessible before the job or job stream is launched.	opens on page 310
onuntil	Specifies the action to take on a job or job stream whose until time has been reached.	until on page 326
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 319
priority	Defines the priority for a job or job stream.	priority on page 313
prompt	Defines prompts that must be replied to before the job or job stream is launched.	prompt on page 314

Table 36. List of scheduling keywords (continued)

Keyword	Description	Page
runcycle	Specifies a label with a friendly name for the run cycle. It is used in conjunction with the following keywords: except For <i>exclusive</i> run cycles, that define when the job stream is not to run. on For <i>inclusive</i> run cycles, that define when the job stream will run.	• except on page 275 • on on page 301
schedule	Assigns a name to the job stream.	schedule on page 317
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 315
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 319
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 323
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 323
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 324
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 325

Table 36. List of scheduling keywords (continued)

Keyword	Description	Page
timezone tz	Specifies the time zone to be used when computing the start time.	timezone on page 325
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 291 .	until on page 326
validfrom	Defines the date from which the job stream instance starts.	validfrom/validto on page 329
validto	Indicates the date on which the job stream instance ends.	validfrom/validto on page 329
vartable	Defines the variable table to be used by the job stream and the run cycle.	Variable table on page 330

**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_PRESIMPLE 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: workstation_name
    name: JOB_name
    jobDefinition:
      Specify the job_definition
```

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 262](#).

Embedded jobs not supported by composer, dataextract, and dataimport commands.

Example

You can create a job stream 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 36: List of scheduling keywords on page 255](#).

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.

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 1011](#) 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 315](#).

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.

0

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:

- Job is shown as late in **conman**.
- A message is issued to the *stdlist* and console logs.

When a job does not complete before its deadline, a warning message is displayed. If this job is not part of a carried forward job stream and you run `JnextPlan` while it is still running, the job is inserted in USERJOBS. In this case, another warning message about the expired deadline is added in the `TWS_home/stdlist/logs/yyyymmdd_TWSMERGE.log` file.



Note: 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.

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 1011](#) 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 120 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 270](#)
- [every \(used for jobs\) on page 271](#)

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 272 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 274 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 274 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 275 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*

$1800 + 0015 + 0015 = \mathbf{1830} < 1833$

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 than 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 301](#) 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 301](#).

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 228](#).

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 $\pm$ time to $\pm$ time|from time $\pm$ day[s] to time $\pm$ day[s]]

Use the following syntax for jobs:

[follows {[netagent::][workstation#]jobstreamname{.jobname}}

[previous|sameday|relative from $\pm$ time to $\pm$ time | from time $\pm$ day[s] to time $\pm$ 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 $\pm$ time to $\pm$ time | from time $\pm$ day[s] to time $\pm$ 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 283](#) 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.

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 87](#).

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.jobx
```

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 USAH0L -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 205](#)), 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 205](#).

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 968](#).



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 stopabendprompt "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 323](#).

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 326](#).

Syntax

```
[jsuntil time [timezone|tz tzname][+n day[s]] [onuntilaction]]
```

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 1011](#) 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 281](#) 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 275](#) for more information.

Syntax

on [**runcycle** *name*

[**valid from** *date*] [**valid to** *date*]

[**description** "*text*"]

[**variable** *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.

variable

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 `updates` calendar:

```
schedule sked3 on 6/15/23,updates
```

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#PIPP0 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.

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.

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.

```
SCHEDULE MDM005#JS_0415
ON RUNCYCLE ACCTRC 06/16/2024
( SCHEDTIME 1445 DEADLINE 1530 EVERY 0010 EVERYENDTIME 1515 )
ONOVERLAP ENQUEUE
:
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 230](#) 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 the HCL 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 backslash **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 230](#) 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 1011](#) 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 87](#).

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. Job *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 1011](#).

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 128](#).

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 128](#).

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 890](#).

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 `FileMgr` 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
VARIABLE MAIN_TABLE
ON RUNCYCLE RC2 08/04/2024
( AT 0800 +1 DAYS )
STARTCOND FILECREATED S_MDM#/Reports/Report.txt USER FileMgr 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 288](#). 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 1011](#) 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 291](#).

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 1011](#) 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 367](#).

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

variable [*folder*]/*table_name*

Arguments

variable[*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 145](#).

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 155](#).

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 369](#) 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.

- 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 rule attributes are:

- Required attributes:

name

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 228](#). 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.

ruleType

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 1087](#).

[Table 38: TWSObjectsMonitor events. on page 336](#) lists the `TWSObjectsMonitor` events.

Table 38. `TWSObjectsMonitor` events.

Event type	When the event is sent
<code>JobStatusChanged</code>	The status of a job changes
<code>JobUntil</code>	The latest start time of a job has elapsed
<code>JobSubmit</code>	A job is submitted
<code>JobCancel</code>	A job is canceled
<code>JobRestart</code>	A job is restarted
<code>JobLate</code>	A job becomes late
<code>JobPromoted</code>	A job in a critical network approaches the critical start time and has not yet started
<code>JobRiskLevelChanged</code>	◦ 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 <code>JOBS</code> .
<code>JobExceededMaximumDuration</code>	A job exceeds the maximum duration time established for the job
<code>JobDidnotReachMinimumDuration</code>	A job does not run long enough to reach the minimum duration time established for the job
<code>JobStreamStatusChanged</code>	The status of a job stream changes
<code>JobStreamCompleted</code>	A job stream has completed
<code>JobStreamUntil</code>	The latest start time of a job stream has elapsed
<code>JobStreamSubmit</code>	A job stream is submitted
<code>JobStreamCancel</code>	A job stream is canceled
<code>JobStreamLate</code>	A job stream becomes late
<code>WorkstationStatusChanged</code>	A workstation is started or stopped

Event type	When the event is sent
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 39: TWSApplicationMonitor events. on page 337](#) lists the `TWSApplicationMonitor` events.

Table 39. TWSApplicationMonitor events.

Event type	When the event is sent
MessageQueuesFilling	A specified mailbox exceeds the percentage full value.

Event type	When the event is sent
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 40: FileMonitor events. on page 338 lists the FileMonitor events.

Table 40. 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 41: DatasetMonitor events. on page 338 lists the DatasetMonitor events.

Table 41. 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)
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 1087](#) 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 1087](#).

Table 42. Action types by action provider.

Action provider	Action types
GenericAction	RunCommand
MailSender	SendMail
MessageLogger	PostOperatorMessage
TECEventForwarder	TECFWD
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 173](#).

onTimeout

The action starts after the time specified in [timeInterval on page 334](#) 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 1102](#) for a list of parameters, or property names, available for every action type.

value

See [Action providers and definitions on page 1102](#) 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 be 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>
      </filteringPredicate>
    </eventCondition>
  </eventRule>
</eventRuleSet>
```

```

        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 166](#) .

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 SECR0LE1, SECR0LE2, SECR0LE3
  USER2 SECR0LE4
  USER3 SECR0LE2, SECR0LE4
END

ACCESSCONTROLLIST FOR SECDOM2
  USER1 SECR0LE1, SECR0LE2
  USER2 SECR0LE3
END

ACCESSCONTROLLIST FOLDER /FOL1/FOL2/
  USER1 SECR0LE1
END

ACCESSCONTROLLIST FOLDER /APPS/APP1
  USER1 SECR0LE1
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=@
```

```
variable    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 43: Security object types on page 353](#) shows the different object types and how they are referenced with **composer** and with the Dynamic Workload Console:

Table 43. 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 43. 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 44: Actions that users or groups can perform on the different objects on page 354 shows the actions that users or groups can perform on the different objects.

Table 44. 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 SECR0LE2
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,
        ALTPRI,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 45. 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)	Variable Tables (vartable)
	Workload Application (wkldappl)
	Workflow Folders (folder)
	Parameters (parameter)
 Note: See in parenthesis the corresponding <i>actions</i> and <i>objects</i> values that you must use when defining role-based security with the composer command-line interface.	

Table 46. 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 46. 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 47. 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 47. 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 <i>actions</i> and <i>objects</i> values that you must use when defining role-based security with the composer command line interface.

Table 48. 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 48. 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 49. 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 49. 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 <i>actions</i> and <i>objects</i> values that you must use when defining role-based security with the composer command line interface.

Table 50. 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 <i>action</i> and <i>object</i> values that you must use when defining role-based security with the composer command-line interface.

Table 51. 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 51. 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 52. 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 53: Attributes for security object types on page 362](#) 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 53. Attributes for security object types

Attribute	Name (name)	Workstation	Custom	JCL (jcl)	JCLtype	Logon	Provider	Type	Host	Port	Folder	CPU Fol
Security object type		(cpu)	(custom)		(jcltype)	(logon)	(provider)	(type)	(host)	(port)	(folder)	der (cpuf older)
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 362](#).

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 (variable)** 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.

\$JCOWNER

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 367](#)
- [RUNCOMPOSER](#) [Running commands from composer on page 374](#)
- [COMPOSER COMMANDS](#) [on page 380](#)

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 aòsdfjasldfkjasdfìòjahsdfjlkasd
```



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 `twc_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 **MAESTROL** 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, you must open the DOS command prompt shell with the "Run As Administrator" option to run composer on your workstation as a generic user different from Administrator or HCL Workload Automation user.

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 390](#) 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 54: Scheduling objects filtering criteria on page 376 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 54. Scheduling objects filtering criteria

Scheduling object	Filter keywords or fields that can be filtered	Description	Example
workstation	<i>[folder/]workstationname</i>	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
	variable	Applies the command to the workstations which refer the specified variable table.	mod ws=@; filter variable =table2
domain	<i>domainname</i>	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	<i>promptname</i>	Applies the command to the global prompts whose name satisfies the criteria.	lock prompt=p@
user	<i>[folder/]workstationname# username</i>	Applies the command to the users whose identifier satisfies the criteria.	list users=cpu1 #operator?
resource	<i>[folder/]workstationname# resourcename</i>	Applies the command to the resources whose identifier satisfies the criteria.	print res=cpu? #operator?

Table 54. Scheduling objects filtering criteria (continued)

Scheduling object	Filter keywords or fields that can be filtered	Description	Example
variable	<i>variablename</i>	Applies the command to the parameters whose name satisfies the criteria.	delete vb=mytable.myparm@
folder	<i>folder</i>	Applies the command to the folders whose name satisfies the criteria.	list folder myfolder
job definition	<i>[folder/]workstationname</i> <i>#folder/jobname</i>	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	<i>[folder/]workstationname</i> <i>#folder/jobstreamname</i>	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
	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

Table 54. Scheduling objects filtering criteria (continued)

Scheduling object	Filter keywords or fields that can be filtered	Description	Example
	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	<i>eventrulename</i>	Applies the command to the event rules that include an action on a specific job or job stream.	list er=@; filter js=accrecjs5
variable	<i>vartablename</i>	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`.

Delimiters and special characters

[Table 55: Delimiters and special characters for composer on page 378](#) lists characters have special meanings in composer commands.

Table 55. Delimiters and special characters for composer

Character	Description
&	Command delimiter. See Running the composer program on page 370 .
;	Argument delimiter. For example: <pre>;info;offline</pre>
=	Value delimiter. For example: <pre>sched=sked5</pre>

Table 55. Delimiters and special characters for composer (continued)

Character	Description
:!	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: <pre>!ls or :ls</pre>
<< >>	Comment brackets. Comments can be placed on a single line anywhere in a job stream. For example: <pre>schedule foo <<comment>> on everyday</pre>
*	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: <pre>*comment or print& *comment</pre>
>	Redirects command output to a file, overwriting the contents of that file. If the file does not exist, it is created. For example: <pre>display parms > parmlist</pre>
>>	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: <pre>display parms >> parmlist</pre>
	Pipes command output to a system command or process. The system command is run whether or not output is generated. For example: <pre>display parms grep alparm</pre>
	Pipes command output to a system command or process. The system command is not run if there is no output. For example: <pre>display parms grep alparm</pre>

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 56. 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.

Table 56. List of composer commands (continued)

Command	Short Name	Description
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 <i>new</i> 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.
<i>system command</i>		Invokes an operating system command.
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 57: Object identifiers for each type of object defined in the database on page 382](#) 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 57. Object identifiers for each type of object defined in the database

Object type	Object identifiers
domain	<i>domainname</i>
workstation	<i>workstationname</i> (checked across workstations and workstation classes)
workstation class	<i>workstationclassname</i> (checked across workstations and workstation classes)
calendar	<i>calendarname</i>
job definition	<i>workstationname</i> and <i>jobname</i>
user	<i>workstationname</i> and <i>username</i>
job stream	<i>workstationname</i> and <i>jobstreamname</i> and, if defined, <i>validfrom</i>
job within a job stream	<i>workstationname</i> and <i>jobstreamname</i> , <i>jobname</i> , and, if defined, <i>validfrom</i>
resource	<i>workstationname</i> and <i>resourcename</i>
prompt	<i>promptname</i>

Table 57. Object identifiers for each type of object defined in the database (continued)

Object type	Object identifiers
variable table	<i>variabletablename</i>
variable	<i>variabletablename.variablename</i>
event rule	<i>eventrulename</i>
access control list	<i>securitydomainname</i>
security domain	<i>securitydomainname</i>
security role	<i>securityrolename</i>

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 58: Object definition update upon deletion of referenced object on page 383](#) shows all cases when a referenced object can be deleted even if other objects reference it:

Table 58. 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 59: Referential integrity check when deleting an object from the database on page 383](#) describes how the product behaves when it is requested to delete an object referenced by another object with using a specific relationship:

Table 59. 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.

Table 59. Referential integrity check when deleting an object from the database (continued)

Object to be deleted	Referenced by object	Relationship	Delete rule
	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 A	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.
	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.

Table 59. Referential integrity check when deleting an object from the database (continued)

Object to be deleted	Referenced by object	Relationship	Delete rule
	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: • in the text of an ad hoc prompt • or in the file name specified in a file dependency	variable A is deleted without checking
	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	variable A is deleted without checking

Table 59. Referential integrity check when deleting an object from the database (continued)

Object to be deleted	Referenced by object	Relationship	Delete rule
		• or in the value specified for streamlogon • or in the value specified for scriptname	
	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.
	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.

Table 59. Referential integrity check when deleting an object from the database (continued)

Object to be deleted	Referenced by object	Relationship	Delete rule
	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 **twc_user1** with password **mypasswd1** from within the **composer** command-line program, run the following command:

```
au username=twc_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 custom 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.

See also

From the Dynamic Workload Console you can perform the same tasks as described in:

the *Dynamic Workload Console User's Guide*.

- To display a list of folders, see
the *Dynamic Workload Console User's Guide*, section about Listing object definitions in the database.

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]=domainname] |
[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] |
[table | vt=[folder/]tablename] |
[wat=[folder/]workloadapplicationtemplatename]
[cpu={[folder/]workstationname [:force] | [folder/]workstationclassname [:force] | domainname}}
[workstation | ws=[folder/]workstationname] [:force] |
[workstationclass | wsc]=[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]
```

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 439](#).

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.

variable | 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/]workstationname#[folder/]jobname* follows, deletes the *[folder/]jobname* job stored on the *[folder/]workstationname* workstation on which the job runs. If *[folder/]workstationname* is omitted, the default is the workstation on which **composer** is running. Wildcard characters are permitted.

sched | jobstream | js

If argument *[folder/]workstationname#[folder/]jstreamname* follows, deletes the *[folder/]jstreamname* job stream on the *[folder/]workstationname* workstation on which the job stream is defined. 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 @.

users | user

If argument *[folder/]workstationname#username* follows, deletes the *username* user of the *[folder/]workstationname* workstation on which the user is defined. If *[folder/]workstationname* 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 be locked. If some matching objects are locked during the command processing, an error message with the list of these objects is shown to the user.

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.

- To delete all other objects, see

the *Dynamic Workload Console User's Guide*, section about Listing object definitions in the database.

the *Dynamic Workload Console User's Guide*, section about Listing object definitions in the database.

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] |
[table | vt=[folder/]tablename] |
[prompts | prom=[folder/]promptname] |
[resources | resource | res=[folder/]workstationname#[folder/]resourcename] |
[runcyclegroup | rcg=[folder/]runcyclegroupname] |
[cpu={[folder/]workstationname | [folder/]workstationclassname | domainname}]
[wat=[folder/]workloadapplicationtemplatename]
[workstation | ws=[folder/]workstationname] |
[workstationclass | wscl=[folder/]workstationclassname] |
[domain | dom=domainname] |
[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.

variable | **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/]workstationname#username* follows, displays the *username* user of the *[folder/]workstationname* workstation on which the user is defined. If *[folder/]workstationname* 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.
- To display all other objects, see

the *Dynamic Workload Console User's Guide*, section about Listing object definitions in the database.

the *Dynamic Workload Console User's Guide*, section about Listing object definitions in the database.

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 in the shell where you plan to run the command
- 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] |
[table | vt=[folder/]tablename] |
[prompts | prom=[folder/]promptname] |
[resources | resource | res=[folder/]workstationname#[folder/]resourcename] |
[runcyclegroup | rcg=[folder/]runcyclegroupname] |
[cpu={[folder/]workstationname | [folder/]workstationclassname | domainname}] |
[workstation | ws=[folder/]workstationname] |
[workstationclass | wscl=[folder/]workstationclassname] |
[domain | dom=domainname] |
[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.

variable | 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 *workstationname#[folder/]jobname* follows, copies the *[folder/]jobname* job, optionally defined in the folder named *[folder/]* of the *workstationname* 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 *workstationname* is omitted, the default is the workstation on which **composer** is running. Wildcard characters are permitted for *workstationname*, [*folder/*], and *jobname*.

sched | jobstream | js

If no argument follows, copies all job stream definitions into the file.

If argument *workstationname#[folder/]jstreamname* follows, copies the *jstreamname* job stream, optionally defined in the folder named [*folder/*], of the *workstationname* 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 *workstationname* is omitted, the default is the workstation on which **composer** is running. Wildcard characters are permitted for *workstationname*, [*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/]workstationname#username* follows, copies the *username* user of the *workstationname* workstation in the specified folder on which the user is defined into the file. If *workstationname* 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
```

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 **MAESTROL** local variable. The print command can be used to send the output to a local printer, if the **MAESTROL** 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] |
[table | vt=[folder/]tablename] |
[prompts | prom=[folder/]promptname] |
[resources | resource | res=[folder/]workstationname#]resourcename] |
[runcyclegroup | rcg=[folder/]runcyclegroupname] |
[cpu={[folder/]workstationname | [folder/]workstationclassname | domainname}]
[wat=workloadapplicationtemplatename]
[workstation | ws=[folder/]workstationname] |
[workstationclass | wscl=[folder/]workstationclassname] |
[domain | dom=domainname] |
[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]}
[: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 421](#) 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.

variable | 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 390](#) for more information about changing folder paths.

jobs | jobdefinition | jd

If no argument follows, lists or prints all job definitions.

If argument *workstationname#[folder/]jobname* follows, lists or prints the *jobname* job stored in the folder named *[folder/]* of the *workstationname* workstation on which the job runs. If *workstationname* is omitted, the default is the workstation on which **composer** is running. Wildcard characters are permitted for *workstationname*, *[folder/]*, and *jobname*.

sched | jobstream | js

If no argument follows, lists or prints all job stream definitions.

If argument *workstationname#[folder/]jstreamname* follows, lists or prints the *jstreamname* job stream stored in the folder named *[folder/]* of the *workstationname* workstation on which the job stream is defined. If

workstationname is omitted, the default is the workstation on which **composer** is running. Wildcard characters are permitted for *workstationname*, [*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/*]*workstationname#username* follows, lists or prints the *username* user of the *workstationname* workstation in the specified folder on which the user is defined. If *workstationname* 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 **MAESTROL**P local variable. The print command can be used to send the output to a local printer, if you set the **MAESTROL**P variable accordingly. Make sure the **MAESTROL**P 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 **MAESTROL**P.

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"
VARIABLE 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.
- To list or print all other objects, see

the *Dynamic Workload Console User's Guide*, section about Listing object definitions in the database.

the *Dynamic Workload Console User's Guide*, section about Listing object definitions in the database.

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 412](#) 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/@
```

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 database objects, see

the *Dynamic Workload Console User's Guide*, section about Listing object definitions in the database.

the *Dynamic Workload Console User's Guide*, section about Listing object definitions in the database.

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] |
[table | vt=[folder/]tablename] |
[prompts | prom=[folder/]promptname] |
[resources | resource | res=[folder/]workstationname#[folder/]resourcename] |
[runcyclegroup | rcg=[folder/]runcyclegroupname] |
[cpu={[folder/]workstationname | [folder/]workstationclassname | domainname}]
[workstation | ws=[folder/]workstationname] |
[workstationclass | wscl=[folder/]workstationclassname] |
[domain | dom=domainname] |
[jobs | jobdefinition | jd=[folder/]workstationname#[folder/]jobname] |
[sched]jobstreamljs= [[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.

variable | 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/]workstationname#[folder/]resourcename* follows, locks 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, 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/]workstationname#[folder/]jobname* follows, locks the *jobname* job of the *workstationname* workstation on which the job runs. If *workstationname* 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/]workstationname#[folder/]jstreamname* follows, locks the *jstreamname* job stream of the *workstationname* workstation on which the job stream is defined. If *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 @.

users | user

If no argument follows, locks all user definitions.

If argument *[folder/]workstationname#username* follows, locks the *username* user of the *workstationname* workstation in the specified folder on which the user is defined. If *workstationname* 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 434](#). 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 390](#) 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]] |
[variable | vt=[folder/]tablename] |
[prompts | prom=[folder/]promptname] |
[resources | resource | res=[folder/]workstationname#[folder/]resourcename] |
[runcyclegroup | rcg=[folder/]runcyclegroupname] |
[cpu={[folder/]workstationname | [folder/]workstationclassname | domainname}]
[wat=workloadapplicationtemplatename]
[workstation | ws=[folder/]workstationname] |
[workstationclass | wscl=[folder/]workstationclassname] |
[domain | dom=domainname] |
[jobs | jobdefinition | jd=[folder/]workstationname#[folder/]jobname] |
```



```
[sched]jobstreamjs= [[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.

variable | 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]/workstationname#[folder]/resourcename* follows, modifies 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, 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/]workstationname#[folder/]jobname* follows, modifies the *jobname* job of the *workstationname* workstation on which the job runs. If *workstationname* 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/]workstationname#[folder/]jstreamname* follows, modifies the *jstreamname* job stream of the *workstationname* workstation on which the job stream is defined. If *workstationname* 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/]workstationname#username* follows, modifies the *username* user of the *workstationname* workstation in the specified folder on which the user is defined. If *workstationname* 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-JOB06
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.

- To modify 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 modify all other objects, see

the *Dynamic Workload Console User's Guide*, section about Listing object definitions in the database.

the *Dynamic Workload Console User's Guide*, section about Listing object definitions in the database.

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 |
variable |
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 427](#).

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.

itext

Inserts text before the character above the **i**.

rtext

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.

>rtext

Replaces characters at the end of the line with text.

Directive Examples

ddd

Deletes the three characters above the **ds**.

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 393](#) 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.

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 database objects, see

the *Dynamic Workload Console User's Guide*, section about Listing object definitions in the database.

the *Dynamic Workload Console User's Guide*, section about Listing object definitions in the database.

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]*] |

[**variable** | **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**=*domainname*] |

```
[jobs | jobdefinition | jd]=[folder/]workstationname#[folder/]jobname |
[sched | jobstream | js]= [folder/]workstationname#
[eventrule | erule | er=[folder/]eventrulename] |
[wat=[folder/]workloadapplicationtemplatenamename]
[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 421](#) and [CHFOLDER on page 390](#) 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 444](#).

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 443](#) 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 **workstationname** 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 **workstationname**:

- 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.
- To rename all other objects, see
the *Dynamic Workload Console User's Guide*, section about Listing object definitions in the database.
the *Dynamic Workload Console User's Guide*, section about Listing object definitions in the database.

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 421](#) and [CHFOLDER on page 390](#) 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 440](#) 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
```


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 all other objects, see
the *Dynamic Workload Console User's Guide*, section about Listing object definitions in the database.
the *Dynamic Workload Console User's Guide*, section about Listing object definitions in the database.

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] |
[table | vt=[folder/]tablename] |
[prompts | prom=[folder/]promptname] |
[resources | resource | res=[[folder/]workstationname#][folder/]resourcename] |
[runcyclegroup | rcg=[folder/]runcyclegroupname] |
[cpu={[[folder/]workstationname | [folder/]workstationclassname | domainname]}]
[workstation | ws=[folder/]workstationname] |
[workstationclass | wscl=[folder/]workstationclassname] |
[domain | dom=domainname] |
[jobs | jobdefinition | jd=[[folder/]workstationname#][folder/]jobname] |
[schedjobstream|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.

variable | 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/]workstationname#[folder/]resourcename* follows, unlocks 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, 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/]workstationname#[folder/]jobname* follows, unlocks the *jobname* job of the *workstationname* workstation on which the job runs. If *workstationname* 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/]workstationname#[folder/]jstreamname* follows, unlocks the *jstreamname* job stream of the *workstationname* workstation on which the job stream is defined. If *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 @.

users | user

If no argument follows, unlocks all user definitions.

If argument *[folder/]workstationname#username* follows, unlocks the *username* user of the *workstationname* workstation in the specified folder on which the user is defined. If *workstationname* 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.
- To unlock all other objects, see
 - the *Dynamic Workload Console User's Guide*, section about Listing object definitions in the database.
 - the *Dynamic Workload Console User's Guide*, section about Listing object definitions in the database.

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.

workstationname

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` variable 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.
- To delete all other objects, see
 - the *Dynamic Workload Console User's Guide*, section about Listing object definitions in the database.
 - the *Dynamic Workload Console User's Guide*, section about Listing object definitions in the database.

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.

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 427](#) 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**Before:**

AA_BB_WS1

AA_BB_WS2

After:

/AA/BB/WORKSTATION1

/AA/BB/WORKSTATION2

```
composer rename ws @_@_ws@ /@/@/workstation@
```

Move a variable table to a child folder using a combination of the "@" and "?" wildcards**Before:**

AA_BB_EF_GHI

AB_CD_GH_IJK

BB_CC_EF_GHI

```
composer rename vt ??_??_@ /??/??/@
```

After:

/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)



Move them from the root to a first-level folder where a wildcard character is used to represent the root and is replaced with a blank

Before: @#/@/@, where @#/@/@ represents WS_NAME#JSNAME and JSNAME is in the root folder

```
composer rename js @#/@/@ @#/F_@/J_@
```

After: WS_NAME#/F_/J_JSNAME

Move them from the root into a hierarchy of folders named using both the prefix and the suffix in the object name

Before: WS_NAME#ABC_DEF_JSNAME_GHI

```
composer rename js @#ABC_DEF_@_GHI @#/GHI/ABC/DEF/@
```

After: WS_NAME#GHI/ABC/DEF/JSNAME

Before: WS_NAME#ABC_DEF_JSNAME

```
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_DADPT, 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 440](#).

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 350](#) 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. If you are updating or upgrading your fault-tolerant agents to version 9.5 Fix Pack 2 or later, these updates are especially important if you plan to use the command line on the fault-tolerant agents to perform operations on the objects in folders.

If you updated your 9.5 or 9.5.0.x agents to version 9.5 Fix Pack 2 or later, then see the topic about completing the security configuration in the section about applying a fix pack in the *Planning and Installation Guide*. The topic refers to updating the security file on the master domain manager and backup master domain manager for the classic security model, but the same updates apply to a fault-tolerant agent.

If you upgraded your agents from a previous product version, for example, 9.4.x, to 9.5 Fix Pack 2 or later, then see the topic about completing the security configuration in the upgrade section in the *Planning and Installation Guide*. The topic refers to updating the security file on the master domain manager and backup master domain manager for the classic security model, but the same updates apply to a fault-tolerant agent.

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 461](#).

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 461](#). 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 471](#) 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 468](#).

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 60. 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 60. 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 the following objects: the name of the external job stream, the job defined in the 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 <code>\${password:ws#user}</code>. 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	Object definition	Table name Variable name	Variable table is created in the database.
Job stream variable tables			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 60. 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 60. 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 60. 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 61. Resolving the mapping file

Definitions file	Mapping file	Reference information file
<pre> <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> </pre>	<pre> #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 </pre>	<pre> 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 </pre>

Table 61. Resolving the mapping file

(continued)

Definitions file	Mapping file	Reference information file
<pre></model:job> </model:jobs> </model:JobStream></pre>	<pre>Agent WORKSTATION_MDM112097_1= MDM112097_1 #This workstation is of type Broker WORKSTATION_MDM112097_DWB= MDM112097_DWB</pre>	

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 462](#). 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 468](#).
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 471](#) for more details about the command usage and syntax. See the topic about importing a workload application template from the Dynamic 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 471](#).

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 regular expressions to customize the mapping file, see [Using regular expressions to modify the mapping file on page 468](#).

-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 478
- [Running commands from conman](#) on page 485
- [JOBSELECT](#) Selecting jobs in commands on page 489
- [JSSELECT](#) Selecting job streams in commands on page 500
- [CONMAN COMMANDS](#) on page 509

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 552.
- `rerunsucc`. For more information, see [RERUNSUC](#) on page 566.

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 **twsl.loggers.trcConman.level** property, available in the `Conman.ini`.

The `Conman.ini` file is located in the following path:

On UNIX™ operating systems

`TWA_DATA_DIR/ITA/cpa/config`

On Windows™ operating systems

`TWA_home\TWS\ITA\cpa\config`

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 `twsl_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` for Bourne and Korn shells
- `./TWS_home/tws_env.csh` for C shells

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, you must open the DOS command prompt shell with the "Run As Administrator" option to run **conman** on your workstation as a generic user different from Administrator or HCL Workload Automation user.

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 390](#) 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=cancelFilters on all jobs that are in **canceled** state.

Delimiters and special characters

Table 62: Delimiters and special characters for conman on page 487 lists characters having special meanings in **conman** commands:

Table 62. 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: <code>sked1(1100 03/05/2023).@~priority=0</code>
~	A delimiter used to select objects for commands. It adds an attribute the object must not have. For example: <code>sked1(1100 03/05/2023).@~priority=0</code>
;	Argument delimiter. For example: <code>;info;offline</code>
,	Repetition and range delimiter. For example: <code>state=hold,sked,pend</code>
=	Value delimiter. For example: <code>state=hold</code>
:! 	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: <code>!ls or :ls</code>
*	Comment prefix. The prefix must be the first character on a command line or following a command delimiter. For example: <code>*comment</code> or

Table 62. Delimiters and special characters for conman (continued)

Char.	Description
	<code>sj& *comment</code>
>	Redirects command output to a file and overwrites the contents of that file. If the file does not exist, it is created. For example: <code>sj> joblist</code>
>>	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: <code>sj >> joblist</code>
	Pipes command output to a system command or process. The system command is run whether or not output is generated. For example: <code>sj grep ABEND</code>
	Pipes command output to a system command or process. The system command is not run if there is no output. For example: <code>sj grep ABEND</code>

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 491](#).

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),(0AAAAAAAAAAAAABTB)]
  (enter "y" for yes, "n" for no)?y
Command forwarded to batchman for WK1#J1[(0600 03/04/19),(0AAAAAAAAAAAAABTB)]
Process WK1#J1[(1010 03/04/19),(0AAAAAAAAAAAAABTC)]
  (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 0AAAAAAAAAAAAABTB 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_ids* and not the *jobstreamnames*. 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 alphanumeric 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),(0AAAAAAAAAABTB)]
(enter "y" for yes, "n" for no)?y
```



```
Command forwarded to batchman for WK1#J1[(0600 03/04/19),(0AAAAAAAAAAAAABTB)]
Process WK1#J1[(1010 03/04/19),(0AAAAAAAAAAAAABTC)]
(enter "y" for yes, "n" for no)
```

In the output only the job stream instance scheduled on (0600 03/04/19) and with identifier 0AAAAAAAAAAAAABTB 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 are 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_ids* and not the *jobstreamnames*. 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 552](#).
- `rerunsucc`. For more information, see [RERUNSUCCE on page 566](#).

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 63. List of conman commands

Command	Short Form	Description	Type
<code>adddep { job sched }</code>	<code>adj ads</code>	Adds job or job stream dependencies.	F
<code>altjob</code>	<code>aj</code>	Modifies a job in the plan before it runs.	F
<code>altpass</code>		Alters a user object definition password.	F
<code>altpri</code>	<code>ap</code>	Alters job or job stream priorities.	F
<code>cancel { job sched }</code>	<code>cj cs</code>	Cancels a job or a job stream.	F
<code>checkhealthstatus</code>	<code>chs</code>	Invokes chkhlst service to check if mailbox can be successfully read by mailman or if there are errors in the mailbox header.	MFS
<code>chfolder</code>	<code>cf</code>	Changes the working directory.	F
<code>confirm</code>	<code>conf</code>	Confirms job completion.	F
<code>console</code>	<code>cons</code>	Assigns the HCL Workload Automation console.	FS
<code>continue</code>	<code>cont</code>	Ignores the next error.	FS
<code>deldep { job sched }</code>	<code>ddj dds</code>	Deletes job or job stream dependencies.	F
<code>deployconf</code>	<code>deploy</code>	Gets the latest monitoring configuration for the event monitoring engine on the workstation.	FS
<code>display { file job sched }</code>	<code>df dj ds</code>	Displays files, jobs, and job streams.	FS
<code>exit</code>	<code>e</code>	Exits conman .	FS
<code>fence</code>	<code>f</code>	Sets HCL Workload Automation job fence.	F
<code>help⁽¹⁾</code>	<code>h</code>	Displays command information.	FS

Table 63. List of conman commands (continued)

Command	Short Form	Description	Type
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
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

Table 63. List of conman commands (continued)

Command	Short Form	Description	Type
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
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
<i>system-command</i>		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.
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=[hhmm] [onmaxdur action]

mindur=[hhmm] [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 `J0BA` in job stream `MLN#/FOLDER2/NEW_TEST(0900 02/19/18)`, run the following command:

```
adj=MLN#/FOLDER2/NEW_TEST(0900 02/19/18).J0BA ; 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 **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`.

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 CUA#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:

- \ – Backslash
- / – Slash (or forward slash)
- & – Ampersand

- `+` – Plus
- `=` – Equals
- `#` – Hash (or number sign)

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 481](#) 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 64. 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	without output conditions SUCC with output conditions SUCC_P and selected output conditions are set to satisfied	without output conditions ABEND with output conditions ABEND_P and selected output conditions are set to satisfied
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

Table 64. State change after confirm command (continued)

Initial Job State	State after confirm ;succ	State after confirm ;abend
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.
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

Table 64. State change after confirm command (continued)

Initial Job State	State after confirm ;succ	State after confirm ;abend
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=[*hh:mm*] [**onmaxdur** *action*]

mindur=[*hh:mm*] [**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 if 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, 2025 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/gll" 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#apwkl;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 Date	Actual Date	Start Time	Log Date	Run Num	Size	Log 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	Size	Log	
Date	Date Time	Date	Num		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	Size	Log	
Date	Date Time	Date	Num		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	Size	Log	
Date	Date Time	Date	Num		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* 0AAAAAAAAAAAAABSE, stored in folder **myfolder**, from all of its dependencies, run the following command:

```
rs myfolder/0AAAAAAAAAAAAABSE; 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 `arpm`, run the following command:

```
reply arpm;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 65: Recovery options retrieval criteria on page 563](#) describes the criteria by which recovery options in the original and **from** job definition are retrieved.

Table 65. 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 564](#) 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 66: Successors status on page 567](#) for a complete list.

Table 66. 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
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

Table 66. Successors status (continued)

Status	Expected behavior
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	10.2.5	US/Pacific	Linux 2.6.5-7.191-s390 #1 SM
FTA1	10.2.5		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-CPU

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_JOB1 HOLD 10 (11/26)                                parms FILE_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. 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 486](#).

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(060002/24/18). JB1 for job J0BA 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 JOBCHECKCALC goes into either ABEND or FAIL state:

```
S_MDM      #JSHOLIDAYS 0600 09/23 ***** READY 10 (00:01)
              (S_AGT#) JOBCHECKCALC          HOLD 19
              (S_AGT#) JOBVACS                HOLD 10

JOBCHECKCALC 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          {0AAAAAAAAAAAAACRZ}
              JDOC SUCC 10 11/26 00:01          0 #J25565
```

The job stream identifier 0AAAAAAAAAAAAACRZ 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. @;login
```

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).JOB
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}
JOB1 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),(0AAAAAAAAAAAAABQM)].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 `jsdl` 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, 2024 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, 2024 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 bankmngnr
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/@

Workstation      Job Stream      SchedTime      State Pr (Est) (Est)  Jobs  Sch
>>              /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 **#Jnnnnn**.
- 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 **#Jnnnnn**.

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 **#Jnnnnn**.

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:

CPU	Schedule	SchedTime	Job	State	Pr	(Est) Start	(Est) Elapse	CP	Crit 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),
      (0AAAAAAAAAAAAABST)]) Are you ready to process job1?
INACT 5(CPUA#SCHED_12[(1010 03/12/18),(0AAAAAAAAAAAAABSU)])
      Are you ready to process job2?
INACT 7(CPUA#SCHED_22[(0600 03/12/18),(0AAAAAAAAAAAAABTR)])
      Are you ready to process job3?
```

To display the status of all **mis prompts** 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),(0AAAAAAAAAAAAABTR)])
      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	EIapse	RetCode
CPUA	#JOBSLOTS	Dependencies are:							
FTAA	#SCHED_F+	0600	03/04	*****	HOLD	55	03/04		[03/04/18];#33
	(CPUA#)	JOBMDM	HOLD	30	03/04				#1(PRMT3);-16 JOBSLOTS-
FTAA	#SCHED_F+	1010	03/04	*****	HOLD	55	03/04		[03/04/18];#34
	(CPUA#)	JOBMDM	HOLD	30	03/04				#1(PRMT3);-16 JOBSLOTS-
FTAA	#SCHED_F+	0600	03/05	*****	HOLD	55	03/05		[03/04/18];#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 486](#).

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   {0AAAAAAAAAAAAACRZ}
```

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),(0AAAAAAAAAAAAABVY)] 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.

Workstation	Job Stream	SchedTime	State	Pr	(Est) Start	(Est) Elapse	Jobs #	OK	Sch 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									

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.

CANCEL

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).JOB1 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 (<) is displayed.
- 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, 2024 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 then the domain manager itself is stopped and the command continues 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 afile**", 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 afile";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 tzone] [+n days | mm/dd[/yy]] | [absolute | abs]

confirmed

critical

deadline=time[timezone | tz tzone][+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 `twc_env.cmd`, whose path contains a blank, on a Windows workstation run:

- In interactive mode:

```
sbf "\"C:\Program Files\HCL\TWS\lucaMDM\twc_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\twc_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[;...]]
 [;variable=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]`

variable=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[;...]]
    [;variable=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 `STTR01234567890`.

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 [mdd[/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.

variable=*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 `CLConfig.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
TELOP>*****
TELOP>*   TWS will be stopped at           *
TELOP>*   4:30 for 15 minutes.             *
TELOP>*****
TELOP>//
```

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, 2025 All rights reserved.
# * Trademark of International Business Machines
# ** Trademark of HCL Technologies Limited
#####
Installed for user "twuser".
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, Orchestration CLI, 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.

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 67. OQL syntax keywords

Keywords	Description
AND	Returns the results that satisfy all the conditions separated by AND . It is case insensitive.

Table 67. OQL syntax keywords (continued)

Keywords	Description
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: • @: matches all characters. • ?: matches a single character in a specific position.

Table 67. OQL syntax keywords (continued)

Keywords	Description
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 <code>true</code> .
false	Represents the Boolean value <code>false</code> .
null	Represents the <code>null</code> 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 68. OQL syntax fields for workstation

Field name	Description
activeFlags z/OS	The flags of the workstation. Supported values are: <pre>AUTOMATION WAIT VIRTUAL JOB_SETUP WTO zCENTRIC DYNAMIC STARTED_TASK PENDING_OFFLINE CONTROL_ON_SERVERS</pre> Example: <pre>activeFlags LIKE '@'</pre>
activeStates	The statuses of the workstation. Supported values are: Distributed <pre>LINKED UNLINKED FULLY_LINKED NOT_APPLICABLE</pre> z/OS <pre>FULLY_LINKED LINKED UNLINKED WS_STATUS_ACTIVE WS_STATUS_OFF WS_STATUS_FAILED WS_STATUS_UNKNOWN</pre>

Table 68. OQL syntax fields for workstation (continued)

Field name	Description
	Example: <pre>activeStates IN ['LINKED', 'FULLY_LINKED']</pre>
available z/OS	The workstation is flagged as available. Supported values are <code>true</code> or <code>false</code>. Example: <pre>available IN ['true']</pre>
description z/OS	The description of the workstation. Example: <pre>description IN ['General Workstation']</pre>
domainName Distributed	The name of the domain within which the workstation is hierarchically organized. Example: <pre>domainName != 'MASTERDM'</pre>
folder Distributed	The name of the folder where the workstation is saved. Example: <pre>folder LIKE '/@A/'</pre>
id	The identifier of the workstation. Example: <pre>id = 'ca15f24a-0e6b-3f1b-9251-854d8d0d3639'</pre>
jobmanUp Distributed	Identifies whether the agent is running or not. Supported values are <code>true</code> or <code>false</code>. Example: <pre>jobmanUp = true</pre>
name	The name of the workstation. Example: <pre>name = 'wksrome'</pre>
remoteEngineType z/OS	The type of remote engine on which the current remote workstation is defined. Supported values are <code>z</code> for z/OS engines, and <code>d</code> for distributed engines.

Table 68. OQL syntax fields for workstation (continued)

Field name	Description
	Example: <pre>remoteEngineType LIKE '@'</pre>
reporting z/OS	The type of reporting. Supported values are: <pre>AUTOMATIC MANUAL_START_STOP MANUAL_COMPLETION_ONLY NON_REPORTING</pre> Example: <pre>reporting LIKE 'AUTOMATIC'</pre>
resource1Info.usedForControl z/OS	The job must be submitted only if the requested resource amount is available. Supported values are <code>true</code> or <code>false</code>. Example: <pre>resource1Info.usedForControl IN ['true']</pre>
resource2Info.usedForControl z/OS	The job must be submitted only if the requested resource amount is available. Supported values are <code>true</code> or <code>false</code>. Example: <pre>resource2Info.usedForControl IN ['false']</pre>
resource2Info.usage z/OS	The type of usage defined for resource 2. Supported values are: <pre>PLANNING CONTROL PLANNING_AND_CONTROL NEITHER_PLANNING_NOR_CONTROL</pre> Example: <pre>resource2info.usage IN ['NEITHER_PLANNING_NOR_CONTROL']</pre>
type	The type of workstation. Supported values are: Distributed <pre>AGENT POOL DYNAMIC_POOL REMOTE_ENGINE MANAGER FTA XA SA WORKLOAD_BROKER DOMAIN_MANAGER</pre> z/OS <pre>GENERAL REM_ENG COMPUTER PRINTER</pre> Example: <pre>type IN ['FTA']</pre>

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 69. OQL syntax fields for job stream

Field name	Description
aliasJobStreamName Distributed	The alias name of the job stream. Example: <pre>aliasJobStreamName LIKE '@rome'</pre>
commonStatus	The external status of the job stream in plan. The supported values are: Distributed <pre>WAITING READY RUNNING SUCCESSFUL ERROR CANCELED HELD UNDECIDED BLOCKED SUPPRESS</pre> z/OS <pre>WAITING READY RUNNING SUCCESSFUL ERROR CANCELED HELD UNDECIDED SUPPRESS</pre> Example: <pre>commonStatus IN ['ERROR','BLOCKED']</pre>
compositeStatus.status	The internal status of the job stream in plan. Supported values are: Distributed <pre>HOLD READY FENCE INTRO FAIL EXEC WAIT SUPPR END SUCC SUCCP ABEND CANCL CANCLP ABENP PEND SUSP STUCK EXTRN</pre> z/OS <pre>WAIT COMPLETE ERROR STARTED UNDECIDED DELETED</pre> Example: <pre>compositeStatus.status NOT IN ['SUCC','ABEND','FAIL']</pre>
dependencies.dependencyStatus Distributed	The status of the job stream dependency. Supported values are:

Table 69. OQL syntax fields for job stream (continued)

Field name	Description
	SATISFIED UNSATISFIED UNDECIDED Example: <pre>dependencies.dependencyStatus = 'SATISFIED'</pre>
dependencies.jobId Distributed	The identifier of the job instance on which the job stream depends. Example: <pre>dependencies.jobId = 'ca15f24a-0e6b-3f1b-9251-854d8d0d3639'</pre>
dependencies.folderOriginalName Distributed	The name of the folder where the job dependency is saved. Example: <pre>dependencies.folderOriginalName NOT LIKE '@JS_@'</pre>
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: <pre>dependencies.jobStreamName LIKE '?a'</pre>
dependencies.jobName Distributed	The name of the job on which the job stream depends. Example: <pre>dependencies.jobName LIKE '@job1'</pre>
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: <pre>dependencies.jobStreamFolder NOT LIKE '/@root/'</pre>
dependencies.jobStreamId Distributed	For dependencies on jobs, the id of the job stream to which the job belongs.

Table 69. OQL syntax fields for job stream (continued)

Field name	Description
	For dependencies on job streams, the id of the dependent job stream. Example: <pre>dependencies.jobStreamId = 'ca15f24a-0e6b-3f1b-9251-854d8d0d3639'</pre>
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: <pre>dependencies.jobStreamName LIKE 'js_exec'</pre>
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: <pre>dependencies.jobStreamWorkstation LIKE '@wks_rome@'</pre>
dependenciesStats.numberOfDependencies Distributed	The number of dependencies of the job stream. Supported operators are = and !=. Example: <pre>dependenciesStats.numberOfDependencies = '3'</pre>
dependenciesStats.numberOfFileDependencies Distributed	The number of dependencies of the job stream of type File. Supported operators are = and !=. Example: <pre>dependenciesStats.numberOfFileDependencies = '1'</pre>
dependenciesStats.numberOfInternetworkDependencies Distributed	Number of dependencies of the job stream of type Internetwork dependency. Supported operators are = and !=. Example: <pre>dependenciesStats.numberOfInternetworkDependencies != '10'</pre>

Table 69. OQL syntax fields for job stream (continued)

Field name	Description
dependenciesStats.numberOfJobDependencies Distributed	The number of job dependencies of the job stream. Supported operators are = and !=. Example: <pre>dependenciesStats.numberOfJobDependencies != '7'</pre>
dependenciesStats.numberOfJobStreamDependencies Distributed	The number of job stream dependencies of the job stream. Supported operators are = and !=. Example: <pre>dependenciesStats.numberOfJobStreamDependencies = '4'</pre>
dependenciesStats.numberOfPromptDependencies Distributed	The number of dependencies of the job stream of type Prompt. Supported operators are = and !=. Example: <pre>dependenciesStats.numberOfPromptDependencies != '9'</pre>
dependenciesStats.numberOfResourceDependencies Distributed	The number of dependencies of the job stream of type Resource. Supported operators are = and !=. Example: <pre>dependenciesStats.numberOfResourceDependencies = '3'</pre>
dependenciesStats.numberOfUnresolvedDependencies Distributed	The number of dependencies of the job stream that have not been satisfied. Supported operators are = and !=. Example: <pre>dependenciesStats.numberOfUnresolvedDependencies != '8'</pre>
folder Distributed	The name of the folder within which the job stream is saved. Example: <pre>folder = '/JSFOLDER'</pre>
folderId Distributed	The identifier of the folder within which the job stream is saved. Example: <pre>folderId = 'be11g38a-0u7m-5f1d-5594-458d8d0d6541'</pre>
id Distributed	The identifier of the job stream.

Table 69. OQL syntax fields for job stream (continued)

Field name	Description
	Example: <pre>id = 'ca15f24a-0e6b-3f1b-9251-854d8d0d3639'</pre>
jobs.dependencies.jobStreamId Distributed	The identifier of the job stream to which the dependency of the job belongs. Example: <pre>jobs.dependencies.jobStreamId = 'ca15f26a-0e9b-3cpl-5594-999d8d0d3639'</pre>
jobStreamStats.numberOfAbendedJob Distributed	The number of jobs in ABEND status within the job stream. Supported operators are = and !=. Example: <pre>jobStreamStats.numberOfAbendedJob != '8'</pre>
jobStreamStats.numberOfCurrentNodes Distributed	The number of current nodes within the job stream. Supported operators are = and !=. Example: <pre>jobStreamStats.numberOfCurrentNodes = '2'</pre>
jobStreamStats.numberOfExecutingJob Distributed	The number of jobs that are currently running within the job stream. Supported operators are = and !=. Example: <pre>jobStreamStats.numberOfExecutingJob != '6'</pre>
jobStreamStats.numberOfFailedJob Distributed	The number of jobs that are in FAIL status within the job stream. Supported operators are = and !=. Example: <pre>jobStreamStats.numberOfFailedJob = '1'</pre>
jobStreamStats.numberOfJob Distributed	The total amount of jobs within the job stream. Supported operators are = and !=. Example: <pre>jobStreamStats.numberOfJob != '3'</pre>
jobStreamStats.numberOfNotRunningJob Distributed	The number of jobs that are currently not running within the job stream. Supported operators are = and !=. Example:

Table 69. OQL syntax fields for job stream (continued)

Field name	Description
	<code>jobStreamStats.numberOfNotRunningJob = '10'</code>
<code>jobStreamStats.numberOfSkelJob</code> Distributed	The number of jobs in <code>SKEL</code> status within the job stream. Supported operators are <code>=</code> and <code>!=</code>. Example: <pre>jobStreamStats.numberOfSkelJob = '1'</pre>
<code>jobStreamStats.numberOfSuccessfulJob</code> Distributed	The number of jobs in <code>SUCCESS</code> status within the job stream. Supported operators are <code>=</code> and <code>!=</code>. Example: <pre>jobStreamStats.numberOfSuccessfulJob != '7'</pre>
<code>jobStreamStats.numberOfUndecidedJob</code> Distributed	The number of jobs in unknown status within the job stream. Supported operators are <code>=</code> and <code>!=</code>. Example: <pre>jobStreamStats.numberOfUndecidedJob = '1'</pre>
<code>jsDefinitionFlags.carriedForward</code> Distributed	The job stream is flagged as <code>carried forward</code>. Supported values are <code>true</code> or <code>false</code>. Example: <pre>jsDefinitionFlags.carriedForward = false</pre>
<code>jsOptions.carryForward</code> Distributed	The job stream is to be carried forward. Supported values are <code>true</code> or <code>false</code>. Example: <pre>jsOptions.carryForward = true</pre>
<code>jsOptions.monitored</code>	The job stream is to be monitored. Supported values are <code>true</code> or <code>false</code>. Example: <pre>jsOptions.monitored = false</pre>
<code>key</code> Distributed	The complete key that identifies the job stream. Example: <pre>key LIKE '@jskey'</pre>
<code>limit</code> Distributed	The limit value set for the job stream. Example:

Table 69. OQL syntax fields for job stream (continued)

Field name	Description
	<code>limit > '1' AND limit <= '3'</code>
name	The name of the job stream. Example: <code>name NOT LIKE '@JS@'</code>
occurrenceToken z/OS	The unique token that identifies the job stream. <code>occurrenceToken IN ['e079702a9547afc2']</code>
originalJobStreamName	The name of the job stream before any change was performed. Example <code>originalJobStreamName NOT LIKE 'job@'</code>
origPriority Distributed	The original priority value set for the job stream. Example: <code>origPriority = '10'</code>
pendingPredecessor Distributed	The job stream has predecessor instances that are pending. Supported values are <code>true</code> or <code>false</code> . Example: <code>pendingPredecessor = false</code>
priority	The priority value set for the job stream. Example: <code>priority = '101'</code>
schedTime	The scheduled time of the job stream. Example: <code>schedTime >= '2025-03-01T00:00:00.000Z'</code>
timeInfo.actualEndTime	The time at which the job stream ended, that is the real end time. Example: <code>timeInfo.actualEndTime < '2025-03-02T10:35:22.000Z'</code>
timeInfo.actualStartTime	The time at which the job stream started, that is the real start time.

Table 69. OQL syntax fields for job stream (continued)

Field name	Description
	Example: <pre>timeInfo.actualStartTime > '2025-02-27T11:15:00.000Z'</pre>
timeInfo.elapsedTime Distributed	The elapsed time of the job stream, that is the real duration. Supported operators are = and !=. Example: <pre>timeInfo.elapsedTime != 'PT6H3S'</pre>
timeInfo.estimatedDuration Distributed	The estimated duration of the job stream. Supported operators are = and !=. Example: <pre>timeInfo.estimatedDuration = 'PT9H45S'</pre>
timeRestrictions.deadlineTime	The time within which the job stream should complete. Example: <pre>timeRestrictions.deadlineTime != '2025-09-13T09:00:00.000Z'</pre>
timeRestrictions.startTime	The start time set for the job stream. Example: <pre>timeRestrictions.startTime > '2025-04-27T10:15:00.000Z'</pre>
timeRestrictions.untilTime Distributed	The latest time the job stream can be launched. Example: <pre>timeRestrictions.untilTime != '2025-02-27T11:35:16.000Z'</pre>
workstation	The workstation where the job stream runs. Example: <pre>workstation = '/WSFOLDER/WS'</pre>

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 70. OQL syntax fields for jobs

Field name	Description
dependencies.dependencyStatus Distributed	The status of the job dependency. Supported values are: <pre>SATISFIED UNSATISFIED UNDECIDED</pre> Example: <pre>dependencies.dependencyStatus != 'UNDECIDED'</pre>
dependencies.jobId Distributed	The identifier of the job instance on which the job depends. Example: <pre>dependencies.jobId = 'ca15f24a-0e6b-3f1b-9251-854d8d0d3639'</pre>
dependencies.jobKey.name Distributed	The name of the job dependency. Example: <pre>dependencies.jobKey.name NOT LIKE '@JS_@'</pre>
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: <pre>dependencies.jobStreamFolder = '/folder/js'</pre>
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: <pre>dependencies.jobStreamId = 'ca15f24a-0e6b-3f1b-9251-854d8d0d3639'</pre>
dependencies.jobStreamName Distributed	For dependencies on jobs, the name of the job stream to which the job belongs.

Table 70. OQL syntax fields for jobs (continued)

Field name	Description
	For dependencies on job streams, the name of the job stream. Example: <pre>dependencies.jobStreamName LIKE 'J?S@'</pre>
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: <pre>dependencies.jobStreamWorkstation LIKE 'wks_@'</pre>
dependenciesStats.numberOfFileDependencies Distributed	The number of dependencies of the job of type <code>File</code>. Supported operators are <code>=</code> and <code>!=</code>. Example: <pre>dependenciesStats.numberOfFileDependencies = '2'</pre>
dependenciesStats.numberOfInternetworkDependencies Distributed	Number of dependencies of the job of type <code>Internetwork</code> dependency. Supported operators are <code>=</code> and <code>!=</code>. Example: <pre>dependenciesStats.numberOfInternetworkDependencies != '4'</pre>
dependenciesStats.numberOfJobDependencies Distributed	The number of job dependencies of the job. Supported operators are <code>=</code> and <code>!=</code>. Example: <pre>dependenciesStats.numberOfJobDependencies = '3'</pre>
dependenciesStats.numberOfJobStreamDependencies Distributed	The number of job stream dependencies of the job. Supported operators are <code>=</code> and <code>!=</code>. Example: <pre>dependenciesStats.numberOfJobStreamDependencies != '8'</pre>
dependenciesStats.numberOfPromptDependencies Distributed	The number of dependencies of the job of type <code>Prompt</code>. Supported operators are <code>=</code> and <code>!=</code>. Example:

Table 70. OQL syntax fields for jobs (continued)

Field name	Description
	<code>dependenciesStats.numberOfPromptDependencies = '1'</code>
<code>dependenciesStats.numberOfResourceDependencies</code> Distributed	The number of dependencies of the job of type <code>Resource</code> . Supported operators are <code>=</code> and <code>!=</code> . Example: <code>dependenciesStats.numberOfResourceDependencies != '5'</code>
<code>dependenciesStats.numberOfUnresolvedDependencies</code> Distributed	The number of dependencies of the job that have not been satisfied. Supported operators are <code>=</code> and <code>!=</code> . Example: <code>dependenciesStats.numberOfUnresolvedDependencies = '2'</code>
<code>folder</code> Distributed	The name of the folder within which the job is saved. Example: <code>folder LIKE '/@A@'</code>
<code>folderId</code> Distributed	The identifier of the folder within which the job is saved. Example: <code>folderId = 'be1lg38a-0u7m-5f1d-5594-458d8d0d6541'</code>
<code>id</code> Distributed	The identifier of the job. Example: <code>id = 'ca15f24a-0e6b-3f1b-9251-854d8d0d3639'</code>
<code>jobDefinition.isCommand</code> Distributed	The job definition is a command. Supported values are <code>true</code> if the job definition is a command, or <code>false</code> if the job definition is a script file. Example: <code>jobDefinition.isCommand = true</code>
<code>jobDefinition.key.name</code> Distributed	The name of the job definition. Example: <code>jobDefinition.key.name LIKE '@jobdef1_'</code>
<code>jobDefinition.recoveryOption</code> Distributed	The recovery options set for the job definition. Supported values are: <code>STOP CONTINUE RERUN</code>

Table 70. OQL syntax fields for jobs (continued)

Field name	Description
	Example: <pre>jobDefinition.recoveryOption = 'CONTINUE'</pre>
jobDefinition.taskType Distributed	The task type of the job definition. Example: <pre>jobDefinition.taskType = 'executable'</pre>
jobDefinition.userLogin Distributed	The username of the job definition. Example: <pre>jobDefinition.userLogin != 'user3'</pre>
jobDefinition.workstation Distributed	The name of the workstation on which the job definition runs. Example: <pre>jobDefinition.workstation NOT LIKE '/wks@'</pre>
jobDefinition.workstationKey Distributed	The name of the workstation on which the job definition runs. Example: <pre>jobDefinition.workstationKey LIKE 'rome1@'</pre>
jobDefinitionFlags.monitored z/OS	The job must be monitored. Supported values are <code>true</code> or <code>false</code>.
jobOptions.critical Distributed	The <code>critical</code> option is set for the job. Supported values are <code>true</code> or <code>false</code>. Example: <pre>jobOptions.critical = true</pre>
jobOptions.every Distributed	The <code>every</code> option is set for the job. Supported values are <code>true</code> or <code>false</code>. Example: <pre>jobOptions.every = false</pre>
jobOptions.noOperation	The <code>No operation</code> option is set for the job. Supported values are <code>true</code> or <code>false</code>. Example: <pre>jobOptions.noOperation = true</pre>

Table 70. OQL syntax fields for jobs (continued)

Field name	Description
jobOptions.requiresConfirmation	The <code>Requires confirmation</code> option is set for the job. Supported values are <code>true</code> or <code>false</code>. Example: <pre>jobOptions.requiresConfirmation = false</pre>
jobruns.actualWorkstation Distributed	The name of the workstation where the job actually runs. Example: <pre>jobruns.actualWorkstation NOT LIKE '@wks@'</pre>
jobruns.commonStatus	The external status of the job runs. Supported values are: Distributed <pre>WAITING READY RUNNING SUCCESSFUL ERROR CANCELED HOLD UNDECIDED BLOCKED SUPPRESS</pre> z/OS <pre>WAITING READY RUNNING SUCCESSFUL ERROR CANCELED HOLD UNDECIDED INTERRUPTED COMPLETED SUPPRESS_BY_COND</pre> Example: <pre>jobruns.commonStatus IN ['WAITING', 'BLOCKED']</pre>
jobruns.compositeStatus.status	The internal status of the job runs. Supported values are: Distributed <pre>HOLD READY FENCE INTRO FAIL EXEC WAIT SUPPR END SUCC SUCCP ABEND CANCL CANCLP ABENP PEND SUSP STUCK EXTRN</pre> z/OS <pre>WAIT ARRVING READY NONREPPRED STARTED INTRPT CANCL UNDCDED COMPLT ERROR SUCCP SUPPR</pre> Example: <pre>jobruns.compositeStatus.status NOT IN ['SUPPR']</pre>
jobruns.elapsedTime Distributed	The elapsed time of the job run instance, that is the real duration. Example: <pre>jobruns.elapsedTime = 'PT5H9M'</pre>
jobruns.estimatedDuration Distributed	The estimated duration of the job run instance.

Table 70. OQL syntax fields for jobs (continued)

Field name	Description
	Example: <pre>jobruns.estimatedDuration = 'PT9H21M'</pre>
jobruns.jobNumber	The total number of jobs runs. Supported operators are = and !=. Example: <pre>jobruns.jobNumber = '3'</pre>
jobruns.maxDurationGone Distributed	The duration of the run is higher than the maximum duration value. Supported values are true or false. Example: <pre>jobruns.maxDurationGone = true</pre>
jobruns.minDurationNotReached Distributed	The duration of the run is lower than the minim duration value. Supported values are true or false. Example: <pre>jobruns.minDurationNotReached = false</pre>
jobruns.reruntype Distributed	The type of rerun. Supported values are: <pre>RERUN RECOVERY_RERUN RERUN_EVERY RERUN_FROM RECOVERY REGULAR RERUN_STEP</pre> Example: <pre>jobruns.reruntype IN ['RERUN', 'RECOVERY']</pre>
jobruns.returnCode Distributed	The return code of the job run instance. Example: <pre>jobruns.returnCode = '1'</pre>
jobruns.step Distributed	The alias name of the job run instance. Example: <pre>jobruns.step = 'JOBNAME'</pre>
jobruns.timeInfo.actualEndTime Distributed	The time at which the job run instance ended. Example: <pre>jobruns.timeInfo.actualEndTime = '2025-12-01T23:00:00.000Z'</pre>
jobruns.timeInfo.actualStartTime Distributed	The time at which the job run instance started.

Table 70. OQL syntax fields for jobs (continued)

Field name	Description
	Example: <pre>jobruns.timeInfo.actualStartTime != '2025-12-01T23:00:00.000Z'</pre>
jobruns.errorCode z/OS	The error code of the first operation in error status within the job run. Example: <pre>jobruns.errorCode LIKE '0S@'</pre>
jobs.jobDefinition.workstationName	The name of the workstation where the job definition referenced by the job instance is saved. Example: <pre>jobs.jobDefinition.workstationName NOT LIKE '@A'</pre>
jobStreamId	The identifier of the job stream within which the job is saved. Example: <pre>jobStreamId = 'ca15f24a-0e6b-3f1b-9251-854d8d0d3639'</pre>
jobStreamName	The name of the job stream within which the job is saved. Example: <pre>jobStreamName NOT LIKE 'A?@'</pre>
jobStreamWorkstation	The name of the workstation on which runs the job stream within which the job is saved. Example: <pre>jobStreamWorkstation != 'WKS_1'</pre>
name	The name of the job. Example: <pre>name LIKE '@A?@'</pre>
priority	The priority value set for the job. Example: <pre>priority > '9' AND priority <= '100'</pre>
jobruns.timeDependent z/OS	The evaluation of the AT time value of the dependency is active. Supported values are <code>true</code> or <code>false</code>.

Table 70. OQL syntax fields for jobs (continued)

Field name	Description
	Example: <pre>jobruns.timeDependent IN ['false']</pre>
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: <pre>SUPPRESS CONTINUE CANCEL</pre> Example: <pre>timeRestrictions.untilAction = 'SUPPRESS'</pre>
timeRestrictions.untilTime Distributed	The latest time a job can be launched. Example: <pre>timeRestrictions.untilTime = '2025-03-02T10:35:22.000Z'</pre>
workstation	The workstation where the job runs. Example: <pre>workstation LIKE '@wks_rome1@'</pre>

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 71. OQL syntax fields for resources

Field name	Description
id	The id of the resource. Example: <pre>id = 'ca15f24a-0e6b-3f1b-9251-854d8d0d3639'</pre>
name	The name of the resource. Example: <pre>name LIKE 'res@'</pre>
folder	The folder of the resource.

Table 71. OQL syntax fields for resources (continued)

Field name	Description
	Example: <pre>folder NOT LIKE '@A'</pre>
workstation	The workstation of the resource. Example: <pre>workstation LIKE '@_1'</pre>
workstationFolder	The folder where the workstation of the resource is saved. Example: <pre>workstationFolder LIKE '/ws_rome@'</pre>
quantity	The entire quantity of the resource. Example: <pre>quantity = '5'</pre>
quantityInUse	The quantity of the resource that is currently in use. Example: <pre>quantityInUse != '2'</pre>
quantityAvailable	The quantity of the resource that is available. Example: <pre>quantityAvailable = '4'</pre>
status	The status of the resource. Supported values are: <pre>AVAILABLE NOT_AVAILABLE UNDECIDED</pre> Example: <pre>status IN ['AVAILABLE']</pre>

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 on 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 on 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 682](#)

[Running commands from Orchestration CLI on page 687](#)

[Orchestration CLI commands on page 689](#)

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 72. 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.
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 688 .
user	Specify the user name of the user who installed the agent.

Table 73. 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 <i>ymd</i> . 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 <i>"/"</i> .
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 789.
logging	
level	You can use this option to filter the log messages that are retrieved in the <code>occli.log</code> 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 <code>C:\Users\user_name\.OCLI\logs</code> . 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 <code>c:\Users\user_name\.OCLI\logs</code>.

Option	Actions
	You can use this option to change the location of the log file. The default location is <code>/home/user_name/.OCLI/logs</code> .
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 <code>./twsd/cli</code> .
protocol	Enter the protocol that you want to use to connect to the remote engine. The default value is https .
insecure	Enter <i>true</i> , to ignore the host name error on SSL certificate. The default value is <i>false</i> .
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.

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 687](#).

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 682](#).
- The required permissions to run the commands. For more information, see overview section of each commands mentioned in [Orchestration CLI commands on page 689](#).

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 689](#).

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 687](#).

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 695](#)
- [new on page 695](#)
- [remove on page 697](#)
- [set on page 697](#)
- [switch on page 698](#)

Model commands

- [add on page 700](#)
- [delete on page 701](#)
- [display on page 703](#)
- [extract on page 709](#)
- [list on page 711](#)
- [listfolder on page 716](#)
- [lock on page 717](#)
- [mkfolder on page 719](#)
- [modify on page 720](#)
- [new on page 723](#)
- [rename on page 724](#)
- [renamefolder on page 726](#)
- [replace on page 726](#)
- [rmfolder on page 727](#)
- [unlock on page 728](#)

Plan commands

- [ADDDEP JOB on page 731](#)
- [ADDDEP SCHED on page 733](#)
- [ALTJOB on page 734](#)
- [ALTRPRI on page 736](#)
- [CANCEL JOB on page 737](#)
- [CANCEL SCHED on page 739](#)
- [CONFIRM on page 741](#)

- [DELDEP JOB on page 744](#)
- [DELDEP SCHED on page 745](#)
- [FENCE on page 746](#)
- [KILL on page 747](#)
- [LIMIT CPU on page 748](#)
- [LIMIT SCHED on page 749](#)
- [LISTFOLDER on page 750](#)
- [RELEASE JOB on page 750](#)
- [RELEASE SCHED on page 751](#)
- [rerun on page 752](#)
- [showcpu on page 754](#)
- [showjobs on page 759](#)
- [showschedules on page 773](#)
- [SUBMIT DOCOMMAND on page 777](#)
- [SUBMIT JOB on page 779](#)
- [SUBMIT SCHED on page 781](#)

Related information

[Configuring Orchestration CLI on page 682](#)

[Running commands from Orchestration CLI on page 687](#)

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	<i>ADD</i>  Important: If you want to perform the action on an existing item, you must have the <i>MODIFY</i> access and if the item is locked by another user, you must also have the <i>UNLOCK</i> access.
delete	<i>DELETE</i>
display	<i>DISPLAY</i>  Important: If you want to run the command with the FULL parameter to view information about job streams then you must also have <i>Display</i> access on job.
extract	<i>DISPLAY</i>  Important: In addition to the permission mentioned above, you must also have <i>MODIFY</i> access if you want to run the command with the lock parameter.
list	<i>LIST</i>
listfolder	<i>LIST</i>
lock	<i>MODIFY</i>
mkfolder	<i>ADD</i>

Orchestration CLI model commands	
Command	Access required on items
modify	<i>MODIFY</i> and <i>DISPLAY</i>  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: • <i>USE</i> access on jobs. • <i>USE</i> access on calendars. Furthermore, if you want to run the command with FULL parameter, you must also have the following permissions: • <i>DISPLAY</i> access on jobs. • <i>USE</i> access on workstation where the job is defined. • <i>ADD</i> access on job, If the job definition is new or <i>MODIFY</i> access on job if the definition is an existing one.
new	<i>ADD</i>
rename	<i>DELETE</i> and <i>ADD</i>
renamefolder	<i>DELETE</i> and <i>ADD</i>
replace	• To replace with a new item: <i>ADD</i> • To replace an existing item: <i>MODIFY</i> • To replace an existing item that is locked by another user: <i>MODIFY</i> and <i>UNLOCK</i>  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:

Orchestration CLI model commands	
Command	Access required on items
	 ◦ <i>USE</i> access on jobs. ◦ <i>USE</i> access on calendars. Furthermore, if you want to run the command with FULL parameter, you must also have the following permissions: ◦ <i>DISPLAY</i> access on jobs. ◦ <i>USE</i> access on workstation where the job is defined. ◦ <i>ADD</i> access on job, If the job definition is new or <i>MODIFY</i> access on job if the definition is an existing one.
rmfolder	<i>DELETE</i>
unlock	• If the item is locked by the same user: <i>LIST</i> • If the item is locked by another user: <i>UNLOCK</i>

Orchestration CLI plan commands	
Command	Access required on items
adddep job/sched	<i>ADDDEP</i>
altjob	<i>SUBMIT</i>
altpass	<i>MODIFY</i>
altpri	<i>altpri</i>
cancel job/sched	<i>CANCEL</i>
confirm	<i>JOB-CONFIRM</i>
deldep job/sched	<i>DELDEP</i>
fence	<i>fence</i>
kill	<i>JOB-KILL</i>
limit sched	<i>limit</i>
listfolder	<i>LIST</i>
release job/sched	<i>RELEASE</i>

Orchestration CLI plan commands	
Command	Access required on items
setonline	<i>LINK</i>
setoffline	<i>UNLINK</i>
showcpu	<i>DISPLAY</i>
showjobs	<i>DISPLAY</i>
showschedules	<i>DISPLAY</i>
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 <i>JOB-SUBMIT</i> or <i>JOB-SUBMITDB</i> 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 <i>SCHEDULE-SUBMIT</i> access. • To submit an existing ad-hoc job definition, you must have the <i>JOB-SUBMITDB</i> and <i>CPU-USE</i> 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 <i>SCHEDULE-SUBMIT</i> access.
submit sched	<i>SCHEDULE-SUBMIT</i>

Orchestration CLI plug-in commands	
Commands	Access required
delete	<i>DELETE_PLUGIN</i>
install	<i>INSTALL_PLUGIN</i>
list	<i>LIST_PLUGIN</i>
update	<i>INSTALL_PLUGIN</i> and <i>DELETE_PLUGIN</i>

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 695](#)
- [new on page 695](#)
- [remove on page 697](#)
- [set on page 697](#)
- [switch on page 698](#)



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 788](#).



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 788](#)

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 788](#)

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 788](#)

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 788](#)

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 788](#)

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 700](#)
- [delete on page 701](#)
- [display on page 703](#)
- [extract on page 709](#)
- [list on page 711](#)
- [listfolder on page 716](#)
- [lock on page 717](#)
- [mkfolder on page 719](#)
- [modify on page 720](#)
- [new on page 723](#)
- [rename on page 724](#)
- [renamefolder on page 726](#)
- [replace on page 726](#)

- [rmfolder on page 727](#)
- [unlock on page 728](#)



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 788](#).

Related information

[Temporarily modifying the parameters in the config.yaml file on page 791](#)

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 788](#)

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 74. Scheduling items

Scheduling items	Description
calendars calendar cal <i>[folder/]calname</i>	You must specify the <i>[folder/]calname</i> that you want to delete. Wildcard characters are permitted.
domain dom <i>domainname</i>	You must specify the <i>domainname</i> that you want to delete. Wildcard characters are permitted.
folder fol <i>foldername</i>	If you do not provide the <i>foldername</i> or if the folder contains scheduling items, an error message is displayed. Wildcard characters are permitted.
jobs jobdefinition jd <i>[[folder/]workstationname#][folder/]jobname</i>	You can specify the <i>jobname</i> that you want to delete. You can provide <i>[folder/]workstationname#</i> and the folder in which the job stream is defined to make the selection more specific. Wildcard characters are permitted.

Table 74. Scheduling items (continued)

Scheduling items	Description
<code>sched jobstream js</code> <code>[[folder/]workstationname#][folder/]jstreamname[valid from date valid to date valid in date date] [;full] [;force]</code>	You must specify the <i>jstreamname</i> that you want to delete. You can provide <i>[[folder/]workstationname#]</i> 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. <i>Valid from date</i>: You can specify a date which will restrict the selection of job streams, that have a <i>valid from date</i> equal to the value specified. The format is <i>mm/dd/yyyy</i>. <i>Valid to date</i>: You can specify a date which will restrict the selection of job streams, that have a <i>valid to date</i> equal to the value specified. The format is <i>mm/dd/yyyy</i>. <i>valid in date date</i>: Specify the time frame at which the job is run. You can specify one of the dates as @. The format is <i>mm/dd/yyyy-mm/dd/yyyy</i>. 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.
<code>prompt pr [folder/]promptname</code>	You must specify the <i>[[folder/]promptname]</i> that you want to delete. Wildcard characters are permitted.
<code>resource re reso [folder/]resourcename</code>	You must specify the <i>[[folder/]resourcename]</i> that you want to delete. You can provide the <i>[[folder/]workstationname#]</i> and the folder in which the resource is defined to make the selection more specific. Wildcard characters are permitted.
<code>runcyclegroup rcg [folder/]rcgname]</code>	You must specify the <i>[[folder/]rcgname]</i> that you want to delete. Wildcard characters are permitted.
<code>users user [[folder/]workstationname#]username [;password]</code>	You must specify the <i>[[folder/]workstationname#]username [;password]</i> that you want to delete. Wildcard characters are permitted.

Table 74. Scheduling items (continued)

Scheduling items	Description
parms parm variable vb [[folder/]tablename.]variablename	You must specify the [[folder/]tablename.]variablename that you want to delete. Wildcard characters are permitted.
variable vt [folder/]tablename	You must specify the [folder/]tablename that you want to delete. Wildcard characters are permitted.
workstation ws cpu {[folder/]workstationname	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 `x1`:

```
ocli model de cpu=x1@
```

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 788](#)

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 75. 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 • 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

Table 75. Output format for displaying items (continued)

items	Details
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
variable table	• Variable Table Name • Default • Updated On • Locked By • Members
workstation	• Workstation Name • Type • Ignored • Updated On • Locked By • Workstation Definition

Table 75. Output format for displaying items (continued)

items	Details
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 76. Scheduling items

Scheduling items	Description
<code>calendars calendar cal [folder/]calname</code>	You can specify the <code>[folder/]calname</code> and optionally, the folder in which the calendar is defined. If you do not specify the <code>[folder/]calname</code> , all the calendar definitions in the database are displayed in order.
<code>domain dom domainname</code>	You can specify the <code>domainname</code> . If you do not provide the <code>domainname</code> , all the domain definitions in the database are displayed. Wildcard characters are permitted.
<code>folder fol foldername</code>	You can specify the name of the folder. If you do not provide the <code>foldername</code> , all the folder definitions are displayed. Wildcard characters are permitted.
<code>jobs jobdefinition jd</code> <code>[[folder/]workstationname#][folder/]jobname</code>	If you provide a specific <code>jobname</code> only that particular job definition is displayed and if not, all the job definitions are displayed. You can provide <code>[folder/]workstationname#</code> and the folder in which the job is defined to make the selection more specific. Wildcard characters are permitted.
<code>sched jobstream js</code> <code>[[folder/]workstationname#][folder/]jstreamname[valid from date valid to date valid in date date][;full]</code>	If you provide a specific <code>jstreamname</code> only that particular job stream definition is displayed and if not, all the job stream definitions are displayed. You can provide

Table 76. Scheduling items (continued)

Scheduling items	Description
	<i>[folder/]workstationame#]</i> 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. <i>Valid from date:</i> You can specify a date which will restrict the selection of job streams, that have a <i>valid from</i> date equal to the value specified. The format is <i>mm/dd/yyyy</i>. <i>Valid to date:</i> You can specify a date which will restrict the selection of job streams, that have a <i>valid to</i> date equal to the value specified. The format is <i>mm/dd/yyyy</i>. <i>valid in date date:</i> Specify the time frame at which the job is run. You can specify one of the dates as @. The format is <i>mm/dd/yyyy-mm/dd/yyyy</i>. full: If you add the option, all the job definitions in the job stream are displayed.
parms parm variable vb <i>[[folder/]tablename.]variablename</i>	You must provide the <i>[[folder/]tablename.]variablename</i> to display a specific variable within a specific table. If you specify the <i>variablename</i> only, the variable definition in the default variable is displayed. If you do not provide the <i>[[folder/]tablename.]variablename</i>, all the global variable definitions in the default variable table are displayed. Wildcard characters are permitted on both <i>[folder/]tablename</i> and <i>variablename</i>.
prompt pr <i>[folder/]promptname</i>	You can specify the <i>[folder/]promptname</i>. If you do not specify a <i>[folder/]promptname</i>, all the prompt definitions in the database are displayed. Wildcard characters are permitted.
resource re reso <i>[folder/]resourcename</i>	You can specify the <i>[folder/]resourcename</i>. You can provide the <i>[folder/]workstationame#]</i> and the folder in which the resource is defined to make the selection more specific. If you do not provide the <i>[folder/]resourcename</i>, all the resource definitions in the database are displayed. Wildcard characters are permitted.

Table 76. Scheduling items (continued)

Scheduling items	Description
<code>runcyclegroup rcg [folder/]rcgname]</code>	You can specify the <code>[folder/]rcgname]</code> . If you do not specify a <code>[folder/]rcgname]</code> , all the run cycle group definitions in the database are displayed. Wildcard characters are permitted.
<code>variable vt [folder/]tablename</code>	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 <code>tablename</code> and add wildcard characters, all the variable table definitions are displayed. Wildcard characters are permitted.
<code>users user [[folder/]workstationname#]username [;password]</code>	If you do not provide any specific <code>username</code> , all the usernames in the database are displayed. You can provide <code>[folder/]workstationname#]</code> and the folder in which the user is defined, to make the selection more specific. Wildcard characters are permitted.
<code>workstation ws cpu {[folder/]workstationname}</code>	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.
<code>workstationclass wsc [folder/]workstationclassname]</code>	You can specify the <code>[folder/]workstationclassname]</code> . If you do not specify a <code>[folder/]workstationclassname]</code> , 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 788](#)

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 77. Scheduling items

Scheduling items	Description
<code>domain dom domainname</code>	If you provide a specific <i>domainname</i> , 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.

Table 77. Scheduling items (continued)

Scheduling items	Description
folder fol <i>foldername</i>	You can specify the name of the folder. If you do not provide the <i>foldername</i> , all the folder definitions are copied into the file. Wildcard characters are permitted.
jobs jobdefinition jd [[<i>folder/</i>] <i>workstationname#</i>][<i>folder/</i>] <i>jobname</i>	If you provide a specific <i>jobname</i> only that particular job definition is copied and if not, all the job definitions are copied into the file. You can provide [<i>folder/</i>] <i>workstationname#</i>] and the folder in which the job is defined to make the selection more specific. Wildcard characters are permitted.
sched jobstream js [[<i>folder/</i>] <i>workstationname#</i>][<i>folder/</i>] <i>jstreamname</i> [<i>valid from date</i> <i>valid to date</i> <i>valid in date date</i>][<i>;full</i>]	If you provide a specific <i>jstreamname</i> only that particular job stream definition is copied and if not, all the job stream definitions are copied into the file. You can provide [<i>folder/</i>] <i>workstationname#</i>] 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. <i>Valid from date</i> : You can specify a date which will restrict the selection of job streams, that have a <i>valid from</i> date equal to the value specified. The format is <i>mm/dd/yyyy</i> . <i>Valid to date</i> : You can specify a date which will restrict the selection of job streams, that have a <i>valid to</i> date equal to the value specified. The format is <i>mm/dd/yyyy</i> . <i>valid in date date</i> : Specify the time frame at which the job is run. You can specify one of the dates as @. The format is <i>mm/dd/yyyy-mm/dd/yyyy</i> . full : If you add the option, all the job definitions in the job stream are displayed.
prompt pr [<i>folder/</i>] <i>promptname</i>	If you provide a specific [<i>folder/</i>] <i>promptname</i> 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 [<i>folder/</i>] <i>resourcename</i>	If you provide a specific [<i>folder/</i>] <i>resourcename</i> only that particular resource definition is copied and if not, all the

Table 77. Scheduling items (continued)

Scheduling items	Description
	resource definitions in the database are copied into the file. Wildcard characters are permitted.
<code>runcyclegroup rcg [folder/]rcgname]</code>	If you provide a specific <code>[folder/]rcgname]</code> 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.
<code>workstation ws cpu [folder/]workstationname]</code>	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.
<code>workstationclass wsc [folder/]workstationclassname]</code>	If you provide a specific <code>[folder/]workstationclassname]</code> 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 788](#)

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 78. 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 • 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

Table 78. Output format for displaying items (continued)

items	Details
	• 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
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

Table 78. Output format for displaying items (continued)

items	Details
	• 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 79. Scheduling items

Scheduling items	Description
calendars calendar cal <i>[folder/]calname</i>	You can specify the name of the calendar and optionally, the folder in which the calendar is defined. If you do not specify the <i>[folder/]calname</i> , all the calendar definitions in the database are listed in order.
domain dom <i>domainname</i>	You can specify the <i>domainname</i> . If you do not provide the <i>domainname</i> , all the domain definitions in the database are listed. Wildcard characters are permitted.
folder fol <i>foldername</i>	You can specify the name of the folder. If you do not provide the <i>foldername</i> , all the folder definitions in the database are listed. Wildcard characters are permitted.
parms parm variable vb <i>[[folder/]tablename.]variablename</i>	You must provide the <i>[[folder/]tablename.]variablename</i> to view a specific variable within a specific table. If you specify the <i>variablename</i> only, the variable definition in the default variable is listed. If you do not provide the <i>[[folder/]tablename.]variablename</i> , all the global variable definitions in the default variable table are listed. Wildcard characters are permitted on both <i>[folder/]tablename</i> and <i>variablename</i> .
prompt pr <i>[folder/]promptname</i>	You can specify the <i>[folder/]promptname</i> . If you do not provide the <i>[folder/]promptname</i> , all the prompt definitions in the database are listed. Wildcard characters are permitted.
resource re reso <i>[folder/]resourcename</i>	You can specify the <i>[folder/]resourcename</i> . You can provide the <i>[folder/]workstationname#</i> and the folder in which the resource is defined to make the selection more specific. If you do not provide the <i>[folder/]resourcename</i> , all the

Table 79. Scheduling items (continued)

Scheduling items	Description
	resource definitions in the database are listed. Wildcard characters are permitted.
<code>runcyclegroup rcg [folder/]rcgname]</code>	You can specify the <code>[folder/]rcgname]</code> . If you do not provide the <code>[folder/]rcgname]</code> , all the run cycle group definitions in the database are listed. Wildcard characters are permitted.
<code>variable vt [folder/]tablename</code>	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 <code>tablename</code> and add wildcard characters, all the variable table definitions are listed. Wildcard characters are permitted.
<code>jobs jobdefinition jd</code> <code>[[folder/]workstationname#][folder/]jobname</code>	If you provide a specific <i>jobname</i> only that particular job definition is listed and if not, all the job definitions are listed. You can provide <code>[folder/]workstationname#]</code> and the folder in which the job is defined to make the selection more specific. Wildcard characters are permitted.
<code>sched jobstream js</code> <code>[[folder/]workstationname#][folder/]jstreamname[valid from date valid to date valid in date date][;full]</code>	If you provide a specific <i>jstreamname</i> only that particular job stream definition is listed and if not, all the job stream definitions are listed. You can provide <code>[folder/]workstationname#]</code> 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. <i>Valid from date:</i> You can specify a date which will restrict the selection of job streams, that have a <i>valid from</i> date equal to the value specified. The format is <i>mm/dd/yyyy</i>. <i>Valid to date:</i> You can specify a date which will restrict the selection of job streams, that have a <i>valid to</i> date equal to the value specified. The format is <i>mm/dd/yyyy</i>. <i>valid in date date:</i> Specify the time frame at which the job is run. You can specify one of the dates as <i>@</i>. The format is <i>mm/dd/yyyy-mm/dd/yyyy</i>. full: If you add the option, all the job definitions in the job stream are listed.

Table 79. Scheduling items (continued)

Scheduling items	Description
users user <code>[[folder/]workstationname#]username</code> <code>[;password]</code>	If you do not provide any specific <i>username</i> , all the usernames in the database are listed. You can provide <code>[[folder/]workstationname#]</code> and the folder in which the user is defined, to make the selection more specific. Wildcard characters are permitted.
workstation ws cpu <code>[[folder/]workstationname]</code>	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 <code>[[folder/]workstationclassname]</code>	You can specify the <code>[[folder/]workstationclassname]</code> . If you do not provide the <code>[[folder/]workstationclassname]</code> , 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 788](#)

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 788](#)

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 80. Scheduling items

Scheduling items	Description
calendars calendar cal <i>[folder/]calname</i>	You must specify the <i>[folder/]calname</i> that you want to lock. Wildcard characters are permitted.
domain dom <i>domainname</i>	You must specify the <i>domainname</i> that you want to lock. Wildcard characters are permitted.
folder fol <i>foldername</i>	You must specify the name of the folder that you want to lock. Wildcard characters are permitted.
jobs jobdefinition jd <i>[[folder/]workstationname#][folder/]jobname</i>	If you provide a specific <i>jobname</i> only that particular job definition is locked and if not, all the job definitions are locked. You can provide <i>[folder/]workstationname#</i> and the

Table 80. Scheduling items (continued)

Scheduling items	Description
	folder in which the job is defined to make the selection more specific. Wildcard characters are permitted.
<code>sched jobstream js</code> <code>[[folder/]workstationname#][folder/]jstreamname[valid from date valid to date valid in date date][;full]]</code>	If you provide a specific <i>jstreamname</i> only that particular job stream definition is locked and if not, all the job stream definitions are locked. You can provide <code>[[folder/]workstationname#]</code> 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. <i>Valid from date:</i> You can specify a date which will restrict the selection of job streams, that have a <i>valid from</i> date equal to the value specified. The format is <i>mm/dd/yyyy</i>. <i>Valid to date:</i> You can specify a date which will restrict the selection of job streams, that have a <i>valid to</i> date equal to the value specified. The format is <i>mm/dd/yyyy</i>. <i>valid in date date:</i> Specify the time frame at which the job is run. You can specify one of the dates as @. The format is <i>mm/dd/yyyy-mm/dd/yyyy</i>. full: If you add the option, all the job definitions in the job stream are displayed for editing.
<code>prompt pr [folder/]promptname</code>	You must specify the <code>[[folder/]promptname]</code> that you want to lock. Wildcard characters are permitted.
<code>resource re reso [folder/]resourcename</code>	You must specify the <code>[[folder/]resourcename]</code> that you want to lock. You can provide the <code>[[folder/]workstationname#]</code> and the folder in which the resource is defined to make the selection more specific. Wildcard characters are permitted.
<code>runcyclegroup rcg [folder/]rcgname]</code>	You must specify the <code>[[folder/]rcgname]</code> that you want to lock. Wildcard characters are permitted.
<code>users user [[folder/]workstationname#]username [;password]</code>	You must specify the <code>[[folder/]workstationname#]username [;password]</code> that you want to lock. Wildcard characters are permitted.
<code>parms parm variable vb</code> <code>[[folder/]tablename.]variablename</code>	You must specify the <code>[[folder/]tablename]</code> that you want to lock. Wildcard characters are permitted.

Table 80. Scheduling items (continued)

Scheduling items	Description
<code>variable vt [folder/]tablename</code>	You must specify the <code>[[folder/]tablename.]variablename</code> that you want to lock. Wildcard characters are permitted.
<code>workstation ws cpu [folder/]workstationname]</code>	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.
<code>workstationclass wsc [folder/]workstationclassname]</code>	You must specify the <code>[folder/]workstationclassname]</code> 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 788](#)

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 788](#)

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 81. Scheduling items

Scheduling items	Description
calendars calendar cal <i>[folder/]calname</i>	You can specify the <i>[folder/]calname</i> that you want to modify. If you do not specify a <i>[folder/]calname</i> , all the calendar definitions in the database are displayed in a file for editing. Wildcard characters are permitted.
domain dom <i>domainname</i>	You can specify the <i>domainname</i> that you want to modify. If you do not specify a <i>domainname</i> , all the domain definitions in the database are displayed in a file for editing. Wildcard characters are permitted.

Table 81. Scheduling items (continued)

Scheduling items	Description
<code>folder fol <i>foldername</i></code>	You can specify the name of the folder. If you do not provide the <i>foldername</i>, all the folder definitions in the database are displayed in a file for editing. Wildcard characters are permitted.
<code>jobs jobdefinition jd</code> <code>[[<i>folder/</i>]<i>workstationname#</i>][<i>folder/</i>]<i>jobname</i></code>	If you provide a specific <i>jobname</i> only that particular job definition is displayed for editing and if not, all the job definitions are displayed. You can provide <code>[[<i>folder/</i>]<i>workstationname#</i>]</code> and the folder in which the job is defined to make the selection more specific. Wildcard characters are permitted.
<code>sched jobstream js</code> <code>[[<i>folder/</i>]<i>workstationname#</i>][<i>folder/</i>]<i>jstreamname</i>[<i>valid from date</i> <i>valid to date</i> <i>valid in date date</i>][<i>;full</i>]</code>	If you provide a specific <i>jstreamname</i> only that particular job stream definition is displayed for editing and if not, all the job stream definitions are displayed. You can provide <code>[[<i>folder/</i>]<i>workstationname#</i>]</code> 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. <i>Valid from date</i>: You can specify a date which will restrict the selection of job streams, that have a <i>valid from</i> date equal to the value specified. The format is <i>mm/dd/yyyy</i>. <i>Valid to date</i>: You can specify a date which will restrict the selection of job streams, that have a <i>valid to</i> date equal to the value specified. The format is <i>mm/dd/yyyy</i>. <i>valid in date date</i>: Specify the time frame at which the job is run. You can specify one of the dates as @. The format is <i>mm/dd/yyyy-mm/dd/yyyy</i>. full: If you add the option, all the job definitions in the job stream are displayed for editing.
<code>prompt pr [<i>folder/</i>]<i>promptname</i></code>	You can specify the <code>[[<i>folder/</i>]<i>promptname</i>]</code> that you want to modify. If you do not specify a <code>[[<i>folder/</i>]<i>promptname</i>]</code>, all the prompt definitions in the database are displayed in a file for editing. Wildcard characters are permitted.
<code>resource re reso [<i>folder/</i>]<i>resourcename</i></code>	You can specify the <code>[[<i>folder/</i>]<i>resourcename</i>]</code> that you want to modify. You can provide the <code>[[<i>folder/</i>]<i>workstationname#</i>]</code>

Table 81. Scheduling items (continued)

Scheduling items	Description
	and the folder in which the resource is defined to make the selection more specific. If you do not specify a <code>[folder/]resourcename</code> , all the resource definitions in the database are displayed in a file for editing. Wildcard characters are permitted.
<code>runcyclegroup rcg [folder/]rcgname]</code>	You can specify the <code>[folder/]rcgname]</code> that you want to modify. If you do not specify a <code>[folder/]rcgname]</code> , all the run cycle group definitions in the database are displayed in a file for editing. Wildcard characters are permitted.
<code>users user [[folder/]workstationname#]username [;password]</code>	You can specify the <code>[[folder/]workstationname#]username [;password]</code> that you want to modify. Wildcard characters are permitted.
<code>parms parm variable vb [[folder/]tablename.]variablename</code>	You can specify the <code>[[folder/]tablename.]variablename</code> that you want to modify. Wildcard characters are permitted.
<code>vartable vt [folder/]tablename</code>	You can specify the <code>[folder/]tablename</code> that you want to modify. Wildcard characters are permitted.
<code>workstation ws cpu [folder/]workstationname]</code>	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 <code>workstationname</code> , all the workstation definitions in the database are displayed in a file for editing. Wildcard characters are permitted.
<code>workstationclass wsc [folder/]workstationclassname]</code>	You can specify the <code>[folder/]workstationclassname]</code> that you want to modify. If you do not specify a <code>[folder/]workstationclassname]</code> , all the workstation class 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 788](#)

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 788](#)

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 82. Scheduling items

Scheduling items	Description
calendars calendar cal <i>[folder/]calname</i>	You can specify the <i>[folder/]calname</i> that you want to rename. Wildcard characters are permitted.
domain dom <i>domainname</i>	You can specify the <i>domainname</i> that you want to rename. Wildcard characters are permitted.
folder fol <i>foldername</i>	You can specify the <i>foldername</i> to rename.

Table 82. Scheduling items (continued)

Scheduling items	Description
jobs jobdefinition jd [[folder/]workstationname#][folder/]jobname	You can specify the <i>jobname</i> to rename. You can provide the <i>[[folder/]workstationname#]</i> path and the folder that contains the job definition to make the selection more specific.
sched jobstream js [[folder/]workstationname#][folder/]jstreamname	You can specify the <i>jstreamname</i> job stream to update. You can provide the <i>[[folder/]workstationname#]</i> path and the folder that contains job stream definition to make the selection more specific.
prompt pr [folder/]promptname	You can specify the <i>[[folder/]promptname]</i> to rename. Wildcard characters are permitted.
resource re reso [folder/]resourcename	You must specify the <i>[[folder/]resourcename]</i> that you want to rename. You can provide the <i>[[folder/]workstationname#]</i> 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 <i>[[folder/]rcgname]</i> that you want to rename. Wildcard characters are permitted.
users user [[folder/]workstationname#]username [;password]	You can specify the <i>[[folder/]workstationname#]username [;password]</i> that you want to rename. Wildcard characters are permitted.
parms parm variable vb [[folder/]tablename.]variablename	You can specify the <i>[[folder/]tablename.]variablename</i> that you want to rename. Wildcard characters are permitted.
variable vt [folder/]tablename	You can specify the <i>[[folder/]tablename]</i> that you want to rename. Wildcard characters are permitted.
workstation ws cpu [folder/]workstationname	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 <i>[[folder/]workstationclassname]</i> 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 788](#)

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 788](#)

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 788](#)

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 788](#)

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 83. Scheduling items

Scheduling items	Description
calendars calendar cal <i>[folder/]calname</i>	You must specify the <i>[folder/]calname</i> that you want to unlock. Wildcard characters are permitted.
domain dom <i>domainname</i>	You must specify the <i>domainname</i> that you want to unlock. Wildcard characters are permitted.
folder fol <i>foldername</i>	You must specify the name of the folder that you want to unlock. Wildcard characters are permitted.

Table 83. Scheduling items (continued)

Scheduling items	Description
<code>jobs jobdefinition jd</code> <code>[[folder/]workstationname#][folder/]jobname</code>	If you provide a specific <i>jobname</i> only that particular job definition is locked and if not, all the job definitions are unlocked. You can provide <code>[[folder/]workstationname#]</code> and the folder in which the job is defined to make the selection more specific. Wildcard characters are permitted.
<code>sched jobstream js</code> <code>[[folder/]workstationname#][folder/]jstreamname[valid from date valid to date valid in date date][;full]</code>	If you provide a specific <i>jstreamname</i> only that particular job stream definition is unlocked and if not, all the job stream definitions are unlocked. You can provide <code>[[folder/]workstationname#]</code> 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. <i>Valid from date:</i> You can specify a date which will restrict the selection of job streams, that have a <i>valid from</i> date equal to the value specified. The format is <i>mm/dd/yyyy</i>. <i>Valid to date:</i> You can specify a date which will restrict the selection of job streams, that have a <i>valid to</i> date equal to the value specified. The format is <i>mm/dd/yyyy</i>. <i>valid in date date:</i> Specify the time frame at which the job is run. You can specify one of the dates as @. The format is <i>mm/dd/yyyy-mm/dd/yyyy</i>. full: If you add the option, all the job definitions in the job stream are displayed for editing.
<code>prompt pr [folder/]promptname</code>	You must specify the <code>[[folder/]promptname]</code> that you want to unlock. Wildcard characters are permitted.
<code>resource re reso [folder/]resourcename</code>	You must specify the <code>[[folder/]resourcename]</code> that you want to unlock. You can provide the <code>[[folder/]workstationname#]</code> and the folder in which the resource is defined to make the selection more specific. Wildcard characters are permitted.
<code>runcyclegroup rcg [folder/]rcgname]</code>	You must specify the <code>[[folder/]rcgname]</code> that you want to unlock. Wildcard characters are permitted.
<code>users user [[folder/]workstationname#]username [;password]</code>	You must specify the <code>[[folder/]workstationname#]username [;password]</code> that you want to unlock. Wildcard characters are permitted.

Table 83. Scheduling items (continued)

Scheduling items	Description
parms parm variable vb <code>[[folder/]tablename.]variablename</code>	You must specify the <code>[[folder/]tablename.]variablename</code> that you want to unlock. Wildcard characters are permitted.
variable vt <code>[folder/]tablename</code>	You must specify the <code>[folder/]tablename</code> that you want to unlock. Wildcard characters are permitted.
workstation ws cpu <code>[folder/]workstationname]</code>	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 <code>[folder/]workstationclassname]</code>	You must specify the <code>[folder/]workstationclassname]</code> 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 788](#)

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 731](#)
- [ADDDEP SCHED on page 733](#)
- [ALTJOB on page 734](#)
- [ALTRPRI on page 736](#)
- [CANCEL JOB on page 737](#)

- [CANCEL SCHED on page 739](#)
- [CONFIRM on page 741](#)
- [DELDEP JOB on page 744](#)
- [DELDEP SCHED on page 745](#)
- [FENCE on page 746](#)
- [KILL on page 747](#)
- [LIMIT CPU on page 748](#)
- [LIMIT SCHED on page 749](#)
- [LISTFOLDER on page 750](#)
- [RELEASE JOB on page 750](#)
- [RELEASE SCHED on page 751](#)
- [rerun on page 752](#)
- [showcpu on page 754](#)
- [showjobs on page 759](#)
- [showschedules on page 773](#)
- [SUBMIT DOCOMMAND on page 777](#)
- [SUBMIT JOB on page 779](#)
- [SUBMIT SCHED on page 781](#)

Related information

[Configuring Orchestration CLI on page 682](#)

[Running commands from Orchestration CLI on page 687](#)

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 `LocaLopt's` 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.



Restrictions:

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 `JOB_A` 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 `JOB1` 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 788](#)

ALTRPRI

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 <code>Cancel Pend</code> 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 <code>Cancel Pend</code> 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, the 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 84. Change in status after confirm command

Initial job status	Status after <code>confirm ;succ</code>	Status after <code>confirm ;abend</code>
READY	No effect, with or without output conditions	No effect, with or without output conditions.

Table 84. Change in status after confirm command (continued)

Initial job status	Status after <code>confirm ;succ</code>	Status after <code>confirm ;abend</code>
HOLD	No effect, with or without output conditions	No effect, with or without output conditions.
EXEC	without output conditions SUCC with output conditions SUCC_P and selected output conditions are set to satisfied	without output conditions ABEND with output conditions ABEND_P and selected output conditions are set to satisfied.
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 84. Change in status after confirm command (continued)

Initial job status	Status after <code>confirm ;succ</code>	Status after <code>confirm ;abend</code>
		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

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
```

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 `follows` 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 sched` command 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 specify 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/@
```

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 persists. To remove the dependency permanently from a job stream, refer to [DELDEP JOB on page 744](#). 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 `myfo lder`, 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
```

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_command"|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 85.

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 1011](#).

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 755](#) 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 file 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, the 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	10.2.5	US/Pacific		Linux 2.6.5-7.191-s390 #1 SM
FTA1	10.2.5			Linux 2.4.9-e.24 #1 Wed May
FTA2	10.2.5			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

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 764](#). 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 486](#).



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 769](#) format. The job streams are displayed in [standard showschedules on page 774](#) format.

- **deps;logon**

If you use the **logon** option with **deps** parameter, the jobs are listed in the [showjobs;logon on page 771](#) format. The job streams are listed in the [standard showschedules on page 774](#) 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 769](#).

step

If you add this option, the job details are displayed in **step** format. For more information, see [step format on page 772](#).

logon

If you add this option, the job details are displayed in **logon** format. For more information, see [Logon format on page 771](#).

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 deferred 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.

showschedules

You can use the `showschedules` command 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 486](#).



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 764](#) and [standard showschedules on page 774](#) 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 769](#) format and the job streams are displayed in the [standard showschedules on page 774](#) format.

- **deps;logon**

The jobs in the results are displayed in [showjobs;logon on page 771](#) format and the job streams are displayed in the [standard showschedules on page 774](#) 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 from 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 {0AAAAAAAAAAAAACRZ}

```

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),(0AAAAAAAAAAAAABVY)] 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 682](#). 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([hmm[ 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([hmm[date]])}><joboption[;...]><varable=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. If 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([hmm[ 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[;...]]><[var[variable]=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 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.

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 86. Special characters as delimiters

Characters	Description
+	You can use the character to include a specific argument. For example, <pre>sj payroll_job + at 0600</pre>
~	You can use the character to exclude a specific argument. For example, <pre>sj payroll_job ~ at 0600</pre>
;	You can use the character as a delimiter between arguments in a command. For example, <pre>adj HCL#ABSENCES_JS.PAYROLL_JOB ; maxdur=80 ;onmaxdur kill</pre>
,	You can use the character to add multiple values for an argument. For example, <pre>state = done, succ, succp</pre>
=	You can use the character to give the specific value for an argument. For example, <pre>state = done</pre>

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 87.

Return code	Description
0	The model command completed successfully. Run the add command for a simple job that will finish successfully: • Run this command: <code>ocli model new job</code> • Define the job as follows and save it: <pre>\$JOBS WS_AGT_0#JOBDEF1 DOCOMMAND "dir" STREAMLOGON Administrator TASKTYPE WINDOWS RECOVERY STOP</pre> • 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: <code>echo %ERRORLEVEL%</code> This returns code result 0
4	The model command generated a warning. For example, try to add an existing item such as <code>simple_job</code>: • Run this command: <code>ocli model add <file_path>\fileName.</code> where <code><file_path></code> is the path of the file that contains the definition of the <code>simple_job</code>. The following error message is displayed:

Table 87. (continued)

Return code	Description
	- 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: <code>echo %ERRORLEVEL%</code> 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 of the definition. - The system responds with this message: > AWSMCG005E The definition does not specify the type of the item. - Run this command: <code>echo %ERRORLEVEL%</code>. 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 the command: <code>ocli plan sj WS_AGT_0#JS_ADJ_115749800</code> 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: <code>echo %ERRORLEVEL%</code>

Table 87. (continued)

Return code	Description
	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: <code>ocli plan sbj WS_AGT_0#invalidJob</code> • The system responds with this message: <code>> AWSMSL009E</code> The criteria specified in the command does not match any item. • To see the return code, run this command: <code>echo %ERRORLEVEL%</code> 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:

```
ocli [model|plan|context] [-file = Location of config.yaml file] command
```

or

```
ocli [model|plan|context] command [-file = Location of config.yaml file]
```

or

```
ocli [-file = Location of config.yaml file] [model|plan|context] command
```


[-file = Location of config.yaml file]

Specify the location of config.yaml file in your computer.

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 700](#)
- [new on page 723](#)
- [modify on page 720](#)

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 703](#)
- [extract on page 709](#)

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 792](#). For more information about using the BmEvents configuration file to configure the integration with external monitoring tools, see [The BmEvents configuration file on page 795](#).

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 88: Workload Automation exposed metrics on page 792](#) shows a list of the metrics retrieved, along with their description.

See [Accessing and visualizing the metrics on page 794](#) to find out where and how to find the metrics.

Table 88. 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 88. Workload Automation exposed metrics (continued)

Metric Display Name	Metric name	Description
Workload	application_wa_JobsInPlanCount_jobs	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
	application_wa_JobsInPlanCount_jobs	Workload throughput (jobs/minute)
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	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 88. 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)

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 89: Events filtered by CHSCHED on page 797](#) lists the events that are filtered by CHSCHED when it is set to LOW.

Table 89. 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
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 90: Logged events on page 799](#) lists the events logged by HCL Workload Automation.

Table 90. 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
mstJobFailed	102	JobFailed	Job failed
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
mstJobSuccp	112	JobSuccp	Job Successful pending
mstJobExtrn	113	JobExtrn	Job extern
mstJobIntro	114	JobIntro	Job in INTRO status
mstJobStuck	115	JobStuck	Job stuck

Table 90. Logged events (continued)

Event	Number	EventName	Description
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
mstJobMaxDurationExceededContinue	123	MaxDurationExceededContinue	Job maximum duration exceeded, job continues to run
mstJobMaxDurationExceededKill	124	MaxDurationExceededKill	Job maximum duration exceeded, Kill action triggered
mstJobMinDurationNotReachedContinue	125	MinDurationNotReachedContinue	Job minimum duration not reached, job continues to run
mstJobMinDurationNotReachedAbend	126	MinDurationNotReachedAbend	Job minimum duration not reached, Abend action triggered
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

Table 90. Logged events (continued)

Event	Number	EventName	Description
mstSchedStuck	152	SchedStuck	Job stream is stuck
mstSchedStart	153	SchedStart	Job stream started
 Note: This event is not logged for Workstation Master.			
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
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

Table 90. Logged events (continued)

Event	Number	EventName	Description
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

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. 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
```


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 205](#).

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 [Managing dynamic scheduling capabilities in your environment on page 858](#).

Prerequisite steps to create job types with advanced options

How to define a new job definitions using the Dynamic Workload Console.

About this task

Perform the following steps before you define and schedule job types with advanced options.

1. Install a number of dynamic agents and add the Java run time

To install dynamic agents, run the installation program. You can install the dynamic agents during the full installation of HCL Workload Automation or in a stand-alone installation of just the agent. During the installation, you have the option of adding the Java run time to run job types with advanced options, both those types supplied with the product and the additional types implemented through the custom plug-ins.

Follow the installation wizard to complete the installation.

See the section about installation options in *Planning and Installation Guide* for descriptions of the installation parameters and options.

2. Organize the dynamic agents in pools and dynamic pools.

Pools and dynamic pools help you organize the environment based on the availability of your workstations and the requirements of the jobs you plan to run.

- a. From the **Design** menu, click **Workload Designer** page.
- b. Select an engine.
- c. Select the workstation type you want to create.
 - To create a pool, define the dynamic agents you want to add to the pool and the workload broker workstation where the pool is hosted.
 - To create a dynamic pool, specify the requirements that each dynamic agent must meet to be added to the dynamic pool.

3. Grant the required authorization for defining job types with advanced options.

The HCL Workload Automation administrator has to grant specific authorizations in the security file to allow the operators to create job types with advanced options.

- a. Navigate to the `TwA_home/Tws` directory from where the **dumpsec** and **makesec** commands must be run.
- b. Run the **dumpsec** command to decrypt the current security file into an editable configuration file.

For more information, see the section about **dumpsec** in *Administration Guide*.

- c. Add display and run access to the workstation, as follows:
 - If the operation is performed on the HCL Workload Automation Connector, display and run access is required on the CPU corresponding to the workstation where the job is created.
 - If the operation is performed on the workstation where the job runs, display access is required on the workload broker workstation.

For more information, see the section about configuring the security file in *Administration Guide*.

- d. Close any open **conman** user interfaces using the **exit** command.
- e. Stop any connectors on systems running Windows operating systems.

f. Run the **makesec** command to encrypt the security file and apply the modifications.

For more information, see the section about **makesec** in *Administration Guide*.

g. If you are using local security, the file is immediately available on the workstation where it has been updated.

- i. If you are using a backup master domain manager, copy the file to it.
- ii. Distribute the centralized file manually to all fault-tolerant agents in the network (not standard, extended, or broker agents), and store it in the *TWA_home/TWS* directory.

iii. Run **JnextPlan** to distribute the Symphony file that corresponds to the new security file.

4. Define the job types with advanced options as described in [Creating advanced job definitions on page 807](#).

Creating advanced job definitions

From the **composer** command line you can create both standard jobs and job types with advanced options. The syntax for creating both job types is similar; the only difference is in the arguments that you use to define the job to be run.

To define standard job types, use the **docommand** or **scriptname** arguments; to define job types with advanced options, use the **task** argument, as described in [Job definition on page 205](#). Each job type with advanced options has specific attributes, which are described in detail in the following sections.

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.

Job definition - z/OS jobs

A z/OS job runs the command you specify in the JCL tab on a JCL system. This job type runs only on HCL Workload Automation Agent for z/OS.

This section describes the required and optional attributes for z/OS jobs. Each job definition has the following format and arguments:

Table 91. Required and optional attributes for the definition of a z/OS job.

Attribute	Description/value	Required
application name	jcl	✓
byDefinition	The type of job submission. This is the only supported submission type.	
jclDefinition	The operation to be performed on the JCL system.	✓

The following example shows a job that returns the status of the JCL system:

```

ZOSAGENT#JCLDEF
TASK
<?xml version="1.0" encoding="UTF-8"?>
<jSDL:jobDefinition xmlns:jSDL="http://www.ibm.com/xmlns/prod/scheduling/1.0/jSDL"
xmlns:jSDLjcl="http://www.ibm.com/xmlns/prod/scheduling/1.0/jSDLjcl">
<jSDL:application name="jcl">
<jSDLjcl:jcl>
<jSDLjcl:JCLParameters>
<jSDLjcl:jcl>
<jSDLjcl:byRefOrByDef>
<jSDLjcl:byDefinition>
<jSDLjcl:jclDefinition>//NORMAL JOB , 'TWS JOB', CLASS=A, MSGCLASS=A, >
// MSGLEVEL=(1,1)
/*
//STEP1 EXEC PGM=IEFBR14</jSDLjcl:jclDefinition>
</jSDLjcl:byDefinition>
</jSDLjcl:byRefOrByDef>
</jSDLjcl:jcl>
</jSDLjcl:JCLParameters>
<jSDLjcl:JOBParameters>
<jSDLjcl:jobStreamName>${tws.jobstream.name}jSDLjcl:jobStreamName>${tws.jobstream.name}>
<jSDLjcl:inputArrival>${tws.job.ia}jSDLjcl:inputArrival>${tws.job.ia}>
</jSDLjcl:JOBParameters>
</jSDLjcl:jcl>
</jSDL:application>
</jSDL:jobDefinition>
DESCRIPTION "Sample JCL Job Definition"

```

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.

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 92. 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 <i>username</i> specified in the database with the user definition (it is applicable to all operating systems on this job type) and key the statement: <pre><jsd1:password>\${password:username}</jsd1:password></pre> The password is retrieved from the <i>username</i> user definition in the database and resolved at runtime. See Using user definitions on job types with advanced options on page 226 for further details. You can also specify the user of a different workstation and use the following syntax for the password: <pre><jsd1:password>\${password:workstation#username}</jsd1:password></pre> - Enter a user and password defined with the <code>param</code> 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 <i>user</i> and a password, the corresponding credential statements would be: <pre><jsd1:userName>\${agent:user}</jsd1:userName> <jsd1:password>\${agent:password.user}</jsd1:password></pre>	✓
password	The password of the authorized user. The password is encrypted when the job is created. See description for <i>userName</i> 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: 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 <code>/home/Administrator</code>.	

Table 92. Required and optional attributes for the definition of a remote command job (continued)

Attribute	Description/value	Required
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.	
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:user>userName</jSDL:user>
      <jSDL:password>{aes}mv0GJq0Hw08lbuhcpFalUL9RkGQKrYvTiAUpKTMgp90=
        </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.

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 93. Required and optional attributes for the definition of an IBM i job.

Attribute	Description/value	Required
application name	ibmi	✓
user name	The user name authorized to run the job on the IBMi 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.	

Table 93. Required and optional attributes for the definition of an IBM i job. (continued)

Attribute	Description/value	Required
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.	
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	

Table 93. Required and optional attributes for the definition of an IBM i job. (continued)

Attribute	Description/value	Required
	accepted for the message. Valid range is from 0 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 SBJJOB 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>QDFTJOB</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>
```

```

    </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 1077](#).

```

$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 94. 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"?>
<jsdsl: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">
  <jsdsl: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 95. 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 95. Required and optional attributes for the definition of an access method job (continued)

Attribute	Description/value	Required
	- Enter a <i>username</i> specified in the database with the user definition (it is applicable to all operating systems on this job type) and key the statement: <pre><jsdL:password>\${password:username}</jsdL:password></pre> The password is retrieved from the <i>username</i> user definition in the database and resolved at runtime. For further details, see Using user definitions on job types with advanced options on page 226. You can also specify the user of a different workstation and use the following syntax for the password: <pre><jsdL:password>\${password:workstation#username}</jsdL:password></pre> - Enter a user and password defined with the <code>param</code> 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 <i>user</i> and a password, the corresponding credential statements is: <pre><jsdL:userName>\${agent:user}</jsdL:userName> <jsdL:password>\${agent:password.user}</jsdL:password></pre> 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 835.	

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/DES8wRzQEijlySQBfUR587QBxM0iwfQ1EWJaDs=</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.

Shadow jobs

A shadow job is a job defined on the local workstation which is used to map a job running on a remote workstation, called a remote job. You can use shadow jobs to integrate the workload running on different engines, which can be HCL Workload Automation for Z engines or HCL Workload Automation engines.

Shadow jobs are defined using XML syntax. The key attributes to identify the remote job instance and the related matching criteria depend on the type of remote engine where the remote job instance is defined. Fields highlighted in bold are those used to identify the remote job instance.

Because z/OS engines support only closest preceding matching criteria the XML template to define a z/OS shadow job is the following:

```
$JOBS
WORKSTATION#ZSHADOW_CLOS_PRE
TASK
  <?xml version="1.0" encoding="UTF-8"?>
  <jSDL:jobDefinition
    xmlns:jSDL="http://www.ibm.com/xmlns/prod/scheduling/1.0/jSDL"
    xmlns:zshadow="http://www.ibm.com/xmlns/prod/scheduling/1.0/zshadow">
    <jSDL:application name="zShadowJob">
    <zshadow:ZShadowJob>
    <zshadow:JobStream>JobStream</zshadow:JobStream>
    <zshadow:JobNumber>JobNumber</zshadow:JobNumber>
    <zshadow:matching>
    <zshadow:previous/>
    </zshadow:matching>
    </zshadow:ZShadowJob>
    </jSDL:application>
    </jSDL:jobDefinition>
  </jSDL:jobDefinition>
DESCRIPTION "Sample Job Definition"
RECOVERY STOP
```



Note: Make sure that you enter valid settings in the **JobStream** and **JobNumber** fields.

Distributed shadow jobs support the four matching criteria available for external dependencies.

The following shows the XML templates you can use to define distributed shadow jobs:

Matching criteria: Closest preceding

XML sample:

```
$JOBS
WORKSTATION#DSHADOW_CLOS_PRE
TASK
  <?xml version="1.0" encoding="UTF-8"?>
  <jSDL:jobDefinition
    xmlns:jSDL="http://www.ibm.com/xmlns/prod/scheduling/1.0/jSDL"
    xmlns:dshadow="http://www.ibm.com/xmlns/prod/scheduling/1.0/dshadow">
    <jSDL:application name="distributedShadowJob">
    <dshadow:DistributedShadowJob>
```

```

    <dshadow:JobStream>JobStream</dshadow:JobStream>
    <dshadow:Workstation>Workstation</dshadow:Workstation>
    <dshadow:Job>Job</dshadow:Job>
    <dshadow:matching>
    <dshadow:previous/>
    </dshadow:matching>
  </dshadow:DistributedShadowJob>
</jsdl:application>
</jsdl:jobDefinition>
DESCRIPTION "Sample Job Definition"
RECOVERY STOP

```

Matching criteria: Within an absolute interval

XML sample:

```

$JOBS
WORKSTATION#DSHADOW_ABSOLUTE
TASK
  <?xml version="1.0" encoding="UTF-8"?>
  <jsdl:jobDefinition
    xmlns:jsdl="http://www.ibm.com/xmlns/prod/scheduling/1.0/jsdl"
    xmlns:dshadow="http://www.ibm.com/xmlns/prod/scheduling/1.0/dshadow">
    <jsdl:application name="distributedShadowJob">
    <dshadow:DistributedShadowJob>
    <dshadow:JobStream>JobStream</dshadow:JobStream>
    <dshadow:Workstation>Workstation</dshadow:Workstation>
    <dshadow:Job>Job</dshadow:Job>
    <dshadow:matching>
    <dshadow:absolute from="0600 -4" to="1100 +3"/>
    </dshadow:matching>
    </dshadow:DistributedShadowJob>
    </jsdl:application>
  </jsdl:jobDefinition>
DESCRIPTION "Sample Job Definition"
RECOVERY STOP

```

Matching criteria: Within a relative interval

```

$JOBS
WORKSTATION#DSHADOW_RELATIVE
TASK
  <?xml version="1.0" encoding="UTF-8"?>
  <jsdl:jobDefinition
    xmlns:jsdl="http://www.ibm.com/xmlns/prod/scheduling/1.0/jsdl"
    xmlns:dshadow="http://www.ibm.com/xmlns/prod/scheduling/1.0/dshadow">
    <jsdl:application name="distributedShadowJob">
    <dshadow:DistributedShadowJob>
    <dshadow:JobStream>JobStream</dshadow:JobStream>
    <dshadow:Workstation>Workstation</dshadow:Workstation>
    <dshadow:Job>Job</dshadow:Job>
    <dshadow:matching>
    <dshadow:relative from="-400" to="+500" />
    </dshadow:matching>
    </dshadow:DistributedShadowJob>
    </jsdl:application>
  </jsdl:jobDefinition>
DESCRIPTION "Sample Job Definition"
RECOVERY STOP

```


Matching criteria: Same scheduled date

XML sample:

```
$JOBS
WORKSTATION#DSHADOW_SAMEDAY
TASK
  <?xml version="1.0" encoding="UTF-8"?>
  <jSDL:jobDefinition
    xmlns:jSDL="http://www.ibm.com/xmlns/prod/scheduling/1.0/jSDL"
    xmlns:dshadow="http://www.ibm.com/xmlns/prod/scheduling/1.0/dshadow">
    <jSDL:application name="distributedShadowJob">
      <dshadow:DistributedShadowJob>
        <dshadow:JobStream>JobStream</dshadow:JobStream>
        <dshadow:Workstation>Workstation</dshadow:Workstation>
        <dshadow:Job>Job</dshadow:Job>
        <dshadow:matching>
          <dshadow:sameDay/>
        </dshadow:matching>
      </dshadow:DistributedShadowJob>
    </jSDL:application>
  </jSDL:jobDefinition>
  DESCRIPTION "Sample Job Definition"
  RECOVERY STOP
```

For more information about the matching criteria, see [Managing external follows dependencies for jobs and job streams on page 87](#).

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.

Variable Table jobs

A Variable Table job adds or modifies a variable in a specified variable table.

Use the Variable Table job type to add or modify a variable in a specified variable table. The Variable Table job type enables variable passing from one job to another, in the same job stream or in a different job stream.

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 Variable Table jobs. Required attributes can be specified either at job definition time, or in the job plug-in properties file. Each job definition has the following format and attributes:

Table 96. Required and optional attributes for the definition of a Variable Table job

Attribute	Description and value	Required
hostname	The host name of the workstation where the HCL Workload Automation master server is installed, on which you want to add or modify a variable.	✓
port	The TCP/IP port number of the workstation where the HCL Workload Automation master server is installed.	✓
protocol	The protocol for connecting the dynamic agent running the Variable Table job and the HCL Workload Automation master server. Supported values are http and https.	✓
user name	The user to be used for accessing the HCL Workload Automation master server.	
password	The password to be used for accessing the HCL Workload Automation master server.	Required if you specify a user name.
keystore file path	The fully qualified path of the keystore file containing the private key that is used to make the connection.	
keystore file password	The password that protects the private key and is required to make the connection.	
number of retries	The number of times that the program retries when connecting to the HCL Workload Automation master server. Default value is 0.	
retry interval (in seconds)	The number of seconds that the program waits before retrying to connect to the HCL Workload Automation master server. Default value is 30 seconds.	
variable table	The name of the variable table, on the HCL Workload Automation master server where you want to add or modify a variable. Use the format <i>[folder/]jobstreamname</i> . If <i>[folder/]</i> is omitted, then the root folder is assumed.	✓
variable list	The list of variables and related values, in the selected variable table, that you want to add or modify. Specify at least one variable.	✓

The following example shows the job definition for a Variable Table job:

```
<?xml version="1.0" encoding="UTF-8"?>
<jSDL:jobDefinition xmlns:jSDL="http://www.ibm.com/xmlns/prod/scheduling/1.0/jSDL"
xmlns:jSDLvariabletable="http://www.ibm.com/xmlns/prod/scheduling
/1.0/jSDLvariabletable"
name="VARIABLETABLE">
<jSDL:application name="myfolder/variabletable">
<jSDLvariabletable:variabletable>
<jSDLvariabletable:VariableTableParameters>
<jSDLvariabletable:Connection>
<jSDLvariabletable:connectionInfo>
<jSDLvariabletable:hostname>nx000140.xxxxlab.yy.zzz.com
</jSDLvariabletable:hostname>
<jSDLvariabletable:port>35116</jSDLvariabletable:port>
<jSDLvariabletable:protocol>https</jSDLvariabletable:protocol>
<jSDLvariabletable:credentials>
<jSDL:user_name>twS_user</jSDL:user_name>
<jSDL:password>{aes}
/2GNMAY8Z2pSx6JXHqcbKwd2xxxxxkyKXD/WNgthhnw=
</jSDL:password>
</jSDLvariabletable:credentials>
<jSDLvariabletable:CertificateGroup>
<jSDLvariabletable:keyStoreFilePath/>
<jSDLvariabletable:keyStorePassword/>

```

```

        </jsdlvariabletable:CertificateGroup>
    </jsdlvariabletable:connectionInfo>
    <jsdlvariabletable:retryOptions>
        <jsdlvariabletable:NumberOfRetries>0
    </jsdlvariabletable:NumberOfRetries>
        <jsdlvariabletable:RetryIntervalSeconds>30</jsdlvariabletable:
            RetryIntervalSeconds>
    </jsdlvariabletable:retryOptions>
</jsdlvariabletable:Connection>
<jsdlvariabletable:Action>
    <jsdlvariabletable:actionInfo>
        <jsdlvariabletable:varTable>TABLE1</jsdlvariabletable:varTable>
        <jsdlvariabletable:varListValues>
            <jsdlvariabletable:varListValue key="DB2Name">HCLDB
        </jsdlvariabletable:varListValue>
        </jsdlvariabletable:varListValues>
    </jsdlvariabletable:actionInfo>
</jsdlvariabletable:Action>
</jsdlvariabletable:VariableTableParameters>
</jsdlvariabletable:variabletable>
</jsdl:application>
</jsdl:jobDefinition>

```

Scheduling a job in HCL Workload Automation

You schedule Variable Table jobs by defining them in job streams. Add the job to a job stream with all the necessary scheduling arguments and submit it.

You can submit jobs by using the Dynamic Workload Console, or the conman command line.

Stopping and restarting a job

Stopping and restarting a Variable Table job are not supported.

Variable TableJobExecutor.properties file

The properties file is automatically generated either when you run a "Test Connection" from the Dynamic Workload Console in the job definition panels, or when you submit the job to run the first time. Once the file has been created, you can customize it. This is especially useful when you need to schedule several jobs of the same type. You can specify the values in the properties file and avoid having to provide information such as credentials and other information, for each job. You can override the values in the properties files by defining different values at job definition time.

The properties file, named `VariableTableJobExecutor.properties`, is located in the following path:

On Windows operating systems

`TWS_INST_DIR\TWS\JavaExt\cfg`

On UNIX operating systems

`TWA_DATA_DIR/TWS/JavaExt/cfg`

The file contains the following properties:

```
#Variable Table properties
hostname=
port=
protocol=http
user=
password=
keyStoreFilePath=
keyStorePassword=
HostNameVerifyCheckbox=false
NumberOfRetries=0
RetryIntervalSeconds=30
varTable=
#add here the variables in the format
VARLISTPROPERTY.<variable_name>=<variable_value>
#For example VARLISTPROPERTY.queueName=default
```

Job properties

You can see the job properties by running `conman sj <job_name>;props`, where `<job_name>` is the Variable Table job name.

Job log content

You can see the job log content by running `conman sj <job_name>;stdlist`, where `<job_name>` is the Variable Table job name.

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.

Job Management jobs

A Job Management job runs actions on a job in a job stream.

Use the Job Management job type to run actions on a job in a job stream.

Actions that you can run on a job are:

- Rerun
- Rerun the job and all its successor jobs
- Rerun the job and its successor jobs in the same job stream
- Release
- Release Dependencies
- Cancel
- Cancel Pending
- Hold
- Kill

- Confirm ABEND
- Confirm SUCC

For more information about the actions that you can run on a job, by using the Dynamic Workload Console, see the [Dynamic Workload Console User's Guide](#), section about Controlling Jobs and Job Streams Processing.

For more information about the actions that you can run on a job, by using conman command line, see the [HCL Workload Automation User's Guide and Reference](#), section about Managing objects in the plan - conman.

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 Job Management jobs. Required attributes must be specified at job definition time. No properties file is available for this plug-in. Each job definition has the following format and attributes:

Table 97. Required and optional attributes for the definition of a Job Management job

Attribute	Description and value	Required
Url	A variable that specifies the web address of HCL Workload Automation master server. You can override this variable with the web address of the master server and any HCL Workload Automation backup master (in case the master is not available). Use a comma or a semicolon to separate the different addresses that you specify. If you do not override this variable at job definition time, the variable is resolved automatically at job execution time. In this case, if the connection to the master server is not available, the plug-in tries to connect automatically to any of the backup masters.	✓
userName	The user to be used for accessing the HCL Workload Automation master server.	
password	The password to be used for accessing the HCL Workload Automation master server.	Required if you specify a user name.
keyStoreFilePath	The fully qualified path of the keystore file containing the private key that is used to make the connection.	
keyStorePassword	The password that protects the private key and is required to make the connection.	
HostnameVerifyCheckbox	Use this attribute to require that the syntax of the HCL Workload Automation master server name, as featured in the keystore file, must match exactly the URL. If they do not match, no authorization is granted to access the server. If this attribute is not specified, the control is not enforced.	
NumberOfRetries	The number of times that the program retries when connecting to the HCL Workload Automation master server. Default value is 0.	
RetryIntervalSeconds	The number of seconds that the program waits before retrying to connect to the HCL Workload Automation master server. Default value is 30 seconds.	
jobname	The name of the job on which you want to run the action.	✓

Table 97. Required and optional attributes for the definition of a Job Management job (continued)

Attribute	Description and value	Required
workstation	A variable that specifies the name of the workstation on which the job runs. You can override this variable with a workstation name in the format <code>/folder_path/workstation_name</code> or <code>/workstation_name</code> , if the workstation is defined in the root (<code>/</code>) folder. If you do not override this variable at job definition time, the variable is resolved automatically at job execution time.	✓
jobstreamid	A variable that specifies the name of the job stream containing the job. You can override this variable with a job stream name. If you do not override this variable at job definition time, the variable is resolved automatically at job execution time.	✓
method	The action that you want to run on the job. Valid values are: • rerun • rerunsuccessors • reruninternalsuccessors • release • releasedependencies • cancel • cancelpending • hold • kill • confirm_abend • confirm_succ	✓
sameworkstation	Specify this parameter only for the rerun action. Use this parameter if you want to rerun the job on the same workstation where it ran previously. This parameter is applicable only to pool and dynamic pool workstations.	
condition	The condition name to confirm the SUCC or ABEND status for the specified output conditions. Any conditions not specified are set to not satisfied.	

The following example shows the job definition for a Job Management job that runs the **rerun** action:

```
<?xml version="1.0" encoding="UTF-8"?>
<jSDL:jobDefinition xmlns:jSDL="http://www.abc.com/xmlns/prod/scheduling/1.0/jSDL"
xmlns:jSDLjobmanagement="http://www.abc.com/xmlns/prod/
scheduling/1.0/jSDLjobmanagement"
name="JOBMANAGEMENT">
<jSDL:application name="jobmanagement">
<jSDLjobmanagement:jobmanagement>
  <jSDLjobmanagement:JobManagementParameters>
    <jSDLjobmanagement:Connection>
<jSDLjobmanagement:connectionInfo>
  <jSDLjobmanagement:Url>${agent-config:master-address}</jSDLjobmanagement:Url>
<jSDLjobmanagement:credentials>
  <jSDL:user_name>twuser1</jSDL:user_name>
  <jSDL:password>{aes}yWIpc7ISIQS9xb7xrzqx0YJn04rNj/d1IfLa20r7Rg=
    </jSDL:password>
</jSDLjobmanagement:credentials>
<jSDLjobmanagement:CertificateGroup>
```

```

    <jsdllibmanagement:keyStoreFilePath></jsdllibmanagement:keyStoreFilePath>
    <jsdllibmanagement:keyStorePassword></jsdllibmanagement:keyStorePassword>
        <jsdllibmanagement:HostnameVerifyCheckbox/>
    </jsdllibmanagement:CertificateGroup>
</jsdllibmanagement:connectionInfo>
<jsdllibmanagement:retryOptions>
    <jsdllibmanagement:NumberOfRetries>0</jsdllibmanagement:NumberOfRetries>
    <jsdllibmanagement:RetryIntervalSeconds>30
        </jsdllibmanagement:RetryIntervalSeconds>
</jsdllibmanagement:retryOptions>
    </jsdllibmanagement:Connection>
    <jsdllibmanagement:Action>
<jsdllibmanagement:informations>
    <jsdllibmanagement:jobname>JOBDIR</jsdllibmanagement:jobname>
    <jsdllibmanagement:workstation>LAPTOP-E0DIBP1_2 (type: Agent, version: 9.4.0.01)
        </jsdllibmanagement:workstation>
    <jsdllibmanagement:jobstreamid>${tw.s.jobstream.id}
        </jsdllibmanagement:jobstreamid>
</jsdllibmanagement:informations>
<jsdllibmanagement:actions>
    <jsdllibmanagement:method>rerun</jsdllibmanagement:method>
</jsdllibmanagement:actions>
<jsdllibmanagement:options>
    <jsdllibmanagement:sameworkstation/>
    <jsdllibmanagement:condition></jsdllibmanagement:condition>
</jsdllibmanagement:options>
    </jsdllibmanagement:Action>
    </jsdllibmanagement:JobManagementParameters>
</jsdllibmanagement:jobmanagement>
</jsdl:application>
</jsdl:jobDefinition>

```

The following example shows the job definition for a Job Management job that runs the **rerunsuccessors** action:

```

<?xml version="1.0" encoding="UTF-8"?>
<jsdllib:jobDefinition xmlns:jsdl="http://www.abc.com/xmlns/prod/scheduling/1.0/jsdl"
xmlns:jsdllibmanagement="http://www.abc.com/xmlns/prod/
scheduling/1.0/jsdllibmanagement"
name="JOBMANAGEMENT">
<jsdllib:application name="jobmanagement">
<jsdllibmanagement:jobmanagement>
    <jsdllibmanagement:JobManagementParameters>
        <jsdllibmanagement:Connection>
<jsdllibmanagement:connectionInfo>
    <jsdllibmanagement:Url>https://localhost:31116/jsdllibmanagement:Url>
    <jsdllibmanagement:credentials>
        <jsdllib:userNametwsuser1</jsdl:userNamet>
        <jsdllib:password{aes}ywIpc7ISIQS9xb7xrzqxoYJn04rNj/d1IfLa20r7Rg=
            </jsdl:password>
    </jsdllibmanagement:credentials>
<jsdllibmanagement:CertificateGroup>
    <jsdllibmanagement:keyStoreFilePath></jsdllibmanagement:keyStoreFilePath>
    <jsdllibmanagement:keyStorePassword></jsdllibmanagement:keyStorePassword>
        <jsdllibmanagement:HostnameVerifyCheckbox/>
    </jsdllibmanagement:CertificateGroup>
</jsdllibmanagement:connectionInfo>
<jsdllibmanagement:retryOptions>
    <jsdllibmanagement:NumberOfRetries>0</jsdllibmanagement:NumberOfRetries>
    <jsdllibmanagement:RetryIntervalSeconds>30

```

```

        </jsdljobmanagement:RetryIntervalSeconds>
    </jsdljobmanagement:retryOptions>
    </jsdljobmanagement:Connection>
    <jsdljobmanagement:Action>
<jsdljobmanagement:informations>
    <jsdljobmanagement:jobname>JOBDIR</jsdljobmanagement:jobname>
    <jsdljobmanagement:workstation>LAPTOP-E0DIBP1_2
        (type: Agent, version: 9.4.0.01)
    </jsdljobmanagement:workstation>
    <jsdljobmanagement:jobstreamid>${tw.s.jobstream.id}
    </jsdljobmanagement:jobstreamid>
</jsdljobmanagement:informations>
<jsdljobmanagement:actions>
    <jsdljobmanagement:method>rerunsuccessors</jsdljobmanagement:method>
</jsdljobmanagement:actions>
<jsdljobmanagement:options>
    <jsdljobmanagement:condition></jsdljobmanagement:condition>
</jsdljobmanagement:options>
    </jsdljobmanagement:Action>
    </jsdljobmanagement:JobManagementParameters>
</jsdljobmanagement:jobmanagement>
</jsdl:application>
</jsdl:jobDefinition>

```

The following example shows the job definition for a Job Management job that runs the **confirm_succ** action:

```

<?xml version="1.0" encoding="UTF-8"?>
<jsdl:jobDefinition xmlns:jsdl="http://www.ibm.com/xmlns/prod/scheduling/1.0/jsdl"
xmlns:jsdljobmanagement="http://www.ibm.com/xmlns/prod/
scheduling/1.0/jsdljobmanagement"
name="JOBMANAGEMENT">
<jsdl:application name="jobmanagement">
<jsdljobmanagement:jobmanagement>
    <jsdljobmanagement:JobManagementParameters>
        <jsdljobmanagement:Connection>
<jsdljobmanagement:connectionInfo>
    <jsdljobmanagement:Url>https://localhost:31116/jsdljobmanagement:Url>
    <jsdljobmanagement:credentials>
        <jsdl:userNam>twuser1/jsdl:userName>
        <jsdl:password>{aes}ywIpc7ISIQSq9xb7xrzqxoYJn04rNj/d1IfLa20r7Rg=
            </jsdl:password>
    </jsdljobmanagement:credentials>
    <jsdljobmanagement:CertificateGroup>
        <jsdljobmanagement:keyStoreFilePath></jsdljobmanagement:keyStoreFilePath>
        <jsdljobmanagement:keyStorePassword></jsdljobmanagement:keyStorePassword>
    </jsdljobmanagement:CertificateGroup>
</jsdljobmanagement:connectionInfo>
<jsdljobmanagement:retryOptions>
    <jsdljobmanagement:NumberOfRetries>0</jsdljobmanagement:NumberOfRetries>
    <jsdljobmanagement:RetryIntervalSeconds>30
        </jsdljobmanagement:RetryIntervalSeconds>
</jsdljobmanagement:retryOptions>
    </jsdljobmanagement:Connection>
    <jsdljobmanagement:Action>
<jsdljobmanagement:informations>
    <jsdljobmanagement:jobname>JOBDIR</jsdljobmanagement:jobname>
    <jsdljobmanagement:workstation>LAPTOP-E0DIBP1_2 (type: Agent, version: 9.4.0.01)
        </jsdljobmanagement:workstation>
    <jsdljobmanagement:jobstreamid>${tw.s.jobstream.id}

```



```

        </jsdljobmanagement:jobstreamid>
    </jsdljobmanagement:informations>
    <jsdljobmanagement:actions>
        <jsdljobmanagement:method>confirm_succ</jsdljobmanagement:method>
    </jsdljobmanagement:actions>
    <jsdljobmanagement:options>
        <jsdljobmanagement:condition>CONF_SUCC_CONDITION/jsdljobmanagement:condition>
    </jsdljobmanagement:options>
        </jsdljobmanagement:Action>
    </jsdljobmanagement:JobManagementParameters>
</jsdljobmanagement:jobmanagement>
</jsdl:application>
</jsdl:jobDefinition>

```

Scheduling a job in HCL Workload Automation

You schedule Job Management jobs by defining them in job streams. Add the job to a job stream with all the necessary scheduling arguments and submit it.

You can submit jobs by using the Dynamic Workload Console, or the `conman` command line.

Stopping and restarting a job

Stopping and restarting a Job Management job are not supported.

Job properties

You can see the job properties by running `conman sj <job_name>;props`, where `<job_name>` is the Job Management job name.

You can export some of the Job Management job properties that you see in the `Extra Information` section of the output command, to a successive job in the same job stream or in a different job stream. For more information about the list of job properties that you can export, see [Table 105: Properties for Job Management jobs on page 851](#).

Job log content

You can see the job log content by running `conman sj <job_name>;stdlist`, where `<job_name>` is the Job Management job name.

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.

Job Stream Submission jobs

A Job Stream Submission job submits a job stream for processing.

The Job Stream Submission job is one of the Automation Utilities that facilitate specific HCL Workload Automation operations. Use the Job Stream Submission job type to automate the submission of a job stream for processing.

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 Job Stream Submission jobs. Required attributes must be specified at job definition time. No properties file is available for this plug-in. Each job definition has the following format and attributes:

Table 98. Required and optional attributes for the definition of a Job Stream Submission job

Attribute	Description and value	Required
Url	A variable that specifies the web address of HCL Workload Automation master server. You can override this variable with the web address of the master server and any HCL Workload Automation backup master (in case the master is not available). Use a comma or a semicolon to separate the different addresses that you specify. If you do not override this variable at job definition time, the variable is resolved automatically at job execution time. In this case, if the connection to the master server is not available, the plug-in tries to connect automatically to any of the backup masters.	✓
userName	The user to be used for accessing the HCL Workload Automation master server.	
password	The password to be used for accessing the HCL Workload Automation master server.	Required if you specify a user name.
keyStoreFilePath	The fully qualified path of the keystore file containing the private key that is used to make the connection.  Note: For Workload Automation on Cloud users, this attribute must be equal to: <pre style="background-color: #f0f0f0; padding: 5px;">\${agent-config:keystore-file}</pre> As an alternative, you can specify this attribute in the following format: <pre style="background-color: #f0f0f0; padding: 5px;">[<key-store-type>:]<key-store-path></pre>	
keyStorePassword	The password that protects the private key and is required to make the connection.  Note: For Workload Automation on Cloud users, this attribute must be equal to: <pre style="background-color: #f0f0f0; padding: 5px;">\${agent-config:keystore-file-password}</pre>	
HostnameVerifyCheckbox	Use this attribute to require that the syntax of the HCL Workload Automation master server name, as featured in the keystore file, must match exactly the URL. If they do not match, no authorization is granted to access the server. If this attribute is not specified, the control is not enforced.  Note: For Workload Automation on Cloud users, this attribute cannot be specified.	
NumberOfRetries	The number of times that the program retries when connecting to the HCL Workload Automation master server. Default value is 0.	

Table 98. Required and optional attributes for the definition of a Job Stream Submission job (continued)

Attribute	Description and value	Required
RetryIntervalSeconds	The number of seconds that the program waits before retrying to connect to the HCL Workload Automation master server. Default value is 30 seconds.	
isMonitored	To synchronize the status of the job with the status of the job stream submitted by the job. This way you can easily monitor the status of the submitted job stream.	
specifyjobstream	To define the job stream that you want to submit for processing.	
workstation	The name of the workstation on which the job stream was defined. Specify the workstation in the format in the format <i>/folder_path/workstation_name</i> or <i>/workstation_name</i> , if the workstation is defined in the root (/) folder.	✓
jobstreamname	The name of the job stream. Use the format <i>[folder/]jobstreamname</i> . If <i>[folder/]</i> is omitted, then the root folder is assumed.	✓
alias	A unique name to be assigned to the job stream in place of jobstreamname.	
resubcurrjobstream	To resubmit the current job stream. This attribute is alternative to specifyjobstream.	
startafter	To define an offset in hours for the start time.	
delayforhours	The offset is calculated from the time of the submission of the Job Stream Submission job. Possible values can range from 00:00 to 23:59. The default value is '00:00'.	
startat	To define the time of day before which the job stream must not start. This attribute is alternative to startafter.	
time	The time of day before which the job stream must not start. Possible values can range from 00:00 to 23:59.	
delayfordays	You can specify an offset in days for the start time. The offset is calculated from the day of the submission of the Job Stream Submission job. The default value is '0'.	
variabletablename	The name of the variable table to be used by the job stream.	
variablelistValues	The list of variables in the variable table, and related values.	

The following example shows the job definition for a Job Stream Submission job:

```
<?xml version="1.0" encoding="UTF-8"?>
<jSDL:jobDefinition xmlns:jSDL="http://www.ibm.com/xmlns/prod/scheduling/1.0/jSDL"
xmlns:jSDLjobstreamsubmission="http://www.ibm.com/xmlns/prod/scheduling/1.0/jSDLjobstreamsubmission"
name="JOBSTREAMSUBMISSION">
  <jSDL:application name="jobstreamsubmission">
    <jSDLjobstreamsubmission:jobstreamsubmission>
      <jSDLjobstreamsubmission:JobstreamSubmissionParameters>
        <jSDLjobstreamsubmission:Connection>
          <jSDLjobstreamsubmission:connectionInfo>

<jSDLjobstreamsubmission:Url>${agent-config:master-address}</jSDLjobstreamsubmission:Url>
          <jSDLjobstreamsubmission:credentials>
            <jSDL:user_name>PayrollJob</jSDL:user_name>
            <jSDL:password>{aes}KB6vt65dWaZRL10khtimdeWRdCK3/t4wha76ozkju54ao=</jSDL:password>
          </jSDLjobstreamsubmission:credentials>
          <jSDLjobstreamsubmission:CertificateGroup>
            <jSDLjobstreamsubmission:keyStoreFilePath>${agent-config:keystore-file}</jSDLjobstreamsubmission:keyStoreFilePath>
            <jSDLjobstreamsubmission:keyStorePassword>${agent-config:keystore-file-password}</jSDLjobstreamsubmission:keyStorePassword>
          </jSDLjobstreamsubmission:CertificateGroup>
        </jSDLjobstreamsubmission:Connection>
      </jSDLjobstreamsubmission:JobstreamSubmissionParameters>
    </jSDLjobstreamsubmission:jobstreamsubmission>
  </jSDL:application>
</jSDL:jobDefinition>
```

```

        </jsdljobstreamsubmission:keyStorePassword>
        </jsdljobstreamsubmission:CertificateGroup>
        </jsdljobstreamsubmission:connectionInfo>
        <jsdljobstreamsubmission:retryOptions>
        <jsdljobstreamsubmission:NumberOfRetries>1</jsdljobstreamsubmission:NumberOfRetries>

<jsdljobstreamsubmission:RetryIntervalSeconds>30</jsdljobstreamsubmission:RetryIntervalSeconds>
        </jsdljobstreamsubmission:retryOptions>
        </jsdljobstreamsubmission:Connection>
    </jsdljobstreamsubmission:Action>
    <jsdljobstreamsubmission:isMonitored/>
    <jsdljobstreamsubmission:method>
    <jsdljobstreamsubmission:specifyjobstream>

<jsdljobstreamsubmission:workstation>/LONDON_DA</jsdljobstreamsubmission:workstation>

<jsdljobstreamsubmission:jobstreamname>JS_EXT_154814973</jsdljobstreamsubmission:jobstreamname>
    <jsdljobstreamsubmission:alias/>
        </jsdljobstreamsubmission:specifyjobstream>
        </jsdljobstreamsubmission:method>
    <jsdljobstreamsubmission:earlieststart>
    <jsdljobstreamsubmission:timeoptions>
    <jsdljobstreamsubmission:startafter>
    <jsdljobstreamsubmission:delayforhours/>
        </jsdljobstreamsubmission:startafter>
        </jsdljobstreamsubmission:timeoptions>
    </jsdljobstreamsubmission:earlieststart>
    <jsdljobstreamsubmission:managevariabletable>
    <jsdljobstreamsubmission:variabletablename/>
        </jsdljobstreamsubmission:managevariabletable>
        </jsdljobstreamsubmission:Action>
    </jsdljobstreamsubmission:JobstreamSubmissionParameters>
</jsdljobstreamsubmission:jobstreamsubmission>
</jsdl:application>
</jsdl:jobDefinition>

```

Status mapping

See table [Table 99: Status mapping on page 832](#) for the mapping between the internal status of the job stream submitted by the Job Stream Submission job and the internal status of the Job Stream Submission job.

Table 99. Status mapping

Internal status of the submitted job stream	External status of the submitted job stream	Internal status of the Job Stream Submission job	External status of the Job Stream Submission job
SUCC	Successful	SUCC	Successful
STUCK	Blocked	EXEC	Running
ABEND	Error	ABEND	Error
HOLD	Waiting	EXEC	Running
READY	Ready	EXEC	Running

Table 99. Status mapping (continued)

Internal status of the submitted job stream	External status of the submitted job stream	Internal status of the Job Stream Submission job	External status of the Job Stream Submission job
EXEC	Running	EXEC	Running
CANCL	Canceled	SUCC	Successful
CANCP	Canceled	EXEC	Running
SUPPR	Suppressed by condition	SUCC	Successful

Scheduling a job in HCL Workload Automation

You schedule Job Stream Submission jobs by defining them in job streams. Add the job to a job stream with all the necessary scheduling arguments and submit it.

Stopping and restarting a job

Stopping and restarting a Job Stream Submission job are not supported.

Job properties

For information about how to display the job properties from the various supported interfaces, see the section about analyzing the job log in *Scheduling Job Integrations with HCL Workload Automation*. For example, you can see the job properties by running `conman sj <job_name>;props`, where `<job_name>` is the Job Stream Submission job name.

You can export some of the Job Stream Submission job properties that you see in the `Extra Information` section of the output command, to a successive job in the same job stream or in a different job stream. For more information about the list of job properties that you can export, see [Table 106: Properties for Job Stream Submission jobs on page 851](#).

Job log content

You can see the job log content by running `conman sj <job_name>;stdlist`, where `<job_name>` is the Job Stream Submission job name.

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.

Return codes

The following is a list of the return codes for job types with advanced options

Database jobs:

RC = 0 -> Job completed successfully
RC = -1 -> SQL statement was run with an exit code different from 1
RC = -2 -> MSSQL Job error
RC = -3 -> SQL statement did not run because of an error in the statement

File transfer jobs:

RC = 0 -> The file transfer completed successfully

RC = -1 -> The file transfer is not performed. The job fails with the following error code: AWKFTE007E

Explanation: An error occurred during the file transfer operation

Possible reasons: Remote file not found or permission denied

RC = -2 -> The file transfer is not performed. The job fails with the following error code: AWKFTE020E

Explanation: Only for SSH or Windows protocols. An error was returned while attempting to convert the code page

Possible reasons: For SSH or Windows protocols, the code page is automatically detected and converted. In this case, there is an error in the code page of the file to be transferred, which is not compliant with the code page of the local system

RC = -3 -> The file transfer is not performed. The job fails with the following error code: AWKFTE015E

Explanation: An error occurred during the file transfer operation

Possible reasons: Local file is not found

RC = -4 -> The file transfer is performed with the default code page. The job fails with the following error code: AWKFTE023E

Explanation: The specified codepage conversion has not been performed. File transfer has been performed with default code pages

Possible reasons: The specified code page is not available

IBM i jobs:

Return code = user return code when retrieved
Return code = 0 -> job completed successfully
Return code > -1 -> job completed unsuccessfully

Java jobs:

RC = 0 -> Job completed successfully
RC = -1 -> The Java application launched by the job failed due to an exception

Web services jobs:

RC = 0 -> Job completed successfully
RC = -1 -> The server hostname contained in the Web Service URL is unknown
RC = -2 -> Web Service invocation error

When the user return code is retrieved, the IBM i Agent Monitor assigns a priority to it.

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 938](#) 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 938](#) utility command. This command has the power to create, delete, and list local variables in dynamic agents. See the [details on page 938](#) 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:\IBM\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

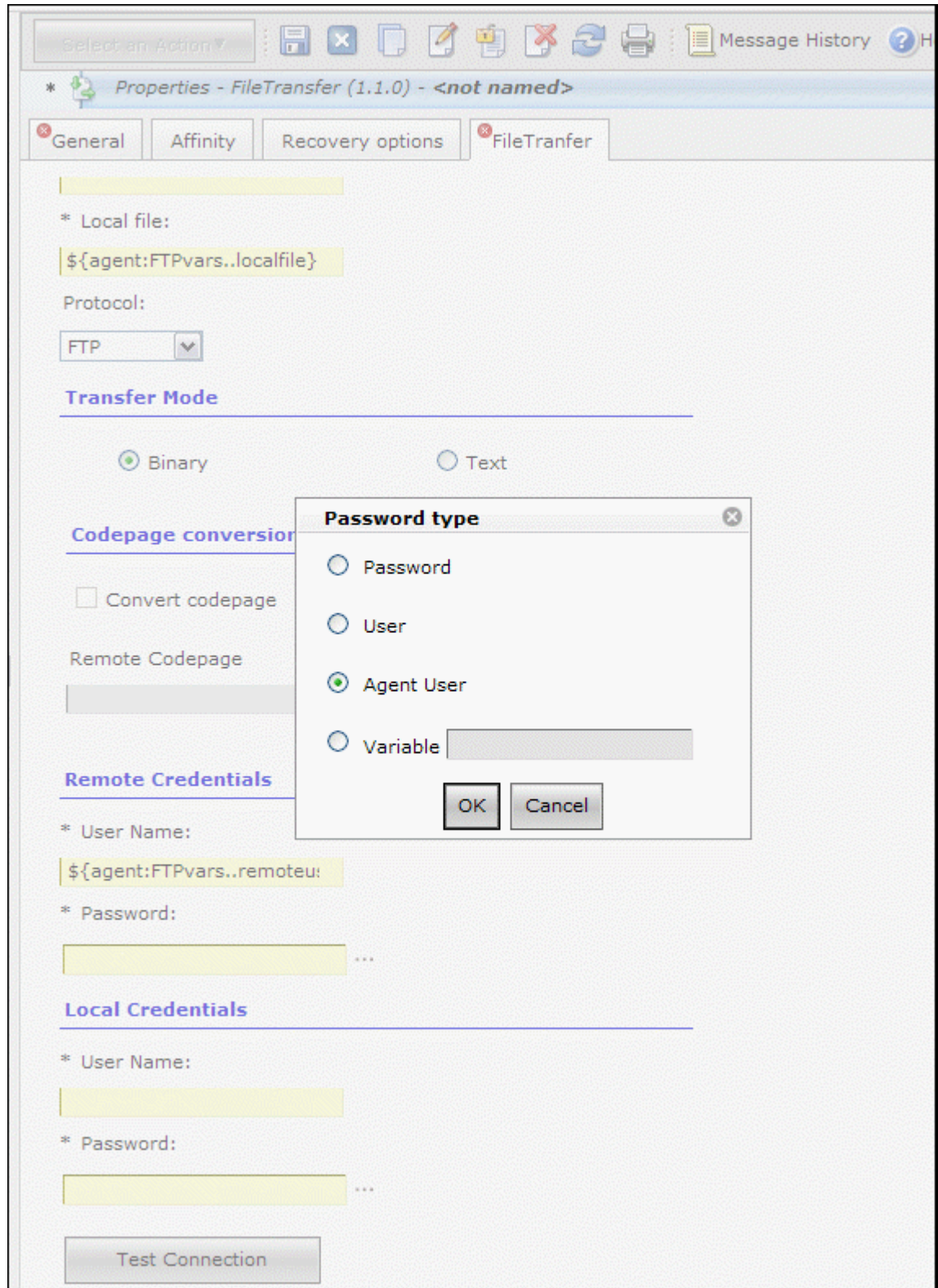
```
E:\HCL\TWA\TWS\ITA\cpa\config\jm_variables_files\FTPvars
```

and checks its contents:

```
localfile = ./npp.5.1.1.Installer.DW.exe
remoteuser = FTPuser

[password]
FTPuser = {aes}XMMYMY2zBHvDEDBo5DdZVmw0Jao60pX1K6x2HhRcovA=
```

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 835](#), 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 839](#). 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 840](#).

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=ITUser;...;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 840](#).

Examples of job definitions with password retrieval from CyberArk

Several examples in which a password vault is used to retrieve the password for a specified user.

The job runs on the local agent named `EU-HWS-WIN14_1` and uses a profile named `vaultProfile`, stored locally on the `EU-HWS-WIN14_1` agent

```
$JOBS
/F3_DA/EU-HWS-WIN14_1#CYBERARK_JOB_135223229
TASK
<?xml version="1.0" encoding="UTF-8"?>
<jsd: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">
  <jsd:application name="remotecommand">
    <jsdremotecommand:remotecommand>
      <jsdremotecommand:RemoteCommandParameters>
        <jsdremotecommand:taskPanel>
          <jsdremotecommand:command>netstat</jsdlremotecommand:command>
        </jsdlremotecommand:taskPanel>
        <jsdremotecommand:environmentPanel>
          <jsdremotecommand:standardOutput/>
          <jsdremotecommand:standardError/>
        </jsdlremotecommand:environmentPanel>
        <jsdremotecommand:serverPanel>
          <jsdremotecommand:serverInfo>
            <jsdremotecommand:serverName>EU-HWS-LNX122</jsdlremotecommand:serverName>
            <jsdremotecommand:port/>
            <jsdremotecommand:protocol>Auto</jsdlremotecommand:protocol>
          </jsdlremotecommand:serverInfo>
          <jsdremotecommand:credentials>
            <jsd:userName>rshLocalUser</jsdl:userName>

            <jsd:password>${vault:/F3_DA/EU-HWS-WIN14_1#VaultProfile:rshLocalUser}</jsdl:password>
          </jsdlremotecommand:credentials>
          <jsdremotecommand:certificates>
            <jsdremotecommand:keystoreFilePath/>
            <jsdremotecommand: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 `EU-HWS-WIN14_1` . Another agent, named `EU-HWS-LNX61_1`, interacts with the password vault using the default profile stored locally on the `EU-HWS-LNX61_1` agent.

```
$JOBS
/F3_DA/EU-HWS-WIN14_1#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>EU-HWS-LNX122</jsdLremoteCommand:serverName>
            <jsdLremoteCommand:port/>
            <jsdLremoteCommand:protocol>Auto</jsdLremoteCommand:protocol>
            </jsdLremoteCommand:serverInfo>
          <jsdLremoteCommand:credentials>
            <jsdL:userName>rshLocalUser</jsdL:userName>
            <jsdL:password>${vault:/F2_DA/EU-HWS-LNX61_1#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>EU-HWS-LNX122</jSDLremoteCommand:serverName>
          <jSDLremoteCommand:port/>
          <jSDLremoteCommand:protocol>Auto</jSDLremoteCommand:protocol>
        </jSDLremoteCommand:serverInfo>
      <jSDLremoteCommand:credentials>
        <jSDL:userName>rshLocalUser</jSDL:userName>

        <jSDL:password>${vault:DP00L4CYBERARK#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 **DP00L4CYBERARK** and applies the default profile stored locally on the **DP00L4CYBERARK** dynamic pool

```

$JOBS
DP00L4CYBERARK#CYBERARK_JOB_135452149
TASK
<?xml version="1.0" encoding="UTF-8"?>
<jSDL:jobDefinition xmlns:jSDL="http://www.ibm.com/xmlns/prod/scheduling/1.0/jSDL"
xmlns:jSDLremoteCommand="http://www.ibm.com/xmlns/prod/scheduling/1.0/jSDLremoteCommand"
name="REMOTECOMMAND">
  <jSDL:application name="remoteCommand">
    <jSDLremoteCommand:remoteCommand>
      <jSDLremoteCommand:RemoteCommandParameters>
        <jSDLremoteCommand:taskPanel>
          <jSDLremoteCommand:command>netstat</jSDLremoteCommand:command>
        </jSDLremoteCommand:taskPanel>
      <jSDLremoteCommand:environmentPanel>
        <jSDLremoteCommand:standardOutput/>
        <jSDLremoteCommand:standardError/>
      </jSDLremoteCommand:environmentPanel>
    <jSDLremoteCommand:serverPanel>
      <jSDLremoteCommand:serverInfo>
        <jSDLremoteCommand:serverName>EU-HWS-LNX122</jSDLremoteCommand:serverName>
        <jSDLremoteCommand:port/>
        <jSDLremoteCommand:protocol>Auto</jSDLremoteCommand:protocol>
      </jSDLremoteCommand:serverInfo>
    <jSDLremoteCommand:credentials>
      <jSDL:userName>rshLocalUser</jSDL:userName>
      <jSDL:password>${vault:DP00L4CYBERARK#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 **EU-HWS-WIN14_1**, uses a CyberArk query to retrieve the password and applies a profile named **vaultProfile** and stored locally on the **EU-HWS-WIN14_1** agent

```

$JOBS
/F3_DA/EU-HWS-WIN14_1#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>EU-HWS-LNX122</jsdlremotecommand:serverName>
                        <jsdlremotecommand:port/>
                        <jsdlremotecommand:protocol>Auto</jsdlremotecommand:protocol>
                    </jsdlremotecommand:serverInfo>
                    <jsdlremotecommand:credentials>
                        <jsdl:user_name>rshLocalUser</jsdl:user_name>

                        <jsdl:password>${vault:/F3_DA/EU-
HWS-WIN14_1#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: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 **EU-HWS-WIN14_1**, 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 **EU-HWS-WIN14_1** agent is used.

```
$JOBS
/F3_DA/EU-HWS-WIN14_1#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>EU-HWS-LNX122</jSDLremoteCommand:serverName>
            <jSDLremoteCommand:port/>
            <jSDLremoteCommand:protocol>Auto</jSDLremoteCommand:protocol>
          </jSDLremoteCommand:serverInfo>
          <jSDLremoteCommand:credentials>
            <jSDL:user>rshLocalUser</jSDL:user>

            <jSDL:password>${vault:/F3_DA/EU-HWS-WIN14_1AppDescs::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 100. Supported HCL Workload Automation variables in JSDL definitions.

Variables that can be inserted in the dynamic workload broker job definition	Description
twc.host.workstation	Name of the host workstation.
twc.job.date	Date of the submitted job.
twc.job.fqname	Fully qualified name of the job (UNISON_JOB).
twc.job.ia	Input arrival time of the job.
twc.job.interactive	Job is interactive. Values can be <code>true</code> or <code>false</code> . Applies only to jobs compatible with earlier versions.
twc.job.login	Credentials of the user who runs the job (LOGIN). Applies only to jobs compatible with earlier versions.
twc.job.name	Name of the submitted job.
twc.job.num	UNISON_JOBNUM.
twc.job.priority	Priority of the submitted job.
twc.job.promoted	Job is promoted. Values can be <code>YES</code> or <code>NO</code> . For more information about promotion for dynamic jobs, see Promoting jobs scheduled on dynamic pools on page 863 .
twc.job.recnum	Record number of the job.
twc.job.resourcesForPromoted	Quantity of the required logical resources assigned on a dynamic pool to a promoted job. Values can be <code>1</code> if the job is promoted or <code>10</code> if the job is not promoted. For more information about promotion for dynamic jobs, see Promoting jobs scheduled on dynamic pools on page 863 .
twc.job.taskstring	Task string of the submitted job. Applies only to jobs compatible with earlier versions.
twc.job.workstation	Name of the workstation on which the job is defined.
twc.jobstream.id	ID of the job stream that includes the job (UNISON_SCHED_ID).
twc.jobstream.name	Name of the job stream that includes the job (UNISON_SCHED).
twc.jobstream.workstation	Name of the workstation on which the job stream that includes the job is defined.
twc.master.workstation	Name of the master domain manager (UNISON_MASTER).
twc.plan.date	Start date of the production plan (UNISON_SCHED_DATE).
twc.plan.date.epoch	Start date of the production plan, in epoch format (UNISON_SCHED_EPOCH).

Table 100. Supported HCL Workload Automation variables in JSDL definitions. (continued)

Variables that can be inserted in the dynamic workload broker job definition	Description
tws.plan.runnumber	Run number of the production plan (UNISON_RUN).

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 execution 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 execution 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 846](#).
- *JobA* exports its standard output value and *JobB* references this standard output as a variable. At execution 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 852](#).
- *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 853](#).
- *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 execution 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 854](#).

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 856](#).



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

The job properties that you can export from one dynamic job to a successive job in the same job stream instance depend on the job type you are defining. To add a job property within another successor job definition, to have it 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.

Only some job types can pass property values to other successor jobs. The following types of job can export variables:

Shadow jobs

[Table 101: Properties for shadow jobs on page 849](#) 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 102: Properties for IBM Sterling Connect:Direct jobs on page 849](#) 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 103: Properties for file transfer jobs on page 850](#) 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 104: Properties for JSR 352 Java Batch jobs on page 850](#) 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 105: Properties for Job Management jobs on page 851](#) 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 106: Properties for Job Stream Submission jobs on page 851](#) 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 107: Properties for database jobs on page 852](#) 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 101. Properties for shadow jobs**Shadow job properties that can be passed in 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 102. Properties for IBM Sterling Connect:Direct jobs**IBM Sterling Connect:Direct properties that can be passed in 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

Table 102. Properties for IBM Sterling Connect:Direct jobs (continued)

IBM Sterling Connect:Direct properties that can be passed in another job definition	IBM Sterling Connect:Direct job Extra Information properties
<code>\${job:<JOB_NAME>.ProcessNumber}</code>	Process Number

Table 103. Properties for file transfer jobs

For the file number <i>counter</i> , the properties that can be passed in another job definition (a set of properties for each transferred file)	For the file number <i>counter</i> , the job Extra Information properties (a set of properties for each transferred file)
<code>\${job:<JOB_NAME>.File.counter.LocalFile}</code>	<code>File.counter.LocalFile</code>
<code>\${job:<JOB_NAME>.File.counter.LocalUser}</code>	<code>File.counter.LocalUser</code>
<code>\${job:<JOB_NAME>.File.counter.Protocol}</code>	<code>File.counter.Protocol</code>
<code>\${job:<JOB_NAME>.File.counter.RemoteFile}</code>	<code>File.counter.RemoteFile</code>
<code>\${job:<JOB_NAME>.File.counter.RemoteUser}</code>	<code>File.counter.RemoteUser</code>
<code>\${job:<JOB_NAME>.File.counter.Size}</code>	<code>File.counter.Size</code>
<code>\${job:<JOB_NAME>.File.counter.TotalTransferTime}</code>	<code>File.counter.TotalTransferTime</code>
<code>\${job:<JOB_NAME>.NumberOfTransferredFiles}</code>	Number of transferred files

Only one value for each job.



Note: The `${job:<JOB_NAME>.RemoteFile}` and `${job:<JOB_NAME>.LocalFile}` properties apply to all actions in the Hadoop Distributed File System job. The remaining properties apply to the **Wait for a file** action only.

Table 104. Properties for JSR 352 Java Batch jobs

JSR 352 Java Batch job properties that can be passed to another job	Label
<code>\${job:<JOB_NAME>.ExitStatus}</code>	Job Exit Status
<code>\${job:<JOB_NAME>.JobInstanceId}</code>	Job Instance Id

Table 104. 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>.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

`${job:<JOB_NAME>.<STEP_NAME>.BatchStatus}`

Step Batch Status

Table 105. 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 106. 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

Table 106. Properties for Job Stream Submission jobs (continued)

Job Stream Submission job properties that can be passed to another job	Label
<code>\${job:<JOB_NAME>.variabletablename}</code>	Variable Table Name
<code>\${job:<JOB_NAME>.variablelistValues}</code>	Variable Table Values {"key1":value1,"key2":value2,...}

Table 107. Properties for database jobs

Database job properties that can be passed to another job	Label
<code>\${job:<JOB_NAME>.NumberOfRows}</code>	Number of rows
<code>\${job:<JOB_NAME>.PPS_ PROPERTY_NAME}</code>	The name of the procedure expressed
<code>\${job:<JOB_NAME>.PPS_ PROPERTY_VALUE}</code>	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:
jSDL="http://www.ibm.com/xmlns/prod/scheduling/1.0/jSDL" name="executable">
<jSDL:application name="executable">
<jSDL:executable interactive="false" path="ls"/>
</jSDL:application>
</jSDL:jobDefinition>
DESCRIPTION "Added by composer for job stream: NC112019#JS_PROP."
RECOVERY STOP

NC112019#JOB B
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:
jSDL="http://www.ibm.com/xmlns/prod/scheduling/1.0/jSDL" XMLSchema:text=
"resolveVariableTable"
name="executable">
<jSDL:application name="executable">
<jSDL: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 935](#).

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#JOB B executable jobs. The NC125133#JOB B 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:
jSDL="http://www.ibm.com/xmlns/prod/scheduling/1.0/jSDL">
<jSDL:application name="executable">
<jSDL:executable interactive="false">
<jSDL:script>#!/bin/sh
. /home/ITAuser/TWA/TWS/tws_env.sh
jobprop VAR1 value1
jobprop VAR2 value2
jobprop VAR3 value3
jobprop VAR4 value4
</jSDL:script>
</jSDL:executable>
</jSDL:application>
</jSDL:jobDefinition>
DESCRIPTION "Sample Job Definition"
RCCONDSUCC "RC">=0"
RECOVERY STOP

NC125133#JOB B
TASK
<?xml version="1.0" encoding="UTF-8"?>
<jSDL:jobDefinition xmlns:jSDL=
"http://www.ibm.com/xmlns/prod/scheduling/1.0/jSDL" xmlns:
jSDL="http://www.ibm.com/xmlns/prod/
scheduling/1.0/jSDL">
<jSDL:application name="executable">
<jSDL:executable interactive="false">
<jSDL:script>
echo VAR1=${job:jobA.VAR1}
echo VAR2=${job:jobA.VAR2}
echo VAR3=${job:jobA.VAR3}
echo VAR4=${job:jobA.VAR4}
</jSDL:script>
</jSDL: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. For more information about defining a VariableTable job, see [Variable Table jobs on page 821](#)

Running a script when a job completes

In many scenarios, when a job completes, you might want to run one or more actions, by using the information related to the job completion. For this purpose, you can write a script file and store it in a directory of the agent file system. The script is run every time that a job completes, either successfully or unsuccessfully. The script runs with the same credentials as the agent user that is running the job.



Note: The agent user must be authorized to access the script file and its directory.

To provide HCL Workload Automation with the path of the script file, you must modify the `JobManager.ini` file as follows:

1. Locate the `JobManager.ini` file on the local agent instance where the script will run. The file is located in the `TWA_home/TWS/ITA/cpa/config` directory on the agent.
2. In the `[NativeJobLauncher]` section of the file, define the value of the **PostJobExecScriptPathName** property, with the fully qualified path of the script file that you want to run when a job completes.
3. Save the changes to the file.

If you do not specify any file path or the script file doesn't exist, no action is taken when the job completes. For details about customizing the **PostJobExecScriptPathName** property, see *Administration Guide*.

The following job variables can be used in the script:

- JOB_ID
- JOB_ALIAS
- JOB_SPOOL_DIR
- JOB_STATUS
- JOB_RETURN_CODE
- JOB_DURATION

- JOB_START_TIME
- JOB_END_TIME

The script is run for any of the following job final statuses:

- SUCCEEDED_ EXECUTION
- UNKNOWN
- CANCELLED
- FAILED_ EXECUTION

In the `JobManager_message.log`, you are notified via a message if the job started successfully, or if any error prevented the job from starting. To analyze the output of the script execution, you can check the `out.log` file in the `post_script` subdirectory of the job `SpoolDir`.

Chapter 18. Managing dynamic scheduling capabilities in your environment

This section explains how you can manage dynamic scheduling capabilities in your environment to schedule both existing HCL Workload Automation jobs and job types with advanced options, both those supplied with the product and the additional types implemented through the custom plug-ins.

Dynamic capabilities help you maintain business policies and ensure service level agreements by:

- Automatically discovering scheduling environment resources
- Matching job requirements to available resources
- Controlling and optimizing use of resources
- Automatically following resource changes
- Requesting additional resources when needed

You can enable dynamic capabilities to your environment by defining a set of workstation types:

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 defined and registered in the HCL Workload Automation database when you install the dynamic agent. You can group dynamic agents in pools and dynamic pools.

In a simple network, dynamic agents connect directly to its master domain manager or through a dynamic domain manager. In more complex network topologies where the master domain manager or the dynamic domain manager cannot directly communicate with the dynamic agent, you can configure your dynamic agents to use a local or remote gateway.

Pool

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

Dynamic pool

A workstation that groups a set of dynamic 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 is hosted by the workload broker workstation. This workstation maps all the dynamic agents in your environment that meet the requirements you specified. The resulting pool is dynamically updated whenever a new suitable dynamic agent becomes available. Jobs run on the first workstation in the dynamic pool which matches all the requirements. Jobs scheduled on this workstation automatically inherit the requirements defined for the workstation. For information about how to create pools and dynamic pools using the Dynamic Workload Console, see

the section on creating a pool of agents in the *HCL Dynamic Workload Console User's Guide*. For more information about how to create pools and dynamic pools using the **composer** command, see the *User's Guide and Reference*, SC32-1274.

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 the section on installing a new agent in the *Planning and Installation Guide*, SC32-1273.

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 schedule job types with advanced options, see [Creating advanced job definitions on page 807](#). For more information about how to create custom plug-ins, see *Extending HCL Workload Automation*.

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.

A business scenario on dynamic capability

This section demonstrates a sample business scenario which outlines the advantages of job types with advanced options and dynamic capability.

An insurance company runs a number of jobs at night to save the data processed during the day in the backup database. They also need to gather all data about the transactions completed during the day from all the workstations in the company branches. They use DB2 databases. Using the job types with advanced options provided in the Workload Designer, they create a job to perform a DB backup and another job to extract the data for the daily transactions. To perform these operations, they use the new database job type with advanced options.

After gathering data from all the company workstations, they copy the resulting data on a single workstation and process it to generate a report. They choose dynamically the best available workstation by defining the requirements necessary to run the job: a workstation with large disk space, powerful CPU and the program required to generate the report.

If the administrator does not want to modify the job stream he used before IBM Workload Scheduler. version 8.6 to run a Java job, for example, he can modify the name of the workstation where he wants the job to run, inserting the name of a pool or dynamic pool of dynamic agents where the Java executable is installed. HCL Workload Automation translates the syntax of the job so that it can be run by the Java program and assigns the job to the best available resource in the pool.

The report highlights how many new contracts were signed and how many customers are late with their payments. A mail is sent to the chief accountant, listing the number of new contracts and late customers.

The company can reach this objective by:

- Using the new workstations with dynamic capabilities to run the jobs the administrator created for the existing HCL Workload Automation workstations. To run these jobs on the new workstations, the administrator changes only the workstation where he wants the job to run. The major advantage is that he can use the workflows he previously created without additional effort.
- Defining several job types with advanced options without having specific skills on the applications where the job runs.

These job types with advanced options run on the following workstations:

dynamic agents

Workstations capable of running both existing jobs and job types with advanced options.

Pools

Groups to which you can add dynamic agents depending on your needs. Jobs are assigned dynamically to the best available agent.

Dynamic pools

Groups of dynamic agents for which you specify your requirements and let HCL Workload Automation select the dynamic agents which meet your needs. Jobs are assigned dynamically to the best available dynamic agent.

Defining file dependencies in dynamic scheduling

You can manage file dependencies with dynamic agents, pools, and dynamic pools

File dependencies introduction

You use file dependencies in dynamic scheduling to control job and job stream processing that is based on the existence of one or more files or directories. When you specify a file dependency, HCL Workload Automation processes check if the specified file or directory exists before job and job stream processing starts.

You can select one or more of the following conditions, which are associated to the file, that must be true before the jobs or job stream processing starts:

- The file exists.
- The file exists and is a directory.
- The file exists and is a regular file.
- The file exists and is readable.
- The file exists and its size is greater than zero.
- The file exists and is writable.

File dependencies behavior

The file dependencies have a different behaviour for dynamic agents, pools, and dynamic pools.

Dynamic agents

HCL Workload Automation manages the file dependency resolution for dynamic agents, in the same way as for the workstation that is defined in the classic scheduling as fault-tolerant agent, master domain manager and its backup, domain manager and its backup, and so on.

If you define the `JOB_A` job on `DYN_A` dynamic agent, which depends on the `FILE_A` file, before `JOB_A` job runs, HCL Workload Automation processes perform a file existence check on the `DYN_A` workstation. When the `FILE_A` file is found, the dependency is resolved and the `JOB_A` job runs on the `DYN_A` dynamic agent.

Pools and dynamic pools

A pool contains several dynamic agent workstations with similar hardware or software characteristics. A dynamic pool contains several dynamic agent workstations that are dynamically defined based on the resource requirements you specify. If you define a file dependency for jobs or job streams that are defined in a pool or a dynamic pool, HCL Workload Automation processes perform a file existence check on a dynamic agent workstation of the pool or dynamic pool, randomly selected.



Note: To check the file dependencies, you must create the file in a file system that is accessible by all the workstations in the pool.

The file dependency is resolved when the first file is found on a dynamic agent workstation. The job does not necessarily run on the dynamic agent workstation where the file is located, but runs on one of the active workstations in the pool or dynamic pool when the file dependency is resolved.

HCL Workload Automation processes perform a file check, and the code selects the workstations that are available inside the pool. However, The file check does not perform for all the workstations.

If you have to make the file dependency work. Ensure the path on which the file has been created; and is accessible for all the workstations in the pool i.e. You have to share the file system on which the file is created, with all the workstations in the pool.

In your environment, you have `P00L_A` that contains `DYN_A1`, `DYN_A2`, `DYN_A3` dynamic agents. If you define the `JOB_A` job in `P00L_A` pool, which depends on the `FILE_A` file, before `JOB_A` job runs, HCL Workload Automation processes perform a file existence check on the `DYN_A1`, `DYN_A2`, and `DYN_A3` workstations. If the `FILE_A` file is found on the `DYN_A2` workstation, the file dependency is resolved. The `JOB_A` job automatically runs on `DYN_A3` which is the active dynamic agent workstation in the `P00L_A` pool.

How to define file dependencies

You can define file dependencies for jobs and job streams on dynamic agents, pools, or dynamic pool workstations, by setting the `opens` keyword in the job or job stream scheduling definition. For more information about the `opens` keyword syntax, see [opens on page 310](#).

Example

The following example shows that HCL Workload Automation processes check that the `/opt/SF1-DIR/myfileSF1.txt` file exists and is readable on the `DYN-AGT-SF1` dynamic agent workstation before the `SF1-JOB-HOSTNAME-0001` job runs on the `DYN-AGT-SF1` workstation:

```
SCHEDULE MDMWKSNY1#NY1-JS1
VARTABLE VARTABLENY
OPENS DYN-AGT-SF1#"/opt/SF1-DIR/myfileSF1.txt" (-r %p)
:
DYN-AGT-SF1#SF1-JOB-HOSTNAME-0001
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" na
me="executable">
    <jsdl:application name="executable">
        <jsdle:executable interactive="false">
            <jsdle:script suffix="">hostname >>/opt/SF1-DIR/myfileSF1.txt
        </jsdle:script>
        </jsdle:executable>
    </jsdl:application>
</jsdl:jobDefinition>
RECOVERY STOP
END
```

See also

You can define the file dependencies for jobs and job streams also, by using the Dynamic Workload Console.

For more information about how to define file dependencies by using Dynamic Workload Console, see:

the *Dynamic Workload Console User's Guide*, section about Designing your Workload.

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:

twc.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.

twc.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: **\${twc.job.resourcesForPromoted}**.

When a job is scheduled on the dynamic pool, the value of the **twc.job.promoted** variable in the job determines the behavior of the dynamic pool:

- If the value of the **twc.job.promoted** variable is **NO**, the value of the **twc.job.resourcesForPromoted** variable on the dynamic pool is 10, which means that few resources match the requirement.
- If the value of the **twc.job.promoted** variable is **YES**, the value of the **twc.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 **twc.job.promoted** variable in the job and performs different actions based on whether or not the job is promoted.

Limitations in dynamic scheduling

Features and properties partially or not supported in dynamic scheduling

Dynamic scheduling supports most of the HCL Workload Automation features for static scheduling. The [Table 108: Features partially or not supported for dynamic scheduling on page 863](#) lists some features or properties that are partially or not supported.

Table 108. Features partially or not supported for dynamic scheduling

Feature	dynamic agent
Event-driven workload automation.	<code>TivoliWorkloadSchedulerObjectMonitor</code> events supported.
	<code>FileMonitor</code> events supported, except for IBM i systems.



Note: For more details about the events type, see *HCL Workload Automation User's Guide* and

Table 108. Features partially or not supported for dynamic scheduling (continued)

Feature	dynamic agent
 <i>Reference: Appendixes - Event-driven workload automation event and action definitions</i>	TivoliWorkloadSchedulerApplicationMonitor events not supported.
Utility commands (datecalc, jobinfo, and so on).	Partially supported.

Chapter 19. 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 109: List of utility commands on page 865](#) contains the list of the utility commands, and for each command, its description and the operating systems it supports.

Table 109. 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 109. List of utility commands (continued)

Command	Description	Operating system
exportserverdata	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®
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.	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 109. 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 <i>TWS_u</i> ser 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 <i>TWS_u</i> ser or root

Table 109. 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 the topic about <code>wa_pull_info</code> command and parameters in <i>User's Guide and Reference</i> .	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:

-queue

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 871](#) 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 869](#).

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 72](#).

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 workstation 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: 10.2.5
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 in the shell where you plan to run the command
- 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\twddb2user 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 in the shell where you plan to run the command
- 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.shhostname [port_number [IS_DA TRUE/FALSE]]
```

On Windows operating systems

```
da_test_connection.bathostname[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: alphanumeric, 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 915](#) 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>
  </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 110. Additional properties that can be used for defining custom events.

Property type	Add into XML event file as shown
boolean	<pre><property type="boolean" required="false" wildcardAllowed="false" multipleFilters="true" minlength="1"> <complexName displayName="Boolean field" name="Boolean"/> <displayDescription>Add a boolean field</displayDescription> </property></pre>
date	<pre><property type="date" required="false" wildcardAllowed="false" multipleFilters="true" minlength="1"> <complexName displayName="Date field" name="Date"/> <displayDescription>Add a date field</displayDescription> </property></pre>
datetime	<pre><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></pre>
datetimeutc	<pre><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></pre>
duration	<pre><property type="duration" required="false" wildcardAllowed="false" multipleFilters="true" minlength="1"> <complexName displayName="Duration field" name="Duration"/> <displayDescription>Add a duration field</displayDescription> </property></pre>
fileSize	<pre><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></pre>
nonnegativeinteger	<pre><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></pre>
numeric	<pre><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></pre>
percentage	<pre><property type="percentage" required="false" wildcardAllowed="false" multipleFilters="true" minlength="1"> <complexName displayName="Percentage field" name="percentage"/> <displayDescription>Add a percentage field</displayDescription> </property></pre>
string	<pre><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></pre>

or

Table 110. Additional properties that can be used for defining custom events.

(continued)

Property type	Add into XML event file as shown
	<pre> <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> </pre>

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>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 10.2.5 (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, 2025 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 150](#).

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, `my\path` is not equivalent to `my\path`.
- 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/my?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} [-modificationCompletedTime seconds]

The event type to be monitored. Supported types are **fileCreated** and **fileModified**. This argument is required when you specify **-path**. If you specify the **fileCreated** or the **fileModified** argument, you can optionally specify the **-modificationCompletedTime** 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, `ModificationCompleted`, is sent.

-event fileCreated -modificationCompletedTime <seconds>

When a file is created, the event is not sent immediately, but only after the interval of time specified by **-modificationCompletedTime <seconds>** has elapsed, and during which 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 immediately, but only after the interval of time specified by **-modificationCompletedTime <seconds>** has elapsed and during which no additional changes were made to the file.

-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.

-tw slog

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),(0AAAAAAAAAAAAEE)].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: **Chmm**, 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 876](#) 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]

[-tw slog]

**[{-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.

-tw slog

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 product 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 883](#) 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 883](#) 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 883](#). 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 Administrators 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 `tw_inst_pull_info` script, which is no longer supported.

Syntax

```
wa_pull_info -?
```

```
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 optional.

-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 924](#) 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 924](#) 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 20. 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 854](#).

[Table 111: List of utility commands for dynamic workstations on page 926](#) contains the list of the utility commands, and for each command, its description and the type of workstation where you can run it.

Table 111. 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 <code>jobprop</code> in one job to another in the same job stream instance on page 854 . It is installed in <code><TWS_INSTALLATION_DIR>/TWS/bin</code> 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 111. List of utility commands for dynamic workstations (continued)

Command	Description	Workstation type
		This command is installed in <code>TWA_home/TWS/CLI/bin.</code>
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 <code>TWA_home/TWS/CLI/bin.</code>
twstrace	Modifies at runtime the settings for tracing on agents	agents This command is installed in <code>TWA_home/TWS/CLI/bin.</code>



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 USERJOBS 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:

on Windows

```
TWA_home/TDWB/bin
```

on UNIX

```
<TWS_DATA>/TDWB_CLI/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 933](#)

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.tdwbdao.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 *ffilename*

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 932](#)

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:
jSDL="http://www.ibm.com/xmlns/prod/scheduling/1.0/jSDL">
    <jSDL:application name="executable">
      <jSDL:executable interactive="false">
        <jSDL:script>#!/bin/sh
. /home/ITAuser/TWA/TWS/tws_env.sh
jobprop VAR1 value1
jobprop VAR2 value2
jobprop VAR3 value3
jobprop VAR4 value4
        </jSDL:script>
      </jSDL: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:
<jSDL="http://www.ibm.com/xmlns/prod/scheduling/1.0/jSDL">
  <jSDL:application name="executable">
    <jSDL:executable interactive="false">
      <jSDL:script>
call C:\Progra~1\HCL\TWA\TWS\twS_env.cmd
jobprop var1 value1
jobprop var2 value2
jobprop var3 value3
jobprop var4 value4
</jSDL:script>
    </jSDL: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 951](#) for details. For more information about the CLIConfig.properties file, see [Command-line configuration file on page 927](#).

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 `CLConfig.properties`.

Authorization

The user name and password for the command are defined in the `CLConfig.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 `CLConfig.properties` file, see the section about the `CLConfig.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:

```
AWKCLII153I 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:

```
AWKCLII165I 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:

```
AWKCLII165I 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:

```
AWKCLII171I Calling the resource repository to perform a query on resources.
```

```
AWKCLII172I "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:

```
AWKCLII165I 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....
AWKCLII171I Calling the resource repository to perform a query on resources.
AWKCLII172I "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
AWKCLII171I 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:

```
AWKCLII153I 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....
AWKCLII171I Calling the resource repository to perform a query on resources.
AWKCLII173I "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....
AWKCLII165I 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....
AWKCLII171I Calling the resource repository to perform a query on resources.
AWKCLII173I "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 927](#).
3. Run the **resource** command. See [Running the resource command on page 952](#).

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
```

ITDWBServerHost

Specify the IP address or the hostname of the master domain manager.

ITDWBServerPort

Specify the number of the WebSphere Application Server Liberty HTTP port.

ITDWBServerSecurePort

Specify the number of the WebSphere Application Server Liberty HTTPS port.

tdwb_user

Specify the user name for a user authorized to perform operations on HCL Workload Automation when security is enabled. This user must be previously defined on WebSphere Application Server Liberty. For more information on security considerations, refer to *HCL Workload Automation: Administration Guide, SC23-9113*.

tdwb_pwd

Specify the password for a user authorized to perform operations on HCL Workload Automation when security is enabled. This password must be previously defined on IBM WebSphere. For more information on security considerations, refer to *HCL Workload Automation: Administration Guide*.

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 883](#) 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 883](#) 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 883](#). 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 915](#).

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] [-infile 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.

infile *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 -level 2000
```


To retrieve the information about the trace level, run the following command:

```
twstrace -level -maxFiles -maxFileBytes
```

```
AWSITA1761 The trace properties are: level="1000",  
maxFiles="3", file size="1024000"
```

Chapter 21. 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 958](#)
- [Command descriptions on page 959](#)
- [Sample report outputs on page 969](#)
- [Report extract programs on page 981](#)
- [Running Dynamic Workload Console reports and batch reports on page 995](#)
- [Running batch reports from the command line interface on page 1004](#)

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:

MAESTROL

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 112. Date formats

<i>date format</i> 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 113: List of report commands on page 959](#):

Table 113. 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 113. 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 **MAESTROL** variable. If **MAESTROL** 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 **In**clude and **Ex**clude 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 8.3 (1.7)          hcl          Page    1
Report 01                                     Job Details Listing      03/06/06

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 03/05/06 at 23:16)
```

```
Maximum      00:00:02      4 (On 03/04/06)
Minimum      00:00:01      4 (On 03/04/06)

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 03/05/06 at 22:22)
Maximum       00:00:01           0 (On 03/05/06)
Minimum       00:00:01           0 (On 03/05/06)

          * * * *   E n d   o f   R e p o r t   * * * *
```

In the output you see the values set in the [Job on page 1111](#) 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 8.3 (1.7)                hcl                Page      1
Report 02                                           Prompt Message Listing        03/06/06

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

[illegible]


```

      .   .   .   .   .   .   .   .   .   .   .   .   .   .   .   .   .   .   .   .
      .   .   .   .   .   .   .   .   .   .   .   .   .   .   .   .   .   .   .
      .   .   .   .   .   .   .   .   .   .   .   .   .   .   .   .   .   .   .
31                                     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
      .                           .   .   .   .   .               .   .   .
      .   .   .   .   .   .   .   .   .   .   .   .   .       .   .   .   .   .   .
      .   .   .   .   .   .   .   .   .   .   .   .   .       .   .   .   .   .   .
      .   .   .   .   .   .   .   .   .   .   .   .   .       .   .   .   .   .   .
      .   .   .   .   .   .   .   .   .   .   .   .   .       .   .   .   .   .   .
      .   .   .   .   .   .   .   .   .   .   .   .   .       .   .   .   .   .   .
      .   31                                     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 8.3 (1.7)          hcl          Page 1
Report 4A                                         User Parameter Listing 03/06/06

Parameter Name          Contents

ACCHOME                 /usr/local/Tivoli/maestro_adm
ACCLLOGIN               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 8.3 (1.7)		hcl	Page 1
Report 4B		TWS Resources Listing	03/06/06
CPU	Resource Name	Number Avail	Description
FTAHP	#DATTAPE\$	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 8.3   (1.13)      hcl      Page      1
Report 07      Job History Listing      03/08/06

Date      Time      Job Stream Name      Elapsed      CPU      Status

Job:MASTER8+#MyJS Runs: Aborted 0   Successful 11 Elapsed Time: Normal 1 Min 1 Max 2

03/03/06   01:46      MASTER8+#JS1      1      4      SU
03/03/06   19:08      MASTER8+#JS2      1      4      SU
03/03/06   19:33      MASTER8+#JS3      1      4      SU
03/03/06   19:37      MASTER8+#JS4      1      4      SU
03/03/06   23:08      MASTER8+#JS5      2      4      SU
03/03/06   05:59      MASTER8+#JS_A      1      4      SU
03/05/06   05:59      MASTER8+#JS_G      1      4      SU
03/06/06   05:59      MASTER8+#JS_H      1      4      SU
03/06/06   21:57      MASTER8+#TIMEJ      2      4      SU
03/06/06   23:16      MASTER8+#SLEEPJ      1      4      SU

Job:MASTER8+#JOB1      Runs: Aborted 0   Successful 1 Elapsed Time: Normal 1 Min 1 Max 1

03/06/06   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 8.3 (1.7)          hcl                      Page    1
Report 08                      Job Histogram 03/05/06 14:05 - 03/06/06 14:04    03/06/05

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
03/06/06

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 10.2.5 (1.7)      hcl                      Page    1
Report 09B  Symnew      Planned Production Detail For 03/06/22          03/06/22

      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
      JOBIWD      10      23:00(03/06/22) 01:00      NETAG#EXTERNAL.E0000000
      Total      00:00

Schedule MYMST #TESTSKE      00:29 10
      TESTCRO+      00:01 10
      NEWTEST      00:29 10 08:30(03/06/22)      TESTCROME
      Total      00:29

```

```
Schedule MYMST #FINAL      00:00  10 05:59(03/07/22)
      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(03/20/22)
```

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{03/20/22}
```

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 03/20/22
```

Report 10B - Actual Production Detail:

Example

TWS for UNIX (AIX) REPORTER 8.3 (1.7)				hcl		Page	1
Report 10B	Symphony		Actual Production Detail For 03/06/19				03/07/19
		Estimated		Actual	CPU	Job	
	Job Name	Run Time	Priority	Start Time	Run Time	Seconds	Status
Schedule	NETAG #EXTERNAL		0				EXTRN
	E0000000		0				ERROR
	Total	00:00			00:00	0	
Schedule	MYMST #MONTHSKE	00:02	10	06:01(03/06/19)	00:03		SUCC
	GETLOGS	00:02	10	06:01(03/06/19)	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(03/06/19) 00:02          STUCK
                   TESTCRO+    00:01      10      06:01(03/06/19) 00:02          #J11613 ABEND
                   NEWTEST      00:29      10          HOLD
                   Total      00:30          00:02      0
Schedule MYMST #FINAL          00:01      10      05:59(03/07/19)          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 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 **ActualRuntime** field is displayed as HH:MM. Assuming the elapsed time of the joblog output is 50 seconds (`hh:mm:ss`) : `0:00:50`, the output of Report 10B is displayed as follows: `00:01` ;



Note: When value of seconds is greater then 30, the duration is reported as 1 minute, to indicate that the job duration is higher than 30 seconds.

Report 11 - Planned Production Schedule:

Example

```

TWS for UNIX (AIX)/REPORT11 10.2.5 (1.7)                                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
SCHD1       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						

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	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						

* * * * End of Report * * * *

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 8.3 (1.7)          hcl                      Page    1
Report 12 Cross Reference Report for the ON, EXCEPT(*) and FREEDAYS(f) options. 03/08/06

CPU: FTAHP

WHEN                Used by the following schedules:
REQUEST             TRFINAL

TWS for UNIX (AIX)/CROSSREF 8.3 (1.7)          hcl                      Page    2
Report 12 Cross Reference Report for the ON, EXCEPT(*) and FREEDAYS(f) options. 03/08/06

CPU: FTAWIN8+

WHEN                Used by the following schedules:
MONTHEND            SCHED1
REQUEST             SCHED1 , SCHEDDAA

TWS for UNIX (AIX)/CROSSREF 8.3 (1.7)          hcl                      Page    3
Report 12 Cross Reference Report for the ON, EXCEPT(*) and FREEDAYS(f) options. 03/08/06

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 8.3 (1.7)hclPage 4
Report 12Cross Reference Report for Job Names.03/08/06

CPU: FTAWIN8+

Job Name	Exists in Schedules
SCHEDDD	SCHED1

TWS for UNIX (AIX)/CROSSREF 8.3 (1.7)hclPage 5
Report 12Cross Reference Report for Job Names.03/08/06

CPU: MASTER8+

Job Name	Exists in Schedules
JnextPlan	FINAL
JOB1	TMP

***** End of Report *****

xref -resource

Example

TWS for UNIX (AIX)/CROSSREF 8.3 (1.7)hclPage 8
Report 12Cross Reference Report for Resource Users.03/08/06

CPU: FTAWIN8+

Resource	Used by the following:
QUKTAPES(N/F)	SCHED1

TWS for UNIX (AIX)/CROSSREF 8.3 (1.7)hclPage 9
Report 12Cross Reference Report for Resource Users.03/08/06

CPU: MASTER8+

Resource	Used by the following:
TAPES(N/F)	TMP

***** End of Report *****

xref -prompts

Example

TWS for UNIX (AIX)/CROSSREF 8.3 (1.7)hclPage 6
Report 12Cross Reference Report for Prompt Dependencies.03/08/06

CPU: FTAWIN8+


```
Prompt                Used by the following:
User defined text      SCHED1

TWS for UNIX (AIX)/CROSSREF 8.3 (1.7)      hcl                Page    7
Report 12                Cross Reference Report for Prompt Dependencies.    03/08/06

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 8.3 (1.7)      hcl                Page   10
Report 12                Cross Reference Report for File Dependencies.    03/08/06

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 114: Report extract programs. on page 981](#):

Table 114. 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

Table 114. Report extract programs. (continued)

Report extract program	Description
rextract	Used to generate Report 04B - Resource Listing
r11xtr	Used to generate Report 11 - Planned Production Schedule
xrxtrect	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 959](#).

jbextract

Extracts information about jobs from the database.

Syntax

jbextract [-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 115: Jbextract output fields on page 983](#).

Table 115. Jbextract 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
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 116: Prxtract output fields on page 984](#).

Table 116. 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 117: Caxtract output fields on page 985](#).

Table 117. 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 118: Paxtract output fields on page 986](#).

Table 118. 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 119: Rextract output fields on page 987](#).

Table 119. 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 120: R11xtr output fields on page 988](#).

Table 120. 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

Table 120. R11xtr output fields (continued)

Field	Description	Max Length (bytes)
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
```

xrxtct

Extracts information about cross-references from the database.

Syntax

xrxtct [-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:

```
xrxtret
```

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 121: Xdep_job output fields on page 990](#).

Table 121. 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 122: Xdep_job output fields \(continued\) on page 990](#).

Table 122. Xdep_job output fields (continued)

Field	Description	Length (bytes)
1	08	2
2	workstation name	16
3	job name	120

Table 122. Xdep_job output fields (continued) (continued)

Field	Description	Length (bytes)
4	not used	128
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

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 123: Xdep_sched output fields on page 991](#).

Table 123. 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 124: Xfile output fields on page 992](#).

Table 124. 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 125: Xjob output fields on page 992](#).

Table 125. 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 125. 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 126: Xprompts output fields on page 993](#).

Table 126. 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 127: Xresource output fields on page 993](#).

Table 127. Xresource output fields

Field	Description	Length (bytes)
1	06	2

Table 127. 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 128: Xsched output fields on page 994](#).

Table 128. 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 128. 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 129: Xwhen output fields on page 995](#).

Table 129. 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 1004](#).

Depending on the interface from where you run the report or the operating system of the engine the following output formats are available:

Table 130. 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 130. 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 131. 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 131. Summary of historical reports (continued)

Rep ort 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 statis tics	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 131. Summary of historical reports (continued)

Rep ort name	Description	Selection criteria	Output content options
	• Average duration • Late and long duration statistics	late, Ended late, and Long duration	◦ Script

Table 131. 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 131. Summary of historical reports (continued)

Rep ort name	Description	Selection criteria	Output content options
load sum mary	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
Works tation work load run ti mes	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 131. Summary of historical reports (continued)

Rep ort 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 132.

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 998](#).

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.

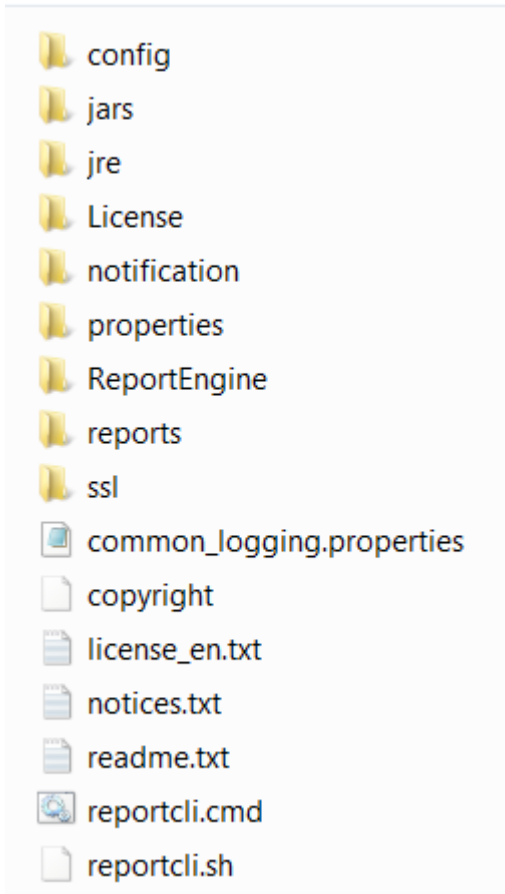
Setting up for command line batch reporting

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 context 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 1005](#), 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 context 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:

```
reportclibatch
-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 22. 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 1011](#)
- [How HCL Workload Automation manages time zones on page 1012](#)
- [Moving to daylight saving time on on page 1014](#)
- [Moving to daylight saving time off on page 1015](#)
- [General rules on page 1015](#)

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.

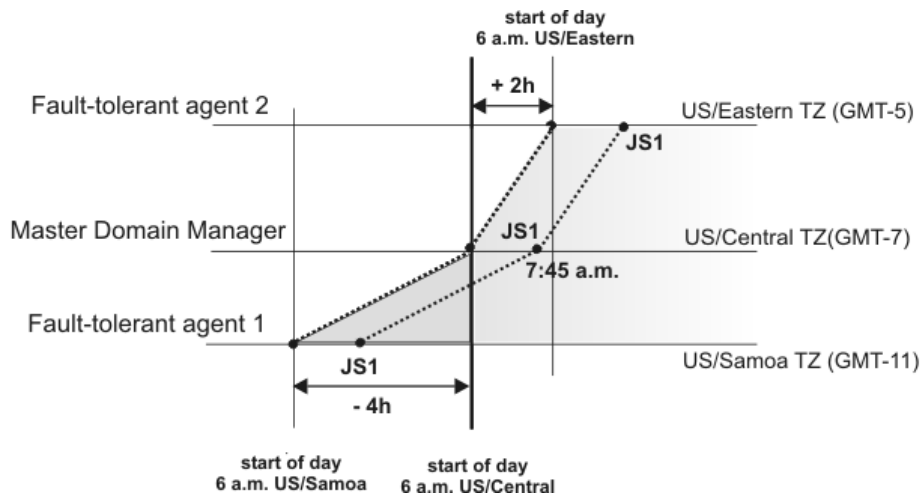
This is why if you use the **conman showsched** or **conman showjobs** commands to see the information about scheduled jobs and job streams you see the time zone values expressed using the time zone set on the workstation where the job or job stream is planned to run. Based on the setup of the *enLegacyStartOfDayEvaluation* 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 1013](#) 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 1013](#).

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

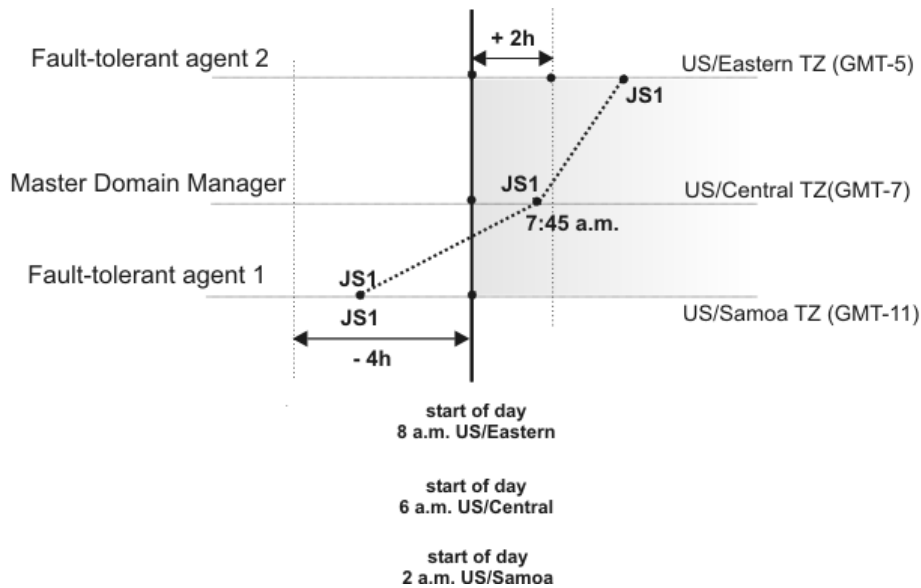


If you set the value of *enLegacyStartOfDayEvaluation* to yes

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 1014 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 1014.

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 23. Defining access methods for agents

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*.

More information about access methods is found in *Scheduling Job Integrations with HCL Workload Automation*.

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 1018](#) for more information.

taskstring

A string of up to 255 characters associated with the task. See [Task options on page 1018](#).

Task options

The task options are listed in [Table 133: Method command task options on page 1019](#). An X means that the option is valid for the task.

Table 133. 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	<i>ljstring</i>
MJ	X	X	X	X	X	X	X	X	X				<i>mjstring</i>
CF	X	X	X							X			<i>cfstring</i>
GS	X	X	X	X		X					X		<i>gsstring</i>

-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 1020](#)) 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 1024](#) and [Manage job task \(MJ\) on page 1025](#).

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 1026](#).

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 134: Launch job task \(LJ\) messages on page 1025](#).

Table 134. Launch job task (LJ) messages

Task	Method Response	HCL Workload Automation Action
LJ and MJ	%CJ <i>state</i> [<i>mjstring</i>]	Sets job state to <i>state</i> . Includes <i>mjstring</i> in any subsequent MJ task.
	%JS [<i>cputime</i>]	Sets job state to SUCC.
	Exit code=non-zero	Sets job state to ABEND.
	%UT [<i>errormessage</i>] and Exit code=2	Sets job state to ABEND and displays <i>errormessage</i> .

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 1024](#) 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 135: Check file task \(CF\) messages on page 1026](#).

Table 135. 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.

cancel

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 1021](#) for more information.

Method responses are summarized in [Table 136: Get status task \(GS\) messages on page 1027](#):

Table 136. Get status task (GS) messages

Task	Method Response	HCL Workload Automation Action
GS	%CJ <i>state</i> [<i>jobid</i>]	Sets job state to <i>state</i> and includes <i>jobid</i> in any subsequent GS task.
	%UT [<i>errormessage</i>] 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 1027](#) for complete command information.

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

```
AWSBH058E 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 **jobmans 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 24. 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 1030](#)
- [Configuring a network agent on page 1032](#)
- [Defining an internetwork dependency on page 1034](#)
- [Managing internetwork dependencies in the plan on page 1035](#)
- [Internetwork dependencies in a mixed environment on page 1038](#)



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 40](#) 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 `ELISCHE` 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

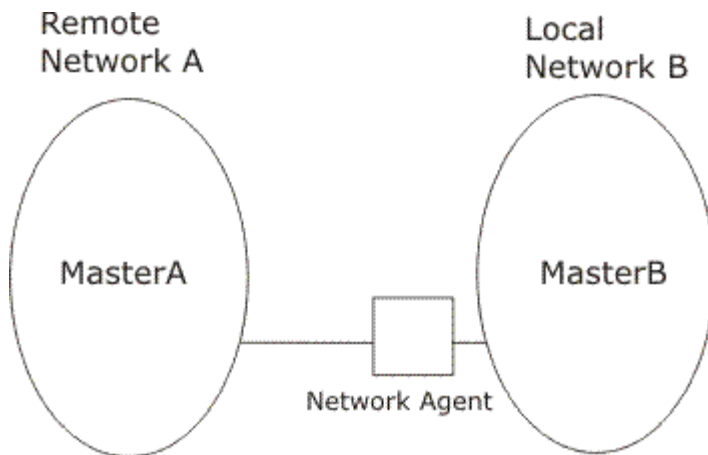
Is the number of seconds, HCL Workload Automation waits for a response before shutting down the access method. The default setting is 300 seconds. The next time **batchman** needs to check the status of the remote predecessor job or job stream, the access method starts automatically.

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 27. Local and remote networks



Assuming that:

- **MasterA** is the master domain manager of the remote network, **Network A**, and that:
 - **twc_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:
 - **twc_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=twc_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 is 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 `rsched`
- 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 afile";follows=netagt::remote1#rsched
```

Internetwork dependencies in a mixed environment

Table 137: Internetwork dependencies in a mixed environment on page 1039 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 137. 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 <code>wks_B</code> as if it had the same version as <code>wks_B</code> .	Net_A sends the information to <code>wks_B</code> in 8.1, 8.2, or 8.2.1 format. The use of the <i>schedtime</i> keyword in the job definition is not supported.	Net_A sends the information to <code>wks_B</code> as if it had the same version as <code>wks_B</code> . If defined, the <i>schedtime</i> keyword in the job definition is automatically removed by Net_A.
Wks_B 8.3 Sym_B back-level	<code>wks_B</code> works as if it had the same version as Net_A.	Net_A sends the information to <code>wks_B</code> . If defined, the <i>schedtime</i> keyword in the job definition is automatically removed by <code>wks_B</code> .	Net_A sends the information to <code>wks_B</code> . If defined, the <i>schedtime</i> keyword in the job definition is automatically removed by <code>wks_B</code> .	Net_A sends the information to <code>wks_B</code> . If defined, the <i>schedtime</i> keyword in the job definition is automatically removed by <code>wks_B</code> .
Wks_B back-level Sym_B 8.3	Not supported.	Not supported.	Not supported.	Not supported.
Wks_B 8.3 Sym_B 8.3	Not supported.	Not supported.	Net_A sends the information to <code>wks_B</code> . If defined, the <i>schedtime</i> keyword is parsed by <code>wks_B</code> .	This is a version 8.3 environment.

Chapter 25. Applying conditional branching logic

With IBM® Workload Scheduler you can define jobs to run when and as often as necessary. Sometimes some jobs might have to wait for other jobs to finish successfully before they start. Add even more flexibility to your job flows by choosing which job to run depending on the result of the job status or output of a previous job. Whenever you have conditions that specify whether or not a segment of your job flow should run, then that is a conditional dependency.

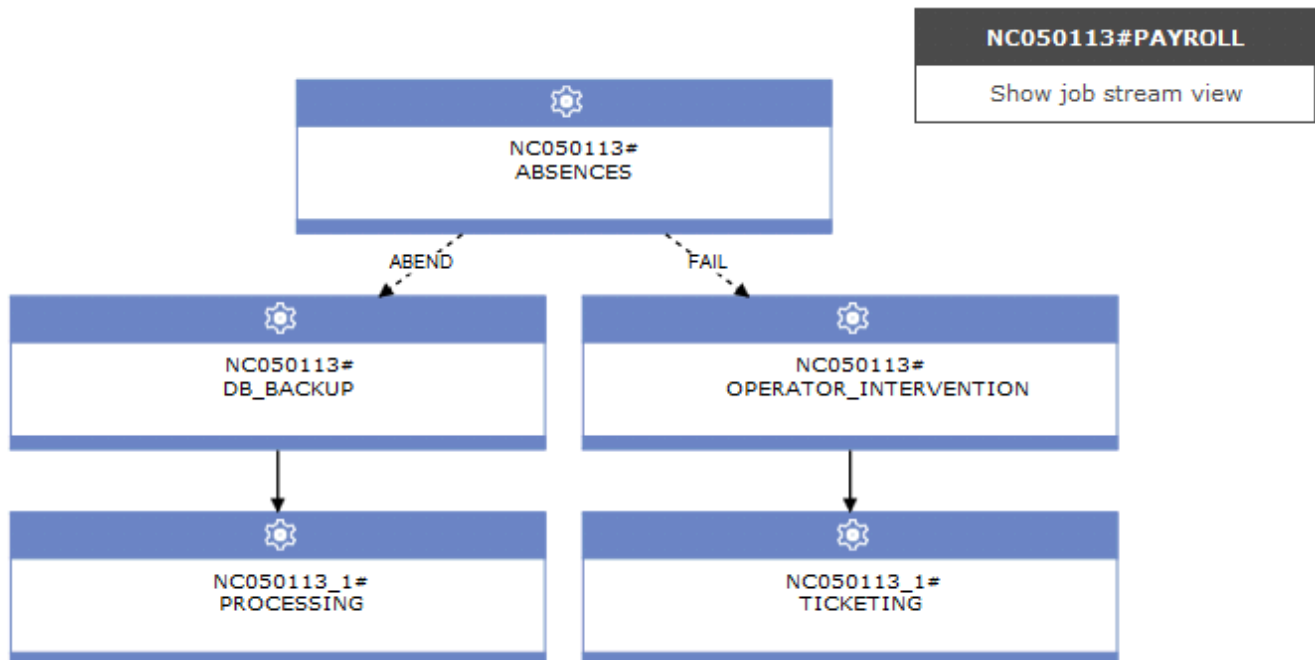
When specifying dependencies, you can define job flows with alternative branches based on conditions, specifically to achieve the same results as using IF/THEN/ELSE statements. You can use return codes, job status, output variables, and job log content as *conditional logic* elements to determine the start of a successor job. In addition to providing flexibility to your job flows, the Graphical View provides a graphical representation of the relationships between the jobs and job streams, including the dependencies and conditions. This at-a-glance view of your job flow is easy to read and you can also edit your job flow from this view.



Note:

Conditional dependencies are evaluated after any standard dependencies in the job or job stream are satisfied.

The following example shows the PAYROLL job stream that starts with the ABSENCES job, which is a predecessor job and is then followed by two possible branches of jobs that can run. The branch that runs depends on the outcome of the initial job, the predecessor ABSENCES job. Possible outcomes of the ABSENCES job are defined in *output conditions*. Any jobs in the flow that do not run, because the output conditions were not satisfied, are put in SUPPRESSED state, which is different from regular dependencies where jobs are put in Hold until the predecessor is in successful (SUCC) state. Predecessors can be either jobs or job streams.



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 in a job log. When the predecessor is a job stream, the conditional dependency can only be a status condition.

Status conditions

These are conditions based on job status, such as if the job started, or if the job completes in FAIL, ABEND, SUCC, or SUPPR state. For job streams, the valid statuses are SUCC, SUPPR, and ABEND.

Other output conditions

Other types of conditions, including successful output conditions, can be specified using a mapping expression, which can be:

- A return code (fault-tolerant and dynamic agents)
- Output variables (dynamic agents)
- Job log content (dynamic agents)

A condition dependency relationship is set up by using a *condition*. You specify the output conditions in the job definition. You can define an unlimited number of conditional dependencies. When you choose to add a conditional dependency on this job, you select the status and output conditions that you want to be considered during the job processing. The following example shows other output conditions defined in the job definition.

Properties - Executable (9.3.0.01) - NC050024_1_1#ABSENCES

General Affinity Recovery Options Credentials Environment Task

* Name: ABSENCES * Workstation: NC050024_1_1

Description:

☐ Variable resolution at runtime

Successful Output Conditions (force the job status to successful):

Condition Name	Condition Value
+	

Other Output Conditions (do not change the job status):

Condition Name	Condition Value
DB_FAIL	RC=1
TEMP_FULL	RC=3
UNKNOWN_ERR	RC>3
WAS_FAIL	RC=2
+	

You can define both successful output conditions, conditions that when satisfied signify that the job completed successfully, and other output conditions, which when satisfied determine which successor job to run. The output conditions are evaluated in "OR".

When this job is added to a job stream as a successor job, and a conditional dependency is added to the job preceding this job (predecessor job), then a selection of the conditions is made. The properties panel for the internal or external dependency is dynamically updated to include the conditions originally specified in the job definition. In addition to the conditions originating from the job definition, you can select conditions based on job status. If the selected conditions are satisfied during job processing, then the corresponding successor job runs.

Properties - Internal Job Dependency - ABSENCES

General

* **Name:**
ABSENCES

* **Workstation:**
NC050024

☒ **Conditional Dependency**

NOTE: The successor job will be suppressed if the conditional dependency is not satisfied. [More information](#)

Conditional Dependency Resolution Criteria

☐ Job started

☒ Successor job runs if the predecessor job or job stream completes with any of these statuses

☐ ABEND ☐ FAIL ☒ SUCC ☐ SUPPRESS

☐ Successor job runs if any of these conditions are satisfied

☒ UNKNOWN_ERR ? [Add a condition](#)

☐ DB_FAIL ?

☐ TEMP_FULL ?

☐ WAS_FAIL ?

You can also join or aggregate conditional dependencies related to different predecessors. 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.

Conditional dependencies are supported only for dependencies where the predecessor is a job or a job stream in the same network and where all the components are at least at the Version 9.3 Fix Pack 2 level. They are not supported on internetwork dependencies, nor on Limited Fault-Tolerant Agents for IBM i.

Setting up conditional dependencies

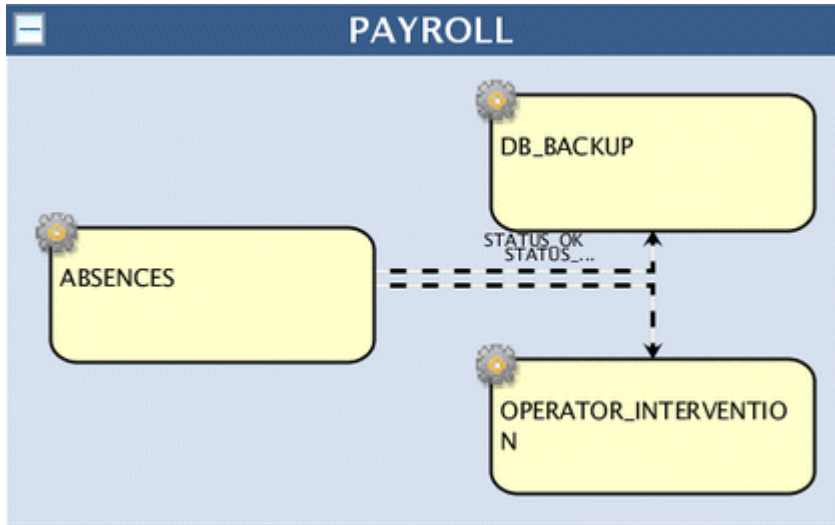
You can set up conditional dependencies to define workflows with alternative branches based on conditions.

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.

Ensure that all the components in the IBM Workload Scheduler environment are at version 9.3, Fix Pack 1, or later.

In the following example, the DB_BACKUP job runs if the ABSENCES job satisfies the condition associated with the STATUS_OK, and the OPERATOR_INTERVENTION job runs if the ABSENCES job satisfies the condition defined for STATUS_ERR1.



To set up this type of job processing, complete the following steps:

1. Create the job definition and define the output conditions.
2. Define the conditional dependency.

To create the PAYROLL job stream, complete the following steps:

1. Add the ABSENCES, DB_BACKUP, and OPERATOR_INTERVENTION jobs to the job stream named PAYROLL.
2. Add a dependency, in this case an internal dependency, to the DB_BACKUP job. In the properties of the internal job dependency, choose to make this a conditional dependency by selecting the **Conditional Dependency** check box.
3. In Conditional Dependency Resolution Criteria, select **Successor job runs if any of these conditions are satisfied** and then select **STATUS_OK**.
4. Add a dependency to the OPERATOR_INTERVENTION job. In the properties of the internal job dependency, choose to make this a conditional dependency by selecting the **Conditional Dependency** check box.
5. In Conditional Dependency Resolution Criteria, select **Successor job runs if any of these conditions are satisfied** and then select **STATUS_ERR1**.
6. Save the job stream.

Example

You can define the same scenario by using the composer command line as follows:

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 the composer command line see [Job definition on page 205](#), [Job stream definition on page 252](#), and [follows on page 281](#).

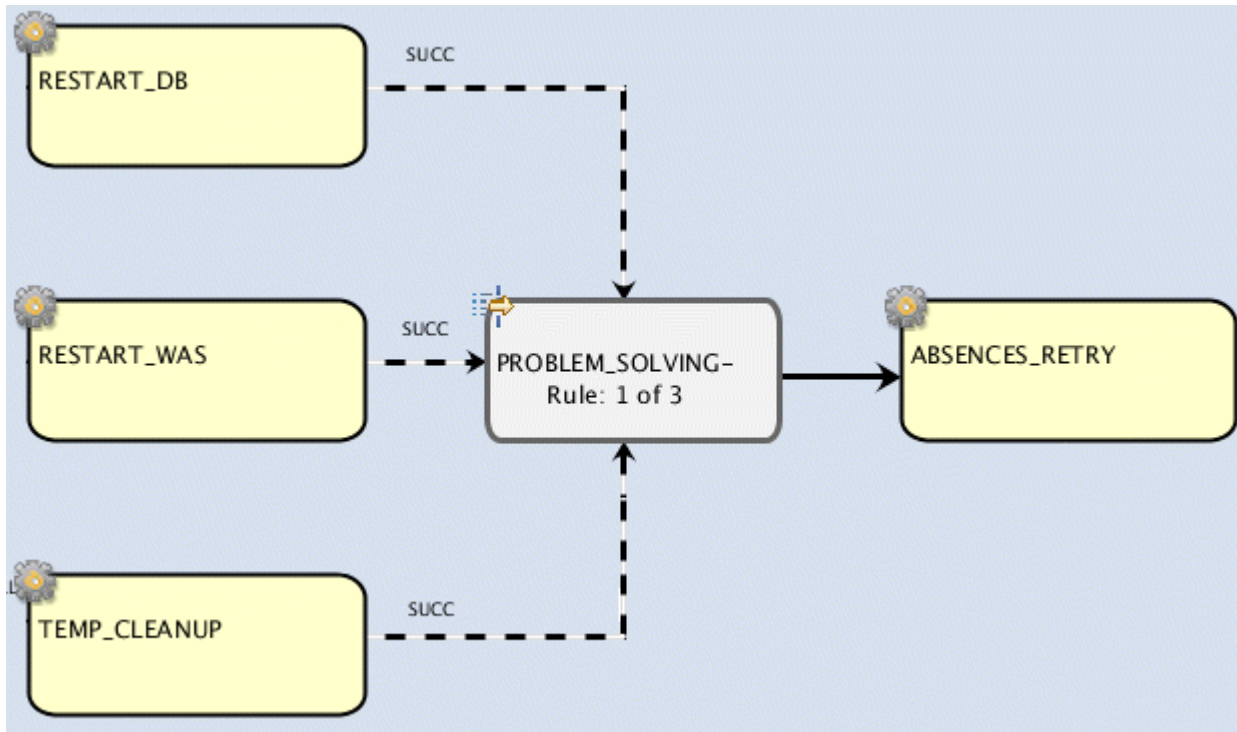
Joining or combining conditional dependencies

You can choose to combine a set of dependencies into a joined dependency.

About this task

You can add multiple dependencies related to different predecessors to a join dependency and then specify how many of those dependencies must be satisfied to consider the join satisfied. When the join is satisfied, then the successor job runs.

In the following example, a join dependency, PROBLEM_SOLVING was inserted into the job stream and aggregates three conditional dependencies, each related to different predecessor jobs. Only one conditional dependency must be satisfied to consider the PROBLEM_SOLVING join satisfied. The ABSENCES_RETRY job follows the PROBLEM_SOLVING join and runs when at least one of the three predecessor jobs completes successfully, thereby satisfying the rule established on the join. If the join is not satisfied then the ABSENCES_RETRY job is suppressed.



To set up the join in this example, complete the following steps:

1. Add the RESTART_DB, RESTART_WAS, TEMP_CLEANUP, and ABSENCES_RETRY jobs to a job stream.
2. Right-click the ABSENCES_RETRY job and select **Add Dependencies > Join Dependencies**.
3. In the properties for the join dependency, assign a name, PROBLEM_SOLVING.
4. The rule to be applied to the PROBLEM_SOLVING join is at least one of the dependencies must be satisfied. Leave the default selection, **At least 1**.
5. Right-click the PROBLEM_SOLVING join dependency in the Details view and select **Add Dependencies > Job in the same Job Stream**.
6. Click **Search** to display all the jobs in the job stream.
7. Select the RESTART_DB, RESTART_WAS, and TEMP_CLEANUP jobs and click **Add**.
8. For each internal job dependency, edit the properties to make it a conditional dependency.
 - a. Select the **Conditional Dependency** check box.
 - b. In **Conditional Dependency Resolution Criteria**, select **Successor job runs if the predecessor job or job stream completes with any of these statuses**.
 - c. Select the **SUCC** check box.
 - d. Save the changes.

Results

The three conditional dependencies on the ABSENCES_RETRY job have been joined together in the PROBLEM_SOLVING join dependency where at least one of the conditions (predecessor job completes in SUCC), must be satisfied for the ABSENCES_RETRY job to run.

Example

You can define the same scenario by using the composer command line as follows:

```
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 the composer command line, see [join on page 289](#).

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.

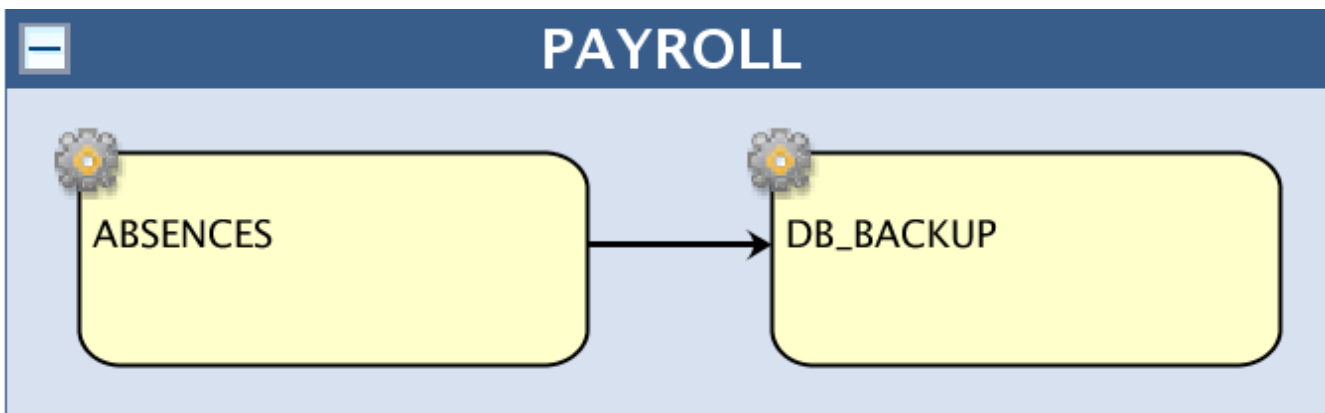
If you rerun a job or job stream, the evaluation of the conditional dependency flow is cleared and all dependencies are evaluated again.

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 28. A follows dependency on the ABSENCES job

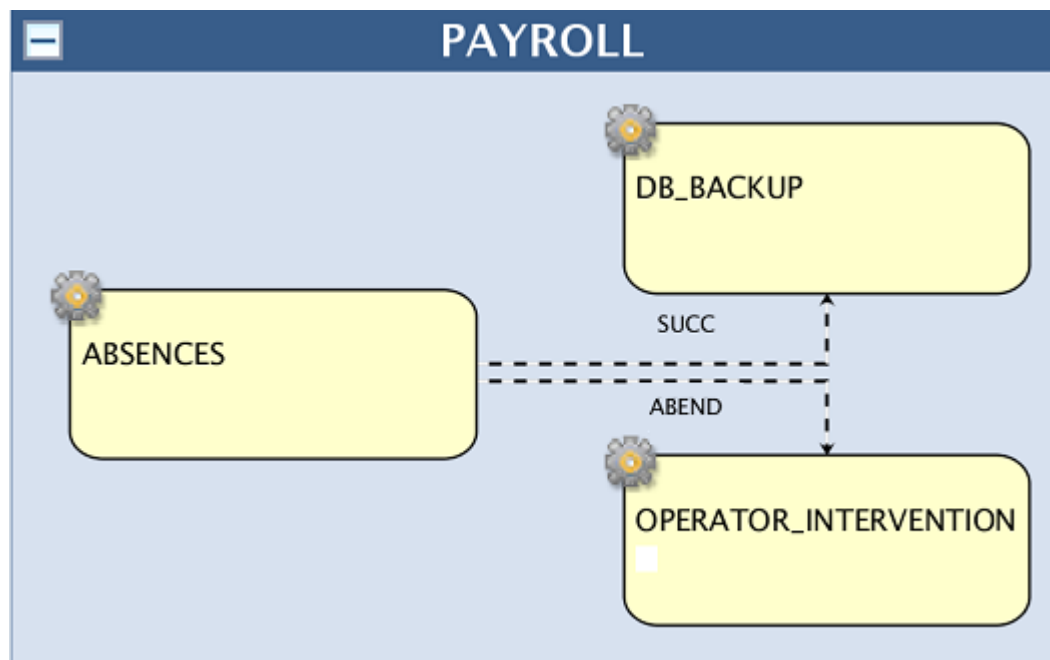


In this example, the DB_BACKUP job can start only after the ABSENCES job completes successfully. If the ABSENCES job does not complete successfully, the DB_BACKUP job remains in HOLD status.

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 29. Two different conditional dependencies on SUCC and ABEND statuses on the ABSENCES job



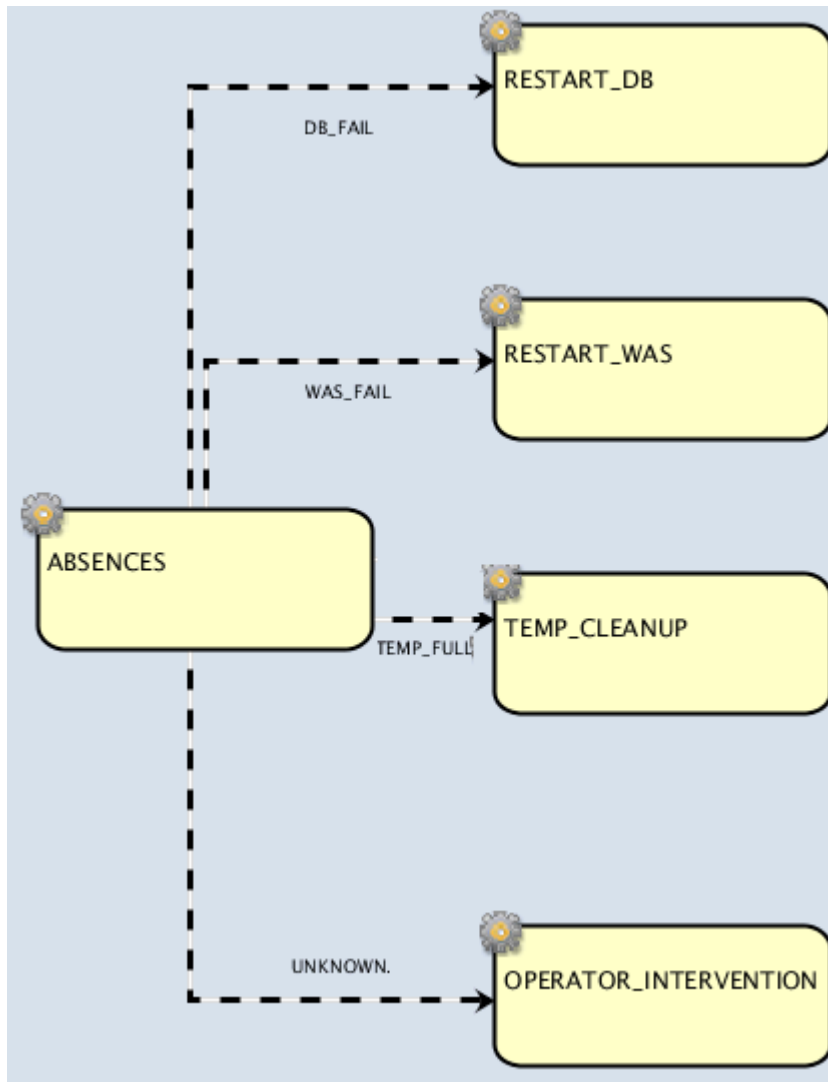
In this example, if the ABSENCES job completes successfully, the DB_BACKUP job runs and the OPERATOR_INTERVENTION job is suppressed. On the contrary, if the ABSENCES job ends in ABEND status, the OPERATOR_INTERVENTION job runs and the DB_BACKUP job is suppressed. If you have a standard follows dependency 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 30. 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.

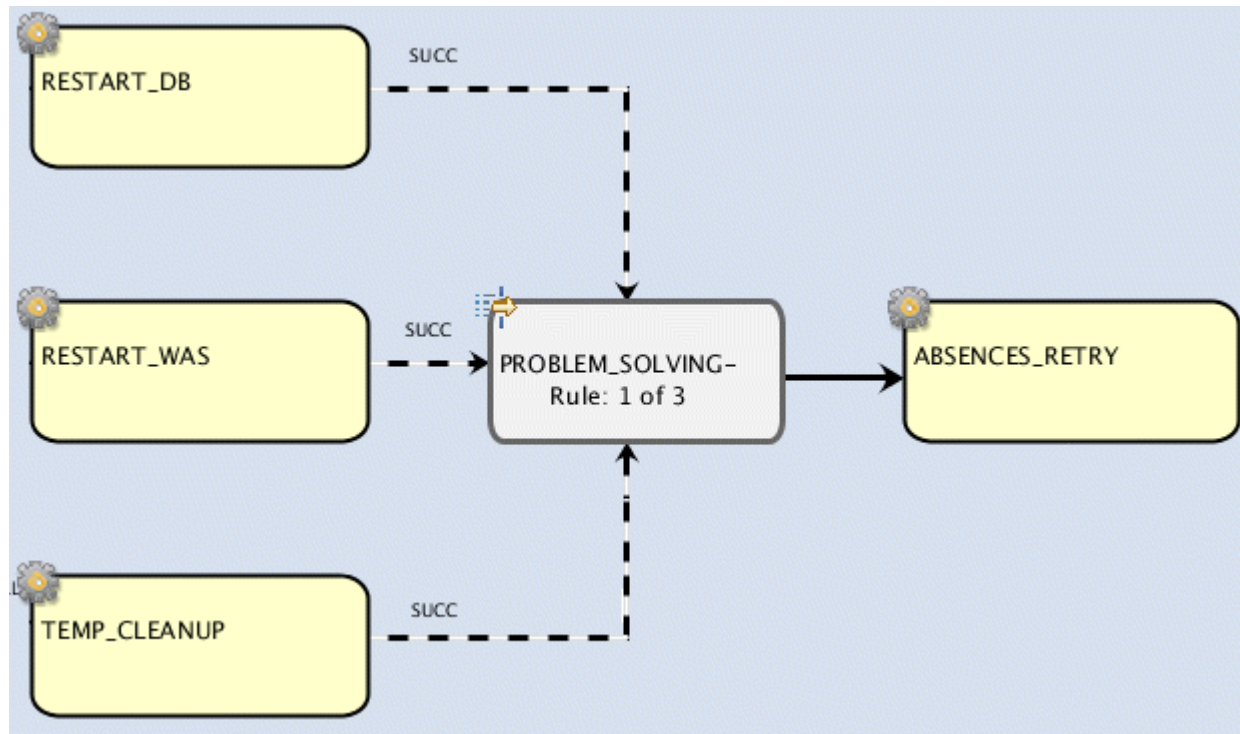
When no output conditions are satisfied, the job or job stream remains in HOLD status.

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 31. 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.

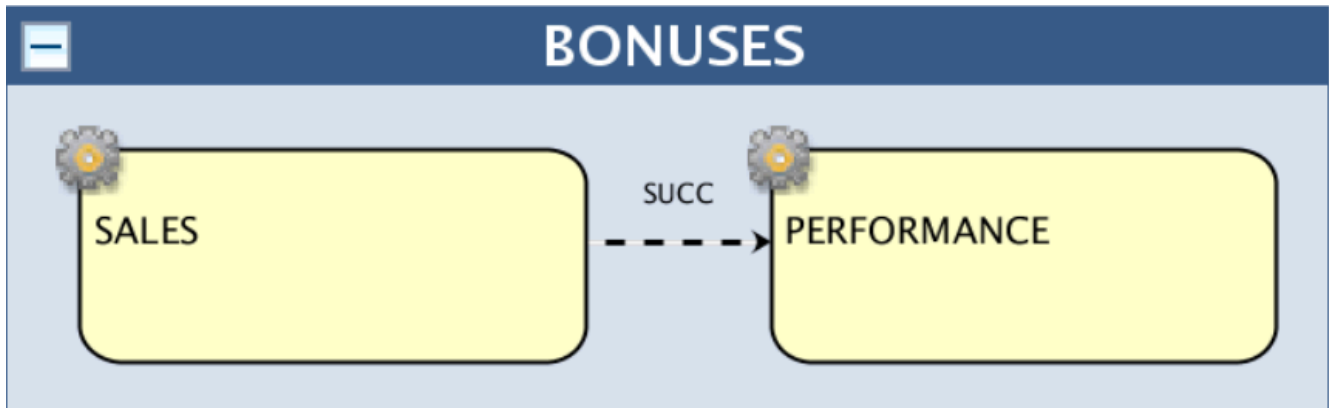
When a job within the job stream is in SUPPR status, its status is evaluated as CANCELLED. If all jobs within the job stream are in SUPPR status, the job stream goes into SUCC status. This is the same behavior that causes a job stream containing only CANCELLED jobs to go into SUCC status.

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 32. Status conditional dependency on a job with **recovery stop** setting

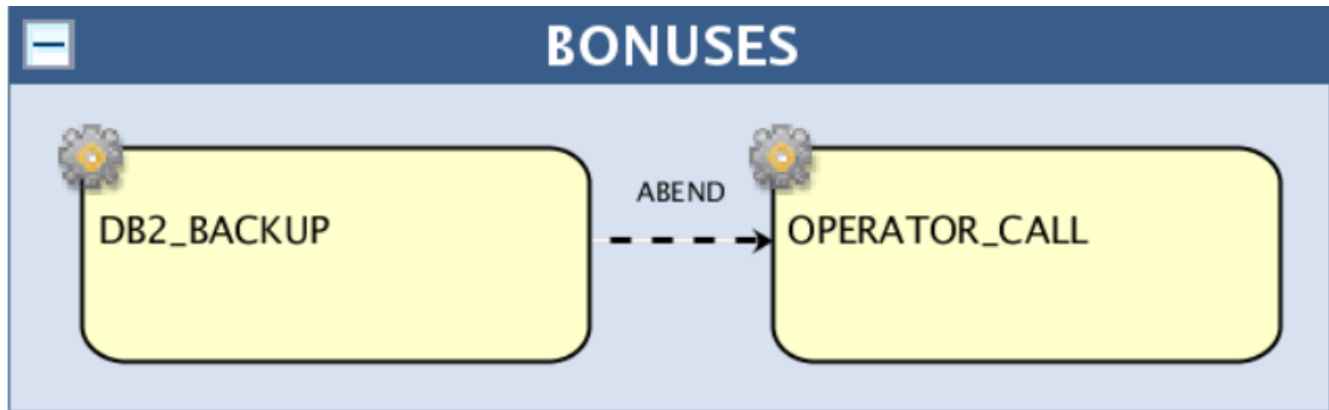


For more information about the **recovery stop** setting, see the section about defining job recovery actions in *User's Guide and Reference*.

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.

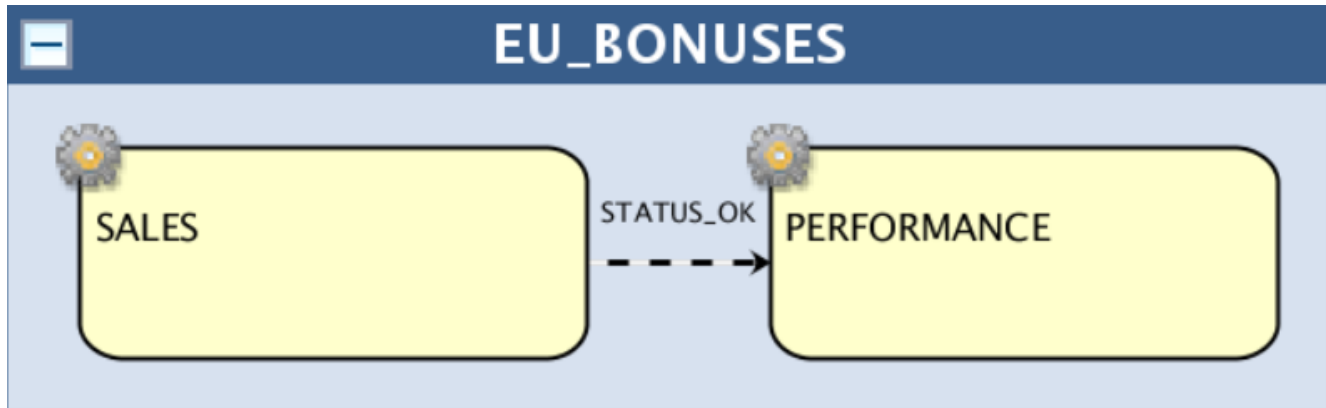
Figure 33. ABEND status conditional dependency



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.

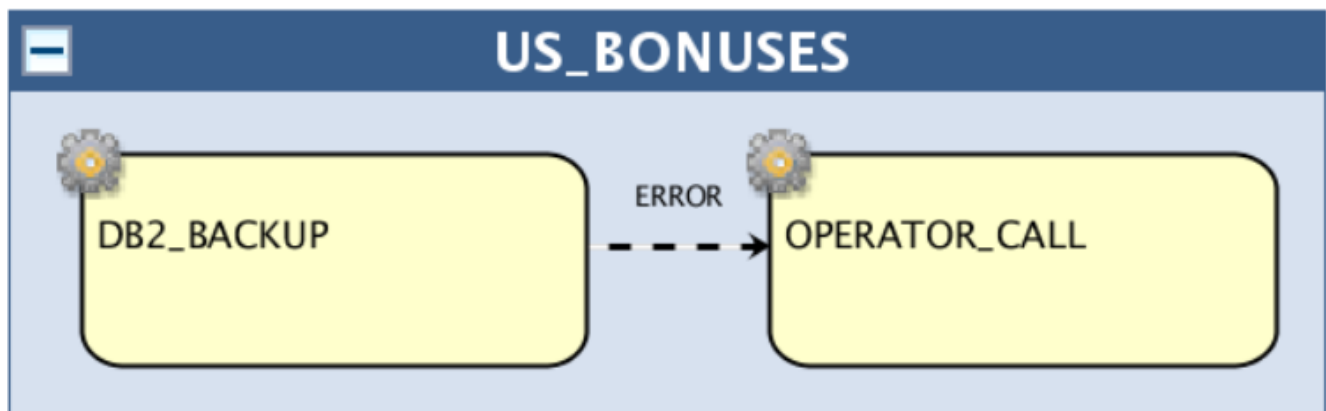
Figure 34. STATUS_OK output condition



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 35. ERROR output condition

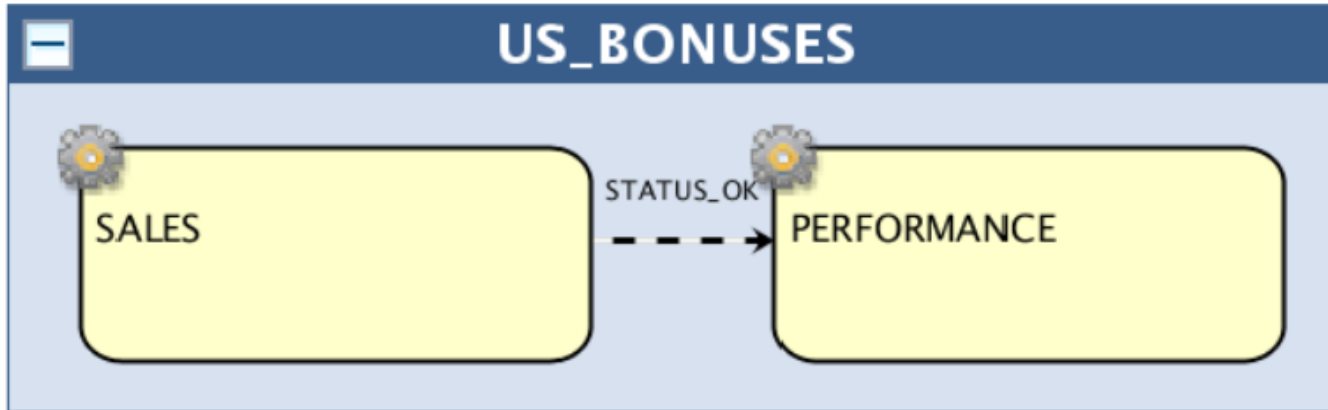


NON-SATISFIED OUTPUT CONDITION AND RECOVERY STOP SETTING CAUSING THE JOB STREAM TO COMPLETE IN ABEND STATUS

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 36. 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:

'PAYROLL' Properties

▼ Monitor Workload (Owner: wasadmin; En... ▶ 'PAYROLL' Properties

Refresh

Close View

Expand All

Collapse All

"PAYROLL" Properties

"PAYROLL" Properties	Value
▼ Properties	
▼ General	
▼ Information	
Name	PAYROLL
Workstation Name	NC050024
Original Name	
Internal Identifier	0AAAAAAAAAAAAAPL7
Scheduled Time	11/19/15 10:33 AM EST
Information	
Limit	
Contains Monitored Job	No
Priority	10
Carry Forward	No
Monitored	No
▼ Status	
Status	 Ready
Internal Status	READY

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
```

Workstation	Job Stream	SchedTime	State	Pr	(Est) Start	(Est) Elapse	Jobs #	Sch 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
```

Workstation	Job Stream	SchedTime	Job	State	Pr	(Est) Start	(Est) 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
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:

Monitor Workload (Owner: wasadmin; En... ▸ 'DB_BACKUP' Dependencies)

Refresh All

Close View

Expand All

Collapse All

Hide Satisfied Dependencies

Displaying satisfied and not satisfied dependencies for job:DB_BACKUP (NC050024#PAYROLL - 11/25/15 5:03 AM EST)

Predecessor Jobs and Job Streams (2, 0, 0)

Add...

Delete

Release

Dependency Status	Internal Status	Job	Job Stream	Workstation (Job Stream)	Scheduled Time	Conditional Dependency	Join	Unsatisfied Join Conditions
Satisfied	SUCC	ABSENCES_RETRY	PAYROLL	NC050024	11/25/15 5:03 AM EST	SUCC	SUCCESS(Rule:1 of 2)	0
Join Satisfied	ABEND	ABSENCES	PAYROLL	NC050024	11/25/15 5:03 AM EST	SUCC	SUCCESS(Rule:1 of 2)	0

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
```

Workstation	Job Stream	SchedTime	Job	State	Pr	(Est) Start	(Est) 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 **REARTART_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
```

Workstation	Job Stream	SchedTime	Job	State	Pr	(Est) Start	(Est) 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 513](#)
- Add a dependency to a job stream: [ADDDEP SCHED on page 516](#)

Delete a dependency

- Delete a dependency from a job: [DELDEP JOB on page 534](#)
- Delete a dependency from a job stream: [DELDEP SCHED on page 536](#)

Release a dependency

- Release a job from its dependencies: [RELEASE JOB on page 557](#)
- Release a job stream from its dependencies: [RELEASE SCHED on page 559](#)

Confirm the completion

Confirm the completion of job in various status: [CONFIRM on page 528](#)

Chapter 26. 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 1058](#)
- [Processing flow across the distributed scheduling environment on page 1060](#)
- [Defining a cross dependency on page 1063](#)
- [How the shadow job status changes until a bind is established on page 1065](#)
- [How a z/OS shadow job is bound on page 1068](#)
- [How the shadow job status changes after the bind is established on page 1071](#)



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 40](#) 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) or 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 new type of 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 HCL Workload Automation 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*.

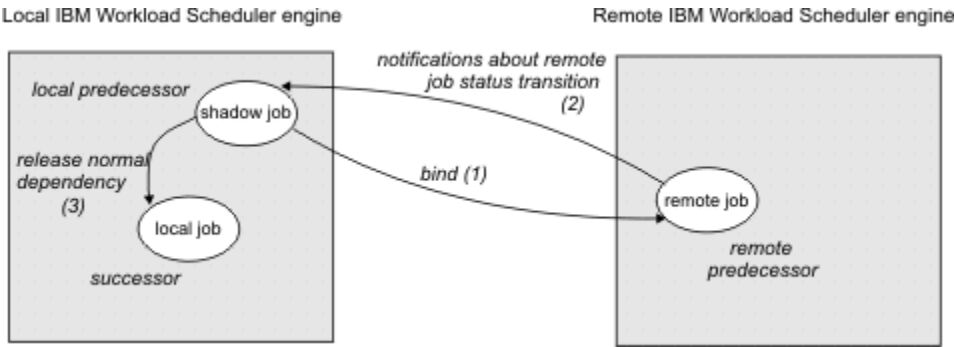
When you create the shadow, consider that

- 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 37: Cross dependency logic on page 1060 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 37. 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 138: Shadow job status transition on page 1060 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 138. 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 138. 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 138. 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 1065](#) and [How the shadow job status changes after the bind is established on page 1071](#).

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 the 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.

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 29](#).



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 1011](#).
- 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 181](#).

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 HCL Workload Automation 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 205](#).

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 139. 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= <i>time_before_scheduled_time</i> to= <i>time_after_scheduled_time</i>
Within an absolute interval	absolute from= <i>interval_start</i> to= <i>interval_end</i>
Same scheduling date	sameDay

For more information about these matching criteria, see [Managing external follows dependencies for jobs and job streams on page 87](#).

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.

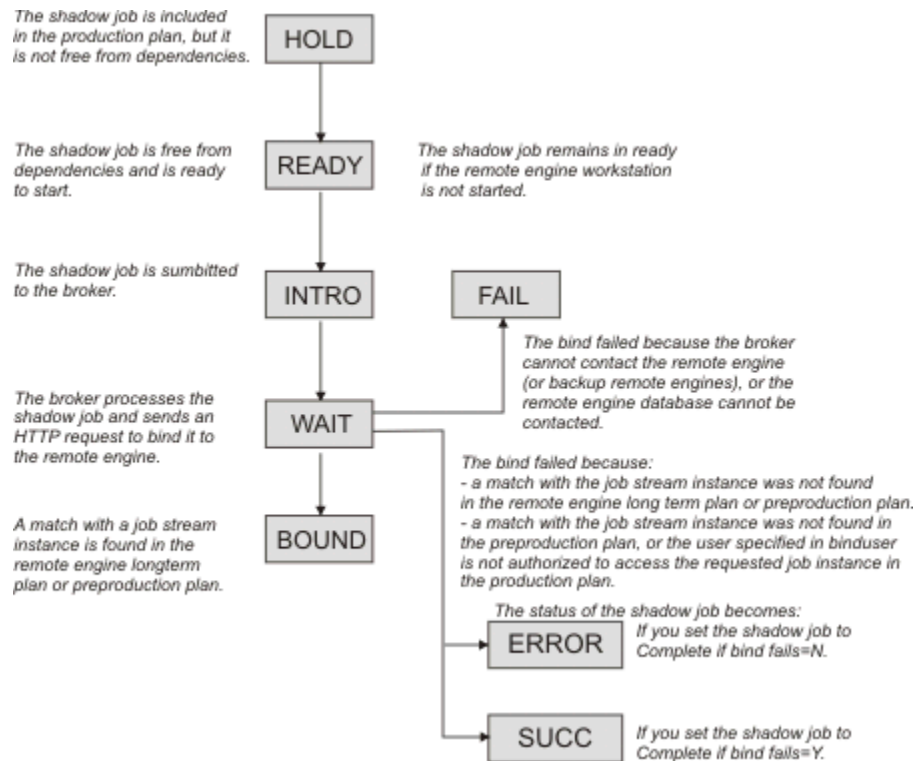
- When the preproduction plan is extended or created and its time frame includes the shadow job scheduled time.

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 38: [Shadow job status transition until the bind is established on page 1066](#) summarizes how a shadow job status changes until the bind is established.

Figure 38. 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 1073](#).

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 1067](#).

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 1068](#).

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:

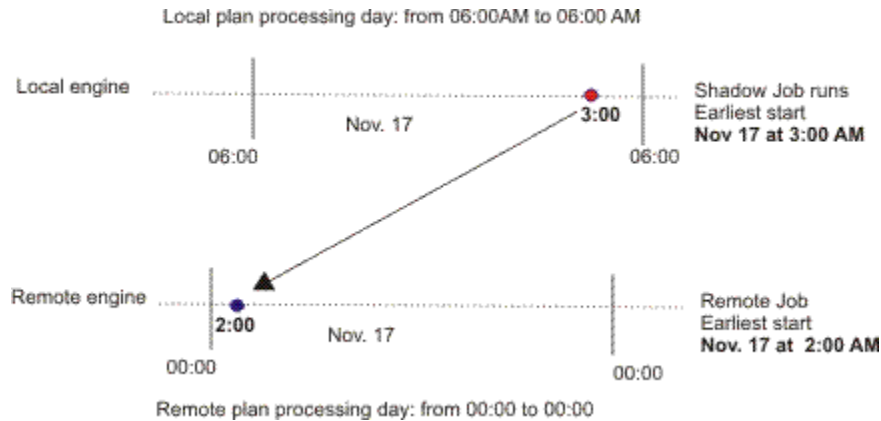
The shadow job scheduled time and the values `[hh]hhmm` and `[HH]HHMM` specified in the clause:

```
<dshadow:matching>
  <dshadow:relative from="+/-[hh]hhmm" to="+/-[HH]HHMM"/>
</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 87](#).

For more information about defining shadow jobs, see [Job definition on page 205](#).

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 39. Instance to be bound if the shadow job scheduled time is included in the CP interval

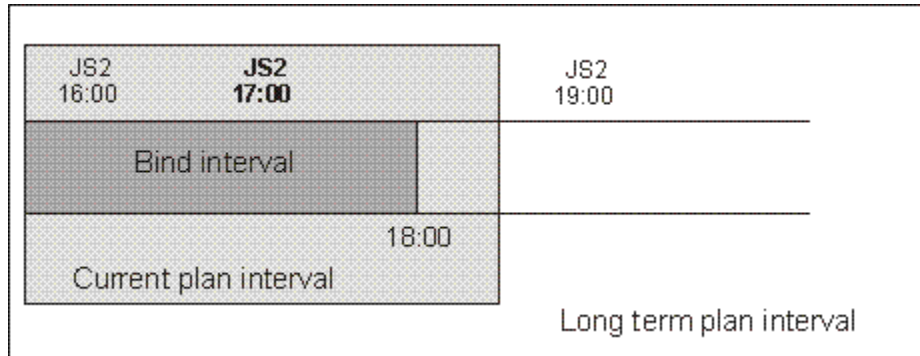


Figure 39: Instance to be bound if the shadow job scheduled time is included in the CP interval on page 1069 shows, highlighted in bold, the **JS2** instance that more closely precedes the shadow job scheduled time. This instance is selected for the bind because the scheduled time is contained in the 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 40. 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

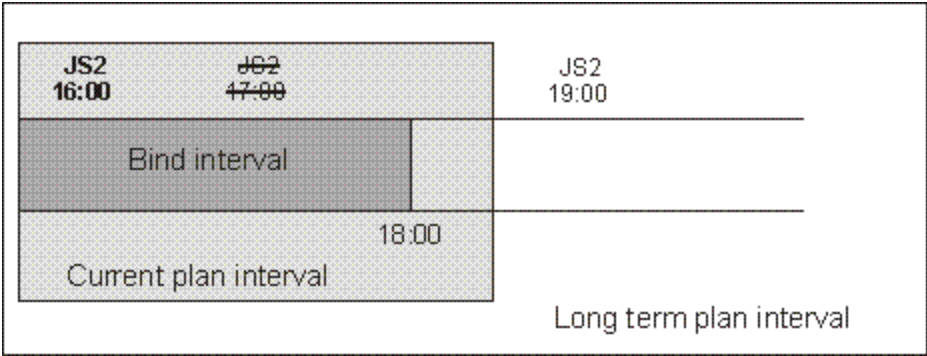


Figure 40: Instance to be bound if the instance that most closely precedes the shadow job scheduled time exists in the LTP but was canceled from the CP on page 1070 shows, highlighted in bold, the **JS2** instance that is selected for the bind, because the occurrence that better matched was deleted.

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 41. The scheduled time of the shadow job is included in the CP but no instance to bind exists

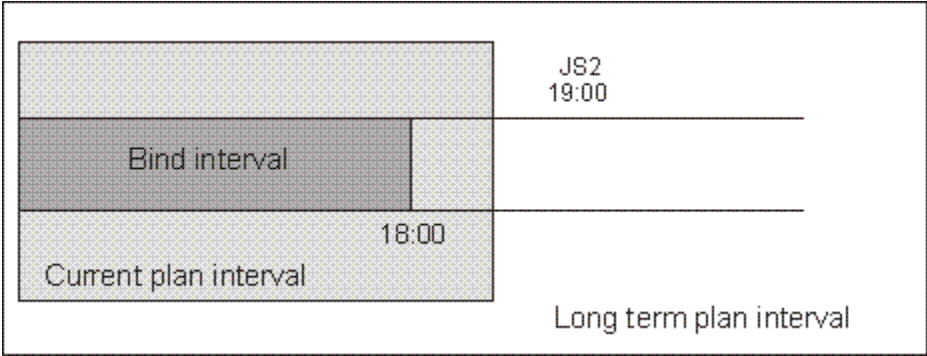


Figure 41: The scheduled time of the shadow job is included in the CP but no instance to bind exists on page 1070 shows that a **JS2** instance that closely precedes the shadow job scheduled time does not exist.

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 42. The instance to be bound exists but it is not yet included in the CP

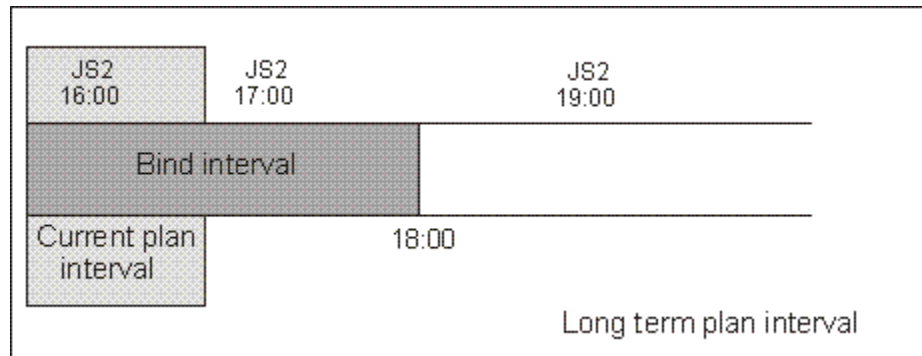


Figure 42: The instance to be bound exists but it is not yet included in the CP on page 1071 shows the **JS2** instance that can be associated with the shadow job, even though the job **JOB2** is not yet in the CP.

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 43. The LTP interval still does not contain the shadow job scheduled time

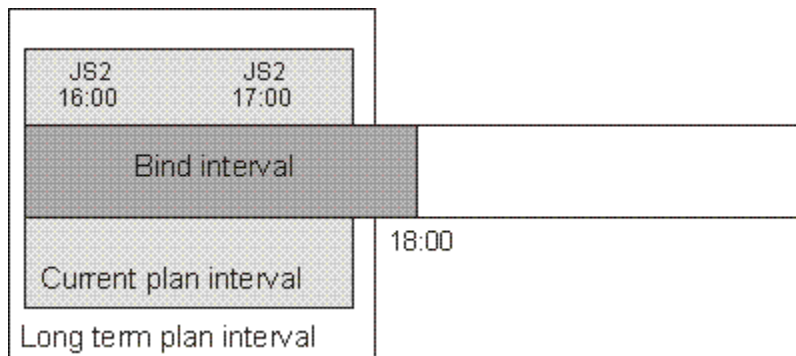


Figure 43: The LTP interval still does not contain the shadow job scheduled time on page 1071 shows that no **JS2** instance can be associated with the shadow job because, until the LTP includes the shadow job scheduled time, closer preceding **JS2** instances can still be added.

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

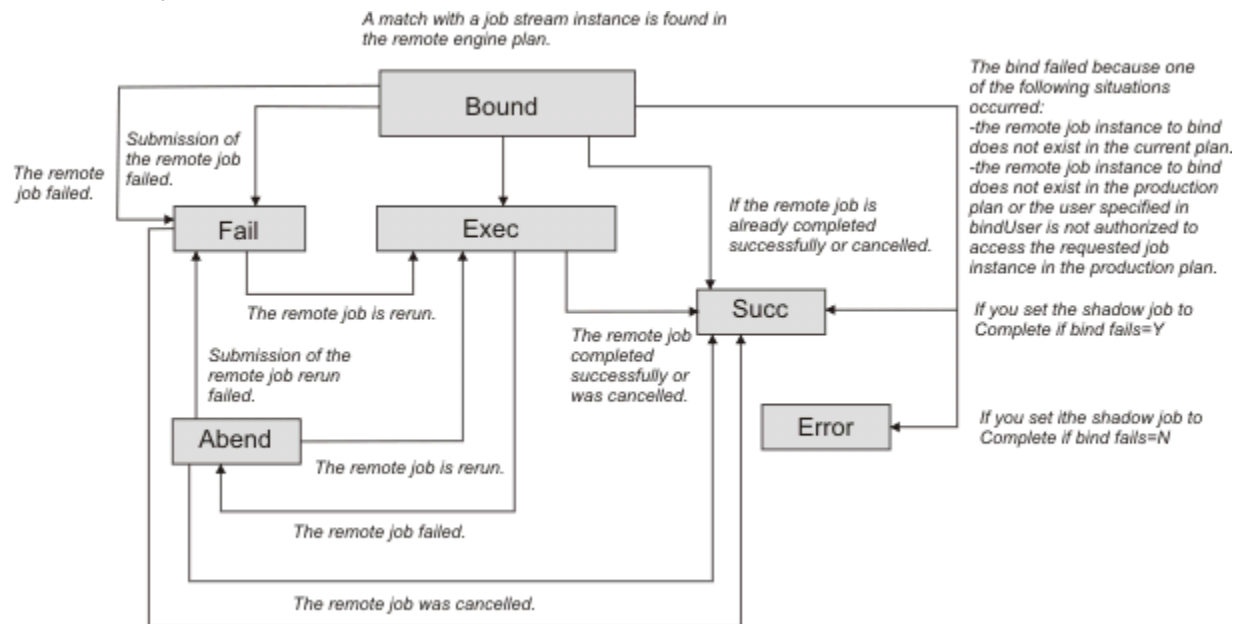
- Job stream name
- Scheduled time
- Job stream workstation
- Job name

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 44: Shadow job status transition chain after the bind was established on page 1072](#) 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 44. 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 1073](#).

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 *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 27. 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 181](#).

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 816](#).

IBM i jobs

To define IBM i jobs that run IBM i operating systems native commands, see [IBM i jobs on page 811](#).

remote command jobs

To define remote command jobs, see [Remote command jobs on page 808](#).

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 205](#).

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 1076](#).

To manage the IBM i agent, use the utilities described in [Using utility commands for agents on IBM i systems on page 1076](#).

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 921](#).

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 919](#).

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 938](#).

twstrace

To use twstrace utility, see [TWSTRACE on page 955](#).

resource

To use resource utility, see [resource on page 941](#).

cpuinfo

To use cpuinfo utility, see [cpuinfo on page 871](#).

version

To use version utility, see [version on page 921](#).

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 start 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 1080](#), 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 1084](#).

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 1084](#).

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 1084](#) 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 1087](#)
- [FileMonitor events on page 1090](#)
- [TWSApplicationMonitor events on page 1099](#)
- [DatasetMonitor events on page 1100](#)

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](#).

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>
      </filteringPredicate>
    </eventCondition>
  </eventRule>
</eventRuleSet>
```

```

    </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</eventType>
      </eventProvider>
    </eventCondition>
    <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>
  </eventRule>
</eventRuleSet>

```

```

</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.

Click [File Monitor](#) to see the Dynamic Workload Console fields for each event type.



Note: PDF users, the above parameter tables are an html file referenced by the PDF. It is not saved locally with the PDF from the infocenter. You must first view it on the infocenter before saving or printing.

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 140. 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 <code>(\< \>)</code>).
\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: <code>\0</code> , <code>\a</code> , <code>\f</code> , <code>\n</code> , <code>\r</code> , <code>\t</code> , <code>\v</code> . To specify a character in hexadecimal, use the <code>\xNN</code> syntax. For example, <code>\x41</code> 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 <code>\[</code> .
[]	Any one of the specified characters in a set. An explicit set of characters may be specified as in <code>[aeiou]</code> as well as character ranges, such as <code>[0-9A-Fa-f]</code> , which match any hexadecimal digit. The dash (-) loses its special meaning when escaped, such as in <code>[A\Z]</code> or when it is the first or last character in a set, such as in <code>[-xyz0-9]</code> . All of the above backslash-escaping rules may be used within <code>[]</code> . For example, the expression <code>[\x41-\x45]</code> is equivalent to <code>[A-D]</code> in ASCII. To use a closing bracket in a set, either escape it using <code>[\]]</code> or use it as the first character in the set, such as <code>[]xyz]</code> . POSIX-style character classes are also allowed inside a character set. The syntax for character classes is <code>[:class:]</code> . The supported character classes are:

Table 140. Regular expression syntax. (continued)

Token	Matches
	• [: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.
+	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 ababab
	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).

i Tip: When defining regular expressions to match multi-byte characters, enclose each multi-byte character in parentheses ().

Table 141: Regular expression examples. on page 1093 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 141. 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
.+	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
\bee	tee	No match

Table 141. Regular expression examples. (continued)

This expression...	Applied to this string...	Results in...
.*thing.*	Paid fee	No match
	feel	No match
	eel	Sub-string match on ee
	The thing is in here	String match
	there is a thing	String match
	it isn't here	No match
a*	thinXXX	No match
	empty string	String match
	aaaaaaaa	String match
	a	String match
	aardvark	Sub-string match on aa
	this string	Sub-string match
((ab)*c)*	empty string	String match
	cccccccc	String match
	ccccabcccabc	String match
a+	empty string	No match
	aaaaaaaa	String match
	a	String match
	aardvark	Sub-string match on aa
(ab)+c)*	this string	No match
	empty string	String match
(ab){2}	ababababcabc	String match
	abab	String match
[0-9]{4,}	cdabababab	Sub-string match on abab
	123	No match
a{0}	a1234	Sub-string match on 1234
	empty string	String match
	a	No match

Table 141. Regular expression examples. (continued)

This expression...	Applied to this string...	Results in...
	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
[0-9A-F]+	BAADF00D	String match
	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

Table 141. Regular expression examples. (continued)

This expression...	Applied to this string...	Results in...
\D\d	x^=^ where ^ represents a space character	No match
	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
	^^^^ 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
\W	x#1	No match
	foo bar	No match
	Hello there	Sub-string match of length 1 on separating space character
	~	String match

Table 141. Regular expression examples. (continued)

This expression...	Applied to this string...	Results in...
	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
	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 World!	No match
	empty string	No match
[:graph:]]+	hello world	Sub-string match on hello
	^ where ^ represents a space character	No match

Table 141. Regular expression examples. (continued)

This expression...	Applied to this string...	Results in...
	<code>^ ^ ^ ! ?</code> where <code>^</code> represents a space character	Sub-string match on <code>! ?</code>
<code>[[:lower:]]+</code>	<code>Animal</code>	Sub-string match on <code>nimal</code>
	<code>ABC</code>	No match
	<code>0123</code>	No match
	<code>foobar</code>	String match
	<code>^ ^ ^ 0 bla H !</code> where <code>^</code> represents a space character	Sub-string match on <code>bla</code>
<code>[_[:lower:]]+</code>	<code>foo_bar</code>	String match
	<code>this_thinG!!!</code>	Sub-string match on <code>_thin</code>
<code>[[:upper:]]+</code>	<code>YES</code>	String match
	<code>#define MAX 100</code>	Sub-string match on <code>MAX</code>
	<code>f00 b4r</code>	No match
<code>[[:print:]]+</code>	<code>hello world</code>	String match
	<code>^ ^ ^ ^ ^</code> where <code>^</code> represents a space character	String match
<code>[[:punct:]]+</code>	<code>didn't</code>	Sub-string match on <code>'</code>
	<code>Animal</code>	No match
<code>[[:xdigit:]]+</code>	<code>43298742432392187ffe</code>	String match
	<code>x = bAAdF00d</code>	Sub-string match on <code>bAAdF00d</code>
	<code>4327afeffegokpoj</code>	Sub-string match on <code>4327afeffe</code>
<code>c:\temp</code>	<code>c:\temp</code>	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.

Click [Application Monitor](#) to see the Dynamic Workload Console fields for each event type.



Note: PDF users, the above parameter tables are stored in an html file referenced by the PDF. It is not saved locally with the PDF from the documentation web site. You must first view it on the web site before saving or printing.

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>
      <parameter name="Severity">
        <value>Warning</value>
      </parameter>
    </action>
  </eventRule>
</eventRuleSet>
```

DatasetMonitor events

Use this function to create event rules and trigger events

DatasetMonitor events are:

- ReadCompleted
- ModificationCompleted

Table 142. SMF events

Event type	Event trigger
ReadCompleted	A data set is closed after it was opened in read mode.
ModificationCompleted	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 143. Parameters of ReadCompleted and ModificationCompleted event types

attributeFilter name	Type	Required	Wildcard 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 1103](#)
- [MailSender on page 1104](#)
- [MessageLogger on page 1104](#)
- [TWSAction on page 1104](#)
- [TWSForZosAction on page 1105](#)

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 176](#).

```
<?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

More information is available in the online version of the current topic in the published documentation at [HCL Workload Automation welcome page](#).

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 a single action named `open Incident` that opens an incident in ServiceNow when a job that matches a defined policy ends in error. 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*.

TWSAction actions

`TWSAction` actions are:

- SubmitJobStream
- SubmitJob
- SubmitAdHocJob
- ReplyPrompt

More information is available in the online version of the current topic in the published documentation at [HCL Workload Automation welcome page](#).

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.

More information is available in the online version of the current topic in the published documentation at [HCL Workload Automation welcome page](#).

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>
  </eventRule>
</eventRuleSet>
```

```

        </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 1107](#)
- [Managing objects in the database on page 1109](#)
- [Managing objects in the plan on page 1124](#)
- [Utility commands on page 1132](#)
- [Report commands on page 1136](#)

Managing the plan

This section describes the operations you can perform against the plan using the **JnextPlan** script and the **planman** command line:

Table 144. Commands used against the plan

Command or script syntax	Action performed
JnextPlan [-from mm/dd/[yy]yy[hh[:]mm[tz <i>timezone</i> tzname]]] {-to mm/dd/[yy]yy[hh[:]mm[tz <i>timezone</i> 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 <i>timezone</i> tzname]]] {-to mm/dd/[yy]yy[hh[:]mm[tz <i>timezone</i> 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 <i>timezone</i> 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 144. Commands used against the plan (continued)

Command or script syntax	Action performed
<pre>[-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}</pre>	
<pre>planman [connection_parameters] exttrial file_name {-to mm/dd/[yy]yy[hh[:]mm[tz timezone tzname]] -for [h]hh[:]mm [-days n] -days n}</pre>	Creates a trial plan of a production plan extension.
<pre>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}</pre>	Creates a forecast plan.
<pre>planman [connection_parameters] unlock</pre>	Unlocks the production plan.
<pre>ResetPlan [connection_parameters] [-scratch]</pre>	Resets the production plan.
<pre>planman reset -scratch</pre>	Removes the preproduction plan while maintaining the Symphony file.
<pre>planman [connection_parameters] resync</pre>	Replicates plan data from the Symphony file to the database.
<pre>planman [connection_parameters] checksync</pre>	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 108](#).

Managing objects in the database

The section is divided into the following subsections:

- [General purpose commands on page 1109](#)
- [Scheduling objects on page 1109](#)
- [Composer commands on page 1117](#)

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 145. 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\"}"]
  [variable [folder/table_name]
  [freedays [folder/calendarname [-sa] [-su]]
  [on [runcycle name
    [validfrom date] [validto date]
    [description "text"]
    [variable [folder/table_name]]
    {date|day|folder/calendar|request|\"icalendar\"|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\"|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"]

variable *[folder/]tablename*

[freedays *[folder/]calendarname [-sa] [-su]*

[on *[runcycle [folder/]name*

[validfrom date] [validto date]

[description "text"]

[variable *[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]]

[onuntilaction]]

[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]] [onuntilaction]]

[every rate {everyendtime time[+n day[s]]}

[deadline time [timezone|tz tzname] [+n day[s]]

end

Variable table

File definition syntax:

```
variable [folder/]table_name
[description "description"]
[isdefault]
members
[variablename "variablevalue"]
...
[variablename "variablevalue"]
end
```

For more information, see [Customizing your workload using variable tables on page 145](#).

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]
[variable 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] [-default twscpu]
        ["command [&[command]] [...]"]

```

where *connection_parameters*, if they are not supplied in the `locaopts` 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 81](#) for more details.

These operations can only be run from any composer client command line installed.

In [Table 146: Composer commands on page 1118](#) 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 146. Composer commands

Command	Syntax	User Authorization
add	{ add a } <i>filename</i> [:unlock]	add or modify
authenticate	{ authenticate au } [username =username password =password]	
chgfolder	{ chfolder cf } <i>foldername</i>	display
continue	{ continue co }	
create extract	{ create cr extract ext } <i>filename</i> from {[calendars calendar cal = <i>[folder/]calname</i>] [eventrule erule er = <i>[folder/]eventrulename</i>] [parms parm vb = <i>[folder/]tablename.]variablename</i>] [vartable vt = <i>[folder/]tablename</i>] [prompts prom = <i>[folder/]promptname</i>] [resources resource res = <i>[folder/]workstationname#[folder/]resourcename</i>] [runcyclegroup rcg = <i>[folder/]runcyclegroupname</i>] [cpu = <i>[folder/]workstationname</i> <i>[folder/]workstationclassname</i> <i>domainname</i>]} [workstation ws = <i>[folder/]workstationname</i>] [workstationclass wscl = <i>[folder/]workstationclassname</i>] [domain dom = <i>domainname</i>] [jobs jobdefinition jd = <i>[folder/]workstationname#[folder/]jobname</i>] [sched jobstream js = <i>[folder/]workstationname#[folder/]jstreamname</i> <i>[valid from date valid to date valid in date date]</i> <i>[:full]</i>] [users user = <i>[folder/]workstationname#username</i> [:password]] [accesscontrollist acl for <i>securitydomainname</i>] [securitydomain sdom = <i>securitydomainname</i>] [securityrole srol = <i>securityrolename</i>] [:lock]	display
delete	{ delete de } {[calendars calendar cal = <i>[folder/]calname</i>] [domain dom = <i>domainname</i>] [eventrule erule er = <i>[folder/]eventrulename</i>] [folder fol = <i>foldername</i>] [parms parm vb = <i>[folder/]tablename.]variablename</i>]	delete

Table 146. Composer commands (continued)

Command	Syntax	User Authorization
	[prompts prom=[folder/]promptname] [resources resource res=[folder/]workstationname#[folder/]resourcename] [runcyclegroup rcg=[folder/]runcyclegroupname] [variable vt=[folder/]tablename] [wat=[folder/]workloadapplicationtemplatename] [cpu=[folder/]workstationname [:force] [folder/]workstationclassname [:force] domainname] [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} {[calendars calendar cal=[folder/]calname] [eventrule erule er=[folder/]eventrulename] [folder fol=foldername] [parms parm vb=variablename.]variablename] [variable vt=[folder/]tablename] [prompts prom=[folder/]promptname] [resources resource res=[folder/]workstationname#[folder/]resourcename] [runcyclegroup rcg=[folder/]runcyclegroupname] [cpu=[folder/]workstationname [folder/]workstationclassname domainname] [wat=[folder/]workloadapplicationtemplatename] [workstation ws=[folder/]workstationname] [workstationclass wscl=[folder/]workstationclassname] [domain dom=domainname] [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]	display

Table 146. Composer commands (continued)

Command	Syntax	User Authorization
	[securityrole srol=securityrolename}] [:offline]	
edit	{edit ed} filename	
exit	{exit e}	
list print	{list l} {[calendars calendar cal=[folder/]calname] [eventrule erule er=[folder/]eventrulename] folder[fol=foldername] [parms parm vb=[folder/]tablename.]variablename] [variable vt=[folder/]tablename] [prompts prom=[folder/]promptname] [resources resource res=[folder/]workstationname#]resourcename] [runcyclegroup rcg=[folder/]runcyclegroupname] [cpu=[folder/]workstationname [folder/]workstationclassname domainname]] [wat=workloadapplicationtemplatename] [workstation ws=[folder/]workstationname] [workstationclass wscl=[folder/]workstationclassname] [domain dom=domainname] [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}] [:offline] [showid]	display
listfolder	{listfolder lf} foldername	list, or list and display
lock	{lock lo} {[calendars calendar cal=[folder/]calname] [eventrule erule er=[folder/]eventrulename] [folder fol=foldername] [parms parm vb=[folder/]tablename.]variablename] [variable vt=[folder/]tablename]	modify

Table 146. 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 domainname}] [workstation ws=[folder/]workstationname] [workstationclass wscl=[folder/]workstationclassname] [domain dom=domainname] [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	<i>add</i> , if new, <i>modify</i> if existing
modify	{modify m} {[calendars calendar cal=[folder/]calname] [eventrule erule er=[folder/]eventrulename] folder fol [parms parm vb=[folder/]tablename.]variablename] [variable vt=[folder/]tablename] [prompts prom=[folder/]promptname] [resources resource res=[folder/]workstationname#[folder/]resourcename] [runcyclegroup rcg=[folder/]runcyclegroupname] [cpu={folder/]workstationname folder/]workstationclassname domainname}] [wat=workloadapplicationtemplatename] [workstation ws=[folder/]workstationname] [workstationclass wscl=[folder/]workstationclassname] [domain dom=domainname] [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]}	<i>modify</i> or <i>add</i>

Table 146. Composer commands (continued)

Command	Syntax	User Authorization
new	new [calendar domain eventrule folder fol job jobstream parameter prompt resource runcyclegroup user variable wat workstation workstationclass accesscontrollist securitydomain securityrole]	<i>add or modify</i>
rename	{rename rn} {calendars calendar cal parms parm vb variable vt prompts prom resources resource res runcyclegroup rcg workstation ws workstationclass wscl domain dom jobs jobdefinition jd jobsched jb eventrule erule er sched jobstream js users user } <i>old_object_identifier new_object_identifier</i>	<i>add and delete</i>
renamefolder	{renamefolder m} <i>previousname newname</i>	<i>delete and add</i>

Table 146. Composer commands (continued)

Command	Syntax	User Authorization
replace	<code>{replace rep} filename [:unlock]</code>	<i>modify or add</i>
rmfolder	<code>{rmfolder rf} foldername</code>	<i>delete</i>
unlock	<code>{unlock un}</code> <code>{[calendars calendar cal=[folder/]calname] </code> <code>[eventrule erule er=[folder/]eventrulename] </code> <code>[folder fol=foldername] </code> <code>[parms parm vb=[folder/]tablename.]variablename] </code> <code>[varitable vt=[folder/]tablename] </code> <code>[prompts prom=[folder/]promptname] </code> <code>[resources resource res=[folder/]workstationname#[folder/]resourcename] </code> <code>[runcyclegroup rcg=[folder/]runcyclegroupname] </code> <code>[cpu=[folder/]workstationname [folder/]workstationclassname domainname]]</code> <code>[workstation ws=[folder/]workstationname] </code> <code>[workstationclass wscl=[folder/]workstationclassname] </code> <code>[domain dom=domainname] </code> <code>[jobs jobdefinition jd=[folder/]workstationname#[folder/]jobname] </code> <code>[schedjobstream js= [folder/]workstationname#[folder/]jstreamname</code> <code> [valid from date valid to date valid in date date]] </code> <code>[users user=[folder/]workstationname#username] </code> <code>[accesscontrollist acl for securitydomainname] </code> <code>[securitydomain sdom=securitydomainname] </code> <code>[securityrole srol=securityrolename]}</code> <code>[:forced]</code>	<i>modify and unlock</i>
update	<code>{update up}</code> <code>{[cpu={folder/]workstationname [folder/]workstationclassname]} </code> <code>[workstation ws=[folder/]workstationname] </code> <code>[workstationclass wscl=[folder/]workstationclassname];</code> <code>[filter workstation_filter_criteria= selection [...]];</code> <code>set [ignore= on off]]</code> <code>[:noask]</code>	<i>modify and display</i>
validate	<code>{validate val} filename [:syntax]</code>	
version	<code>{version v}</code>	

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 81](#).

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 147. Commands that can be run from conman

Command	Syntax	Worksta tion types	User Authorization
adddep job	{adddep job adj} = <i>jobselect</i> ;dependency[:...] [:noask]	F	<i>adddep</i> - (use when using prompts and needs)
adddep sched	{adddep sched ads} = <i>jstreamselect</i> ;dependency[:...] [:noask]	F	<i>adddep</i> - (use when using prompts and needs)
altpass	altpass [[folder/]workstation#] username ["password"]	F	<i>altpass</i>
altpri	{altpri ap} <i>jobselect</i> <i>jstreamselect</i> [:pri] [:noask]	F	<i>altpri</i>
cancel job	{cancel job cj} <i>jobselect</i> [:pend] [:noask]	F	<i>cancel</i>
cancel sched	{cancel sched cs} <i>jstreamselect</i> [:pend] [:noask]	F	<i>cancel</i>
checkhealthstatus	{checkhealthstatus chs} [[folder/]workstation]	M,F,S	
chfolder	{chfolder cf} <i>foldername</i>	F	display

Table 147. Commands that can be run from conman (continued)

Command	Syntax	Workstation types	User Authorization
confirm	{confirm conf} <i>jobselect</i> ; {succ abend} [IF ' <i>output_condition_name</i> [' <i>output_condition_name</i> [, ...]'] ; noask]	F	<i>confirm</i>
console	{console cons} [sess sys] ; level=msglevel]	F-S	<i>console</i>
continue	{continue cont}	F-S	
deldep job	{deldep job ddj} <i>jobselect</i> ; <i>dependency</i> [:...] ; noask]	F	<i>deldep</i>
deldep sched	{deldep sched dds} <i>jstreamselect</i> ; <i>dependency</i> [:...] ; noask]	F	<i>deldep</i>
deployconf	{deployconf deploy} [<i>domain!</i>][<i>folder/</i>] <i>workstation</i>	F,S	Permission to start actions on <i>cpu</i> objects
display	{display file df} <i>filename</i> ; offline] {display job dj} <i>jobselect</i> ; offline] {display sched ds} <i>jstreamselect</i> [valid {at date in date date} ; offline]	F-S ¹	<i>display</i>
exit	{exit e}	F-S	
fence	{fence f} <i>workstation</i> ; <i>pri</i> ; noask]	F	<i>fence</i>

Table 147. Commands that can be run from conman (continued)

Command	Syntax	Worksta tion types	User Authorization
help (UNIX® only)	{help h} { <i>command</i> <i>keyword</i> }	F-S	
kill	{kill k} <i>jobselect</i> [; <i>noask</i>]	F	<i>kill</i>
limit cpu	{limit cpu lc } [<i>folder/</i>] <i>workstation</i> ; <i>limit</i> [; <i>noask</i>]	F	<i>limit</i>
limit sched	{limit sched ls } <i>jstreamselect</i> ; <i>limit</i> [; <i>noask</i>]	F	<i>limit</i>
link	{link lk} [<i>domain!</i>][<i>folder/</i>] <i>workstation</i> [; <i>noask</i>]	F-S	<i>link</i>
listfolder	{listfolder lf} <i>foldername</i>	F	If <i>enListSecChk</i> global option is set to yes on the MDM, then, you must have either <i>list</i> access, or <i>list</i> and <i>display</i> access.
listsym	{listsym lis} [<i>trial</i> <i>forecast</i>] [; <i>offline</i>]	F	
recall	{recall rc} [<i>folder/</i>] <i>workstation</i> [; <i>offline</i>]	F	<i>display</i>
redo	{redo red}	F-S	
release job	{release job rj} <i>jobselect</i> [; <i>dependency</i> [;...]] [; <i>noask</i>]	F	<i>release</i>

Table 147. Commands that can be run from conman (continued)

Command	Syntax	Worksta tion types	User Authorization
release sched	{release sched rs} <i>jstreamselect</i> [;dependency[;...]] [;noask]	F	<i>release</i>
reply	{reply rep} { <i>promptname</i> [[<i>folder</i>]/ <i>workstation#</i>] <i>msgnum</i> } ; <i>reply</i> [;noask]	F	<i>reply</i>
rerun	{rerun rr} <i>jobselect</i> [[;from=[[<i>folder</i>]/ <i>wkstat#</i>] <i>job</i> [;at= <i>time</i>] [;pri= <i>prl</i>]] [[;streamlogon logon= <i>new_logon</i>] [;docommand=" <i>new_command</i> ";script=" <i>new_script</i> "]] [;step= <i>step</i>]] [;sameworkstation=] [;noask]	F	<i>rerun</i>
resource	{resource reso} [[<i>folder</i>]/ <i>workstation#</i>] [<i>folder</i>]/ <i>resource</i> ; <i>num</i> [;noask]	F	<i>resource</i>
setsym	{setsym set} { <i>trial</i> <i>forecast</i> } [<i>filenum</i>]	F	
showcpus	{showcpus sc} [[<i>domain!</i>]/ <i>folder</i>]/ <i>workstation</i>] [;info];link [;offline] [;showid]	F-S	<i>list</i> ²
showdomain	{showdomain showdom sd} [<i>domain</i>] [;info] [;offline]	F-S	<i>list</i> ²
showfiles	{showfiles sf} [[<i>folder</i>]/ <i>workstation#</i>] <i>file</i>] [;state[;...]] [;keys] [;offline]	F	

Table 147. Commands that can be run from conman (continued)

Command	Syntax	Worksta tion types	User Authorization
	{showfiles sf} <i>[[folder/]workstation#]file</i> [:state[:...]] [:deps[:keys info logon]] [:offline]		
showjobs	{showjobs sj} <i>[jobselect]</i> [:deps[:keys info logon]] [:short single] [:offline] [:showid] [:props] {showjobs sj} <i>[jobselect </i> <i>[[folder/]workstation#]jobnumber.hhmm</i> [:stdlist[:keys]] [:short single] [:offline] [:showid] [:props]	F	<i>list</i> ²
showprompts	{showprompts sp} <i>[[folder/]promptselect]</i> [:keys] [:offline] [:showid] {showprompts sp} <i>[[folder/]promptselect]</i> [:deps[:keys info logon]] [:offline] [:showid]	F	<i>list</i> ²
showresources	{showresources sr} <i>[[folder/]workstation#][folder/]resourcename]</i> [:keys] [:offline] [:showid] {showresources sr} <i>[[folder/]workstation#][folder/]resourcename]</i> [:deps[:keys info logon]]	F	<i>list</i> ²

Table 147. Commands that can be run from conman (continued)

Command	Syntax	Worksta tion types	User Authorization
	<code>[:offline]</code> <code>[:showid]</code>		
showschedules	<code>{showscheds ss} [jstreamselect]</code> <code>[:keys]</code> <code>[:offline]</code> <code>[:showid]</code> <code>{showscheds ss} [jstreamselect]</code> <code>[:deps[:keys info login]]</code> <code>[:offline]</code> <code>[:showid]</code>	F	<i>list</i> ²
shutdown	<code>{shutdown shut} [:wait]</code>	F-S	<i>shutdown</i>
start	<code>start [domain!][folder/]workstation</code> <code>[:mgr]</code> <code>[:noask]</code> <code>[:demgr]</code>	F-S	<i>start</i>
startappserver	<code>startappserver [domain!][folder/]workstation</code> <code>[:wait]</code>	F-S	Permission to <i>start</i> actions on <i>cpu</i> objects
startevtp	<code>{starteventprocessor startevtp} [domain!]workstation</code>	M ⁴	Permission to <i>start</i> actions on <i>cpu</i> objects
startmon	<code>{startmon startm} [domain!][folder/]workstation</code> <code>[:noask]</code>	F-S	Permission to <i>start</i> actions on <i>cpu</i> objects
status	<code>{status stat}</code>	F-S	<i>appserver</i>
stop	<code>stop [domain!][folder/]workstation</code> <code>[:wait]</code> <code>[:noask]</code>	F-S	<i>stop</i>
stop ;progressive	<code>stop ;progressive</code>		<i>stop</i>

Table 147. Commands that can be run from conman (continued)

Command	Syntax	Worksta tion types	User Authorization
stopappserver	{stopappserver stopapps} [domain!][folder/]workstation [;wait]	F-S	Permission to <i>stop</i> actions on <i>cpu</i> objects
stopevtp	{stopeventprocessor stopevtp} [domain!][folder/]workstation	M ⁴	Permission to <i>stop</i> actions on <i>cpu</i> objects
stopmon	{stopmon stopm} [domain!][folder/]workstation [;wait] [;noask]	F-S	Permission to <i>stop</i> actions on <i>cpu</i> objects
submit docommand	{submit docommand = sbd} [folder/]workstation#"cmd" [;alias[=name]] [;into=[folder/]workstation#] {jobstream_id;schedid [folder/]jobstreamname ([hhmm[date]])} [;joboption[...]]	F-S	<i>submit</i> - (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	<i>submit</i> - (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[...]] [;varable=tablename] [;noask]	F-S ³	<i>submit</i> - (use when using prompts and needs)
submit sched	{submit sched = sbs = } [folder/]workstation#[folder/]jstreamname [;alias[=name]] [;jstreamoption[...]]	F-S ³	<i>submit</i> - (use when using prompts and needs)

Table 147. Commands that can be run from conman (continued)

Command	Syntax	Worksta tion types	User Authorization
	<code>[:;variable=tablename]</code> <code>[:;noask]</code>		
switchevtp	<code>{switcheventprocessor switchevtp} [folder/]workstation</code>	M ⁴	Permission to <i>start</i> and <i>stop</i> actions on <i>cpu</i> objects
switchmgr	<code>{switchmgr switchm} domain;newmgr</code>	F	<i>start stop</i>
system	<code>[: !] system-command</code>	F-S	
tellop	<code>{tellop to} [text]</code>	F-S	
unlink	<code>unlink [domain!][folder/]workstation</code> <code>[:;noask]</code>	F-S	<i>unlink</i>
version	<code>{version v}</code>	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 148. Utility commands available for both UNIX® and Windows®

Command	Syntax
cpuinfo	cpuinfo -V -U cpuinfo <i>workstation</i> [<i>infotype</i>] [...]
datecalc	datecalc -V -U datecalc <i>base-date</i> [<i>offset</i>] [pic <i>format</i>][freedays [<i>folder/</i> Calendar_Name [- sa] [- su]] datecalc -t <i>time</i> [<i>base-date</i>] [<i>offset</i>] [pic <i>format</i>] datecalc <i>yyyymmddhhtt</i> [<i>offset</i>] [pic <i>format</i>]
delete	delete -V -U delete <i>filename</i>
evtdef	evtdef -U -V evtdef [<i>connection parameters</i>] dumpdef <i>file-path</i> evtdef [<i>connection parameters</i>] loaddef <i>file-path</i>
evtsize	evtsize -V -U evtsize <i>filename</i> <i>size</i> evtsize -compact <i>filename</i> [<i>size</i>] evtsize -info <i>filename</i> evtsize -show <i>filename</i> evtsize -info -show pobox
filemonitor	filemonitor -V -U filemonitor -path <i>path_to_monitor</i> -event <i>event_to_monitor</i> { fileCreated fileModified [- modificationCompletedTime <i>seconds</i>]} [- repositoryName <i>repository_name</i>]

Table 148. 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}] [-tw slog] [-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] [-tw slog] [-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 148. Utility commands available for both UNIX® and Windows® (continued)

Command	Syntax
	parms {-replace -extract} filename parms [-d][folder/]parameternameparms -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 -? wa_pull_info [-component DWC TWS] [-date yyyy-mm-dd] [-isroot=true false] [-output output_path]

Table 148. Utility commands available for both UNIX® and Windows® (continued)

Command	Syntax
	<code>-user <i>userid</i></code> <code>[-workdir <i>working_directory</i>]</code>

Utility commands available for UNIX® operating system only

Table 149. Utility commands available for UNIX® only

Command	Syntax
at	<code>at -V -U</code> <code>at -sjstream -qqueuetime-spec</code>
batch	<code>batch -V -U</code> <code>batch [-s <i>jstream</i>]</code>
showexec	<code>showexec [-V -U -info]</code>
version	<code>version -V -U -h</code> <code>version [-a] [-f <i>vfile</i>] [<i>file</i> [...]]</code>

Utility commands available for Windows® operating system only

Table 150. Utility commands available for Windows® only

Command	Syntax
listproc	<code>listproc</code>
(UNSUPPORTED)	
killproc	<code>killproc <i>pid</i></code>
(UNSUPPORTED)	
shutdown	<code>shutdown [-V -U] [-appsrv]</code>

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 151. Report commands

Name	Output produced	Syntax
rep1	Reports 01 - Job Details Listing	rep [<i>x</i>] [-V -U]
rep2	Report 02 - Prompt Listing	rep [<i>x</i>] [-V -U]
rep3	Report 03 - Calendar Listing	rep [<i>x</i>] [-V -U]
rep4a	Report 04A - Parameter Listing	rep [<i>x</i>] [-V -U]
rep4b	Report 04B - Resource Listing	rep [<i>x</i>] [-V -U]
rep7	Report 07 - Job History Listing	rep7 -V -U rep7 [-c <i>wkstat</i>] [-s <i>jstream_name</i>] [-j <i>job</i>] [-f <i>date</i> -t <i>date</i>] [-l]
rep8	Report 08 - Job Histogram	rep8 -V -U rep8 [-f <i>date</i> -b <i>time</i> -t <i>date</i> -e <i>time</i>] [-i <i>file</i>] [-p] rep8 [-b <i>time</i> -e <i>time</i>] [-i <i>file</i>] [-p]
rep11	Report 11 - Planned Production Schedule	rep11 -V -U rep11 [-m <i>mm</i> [<i>yy</i>] [...]] [-c <i>wkstat</i> [...]] [-s <i>jstream_name</i>] [-o <i>output</i>]
reptr	Report 09A - Planned Production Summary	reptr [-V -U] reptr -pre [-{summary detail}] [<i>symfile</i>]
	Report 09B - Planned Production Detail	reptr -post [-{summary detail}] [<i>logfile</i>]
	Report 10A - Actual Production Summary	

Table 151. Report commands (continued)

Name	Output produced	Syntax
	Report 10B - Actual Production Detail	
xref	Report 12 - Cross Reference Report	xref [-V -U] xref [-cpu <i>wkstat</i>] [-s <i>jstream_name</i>] [-depends -files -jobs -prompts -resource -schedules -when [...]]

Report extract programs**Table 152. Report extract programs**

Extract Program	Used to generate	Syntax
jbextract	Report 01 Report 07	jbextract [-V -U] [-j <i>job</i>] [-c <i>wkstat</i>] [-o <i>output</i>]
prxtract	Report 02	prxtract [-V -U] [-o <i>output</i>] prxtract [-V -U] [-m <i>mm[yyyy]</i>] [-c <i>wkstat</i>] [-o <i>output</i>]
caxtract	Report 03	caxtract [-V -U] [-o <i>output</i>]
paxtract	Report 04A	paxtract [-V -U] [-o <i>output</i>]
rextract	Report 04B	rextract [-V -U] [-o <i>output</i>]
r11xtr	Report 11	r11xtr [-V -U] [-m <i>mm[yyyy]</i>] [-c <i>wkstat</i>] [-o <i>output</i>] [-s <i>jstream_name</i>]
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 367](#) 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, 2025

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) (2025).

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 75
- \$MANAGER keyword 191
- \$MASTER keyword 191

A

- abend
 - job state 498
 - job stream state 507
- abend prompt 28
- abenp
 - job state 498
- access
 - extended and network agents 191
 - workstation definition 191
- access control list definition
 - security access control list 348
- access method 1018
 - agent
 - syntax 1018
 - dynamic agent
 - option file 1021
 - extended agent
 - option file 1021
 - HCL Workload Automation Agent
 - option file 1021
 - interface 1018
 - task options 1018
- access method for dynamic agent
 - overview 1017
- access method for extended agent
 - overview 1017
- access method jobs 805
- accessibility xv, 1139
- actions on security objects
 - specifying actions on security objects 356
- ad-hoc prompt 28
- add
 - job state 498, 498
 - job stream state 507
- add command 387
- adddep job command 513
- adddep sched command 516
- advanced rerun options 44
- agent
 - 29, 32
 - access method
 - syntax 1018
 - defining Windows user 225
 - starting 58
 - stopping 58
 - workstation definition 193
- aggregate
 - dependencies 1044
- altjob command 519
- altpass command 520
- altpri command 521
- anomaly detection
 - metrics 792
- API key authentication 370, 471, 481
- application job plug-ins 209
 - scheduling 32, 805, 858
- application server
 - stopping 630
- Appserverbox.msg 61
- appservman

- stopping 623
- architecture 54
- archiving
 - job instances 936
- at command 868
 - ATSCRIPT variable 869
- at keyword 255, 263
- auditbox.msg 61
- authenticate command 389
- authenticating with API key 370, 471, 481
- autolink
 - workstation definition 195
- automating plan processing
 - final job stream 132
- automating processing
 - production plan 132
- autostart monman 633

B

- backup master domain manager 29
- batch command 868
- batch reports
 - logs 1010
 - sample scenario 1004
 - traces 1010
- batchman process 55
- behindfirewall
 - workstation definition 195
- bind
 - definition 1058
- bind process
 - for distributed shadow job 1067
- broker
 - workstation definition 193
- broker jobs promotion 863
- broker promotion variables 863

C

- calendar
 - freedays 28, 28
 - holidays 28, 28
 - run cycle 28, 28, 243, 276, 302
- calendar definition 227
- call to a Web service 859
 - sample JSDL files 805
- cancel command 522
- cancel sched command 524
- carry forward
 - remote job 1073
 - shadow job 1073
- carry forward settings
 - conditional dependencies 100
- carryforward
 - customizing 100
 - job stream keywords 100
 - stageman 100
 - variable 126
- carryforward keyword 265
- carryStates
 - variable 100, 104
- caxtract command 985
- check file task
 - extended agent
 - syntax 1026
- checkhealthstatus command 526
- checking mailbox health 526
- chfolder command 390, 526
- child job on i5/OS 1080

- child job on IBM i 1080
- child job settings on AS/400 for performances 1080
- child job settings on i5/OS for performance 1080
- child job settings on IBM i for performances 1080
- child jobs settings for performances on AS/400 1080
- class
 - workstation 33
- cleanuserjobs command 931
- CLIconfig.properties file
 - command-line configuration 927
- closest preceding
 - follows 281
 - follows previous 87
 - join 289
 - matching criteria 87
- combine
 - dependencies 1044
- command
 - Cpuinfo 1027
 - logman 128
 - stageman 125
- command line
 - composer 49
 - conman 49
 - optman 49
- command line interface
 - setting 81
- command line reporting
 - setting up 1005
- command-line authentication 370, 471, 481
- command-line configuration
 - CLIconfig.properties file 927
- commands 687
 - adddep job 513
 - adddep sched 516
 - altjob 519
 - altpass 520
 - altpri 521
 - at 868
 - batch 868
 - cancel job 522
 - cancel sched 524
 - caxtract 985
 - checkhealthstatus 526
 - cleanuserjobs 931
 - confirm 528
 - console 532
 - continue (composer) 392
 - continue (conman) 533
 - cpuinfo 871
 - da_test_connection 881
 - dataexport 874
 - dataimport 875
 - datecalc 876
 - deldep job 534
 - deldep sched 536
 - delete 882
 - deployconf 538
 - display 538
 - evtdef 883
 - evtsize 888
 - exit 541
 - exportserverdata 932
 - fence 542

- getmon 572
- help 543
- importserverdata 933
- jbextract 982
- jobinfo 896
- jobprop 935
- jobstdl 898
- kill 544
- limit cpu 545, 545
- limit sched 547
- link 548
- listsucc 552
- listsym 550
- maestro 901
- makecal 901
- metronome 904, 923
- morestdl 904
- movehistorydata 936
- param 938
- parms 906
- paxtract 985
- prxtract 984
- r11xtr 987
- recall 554
- redo 555
- release 909
- release job 557
- release sched 559
- rep1 960
- rep11 965
- rep2 960
- rep3 960
- rep4a 960
- rep4b1 960
- rep7 962
- rep8 963
- reply 561
- reptr 966
- rerun 562
- rerunsucc 566
- resource 570, 941
- rextract 986
- rmstdlist 914
- sendevent 915, 953
- setsym 571
- showcpus 572
- showdomain 581
- showexec 917
- showfiles 583
- showjobs 586
- showprompts 610
- showresources 613
- showschedules 617
- shutdown 623
- start 624
- startappserver 626
- starteventprocessor 626
- startmon 627
- StartUp 920
- StartUpLwa 921
- status 628
- stop 629
- stop ;progressive 630
- stopappserver 630
- stopeventprocessor 632
- stopmon 632
- submit docommand 633
- submit file 638
- submit job 642
- submit sched 646
- switcheventprocessor 651

- switchmgr 653
- tellop 654
- unlink 655
- version 657
- version utility command 921
- xref 968
- xxtrct 989
- comment keyword 266
- compiler
 - messages 1028
- composer
 - command line 49
 - messages 1028
- composer program 367
 - command line return codes 379
 - connection parameters 370
 - control characters 373
 - delimiters 378
 - editor 369
 - filters 374
 - list of commands 380
 - offline output 368
 - prompt 369
 - running commands 370
 - setup 368
 - variables 958
 - special characters 378
 - terminal output 368
 - wildcards 374
 - XML editor 369
- composer reference 177
- computer association
 - retrieving 941
- COMPUTERNAME variable 69
- computers associated to a resource
 - retrieving 941
- condition-based automation 150
- condition-based workload automation
 - defining 153
- conditional automation 150
- conditional dependencies
 - carry forward behavior 100
 - defining 1042
 - definition 1040
 - join 1044
 - plan replication 1056
- conditional logic 1040
- CONFIG FILE 682
- configuration scripts
 - .jobmanrc 75
 - djobmanrc.cmd 79
 - jobmanrc 72
 - jobmanrc.cmd 77
- configuring
 - local properties 68
- confirm command 528
- confirmed keyword 266
- conman
 - command line 49
 - filters 486
 - wildcards 486
- conman program 478
 - control characters 484
 - delimiters 487
 - filters 486
 - list of commands 509
 - offline output 480
 - processing 488
 - prompt 480
 - running commands 481, 485
 - selecting job 489

- arguments 489
- jobstream_ID 490
- jobstreamname 489
- schedid 490
- using at 491
- using confirm 492
- using critical 492
- using critnet 492
- using deadline 492
- using every 493
- using finished 493
- using follows 494
- using logon 495
- using needs 495
- using opens 496
- using priority 496
- using prompt 496
- using recovery 497
- using scriptname 497
- using started 497
- using state 498
- using until 499
- selecting job streams 500
 - arguments 501
 - jobstream_id 501
 - jobstreamname 501
 - schedid 501
 - using at 502
 - using carriedforward 503
 - using carryforward 503
 - using finished 503
 - using follows 504
 - using limit 505
 - using needs 505
 - using opens 505
 - using priority 506
 - using prompt 506
 - using started 506
 - using state 507
 - using until 508
- setup 479
 - set variables 479
- special characters 487
- terminal output 479
- user prompting 484
- wildcards 486
- Conman program
 - Conman command line return codes 509
- conman reference 478
- conman startappserver
 - JnextPlan 108
- connection parameters
 - setting 81
- console command 532
- Console Manager
 - messages 1028
- context 687
- continue command (composer) 392
- continue command (conman) 533
- controlling
 - job processing
 - edit job definition 46
- conventions, typeface xv
- Courier.msg 61
- cpuclass
 - workstation class definition 202
- Cpuinfo
 - command 1027
- cpuinfo command 871
- cpuname
 - workstation definition 188

- create
 - folder 427
- create command 405
- CreatePostReports
 - JnextPlan 109
- creating forecast
 - planman command line 119
- creating trial
 - planman command line 116
- critical dynamic job promotion 863
- critical
 - HCL Workload Automation
- jobs
 - 220
- critical job
 - prioritization 220
 - promotion 220
- critical job priority
 - enhancing 220
- critical jobs
 - global options 136
 - local options 138
 - security file 139
- critical keyword 267
- critical path
 - job promotion 220
- cross dependencies
 - defining 1058
 - managing 1058
- cross dependency
 - as dependency on a shadow job 1063
 - definition 1063
 - how to add to the plan 1065
 - information flow 1060, 1068
 - introduction 1058
 - logic 1058
 - monitoring resolution in plan 1065
 - production plan 1065
 - remote engine workstation 1058
 - remote job 1058
 - shadow job 1058
 - steps to define 1063
- crucial dynamic jobs 863
- custom events
 - defining 176, 883
 - sending 176, 915
 - sending from dynamic agents 953
- customizing the workload
 - using variable table 145
- CyberArk integration 838
 - job definition example 840
- CyberArk query format 839

D

- d-pool
 - workstation 194
- da_test_connection command 881
- daily run cycle 22
- data integrity
 - variable table 147
- database
 - replicating plan 47
- database data extract 805, 859
- database data validation 805, 859
- database job return codes 833
- database objects 455
 - access method jobs 817
 - add command 387
 - authenticate command 389
 - calendars 227
 - continue command 392
 - create command 405
 - delete command 393
 - display command 398
 - displaying composer banner 455
 - domains 203
 - edit command 404
 - event rules 330
 - executable jobs 816
 - exit command 405
 - extract command 405, 405
 - folders 228
 - help command 411
 - IBM i jobs 811
 - JCL jobs 807
 - job stream 252
 - Job Stream submission jobs 829
 - JobManagement 824
 - jobs 205
 - list command 412
 - lock command 422
 - modify command 428
 - new command 434
 - print command 412
 - prompts 28, 237
 - redo command 437
 - rename command 440
 - replace command 445
 - resources 239
 - run cycle group 240
 - Security access control list definition 348
 - Security domain definition 350
 - Security role definition 352
 - shadow jobs 819
 - twstrace command 955
 - unlock command 447
 - update command 451
 - users 222
 - validate command 454
 - variable table 235
 - variables 230
 - VariableTable 821
 - wappman command 471
 - workstation classes 201
 - workstations 181
- database operations 859
 - sample JSDL files 805
- database performance
 - improving 936
- database stored procedure
 - database jobs
 - 859
 - sample JSDL files 805
 - sample JSDL files 805
- dataexport command 874
- datagather tool 923
 - replaced by wa_pull_info 923
- dataimport command 875
- date
 - run cycle 243, 275, 302
- datecalc command 876
- day
 - run cycle 243, 276, 302
- DB tables maintenance
 - movehistorydata command 936
- deadline keyword 267
- default variable table
 - using 146
- defining
 - condition-based workload automation 153
 - conditional dependencies 1042
 - database objects

- access method jobs 817
- AS/400 jobs 811
- calendars 227
- domains 203
- event rules 330
- executable jobs 816
- folders 228
- i5/OS jobs 811
- IBM i jobs 811
- JCL jobs 807
- job stream 252
- Job Stream Submission job 829
- JobManagement 824
- jobs 205
- prompts 28, 237
- resources 239
- run cycle group 240
- Security access control list 348
- Security domain 350
- Security role 352
- shadow jobs 819
- variables 230
- VariableTable 821
- windows users 222
- workstation classes 201
- workstations 181
- defining security objects in the
 - database 348
- dependencies
 - follows 281
 - join 289
 - needs 299
 - opens 310
 - prompts 314
- remote command jobs 808
- security objects in the database 348
- Defining agents on AS/400 systems 1075
- Defining agents on i5/OS systems 1075
- Defining agents on IBM i systems 1075, 1075
- Defining IBMi jobs 1075
- Defining jobs on AS/400 1075
- Defining jobs on i5/OS 1075
- Defining jobs on IBM i 1075
- defining non-operational jobs in job
 - streams 300
- Defining objects
 - in the database 177
- definition
 - variable table 235
- deldep job command 534
- deldep sched command 536
- delete
 - folders 439
- delete command 393, 882
- dependencies
 - conditional 1040
 - join, combine 1044
 - orphaned 91
- dependency
 - cross 1068
 - internetwork 1030, 1035
- deployconf command 538
- Deploying rules
 - planman command line 120
- description keyword 268
- directory names, notation xvi
- display command 398, 538
- djobmanrc configuration script 79
- docommand
 - job definition 209
- domain 34

- workstation definition 191
- domain definition 203
 - ismaster 204
 - manager 204
 - parent 204
- domain manager 29
- done
 - job state 498
- draft keyword 269
- dynamic agent
 - access method
 - option file 1021
 - gateway 32
 - overview 1017
 - workstation definition 193
- dynamic agents
 - 805, 858, 858
- dynamic capabilities 805, 858
- dynamic database jobs 205, 807
- dynamic file transfer jobs 205, 807
- dynamic java jobs 205, 807
- dynamic job creation 205, 807
- dynamic job promotion 863
- dynamic jobs 181, 205, 209, 807
 - access method jobs 817
 - AS/400 jobs 811
 - executable jobs 816
 - i5/OS jobs 811
 - IBM i jobs 811
 - JCL jobs 807
 - Job Stream Submission job 829
 - JobManagement job 824
 - remote command job 808
 - VariableTable job 821
- dynamic pool 29, 32, 858
 - defining 181
 - defining Windows user 225
 - workstation 194
- dynamic pools
 - scheduling
 - job types with advanced options
 - 805, 858
- dynamic scheduling 29, 32, 32, 805, 858, 858, 858, 858
 - job definition 205, 807
 - job types with advanced options
 - 805, 858
 - task job definition 209
 - workstation definition 181
- dynamic web service jobs 205, 807
- dynamic workload broker instance
 - URI
 - 932, 934
- Dynamic Workload Console
 - accessibility xv
- dynamic workstations 29, 181

E

- edit command 404
- enabling
 - SSL communication 196
 - time zone 1011
- enCarryForward
 - variable 100, 104
- enCFInterNetworkDeps
 - variable 105
- enCFResourceQuantity
 - variable 105
- end keyword 269
- enLegacyStartOfDayEvaluation
 - variable 107, 1012
- enPreventStart
 - variable 106
- enTimeZone
 - variable 107, 1011
- environment variables
 - job promotion 863
- environment variables, notation xvi
- error
 - job state 498
- event rule 37
- event rule definition 330
 - eventRules.xsd 331
 - keywords
 - actionProvider 340
 - actionType 340
 - activeTime 334
 - correlationAttributes 339
 - daylight saving time 334
 - description 333, 342
 - eventCondition 335
 - eventProvider 335
 - eventRule 332
 - eventType 335
 - filteringPredicate 338
 - isDraft 333
 - name 333
 - onDetection 341
 - onTimeOut 341
 - operator 339
 - responseType 341
 - ruleType 333
 - scope 338, 342
 - timeInterval 334
 - timeZone 334
 - validity 334
- event rules
 - instances 175
 - sample scenarios 156, 166
 - timeout option 164
 - variable substitution 165
- every keyword 270
 - used in job definitions 271
 - used in job stream definitions 270
- evtdef command 883
- evtsize command 888
- except keyword 275
- exclusive run cycle 22
- exec
 - job state 498
 - job stream state 508, 508
- executable jobs 805
- existing job types
 - definition 20
- exit command 405, 541
- exportserverdata command 932
- extended agent
 - access method
 - option file 1021
 - response messages 1020
 - running 1024
 - troubleshooting 1027
 - check file task
 - syntax 1026
 - get status task
 - syntax 1026
 - overview 1017
 - workstation definition 192
- EXTERNAL
 - job stream 1035
 - jobs 1036
- extract command 405

- extrn
 - job state 499

F

- fail
 - job state 499
- fault-tolerant agent 29
- fdignore
 - except 279
 - on 247, 306
- fdnext
 - except 279
 - on 247, 306
- fdprev
 - except 279
 - on 247, 306
- fence
 - job state 499
- fence command 542
- file dependence
 - defining 860
- file dependencies 860, 860, 860
 - defining 860, 860, 860
 - file dependencies 860
- file transfer job return codes 833
- file transfer jobs 859
 - sample JSDL files 805
- file transfer operations 859
 - sample JSDL files 805
- files
 - Appserverbox.msg 61
 - at.allow 870
 - at.deny 870
 - auditbox.msg 61
 - Courier.msg 61
 - Intercom.msg 61
 - Mailbox.msg 62
 - mirrorbox.msg 62
 - Monbox.msg 62
 - Moncmd.msg 62
 - NetReq.msg 62
 - PlanBox.msg 62
 - Server.msg 62
- filters
 - composer 374
- final job stream
 - automating plan processing 132
- FNCJSI
 - preproduction plan 85
- Folder
 - Folder definition syntax 1111
- folder definition 228
- folder keyword 281
- folders
 - chfolder command 390, 526
 - create 427
 - delete 439
 - list command 421, 549
 - mkfolder command 427
 - remove command 439
 - renamefolder command 444
- follows
 - matching criteria 281
- follows absolute to
 - matching criteria 89
 - within an absolute interval 89
- follows keyword 281
- follows previous
 - closest preceding 87
 - matching criteria 87
- follows relative to

- matching criteria 88
 - within a relative interval 88
- follows sameday
 - matching criteria 87
 - same day 87
- forecast plan
 - creating 119
 - description 103
 - earliest start time calculation 103
 - loop detection 133
- forecast start time enablement 103
- freedays keyword 285
- freedays run cycle 22
- fta
 - workstation definition 192
- fullstatus
 - workstation definition 195

G

- generic Java job 859
 - template 805
- generic Web service call 859
 - template 805
- get status task
 - extended agent
 - syntax 1026
- global options
 - carryforward variable 126
 - carryStates 100
 - carryStates variable 104
 - enCarryForward variable 100, 104
 - enCFInterNetworkDeps variable 105
 - enCFResourceQuantity variable 105
 - enLegacyStartOfDayEvaluation variable 107, 1012
 - enPreventStart variable 106
 - enTimeZone variable 107, 1011
 - logmanMinMaxPolicy variable 106
 - logmanSmoothPolicy variable 106
 - maxLen variable 104
 - minLen variable 104
 - startOfDay variable 104, 1012
 - untilDays 105
- global options file
 - name 1022
- global parameter 145
 - definition 230
 - variable table 145
- global parameters 145, 145
- global prompt 28

H

- HCL Workload Automation
 - architecture 54
 - basic concepts 19
 - controlling job processing 40
 - critical jobs 220
 - critical path 220
 - defining activities 40
 - issuing commands on Windows 54
 - managing production 47
 - network 38
 - object 19
 - overview 19
 - processes 54
 - quick start 50
 - running event management 48
 - runtime environment 39
 - user interfaces 49, 49
- HCL Workload Automation Agent
 - access method
 - option file 1021

- HCL Workload Automation
 - jobs
 - prioritizing 220
 - help command 411, 543
 - historical data
 - alerts 792
 - metrics 792
 - hold
 - job state 499
 - job stream state 508
 - hold status 557
 - HOME variable 69, 70, 78
 - HOMEDRIVE variable 69
 - HOMEPAATH variable 69
 - host
 - extended agents 191
 - workstation definition 191

I

- IBM i jobs 805
 - AS400 jobs 805
- icalendar
 - run cycle 244, 277, 303
- identifying job stream instances
 - in the plan 86
 - at 86
 - scheddateandtime 86
 - preproduction plan 86
- ignore
 - workstation class definition 202
- important dynamic jobs 863
- importserverdata command 933
- improving database performance 936
- inclusive run cycle 22
- Integration with IBM
- IBM Z Monitoring
 - BmEvents.conf
 - FILE parameter 798
- integrity of data
 - variable table 147
- interactive
 - job definition 210
- Intercom.msg 61
- interface 1018
- intermediate plan
 - extending with 114
 - generating 113
- internetwork dependency
 - creating 1034
 - managing using conman 1035
- intro
 - job state 499
- ismaster
 - domain definition 204
- isservice keyword 288

J

- J2EE jobs 805
- Java API 50
- Java job return codes 833
- Java jobs 859
 - sample JSDL files 805
- Java operations 859
 - sample JSDL files 805
- jbxttract command 982
- JnextPlan
 - conman startappserver 108
 - CreatePostReports 109
 - MakePlan 108
 - SwitchPlan 109
 - UpdateStats 109
- job 20

- controlling process
 - edit job definition 46
- job creation 205, 807
- job definition 205
 - access method jobs 817
 - creating 805
 - docommand 209
 - executable jobs 816
 - IBM i jobs 811
 - interactive 210
 - JCL jobs 807
 - JobManagement jobs 824
 - recovery option 213
 - remote command jobs 808
 - scriptname 208
 - shadow jobs 819
 - streamlogon 209
 - task 209
 - tasktype 210
 - using variables and parameters 219
 - VariableTable jobs 821
- job environment on AS/400 1084
- job environment on i5/OS 1084
- job environment on IBM i 1084
- job executor return codes 833
- job in job stream
 - onlate 308
- job instances
 - archiving 936
- job loop detection 133
- job processing
 - configuring 79
- job promotion 136, 220
 - environment variables 863
- job promotion on dynamic pools 863
- job statement
 - in job streams 287
- job states
 - abend 498
 - abenp 498
 - add 498, 498
 - done 498
 - error 498
 - exec 498
 - extrn 499
 - fail 499
 - fence 499
 - hold 499
 - intro 499
 - pend 499
 - ready 499
 - sched 499
 - succ 499
 - succp 499
 - wait 499
- job stream 20
 - EXTERNAL 1035
- job stream definition 252
- job stream keywords
 - at 263
 - carryforward 100, 265
 - comment 266
 - confirmed 266
 - critical 267
 - deadline 267
 - description 268
 - draft 269
 - end 269
 - every 270
 - used in job definitions 271
 - used in job stream definitions 270

- except 275
- folder 281
- follows 281
- freedays 285
- isservice 288
- job statement 287
- join 289
- jsuntil 291
- keyjob 293
- keysched 294
- limit 294
- matching 295
- maxdur 296
- mindur 298
- needs 299
- nop 300
- on 301, 308
- onoverlap 309
- opens 310
- priority 313
- prompt 314
- schedtime 315
- schedule 317
- servicedescription 323
- servicename 323
- servicetags 324
- servicevariables 325
- startcond 319
- statisticstype custom 318
- timezone 325
- until 326
- validfrom 329
- varable 330
- job stream loop detection 133
- job stream states
 - abend 507
 - add 507
 - exec 508, 508
 - hold 508
 - ready 508
 - stuck 508
 - succ 508
- job stream submission
 - Job Stream Submission jobs 829
- Job Stream Submission job
 - external status 829
 - internal status 829
 - status mapping 829
- job streams
 - move 455
 - rename 455
- job streams keywords
 - at 255
 - jsuntil 291
 - onmaxdur 296
 - onmindur 298
 - onuntil 248, 250, 292, 326, 326
 - startcond 319
- job types 859
 - template 805
- job types with advanced options
 - 805, 858, 859
 - definition 20
 - dynamic scheduling 20
 - sample JSDL files 805
 - scheduling 32, 209
 - scheduling dynamically 805, 858
 - scheduling statically 805, 858
 - static scheduling 20
 - template 805
- job variables 219

- job with advanced options return codes 833
- jobinfo command 896
- jobman
 - environment variables 69
- Jobman
 - messages 1029
- jobman process 56
 - limit cpu 56
- jobmanrc configuration script 72, 77
- jobprop command 935
- jobs
 - move 455
 - rename 455
- jobstdl command 898, 904
- join
 - dependencies 1044
 - matching criteria 289
- join keyword 289
- jsuntil keyword 291, 291

K

- keyjob keyword 293
- keysched keyword 294
- keywords
 - at 255, 263
 - carryforward 265
 - comment 266
 - confirmed 266
 - critical 267
 - deadline 267
 - description 268
 - draft 269
 - end 269
 - every 270
 - used in job definitions 271
 - used in job stream definitions 270
- except 275
- folder 281
- follows 281
- freedays 285
- isservice 288
- join 289
- jsuntil 291
- keyjob 293
- keysched 294
- limit 294
- matching 295
- maxdur 296
- mindur 298
- needs 299
- nop job in job stream 300
- on 301
- onlate 308
- onoverlap 309
- opens 310
- priority 313
- prompt 314
- schedtime 315
- schedule 317
- servicedescription 323
- servicename 323
- servicetags 324
- servicevariables 325
- startcond 319
- statisticstype custom 318
- timezone 325
- until 326
- validfrom 329
- varable 330
- kill command 544

L

- LANG variable 69, 70
- Language setting 682
- late status 267
- LD_LIBRARY_PATH variable 70
- LD_RUN_PATH variable 70
- licensetype
 - licensing 188
- licensing
 - licensetype 188
- licensing model
 - actual workstation 603
 - pricing model 603
- limit cpu
 - jobman process 56
- limit cpu command 545
- limit keyword 294
- limit sched command 547
- link command 548
- list command 412
- listfolder command 421, 549
- listsucc command 552
- listsym command 550
- local option
 - mm retry link variable 131
- local options file
 - name 1022
- local parameter
 - database 906
 - definition 230
 - exporting 906
 - importing 906
 - managing 906
- local password
 - managing on dynamic agents 938
- local prompt 28
- local properties 68
- local variable
 - managing on dynamic agents 938
- LOCAL_RC_OK variable 73, 78
- lock command 422
- lock mechanism
 - variable table 147
- log level on AS/400 1077, 1079
- log level on i5/OS 1077, 1079
- log level on IBM i 1077, 1079
- log settings on AS/400 1077, 1079
- log settings on i5/OS 1077, 1079
- log settings on IBM i 1077, 1079
- logging job statistics 128
- logical resource 37
- logical resource association
 - retrieving 941
- logical resource information
 - retrieving 941
- logmanMinMaxPolicy
 - variable 106
- logmanSmoothPolicy
 - variable 106
- LOGNAME variable 69, 70
- logs observability
 - OpenTelemetry 802
- long term plan
 - preproduction plan 85
- loop detection
 - forecast plan 133
 - trial plan 133

M

- maestro command 901
- MAESTRO_OUTPUT_STYLE variable 69, 70
- MAIL_ON_ABEND variable 73, 74, 78

- on a Windows workstation 78
- mailbox files
 - Appserverbox.msg 61
 - auditbox.msg 61
 - Courier.msg 61
 - Intercom.msg 61
 - Mailbox.msg 62
 - mirrorbox.msg 62
 - Monbox.msg 62
 - Moncmd.msg 62
 - NetReq.msg 62
 - PlanBox.msg 62
 - Server.msg 62
 - setting size 888
- Mailbox.msg 62
- mailman process 55
 - ServerID 55
- makecal command 901
- MakePlan
 - JnextPlan 108
- manager
 - workstation definition 192
- managing
 - external follows dependencies 87
 - matching criteria 87
 - objects in the database 177
 - production cycle 83
 - shadow job in the plan 1074
 - workload applications 346
- Managing agents on AS/400 systems 1076
- Managing agents on i5/OS systems 1076
- Managing agents on IBM i systems 1076, 1076
- managing events
 - starting the event processing server 626
 - starting the monitoring engine 627
 - stopping the event processing server 632
 - stopping the monitoring engine 632
 - switching the event processing server 651
- managing jobs and agents on AS/400 1075
- managing jobs and agents on i5/OS 1075
- managing jobs and agents on IBM i 1075
- managing jobs and agents on IBM i dynamic environment 1075
- managing objects
 - command line 367
 - in plan 478
- managing plan
 - adding dependency to job streams 516
 - adding dependency to jobs 513
 - altering priority 521
 - altering user password 520
 - assigning console 532
 - cancelling job streams 524
 - cancelling jobs 522
 - confirming job completion 528
 - deleting dependency to job streams 536
 - deleting dependency to jobs 534
 - displaying conman banner 657
 - displaying help information 543
 - displaying jobs or job streams 538
 - displaying production plan status 628
 - displaying workstation information 572
 - exiting conman 541
 - get active monitors 572
 - ignoring command 533
 - limiting jobs running in job stream 547
 - linking workstations 548
 - listing job successors 552
 - listing processed plans 550
 - listing unresolved prompts 554

- modifying job fence 542
- modifying jobs 519
- modifying jobs running on workstation 545
- modifying resource units 570
- releasing job streams from dependency 559
- releasing jobs from dependency 557
- replying to prompts 561
- rerunning commands 555
- rerunning job successors 566
- rerunning jobs 562
- selecting processed plan 571
- sending messages to operator 654
- setting message level 532
- showing domain information 581
- showing file dependencies 583
- showing job information 586
- showing job streams information 617
- showing prompts information 610
- showing resource information 613
- shutting down workstation processes 623
- starting the application server 626
- starting workstation processes 624
- stopping behindfirewall workstation processes 630
- stopping jobs 544
- stopping the application server 630
- stopping workstation processes 629
- submitting commands as jobs 633
- submitting file as jobs 638
- submitting job streams 646
- submitting jobs 642
- switching domain management 653
- unlinking workstations 655
- updating the monitoring configuration file 538
- managing time zone 1011
 - time zone name
 - with variable length 1011
- managing workload applications 460
- mapping file, regular expressions 468
- mapping file, workload applications 462
- master domain manager 29
- matching criteria
 - closest preceding 87, 295
 - follows 281
 - follows absolute to 89
 - follows previous 87
 - follows relative to 88
 - follows sameday 87
 - join 289
 - pending predecessor 90
 - predecessor 90
 - same day 87, 295
 - successor 90
 - within a relative interval 88, 295
 - within an absolute interval 89, 295
- matching keyword 295
- maxdur keyword 296
- maxLen
 - variable 104
- mechanism of lock
 - variable table 147
- members
 - workstation class definition 203, 350, 352
 - workstation definition 198
- messages
 - compiler 1028
 - composer 1028
 - Console Manager 1028
 - Jobman 1029

- metrics
 - AI Data Advisor 792
 - AIDA 792
 - authorizing access 792
 - exposing 792
 - Grafana 792
 - list 792
 - OpenMetrics standard 792
 - Prometheus 792
 - visualizing 792
- metrics observability
 - OpenTelemetry 802
- metronome command 904, 923
- migrating 145, 145
- mindur keyword 298
- minLen
 - variable 104
- mirrorbox.msg 62
- mirroring 47
- mkfolder command 427
- model 687, 700, 701, 703, 709, 711, 717, 719, 720, 723, 724, 726, 728, 782, 783
- modify command 428
- Monbox.msg 62
- Moncmd.msg 62
- monman process 55
- move
 - job streams 455
 - jobs 455
 - resources 457
 - variable table 457
 - workstation 456
- movehistorydata command 936
- MSSQL jobs 805

N

- name
 - global options file 1022
 - local options file 1022
- named prompt 28
- needs keyword 299
- netman process 55
- netmth access method 1032
- NetReq.msg 62
- network agent
 - access method
 - options file 1032
 - access method netmth 1032
 - definition 1032
 - EXTERNAL 1035
 - EXTERNAL state
 - ERROR 1035, 1035
 - EXTRN 1036
 - internetwork dependency 1030
 - creating 1034
 - managing using conman 1035
 - overview 1030
 - reference 1030
 - sample scenario 1033
- network communication 64
 - job processing 64
 - start of day 64
- new command 434
- new executor 859
- new executors 209
 - access method jobs 817
 - AS/400 jobs 811
 - executable jobs 816
 - i5/OS jobs 811
 - IBM i jobs 811
 - JCL jobs 807

- Job Stream Submission job 829
- JobManagement job 824
- remote command job 808
- scheduling 32, 805, 858
- template 805
- VariableTable job 821
- new plug-ins 805, 859
 - access method jobs 817
 - AS/400 jobs 811
 - executable jobs 816
 - i5/OS jobs 811
 - IBM i jobs 811
 - JCL jobs 807
 - Job Stream Submission job 829
 - JobManagement job 824
 - remote command job 808
 - template 805
 - VariableTable job 821
- node loop detection 133
- nop keyword 300
- notation
 - environment variables xvi
 - path names xvi
 - typeface xvi

O

- object attribute values
 - specifying object attribute values 362
- object attributes
 - attributes for object types 361
- ocli 687
- ocli command 700, 701, 703, 709, 711, 717, 719, 720, 723, 724, 726, 728, 782, 783
- offset-based run cycle 22
- on
 - run cycle 301
- on keyword 301
 - run cycle 301
- onlate
 - job in job stream 308
- onlate keyword 308
 - run cycle 308
- onmaxdur keyword 296
- onmindur keyword 298
- onoverlap keyword 309
- onuntil keyword 248, 250, 292, 326, 326
- opens keyword 310
- OpenTelemetry
 - bootstrap.properties 802
 - configuration 802
 - enabling after upgrade 802
 - logs observability 802
 - metrics observability 802
 - traces observability 802
- option file
 - dynamic agent
 - access method 1021
 - extended agent
 - access method 1021
 - HCL Workload Automation Agent
 - access method 1021
- options
 - untilDays 105
- optman
 - command line 49
- OQL 658
- Orchestration CLI 687, 700, 701, 703, 709, 711, 717, 719, 720, 723, 724, 726, 728, 782, 783
- Orchestration CLI
- program

- Orchestration CLI
 - command line return codes 785
- orphaned
 - dependencies 91
- os type
 - workstation definition 189
- overview
 - access method for dynamic agent 1017
 - access method for extended agent 1017
 - dynamic agent 1017
 - extended agent 1017

P

- param command 938
- parameter 37
- parameter definition 230
- parameters
 - in job definitions 219
- parent
 - domain definition 204
- parms command 906
- password
 - defining on dynamic agent 835
 - job types with advanced options 835
 - resolving on dynamic agent 835
- password management 838
 - job definition example 840
- password resolver 838
 - job definition example 840
- password retrieval
 - using queries 839
- password solver 838
 - job definition example 840
- password storage 838
 - job definition example 840
- password vault 838
 - job definition example 840
- path names, notation xvi
- PATH variable 71
- paxtract command 985
- pend
 - job state 499
- pending predecessor
 - matching criteria 90
 - orphaned dependencies 91
 - successor 90
- physical resource 37
- plan 687
 - quick start 50
- plan data 47
- plan management
 - basic concepts 83
 - customizing 100, 104
 - logman 128
 - stageman 125
- plan replication
 - conditional dependencies 1056
- PlanBox.msg 62
- planman command line
 - connection parameters 111
 - creating forecast 119
 - creating trial 116
 - Deploying rules 120
 - intermediate plan 113, 114
 - monitor replication 124
 - removing plan 122
 - replicating plan data 123

- resetting plan 121
- retrieving plan info 115
- trial extension 118
- unlocking plan 121
- planman deploy 164
- plug-ins 50
- pool 29, 32, 858
 - defining 181
 - defining Windows user 225
 - workstation 194
- pools
 - scheduling
 - job types with advanced options 805, 858
- POSIXHOME variable 78
- predecessor
 - matching criteria 90
 - successor 85, 90
- preproduction plan
 - description 85
 - FNCJSI 85
 - long term plan 85
 - removing plan 122
- print command 412
- priority keyword 313
- problem prevention
 - metrics 792
- processes
 - batchman 55
 - jobman 56
 - mailman 55
 - monman 55
 - netman 55
 - ssmagent 55
 - writer 55
- production cycle 83
 - identifying job stream instances 86
 - managing 83
 - planman command line 111
- production plan
 - automating processing 132
 - description 99
 - generating 108, 109
 - JnextPlan 83, 108, 109
 - monitor replication 124
 - replicating data 123
 - resetting plan 121
 - retrieving info 115
 - starting processing 131
 - Symphony file 83, 108, 109
 - unlocking plan 121
- promoting a job 136
- prompt
 - abend 28
 - ad-hoc 28
 - global 28
 - local 28
 - named 28
 - recovery 28
- prompt definition 28, 237
- prompt keyword 314
- prompts
 - unique ID 611
- protocol
 - workstation definition 198
- Proxy user 682
- prxtract command 984

Q

- queries for password retrieval 839
- Querying with the OQL syntax 658

R

- r11xtract command 987
- ready
 - job state 499
 - job stream state 508
- recall command 554, 554
- recovery
 - job definition 213
- recovery options
 - continue 44
 - recovery jobs 44
 - rerun 44
 - stop 44
- recovery prompt 28
- redo command 437, 555
- referential integrity check 382
- release command 909
- release job command 557, 561
- release sched command 559
- rem-eng
 - workstation 194
- remote engine
 - cross dependency 1068
 - how it is bound 1067
 - workstation 182, 194
- remote engine workstation 29, 33, 38
 - defining 1058, 1063
- remote job
 - carry forward 1073
 - defining 1058
 - failed 1073
 - status transition during recovery 1073
- removing plan
 - planman command line 122
- rename
 - job streams 455
 - jobs 455
 - resources 457
 - variable table 457
 - workstation 456
- rename command 440
- renamefolder command 444
- rep1 command 960
- rep11 command 965
- rep2 command 960
- rep3 command 960
- rep4a command 960
- rep4b command 960
- rep7 command 962
- rep8 command 963
- replace command 445
- replicate plan 47
- report commands 958
 - Actual Production Detail
 - sample output 976
 - Actual Production Details 966
 - Actual Production Summary 966
 - Calendar Listing 960
 - sample output 972
 - changing date format 959
 - Cross Reference 968
 - sample output 979
 - extract programs 981
 - caxtract 985
 - jbextract 982
 - paxtract 985
 - prxtract 984
 - r11xtr 987
 - rextract 986
 - rxrtract 989

- Job Details Listing 960
 - sample output 969
- Job Histogram 963
 - sample output 975
- Job History Listing
 - sample output 974
- Parameter Listing
 - sample output 973
- Parameters Listing 960
- Planned Production Detail
 - sample output 975
- Planned Production Details 966
- Planned Production Schedule 965
 - sample output 977
- Planned Production Summary 966
- Prompt Listing 960
 - sample output 972
- Resource Listing 960
 - sample output 973
- sample outputs 969
- setup 958

- reports commands
 - Job History Listing 962
 - list of commands 959
- reptr command 966
- requirements
 - workstation definition 198
- rerun command 562
- rerun with successors 44
- rerunsucc command 566
- reserved keywords
 - for job streams 179
 - for user definitions 180
 - for workstations 180
- reserved words
 - for job streams 179
 - for user definitions 180
 - for workstations 180
- resetFTA command 569
- resetting plan
 - planman command line 121
- resolution
 - variable 148
- resource
 - logical 37
 - physical 37
 - scheduling 37
- resource command 570, 941
 - running from agent
 - CLIconfig.properties setup 951
 - requirement 951
- resource definition 239
- resources
 - unique ID 614
- return code on AS/400 1084
- return code on i5/OS 1084
- return code on IBM i 1084
- Reusing a workload in another environment 461, 461
- rextract command 986
- rial plan
 - loop detection 133
- rmfolder command 439
- rmstdlist command 914
- rule 37
- rule-based run cycle 22
- run cycle
 - calendar 243, 276, 302
 - daily 22
 - date 243, 275, 302
 - day 243, 276, 302

- exclusive 22, 28, 28
- freedays 22
- icalendar 244, 277, 303
- inclusive 22, 28, 28
- offset-based 22
- on 301
- rule-based 22
- run cycle group 246, 279, 305
- simple 22
- weekly 22
- yearly 22
- run cycle group 240
 - file definition syntax 1115
 - run cycle 246, 279, 305
- run cycle group definition 240
- Running commands 687
- running system commands
 - from composer 446
 - from conman 484, 654

S

- same day
 - follows 281
 - follows sameday 87
 - join 289
 - matching criteria 87
- sched
 - job state 499
- schedtime keyword 315
- schedule keyword 317
- scheduling
 - job types with advanced options 805, 858
 - scheduling language 252
 - scheduling objects
 - moving to folders in batch mode 455
 - scheduling resource 37
- scriptname
 - job definition 208
- secureaddr
 - workstation definition 190
- security
 - variable tables 147
- security access control list
 - security access control list definition 348
- Security access control list definition 348
- security domain
 - security domain definition 350
- security domain definition
 - security domain 350
- Security domain definition 350
- security role
 - security role definition 352
- security role definition
 - security role 352
- Security role definition 352
- securitylevel 196
 - workstation definition 196
- sendevent command 915, 953
- Server.msg 62
- ServerID
 - mailman process 55
 - workstation definition 197
- servicedescription keyword 323
- servicename keyword 323
- ServiceNow
 - event actions 1104
- servicetags keyword 324
- servicevariables keyword 325
- setsym command 571
- setting

- connection parameters 81
- SETTING UP OCLI 682
- setup
 - command line reporting 1005
- shadow job 20
 - carry forward 1073
 - defining 1058, 1063
 - definition 819
 - during remote job recovery 1073
 - failed 1073
 - managing in the current plan 1074
 - status fail 1073
 - status transition after bind 1071
- SHELL_TYPE variable 74
- showcpus command 572
- showdomain command 581
- showexec command 917
- showfiles command 583
- showjobs command 586
- showprompts command 610
- showresources command 613
- showschedules command 617
- shutdown
 - utility command 919
- shutdown command 623
- ShutDownLwa
 - utility command 919
- simple run cycle 22
- skipping objects when importing 461, 475
- slow database access 936
- specific job types 859
 - sample JSDL files 805
- SSL communication
 - enabling 196
- stageman
 - carryforward 100
 - SwitchPlan 125, 125
- standard agent
 - workstation definition 192
- start command 624
- start condition 150
- start of day
 - establishing communication 65
- startappserver command 626
- startcond keyword 319, 319
- starteventprocessor command 626
- starting
 - WebSphere Application Server Liberty 58
 - workstation processes 58
- Starting and stopping agents on AS/400 systems 1076
- Starting and stopping agents on i5/OS systems 1076
- Starting and stopping agents on IBM i 1076
- Starting and stopping agents on IBM i systems 1076
- starting processing
 - production plan 131
- startmon command 627
- startOfDay
 - variable 104, 1012
- StartUp command 920
- StartUpLwa command 921
- statisticstype custom, keyword 318
- status
 - late 267
- status command 628
- stop command 629
- stop; progressive command 630
- stopappserver command 630

- stopeventprocessor command 632
- stopmon command 632
- stopping
 - WebSphere Application Server Liberty 58
 - workstation processes 58
- streamlogon
 - job definition 209
 - windows user definition 222
- stuck
 - job stream state 508
- submit docommand command 633
- submit file command 638
- submit job
 - return code 509, 785
- submit job command 642
- submit sched command 646
- submitted job stream
 - external status 829
 - internal status 829
 - status mapping 829
- succ
 - job state 499
 - job stream state 508
- successor
 - matching criteria 90
 - pending predecessor 90
 - predecessor 85, 90
- succp
 - job state 499
- switcheventprocessor command 651
- switching extended agents
 - \$MANAGER keyword 191
 - \$MASTER keyword 191
- switchmgr command 653
- SwitchPlan
 - JnextPlan 109
- Symphony corruption
 - resetFTA command 569
- Symphony file
 - JnextPlan 108, 109
 - production plan 99, 108, 109
- syntax 658
 - agent
 - access method 1018
 - extended agent
 - check file task 1026
 - get status task 1026
- SystemDrive variable 69
- SystemRoot variable 69

T

- table of variables
 - using 145
- task
 - job definition 209
- task options
 - access method 1018
- tasktype
 - job definition 210
- tcpaddr
 - workstation definition 190
- tellop command 654
- TEMP variable 69
- templates
 - for scheduling object definitions 181
- time zone
 - enabling 1011
- timezone
 - in job streams 325
 - workstation definition 190

- timezone keyword 325
- TIVOLI_JOB_DATE variable 69, 71
- TMPDIR variable 69
- TMPTIME variable 69
- traces observability
 - OpenTelemetry 802
- trial extension
 - planman command line 118
- trial plan
 - creating 116
 - description 102
 - extension 118
- trialsked
 - forecast plan 103
 - trial plan 102
- trigger action 37
- troubleshooting tool 923
- twinst_pull_info
 - deprecated 923
 - replaced by wa_pull_info 923
 - wa_pull_info 923
- twinst_pull_info command 923
- TWS_PROMOTED_JOB 1025
- TWS_PROMOTED_JOB variable 69, 71
- TWS_TISDIR variable 71
- twstrace command 955
- type
 - workstation definition 192
- typeface conventions xv
- TZ variable 69, 71

U

- UNISON_CPU variable 69, 71
- UNISON_DATE variable 70
- UNISON_DATE_FORMAT variable 72
- UNISON_DIR variable 69, 71
- UNISON_EXEC_PATH variable 69, 71
- UNISON_EXIT variable 73
- UNISON_HOST variable 69, 71
- UNISON_JCL variable 73
- UNISON_JOB variable 70, 71
- UNISON_JOBNUM variable 70, 71
- UNISON_MASTER variable 70, 71
- UNISON_RUN variable 70, 71
- UNISON_SCH variable 70, 71
- UNISON_SCH_DATE variable 71
- UNISON_SCH_EPOCH variable 70, 71
- UNISON_SCH_IA variable 70, 71
- UNISON_SCH_ID variable 70, 71
- UNISON_SHELL variable 70, 71
- UNISON_STDLIST variable 70, 71, 73
- UNISON_SYM variable 70, 72
- UNISONHOME variable 69, 71
- UNIXTASK 210
- UNKNOWN 210
- unlink command 655
- unlock command 447
- unlocking plan
 - planman command line 121
- until keyword 326
- untilDays
 - option 105
- update command 451
- UpdateStats
 - JnextPlan 109
- upgrading 145
- USE_EXEC variable 74
- user 37
- user definition 222
 - trusted domain 226

- using on job types with advanced options 226
- user interfaces
 - composer 50
 - conman 50
 - Dynamic Workload Console 49
 - Java API 50
 - optman 50
 - planman 50
 - plug-ins 50
- user return code on AS/400 1084, 1085
- user return code on i5/OS 1084, 1085
- user return code on IBM i 1084, 1085
- USERDOMAIN variable 70
- USERNAME variable 70
- USERPROFILE variable 70
- using
 - default variable table 146
 - variable table 145
- Using utility commands on agents on AS/400 systems 1076
- Using utility commands on agents on i5/OS systems 1076
- Using utility commands on agents on IBM i systems 1076, 1076
- utility commands 865
 - agents
 - 926
 - at 868
 - at.allow file 870
 - at.deny file 870
 - ATSCRIPT variable 869
 - batch 868
 - changing date format 876
 - checking
 - dynamic agent
 - connection
 - 881
 - creating and managing variables and passwords locally on dynamic agents 938
 - creating calendars 901
 - defining custom events 883
 - deleting files 882
 - displaying content of standard list files 904
 - displaying product version 921
 - displaying running jobs 917
 - displaying standard list files 914
 - dynamic 926
 - dynamic domain manager 926
 - exporting data 874
 - getting HTML reports 904, 923
 - getting job information 896
 - getting TWS_home path 901
 - getting workstation information 871
 - importing data 875
 - list of commands 865
 - listing standard list files 898
 - managing parameters locally 906
 - releasing resource units 909
 - removing standard list files 914
 - sending custom events 915
 - setting mailbox file size 888
 - setting variables locally on dynamic agents 935
 - shutdown 919
 - ShutDownLwa 919
 - starting up netman 920, 921
- utility commands for dynamic agents
 - sending custom events 953

- validate command 454
- validfrom keyword 329
- variable 38
 - defining on dynamic agent 835
 - definition 230
 - in job definitions 219
 - job types with advanced options 835
 - resolution 148
 - resolving on dynamic agent 835
- variable management 219
- variable table 38
 - data integrity 147
 - default 146
 - definition 235
 - lock mechanism 147
 - using 145
- variable table definition
- vartable 189
- variable tables
 - security 147
 - security file migration 146
- variables
 - ATSCRIPT 869
 - carryforward 126
 - carryStates 100, 104
 - COMPUTERNAME 69
 - defining and using 219
 - Dynamic workload broker
 - 219
 - enCarryForward 100, 104
 - enCFInterNetworkDeps 105
 - enCFResourceQuantity 105
 - enLegacyStartOfDayEvaluation 107, 1012
 - enPreventStart 106
 - enTimeZone 107, 1011
 - exported locally by .jobmanrc 72, 75, 77, 79
 - exported on UNIX 70
 - HOME 69, 70, 78
 - HOMEDRIVE 69
 - HOMEPath 69
 - LANG 69, 70
 - LD_LIBRARY_PATH 70
 - LD_RUN_PATH 70
 - local variables 77, 79
 - LOCAL_RC_OK 73, 78
 - logmanMinMaxPolicy 106
 - logmanSmoothPolicy 106
 - LOGNAME 69, 70
 - MAESTRO_OUTPUT_STYLE 69, 70
 - MAIL_ON_ABEND 73, 74, 78, 78
 - maxLen 104
 - minLen 104
 - PATH 71
 - POSIXHOME 78
 - SHELL_TYPE 74
 - startOfDay 104, 1012
 - SystemDrive 69
 - SystemRoot 69
 - TEMP 69
 - TIVOLI_JOB_DATE 69, 71
 - TMPDIR 69
 - TMPTIME 69
 - TWS_PROMOTED_JOB 69, 71
 - TWS_TISDIR 71
 - TZ 69, 71
 - UNISON_CPU 69, 71
 - UNISON_DATE 70
 - UNISON_DATE_FORMAT 72
 - UNISON_DIR 69, 71
 - UNISON_EXEC_PATH 69, 71
 - UNISON_EXIT 73

- UNISON_HOST 69, 71
- UNISON_JCL 73
- UNISON_JOB 70, 71
- UNISON_JOBNUM 70, 71
- UNISON_MASTER 70, 71
- UNISON_RUN 70, 71
- UNISON_SCHED 70, 71
- UNISON_SCHED_DATE 71
- UNISON_SCHED_EPOCH 70, 71
- UNISON_SCHED_IA 70, 71
- UNISON_SCHED_ID 70, 71
- UNISON_SHELL 70, 71
- UNISON_STDLIST 70, 71, 73
- UNISON_SYM 70, 72
- UNISONHOME 69, 71
- untilDays 105
- USE_EXEC 74
- USERDOMAIN 70
- USERNAME 70
- USERPROFILE 70
- variables, environment, notation xvi
- vartable
 - variable table definition 189
- vartable keyword 330
- version
 - utility command 921
- version command 455, 657
 - displaying composer banner 455

W

- wa_pull_info command 923
- wait
 - job state 499
- wappman command 471
 - logs and traces 471
- WAS
 - stopping 630
- Web service jobs 859
 - sample JSDL files 805
- web services job return codes 833
- WebSphere Application Server Liberty
 - infrastructure 54
 - starting 58
 - stopping 58, 630
- weekly run cycle 22
- wildcards
 - composer 374
- Windows command prompt
 - privilege level to issue
 - HCL Workload Automation commands
 - 54
- Windows operating systems
 - privilege level to issue
 - HCL Workload Automation commands
 - 54
- Windows OS
 - special characters, handling 130
- Windows user
 - defining 225
 - definition 225
 - running jobs on a dynamic pool 225
 - running jobs on a pool 225
 - running jobs on an agent
 - 225
 - scheduling on a dynamic pool 225
 - scheduling on a pool 225
 - scheduling on an agent

- 225
- WINDOWSTASK 210
 - within a relative interval
 - follows 281
 - follows relative to 88
 - join 289
 - matching criteria 88
 - within an absolute interval
 - follows 281
 - follows absolute to 89
 - join 289
 - matching criteria 89
- workload analysis
 - alerts 792
 - metrics 792
- workload application
 - command line 471
 - definition 21
 - skipping objects 461, 475
- workload applications
 - defining 346
 - importing 470
 - logs and traces 471
 - mapping 470
 - mapping file 462
 - regular expressions 468
 - skipping objects 471
- workload applications managing 460
- workload automation 150
- workload customizing
 - using variable table 145
- workload monitoring
 - alerts 792
 - metrics 792
- workload service assurance
 - 220, 220
 - calculating job start times 103
 - forecast plan 103
- workstation
 - backup master domain manager 29
 - class 33
 - creating 181
 - d-pool type 194
 - defining 181
 - domain manager 29
 - dynamic pool 29
 - dynamic pool type 194
 - fault-tolerant agent 29
 - mailbox files
 - NetReq.msg 61
 - master domain manager 29
 - pool 29
 - pool type 194
 - processes 54
 - remote engine type 182, 194
 - remote engine workstation 29
- workstation class 33
- workstation class definition 201
 - cpuclass 202
 - ignore 202
 - members 203, 350, 352
- workstation definition 181, 196
 - access 191
 - agent 193
 - autolink 195
 - behindfirewall 195
 - broker 193
 - cpuname 188
 - domain 191
 - dynamic agent 193
 - extended agent 192

- fta 192
- fullstatus 195
- host 191
- manager 192
- members 198
- os type 189
- protocol 198
- requirements 198
- secureaddr 190
- ServerID 197
- standard agent 192
- tcpaddr 190
- timezone 190
- type 192
- workstation links status 576
- workstation process status 576
- workstation processes 55, 55
 - batchman 55
 - inter-process communication 61
 - jobman 56
 - mailman 55
 - ServerID 55
 - managing change of job states 66
 - monman 55
 - netman 55
 - processes tree on UNIX 56
 - processes tree on Windows 57
 - start of day
 - establishing communication 65
 - starting 58
 - stopping 58
 - writer 55
- workstation properties 581
- workstation status 576, 581
- workstations
 - unique ID 417, 573
- writer process 55

X

- XATASK 210
- xref command 968
- xxtrct command 989

Y

- yearly run cycle 22