HCL Software

HCL Workload Automation for Z
Workload Automation Programming Language for z/OS User's Guide and Reference
Version 10.2 Fix Pack 2

Note

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

This edition applies to version 10, release 2, modification level 0 of HCL Workload Automation for Z (program number 19OP1249) and to all subsequent releases and modifications until otherwise indicated in new editions.

Contents

List of Figures

List of Tables

About this publication

Workload Automation Programming Language for z/OS User's Guide and Reference shows you how to use the Workload Automation Programming Language to easily access the features of the HCL Workload Automation for Z programming interface (PIF).

This guide is part of a set of guides that allows you to program many aspects of working with the products in the HCL Workload Automation family. These guides comprise:

- *Developer's Guide: Driving HCL Workload Automation for ZDeveloper's Guide: Extending HCL Workload Automation*
- *Developer's Guide: Driving HCL Workload Automation*

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

Conventions used in this publication.

The publication uses several typeface conventions for special terms and actions. Technical changes to the text are indicated by a vertical line to the left of the change. These conventions have the following meanings:

Information type	Style convention	Example
Commands	All capital letters	CREATE
References in the text to fields on panels	All capital letters	QUANTITY
File and directory names, input you should type in panel fields	Monospace	`MYAPPLICATION`
First time new term introduced, publication titles	Italics	*Application*

Chapter 1. Overview

The Workload Automation Programming Language is a programming language that provides you with easy access to the features of the HCL Workload Automation for Z Program Interface (PIF).

Workload Automation Programming Language gives you full access to all the PIF commands in an easy-to-use syntax, as well as a Batch Loader feature for all elements of the database. Extended PIF commands perform more complex functions from a single statement, such as determining the status of elements, and access to the HCL Workload Automation for Z TSO commands from within the same command stream.

Workload Automation Programming Language handles all the complicated elements of communication blocks and data exchange, allowing you to communicate with HCL Workload Automation for Z without having any specialized knowledge of memory manipulation or data definitions, allowing you to integrate HCL Workload Automation for Z with your operational environment with minimum effort.

After installing APAR PI79321, you can call Workload Automation Programming Language in the following ways:

- As a stand-alone batch job calling the main compiled REXX EXEC entry point EQQYXTOP with commands passed as SYSIN. The JCL procedure EQQYXJPX is provided for this purpose.
- As a batch job, called from your own REXX routines calling the main compiled REXX EXEC entry point EQQYXTOP, passing commands, and receiving output through the REXX stack. The JCL procedure EQQYXJPX is provided for this purpose.
- As a started-task workstation operation, passing the commands through operation user fields. Samples EQQWCMD1 and EQQWCMD2 are provided to set up started-task workstation operations.
- As a load module, named EQQWAPL, with commands passed either through SYSIN or using a control block created by a calling program. The JCL procedure EQQYXJPL is provided for this purpose. Online within TSO as part of a dialog process, calling the main compiled REXX EXEC entry point EQQYXTOP, passing commands, and receiving output through the REXX stack. Samples EQQWTS* are provided to set up calling WAPL from within TSO dialogs.

Note: The load module EQQWAPL runs without a TSO environment, therefore not all the commands and options are available to it. For details about the limitations, see Running Workload Automation Programming Language as a load module on page 36.

Workload Automation Programming Language works as a session-based interface to HCL Workload Automation for Z, sending a sequence of commands to a specified HCL Workload Automation for Z subsystem. It communicates with only one controller at a time, but can communicate with multiple subsystems in a single run, opening and closing communication in turn.

Note: Although Workload Automation Programming Language can perform functions across many items in the database and plans in a single run, the main design criteria for the processing engine is focused on flexibility, simple syntax, and ease of use. Because of this, running Workload Automation Programming Language against large

amounts of data in a single run can result in long run times. For large data handling, use EQQYCAIN or bespoke PIF programs instead.

Similarities to the Scheduling Operational Environment

Workload Automation Programming Language is a supported version of Scheduling Operational Environment, but not all the original features of Scheduling Operational Environment are supported as part of the product. Some unsupported features have been superseded by improved features in the language; other unsupported features were used for migration purposes only, not for normal usage.

Some of the unsupported features are still provided with the product, but are not documented. Others are not provided, but can still be downloaded from the z/Glue community website.

The following features are not currently supported by Workload Automation Programming Language:

Segment processing exits

This is an obsolete way to manipulate data from Workload Automation Programming Language. Basic data translation can be performed by the `TRANSLATE` command, and any other data manipulation can be performed by creating an Object variable and using the native Workload Automation Programming Language commands to manipulate that data.

WAPLEXEC programming

Writing your own external REXX programs calling Workload Automation Programming Language is no longer required, because the native Workload Automation Programming Language engine can perform many of the same processes that an external REXX routine could.

Some WAPLEXEC routines

Many of the original WAPLEXEC routines have now been superseded by native Workload Automation Programming Language commands. Although the syntax might be different, the functionality is broadly similar. Table 1: Old and new routines on page 20 shows the old and current routines.

Table 1. Old and new routines

Old routine	Now replaced by...
`EQQWXADD`	Workload Automation Programming Language **`ADD`** command.
`EQQWXJBF`	Workload Automation Programming Language **`LISTJOB`** command. Although the output format is different, the core function is the same, to find jobs in the database.
`EQQWXMOD`	Workload Automation Programming Language Current Plan Operation commands. For detailed information, see Current Plan Operation commands on page 185.

Table 1. Old and new routines (continued)

Old routine	Now replaced by...
`EQQWXLNK`	Workload Automation Programming Language `ADD` command with `LINK(YES)`.

The SUBSYSTEM specific input stream, set by `OPTIONS SUBSYS`

It is no longer supported because subsystem-specific options can now be set in the default options member by using the `ZSUBSYS` variable and `IF` statements.

EQQYXPRC procedure

The EQQYXPRC procedure that allows ISV product steps to be run in parallel with Workload Automation Programming Language commands is part of migration processing, therefore is not provided with Workload Automation Programming Language nor supported.

SEQQWAPL and WAPLEXEC member names

The PARM members of the Scheduling Operational Environment have been moved into a library called SEQQWAPL. As official product delivery enforces a member naming prefix of EQQ for delivery, all of these members have been renamed. The sample job EQQWPLCO will create copies of the new members under the original name. The remaining WAPLEXEC programs have also been renamed; for details, see WAPLEXEC programs on page 453. The original SOEEXEC library can still be used with Workload Automation Programming Language under the original names.

Note: The ILSON utility to convert HCL Workload Automation for Z data to ISPF tables is not documented or supported as part of Workload Automation Programming Language.

For all the supported WAPLEXEC commands, see WAPLEXEC programs on page 453.

Version compatibility

Workload Automation Programming Language is compatible with all the supported versions of HCL Workload Automation for Z.

By default, if you do not specify `VER` and `MOD` in the JCL or `OPTIONS` statement, Workload Automation Programming Language operates at the latest level of code installed, communicating and generating output for the equivalent level of HCL Workload Automation for Z.However, Workload Automation Programming Language can work with HCL Workload Automation for Z from version 9.x to the current release, but only supported for versions whose Service date is not passed.

For cross-version compatibility, you can define to Workload Automation Programming Language which version of HCL Workload Automation for Z to operate as. In this way, Workload Automation Programming Language will allow you to perform only actions applicable to that release, and will generate only output compatible with that release. Performing database updates when specifying an HCL Workload Automation for Z version other than the version of the database might result in update failures if the object being updated has different structures between the two releases.

You can specify the version to operate as by appending the version to the subsystem name with a hyphen when you call EQQYXTOP, for example `SUBSYS='ZWSC-950'`. Otherwise Workload Automation Programming Language loads support for the latest version that was generally available at the time it was released.

You can change the version during a session with the **`OPTIONS VERSION`** statement, which causes the operating mode to be initialized again. This would allow you to perform some complex actions, for example, performing some V10.1 specific actions with a V10.1 controller, and then later generating some V9.5-compatible output within the same session from a V10.1 controller.

The version itself can be specified in a variety of formats, for example 9.5, 950, V950, 9.5.0, or V9R5M0. If you use one of these formats as part of your naming convention for HCL Workload Automation for Z data sets, such as your message library or load library, then you will be able to use the `&VER` symbol in the EQQYXJPX procedure to form part of the names.

There is a special case of * that you can use either as **`VER=*`** from the JCL (only if you are not using it in data set names), or as **`OPTIONS VERSION(*)`**. This causes Workload Automation Programming Language to start up using the latest supported version available, but when it is connected to an HCL Workload Automation for Z subsystem it will detect what version the subsystem is using and automatically switch to that version.

Note: You can set **`VER=*`** only if you are using HCL Workload Automation for Z V9.5, or later.

As well as major versions of the product, some new features could be released between versions by applying PTFs. When this occurs, module EQQYXTOP is given a new modification (MOD) level, as shown in the EQQB001I message at the beginning of the Workload Automation Programming Language log. The MOD level is shown after the version as a plus sign (+) followed by 3 digits:

```
EQQI001I ADCDMST1,JOB06592 Starting WAPL v9.5 +001
```

By default Workload Automation Programming Language always runs with the latest version and MOD level. For any specifically selected version, without selecting a MOD level Workload Automation Programming Language always operates at the highest MOD level for that version.

The following features have been delivered by MOD levels:

- V930+001 October 2016 – Enable NOP, and HOLD in the database.

To select a specific MOD level within Workload Automation Programming Language, specify the MOD level in the **`OPTIONS VERSION`** keyword. For example, **`OPTIONS VERSION(V950+000)`** selects MOD level +000, which is the original version 9.5 of the product, and suppresses the generation of any Batch Loader keywords relating to NOP or Manually Hold in the database, along with any other new features added to version 9.5 after the GA version.

Workload Automation Programming Language automatically sets the appropriate value for the USRLEV keyword for any INIT statements it generates, and any USRLEV statements that do not include a USRLEV keyword. Workload Automation Programming Language does not override any USRLEV keywords already present in an INIT statement.

To select the MOD level from JCL use the MOD symbolic parameter in the JCL, as shown in the following example:

```
//RUNWAPL EXEC EQQYXJPX,
//        VER='V950'
//        MOD='+000'
```

During the rollout of a version upgrade, you might have controllers operating at different versions. You can manage this issue by setting **OPTIONS VERSION** statements in your site default member, according to the controller version that you are using, as shown in the following example:

```
VARSUB SCAN
IF !ZSUBSYS = "WSMC" THEN
   OPTIONS VERSION(V950+001)
IF !ZSUBSYS = "WSLC" THEN
   OPTIONS VERSION(V930)
```

Ensure that the values for **VER** and **MOD** in the PROC statement of the EQQYXJPX and EQQJXJPL procedures are always set to null values, as shown in the following example. This assures that you always operate at the latest version of Workload Automation Programming Language installed (unless the version value is overwritten by the setting of a job). Set the **VER** and **MOD** keywords to a significant value only if you are working with a controller whose version is earlier than Workload Automation Programming Language.

```
//EQQYXJPX PROC @=,
//         ARGS='',
//         CMD=EQQYXTOP,
//         REG=4M,
//         SUBSYS=,
//         VER=,
//         MOD=,
//         #=
```

Small product enhancements

For some versions of HCL Workload Automation for Z, new features were added via PTF as Small Product Enhancements (SPEs). Some SPEs can modify the structure of data within HCL Workload Automation for Z. Support for these features can be turned on selectively.

To list Small Product Enhancements, use the **SHOW SPE** command.

To turn on an SPE delivered feature use **OPTIONS SPE**, as in the following example:

```
OPTIONS SPE(WLM,PEND)
```

To turn off an SPE delivered feature, turned on earlier in the session use **OPTIONS SPE**; for example, `OPTIONS SPE(WLM=N)`.

Any options added to an earlier version by SPE are automatically included in later versions.

Using **OPTIONS SPE** to turn on an SPE that is automatically available for that release is ignored. Using **OPTIONS SPE** to turn on an SPE that is not eligible for use with the version of HCL Workload Automation you are running, generates an error.

Note: If you use the **OPTIONS SUBSYS** keyword to create subsystem-specific OPTIONS members, the **SPE** keyword must be maintained within these members, because the relevant enhancements are installed on each subsystem.

Setting up the Workload Automation Programming Language environment

If you run a version of SOE or Workload Automation Programming Language JCL that refers to SEQQWAPL members without the EQQ prefix, ensure that you set up the environment by running the EQQWPLCO sample job.

Note:

1. When you run Workload Automation Programming Language jobs under the control of current plan, to enable the use of Workload Automation Programming Language occurrence variables, or of the == short form, ensure that you run the WAPL job with a user ID that has read access to the CP occurrence or operation controlling the job. Otherwise, a message similar to the following might be issued:

   ```
   EQQI012A Job ZUSRDH1W,JOB02218 is external to IWS
   ```

2. To use Operation User fields to contain Workload Automation Programming Language commands, ensure that you run the job with a user ID that has read access to the CP occurrence or operation controlling the job, and read access to the user fields containing the commands. Otherwise, a message similar to the following might be issued:

   ```
   EQQI145E No user fields match EQQ-SYN-*
   ```

3. To ensure accurate use of queries for input arrival, set CWBASE (century window base) and HIGHDATE to 00 and 711231, respectively (these are the default values). If CWBASE and HIGHDATE are set to values different than the defaults, any queries against input arrival fields fail. If the EQQCPOP DD statement is used together with the supplied variable &OYMD1, Workload Automation Programming Language cannot locate itself in the current plan, message EQQI012A is issued, and the supplied occurrence variables or user fields are not accessed.

   For details about the settings of CWBASE and HIGHDATE in the INIT statement, see *Customization and Tuning*.

4. When you use Workload Automation Programming Language, do not set DATINT to YES in the INIT statement of EQQYPARM. This setting would create a lock conflict when performing **OPTIONS DBMODE(UPDATE)** or **OPTIONS DBMODE(COPY)**, and data could get lost.

Language support

Command syntax is in English, but output messages might be displayed in other languages in later releases.

The language can be selected by the LANG symbolic parameter in the JCL.

Note: Workload Automation Programming Language does not support DBCS.

Command language

The WAPL command language is a combination of Workload Automation Programming Language core commands, Data Access commands (native PIF requests), Current Plan Operation commands, Function Based commands (extended PIF requests), HCL Workload Automation for Z TSO commands, and Batch Loader. The syntax uses the same conventions for all commands, which is easy to learn and to use.

Commands can be passed to Workload Automation Programming Language in many ways. Workload Automation Programming Language reads commands from the following input streams, in the following order:

1. EQQOPTS. You can use the optional EQQOPTS DD statement to set the defaults for your environment when Workload Automation Programming Language starts. If you define an EQQOPTS DD statement, it is read and run before any other statements. Although EQQOPTS is intended to run **`OPTIONS`** statements, you can use it to run any command; therefore you can decide to set the **`OPTIONS`** and variables specific to the LPAR where the job is running.
2. The ARG= symbolic parameter in the JCL. This only passes keywords for an **`OPTIONS`** statement. There are some immediate **`OPTIONS`** that can be specified *only* as an argument, these take effect as Workload Automation Programming Language starts and are not valid in `OPTIONS` statements elsewhere. This allows Site and Subsystems to be overridden at Workload Automation Programming Language startup in the program call or JCL.
3. EQQFILE. You can use the optional EQQFILE DD statement to set the default file specifications for all your output files. Use of this DD statement is deprecated, use of the **`LOADDEF`** statement in the command stream is recommended instead. The SEQQWAPL library contains an assortment of predefined members that can be used to load groups of **`OUTPUT`** statements. The EQQFLALL member loads **`OUTPUT`** statements for every segment; the **`LOADDEF`** equivalent of this is **`LOADDEF *`**. The name of the DD statement can be overridden by **`OPTIONS FILESPEC(<dd>)`**.
4. External data queue. If data is detected on the external data queue (REXX stack) it is read as command input. If you use the **`INPUT`** keyword to pass a control block address in a previous input stream, the contents of the control block are added to the external data queue at this point, before it starts reading. It processes only what is in the external data queue at the time it starts reading; any lines added during the processing of the commands are not read at this stage.
5. SYSIN. Use this optional DD statement to specify the commands for the session you are running. By default, the DD name for this input source is called SYSIN in batch, and INPUT when processed in the foreground, but you can set an alternative name with the **`OPTIONS INPUT(<ddname>)`** statement specified in any of the previous input sources.
6. Post Processing. If **`OPTIONS POSTPROC(YES)`** is set, any further content of the external data queue will be processed at this point. The queue will be processed until it is empty, so any additional lines created whilst processing the queue will also be read.

Although there are many different streams through which you *can* specify command input, these are intended to let you separate your statements into standardized and reusable blocks. You can specify all your command input in one simple single stream, making it simple to see exactly what you are asking Workload Automation Programming Language to do in one place.

Note:

1. Each input stream is parsed in its entirety before any statement within it is run, therefore any **OPTIONS** statement that has an impact on syntax does not take effect until the start of the next input stream.
2. You must specify any syntax options in the **OPTIONS** or **ARG** streams.
3. You can set any subsystem-specific options in the **OPTIONS** stream by using the ZSUBSYS variable and IF statements.
4. If EQQYXTOP is being executed in foreground mode (rather than batch), a command source is not read if it is referring to the SYSIN DD statement, because this could lead to the program stalling if SYSIN is left allocated to the terminal.

Output files

By default, Workload Automation Programming Language does not generate any output from a **LIST** or **SELECT** command. To generate an output, you must specify **OUTPUT** statements to define the segments and fields to be extract.

You can produce two kinds of output:

DATA

It is produced in ISPF Streamed Output Notation (ILSON). This notation allows you to easily read and update the records from your own programs, without having to know much about the underlying record structure. DATA records are written as a result of **LIST** and **SELECT** requests. If **LIST** is performed with `SELECT(Y)`, the output is written only for the subsequent **SELECT** action, avoiding duplication of common segments.

LOADER

Also known as Batch Loader, this is a structured text representation of a database object that you can use to define an object in the database. Batch Loader records are written only as a result of a **SELECT** request.

Use the **OUTPUT** statement to define what records you want to extract, and in what format. This statement defines also where the output is sent.

Use **LOADDEF** to load predefined sets of statements. The predefined statements generate both DATA and LOADER output for every field available for each segment. By default, DATA is sent to DD statement OUTDATA and LOADER is sent to DD statement OUTBL.

For example, to load:

- **OUTPUT** statements for every segment available, specify `LOADDEF *`
- **OUTPUT** statements for all the segments in an application definition, specify `LOADDEF AD*`
- Application definition statements and turn off batch loader, specify `LOADDEF AD* LOADER(-)`
- Application definition statements, turn off batch loader, and send DATA output to the external data queue, specify `LOADDEF AD* LOADER(-) DATA(*)`

Batch loader

Workload Automation Programming Language can extract all HCL Workload Automation for Z database objects and Current Plan JCL into a Batch Loader format.

To generate Batch Loader, you need a combination of **OUTPUT** statements (or **LOADDEF** statements to load pre-built **OUTPUT** statements) and a **SELECT** request. The **SELECT** request can either be a direct **SELECT** statement, or a **LIST** statement with SELECT(Y). The **SELECT** request must select the complete record of the object for which you require batch loader output.

The output can be complete batch loader, containing every complete field, including blanks. If you use **OPTIONS STRIP(Y)** and **SHOWDFLT(N)** the leading zeroes, trailing blanks, and the keywords set to their default values, are removed.

The following example shows how to generate compact Batch Loader for today's version of an application:

```
OPTIONS STRIP(Y) SHOWDFLT(N)
LOADDEF AD* DATA(-)
LIST ADCOM ADID(MYAPPL) VALID(=) SELECT(Y)
```

You can make the Batch Loader format compatible with EQQYLTOP for Application Definitions and Operator instructions by using **OPTIONS SYNTAX(LEGACY)**; it uses similar constructs for all other objects. You can then use this Batch Loader as input commands to Workload Automation Programming Language to re-create these objects within HCL Workload Automation for Z.

Use the **OUTPUT** statements to influence what keywords appear in Batch Loader statements generated from the database. Use the **TRANSLATE** command to transform fields based on rules, to perform lifecycle management of database objects.

You can interchange Batch Loader and Command Language in the same step, in any sequence you like, the only consideration being that you cannot place Command Language statements within the scope of Batch Loader statements for any individual HCL Workload Automation for Z object.

Command Language statements can cause Batch Loader statements to be generated that have been modified by **TRANSLATE** processing. These Batch Loader statements can be output to the external data queue to be run at the end of the step by specifying **OPTIONS POSTPROC(YES)**. This means that you can extract an object from the HCL Workload Automation for Z database and reload it into the database, adjusted for your development lifecycle, in a single step.

Use **OPTIONS DBMODE** to instruct Workload Automation Programming Language about how to process the Batch Loader statements.

- Specify **ADD** or **REPLACE** to take the object exactly as described in the text and update the database.
- Specify **UPDATE** or **COPY** to read the existing object from the database and apply the changes as described in the text. Specify enough detail to clearly identify each segment to update and the keywords for the fields to be changed; the whole object does not need to be specified in text form. For multilevel objects, set a Batch Loader statement for each segment level leading to the segment being updated; for example, to update a dependency interval you need to specify **ADSTART**, **ADOP**, and **ADXIV**.
- Specify **EXPORT** to use Batch Loader as input, pass through **TRANSLATE** statements and generate new translated Batch Loader as output.
- Specify **SCAN** to check Batch Loader statements for basic syntax only.

HCL Workload Automation for Z PIF concepts

The following sections explain the basic concepts of the HCL Workload Automation for Z programming interface (PIF) and data structure.

Data sources and structures

The use of the HCL Workload Automation for Z PIF is all about reading and writing data from HCL Workload Automation for Z.

Data can come from one of the following:

- Database
- Current Plan (including JCL)
- Long-Term Plan (including JCL)

Despite the different sources, the data follows the same basic rules.

Each entity in HCL Workload Automation for Z is represented by a single record; for example, an Application record, an ETT rule record, and a Calendar record. The record is formatted into one or more Segments.

A Segment is a subdivision of a record for a particular type of information. Sometimes a record has only one segment of a particular type, sometimes it has several. For example, an Application might have one common segment (ADCOM) and several run cycle segments (ADRUN) and operation segments (ADOP). The operations might have several special resources (ADSR), and so on.

The HCL Workload Automation for Z record structure is hierarchical, this means that the segments are ordered in parent-child relationships. Hence, an operation’s child segments follow their parent operation segment and precede the next operation segment.

Every multi-segment object always begins with a Common Segment that defines the key and high level information about the object. The easiest way to visualize what is in the Common Segment is to think about what information appears in the non-scrollable portion of the first panel you see when you browse an object.

The following example shows the relationships of the AD Record:

```
ADCOM -+- Common segment (1 per appl)
       |       |
       += ADAPD =+= Application dependency(ies)
       |
       += ADRUN =+= Run Cycle(s)
       |         |
       |         +- ADRULE - Rule (1 per run cycle)
       |
       += ADOP =+= Operation(s)
                 |
                 += ADDEP = Dependency(ies)
                 |
                 += ADXIV = External dependency interval(s)
                 |
                 += ADSR = Special resource(s)
```

```
                |
                +- ADOPEXT - Extended name (1 per op)
                |
                +- ADOPSAI - System Automation (1 per op)
                |
                += ADCNC = Condition(s)
                |
                += ADCNS = Conditional dependency(ies)
                |
                += ADCIV = Conditional dependency interval(s)
                |
                += ADUSRF = User field(s)
                |
                +- ADVDD = Variable durations and deadlines
                |
                +- ADRE = Remote job
                |
                +- ADLAT = User-defined late information
```

The following example shows the logical layout of the AD record (either an application or job stream). An AD record could be a sequence of segments:

```
ADCOM ADAPD ADRUN ADRULE ADRUN ADRULE ADOP ADOP ADDEP ADSR ADSR ADOP ADDEP
```

When the programming interface retrieves a record it presents a header that lists each segment name and the offset to where the segment starts. Workload Automation Programming Language automatically processes the header and decodes each segment in turn, allowing you to decide the segments to be processed. If you want to use a Segment Processing Exit, the exit is automatically called for every segment you referenced with the **OUTPUT** statement.

Use the **SHOW OBJECT** command to understand the structure of any HCL Workload Automation for Z object.

For example, the **SHOW OBJECT(CL)** command shows all the available object variables for a calendar with **-n-** showing where sequence numbers fit into the syntax:

```
08/22 10.47.39 EQQI200I SHOW OBJECT(CL)
08/22 10.47.39 EQQI601A Object: @OBJ-CLNAME
08/22 10.47.39 EQQI601A Object: @OBJ-CLDAYS
08/22 10.47.39 EQQI601A Object: @OBJ-CLSHIFT
08/22 10.47.39 EQQI601A Object: @OBJ-CLDESC
08/22 10.47.39 EQQI601A Object: @OBJ-CLVERS
08/22 10.47.39 EQQI601A Object: @OBJ-CLLDATE
08/22 10.47.39 EQQI601A Object: @OBJ-CLLTIME
08/22 10.47.39 EQQI601A Object: @OBJ-CLLUSER
08/22 10.47.39 EQQI601A Object: @OBJ-CLLUTS
08/22 10.47.39 EQQI601A Object: @OBJ-#CLSD
08/22 10.47.39 EQQI601A Object: @OBJ-CLSD-n-CLSDDATE
08/22 10.47.39 EQQI601A Object: @OBJ-CLSD-n-CLSDSTAT
08/22 10.47.39 EQQI601A Object: @OBJ-CLSD-n-CLSDDESC
08/22 10.47.39 EQQI601A Object: @OBJ-#CLWD
08/22 10.47.39 EQQI601A Object: @OBJ-CLWD-n-CLWDDAY
08/22 10.47.39 EQQI601A Object: @OBJ-CLWD-n-CLWDNUM
08/22 10.47.39 EQQI601A Object: @OBJ-CLWD-n-CLWDSTAT
08/22 10.47.39 EQQI601A Object: @OBJ-CLWD-n-CLWDDESC
08/22 10.47.39 EQQI299I Statement completed - RC=0 (00000014)
```

HCL Workload Automation for Z PIF requests

A command within the HCL Workload Automation for Z PIF is known as a request. Table 2: HCL Workload Automation for Z PIF requests on page 30 shows, at a high level, different types of PIF requests.

Table 2. HCL Workload Automation for Z PIF requests

Request	Read/Write	Acts upon	Purpose
DELETE	WRITE	Database and Plans	Deletes objects
EXECUTE	WRITE	Current Plan	Commits updates to the Current Plan
INIT			Starts a session with a controller
INSERT	WRITE	Database and Plans	Creates a new record or segment
LIST	READ	Database and Plans	Searches for records
MODIFY	WRITE	Plans	Updates records in the plans
OPTIONS			Sets session options
REPLACE	WRITE	Database	Updates records in the database
RESET		Current Plan	Abandons updates to the Current Plan
SELECT	READ	Database and Plans	Reads a single record
SETSTAT	WRITE	Current Plan	Changes a condition status
TERM			Terminates a session with a controller

INIT and **TERM** requests are done for you automatically, though you can do your own **INIT** and **TERM** requests many times in a Workload Automation Programming Language session to communicate with different controllers.

For the requests that directly read or write data, you must specify enough of an item key to uniquely identify a single object, with the exception of **LIST**. The **LIST** statement is used to find records. From the returned **LIST** you can derive the unique key for every object found, against which you could then take an action (for example, **SELECT**, **DELETE**, **MODIFY**).

LIST requests are always performed against the Common Segment. However, **SELECT** requests can be performed against the entire record, to retrieve all the segments or just the common segment.

The requests that write data work in one of the following ways:

- Database updates are performed by creating the whole record in storage before sending it to HCL Workload Automation for Z. Database updates are handled in Workload Automation Programming Language by Batch Loader constructs.
- Plan updates are performed by using keywords with the **`MODIFY`** or **`INSERT`** instructions to alter the values of fields within the object. Plan updates are handled in Workload Automation Programming Language by commands.

Chapter 2. Running Workload Automation Programming Language

You can run Workload Automation Programming Language as a batch job by using the compiled REXX exec, as a load module, online within a TSO session, as a started task workstation, or as a console command. A user program must run as an authorized program.

By installing APAR PI79321, you can run Workload Automation Programming Language without the need of reference files at startup time, and with a JCL that is simpler and more efficient. Therefore, the old procedure EQQYXJCL is deprecated; you can use procedure EQQYXJPX, instead.

You can run Workload Automation Programming Language with the following methods:

EQQYXTOP

Use this compiled REXX EXEC in batch or online TSO. It can be called by other REXX EXEC programs and receive commands through the REXX stack. It can also send data to calling EXEC programs through the REXX stack. The EQQYXJPX procedure is provided to run EQQYXTOP.

This should be the primary method to run Workload Automation Programming Language.

EQQWAPL

Use this load module in batch, or call it from other load modules. Commands are passed through a control block and data is received by a control block. The load module does not create an internal TSO environment, so some commands and options are not available. The EQQYXJPL procedure is provided to run EQQWAPL.

Table 3: DD statements used by Workload Automation Programming Language on page 32 shows the DD statements used by Workload Automation Programming Language and their description. You can specify alternative DD statements as destinations for the output with the `LOADDEF` statement. The following example shows how to send ILSON data to MYDATA and Batch loader output to the REXX stack:

```
LOADDEF AD* DATA(MYDATA) LOADER(*)
```

Table 3. DD statements used by Workload Automation Programming Language

DD statement	Description

EQQCPOP	Optional. Use this DD statement to identify the occurrence or operation that is controlling the job in the current plan. This prevents Workload Automation Programming Language from unnecessary searching in the CP.
	
	For example, to identify the jobs that are submitted by HCL Workload Automation for Z, set EQQCPOP as follows:
	`//EQQCPOP DD *`
	`&OADID. &OYMD1.&OHHMM. &OOPNO`
	
	To identify the jobs that are not submitted by HCL Workload Automation for Z, set the EQQCPOP to a single hyphen (-):

Table 3. DD statements used by Workload Automation Programming Language (continued)

DD statement	Description
	`//EQQCPOP DD *` `-`
EQQDUMP	Optional. A diagnostic log, which might include additional information about problems. Use this DD statement if you specified **TRACE** in EQQYPARM.
EQQMLIB	Required. The HCL Workload Automation for Z message library, which is normally delivered in the SEQQMSG0 library.
EQQMLOG	Required. The log for any message issued from HCL Workload Automation for Z.
EQQOPTS	Optional. Use this DD statement to set the options specific for your environment. It can be either a sequential file, a member in a partitioned data set, or a library. When you use EQQWAPL as a service within a permanently available started task, do *not* use EQQOPTS, because if the started task tries to run EQQWAPL in parallel this might produce I/O problems.
EQQSMTP	Optional. This DD statement is needed only if you are issuing the **SENDMAIL** command. Define a SYSOUT DD statement pointing at the SMTP class and writer, for example `//EQQSMTP DD SYSOUT=(B,SMTP)`
EQQYLOG*x*	Optional. If you define an EQQYLOG*x* DD statement, all Workload Automation Programming Language messages are directed to this file. *x* can be a number from 1 to 5, and determines the message level used. For details, see MSGLEVEL – Output message level on page 429.
EQQYPARM	Optional. Use it to provide an override **INIT** statement, to set running options.
OUTBL	Optional. The default output destination for batch loader statements when you use **LOADDEF** to load predefined output statements.
OUTDATA	Optional. The default output destination for ILSON data when you use **LOADDEF** to load predefined output statements.
STEPLIB	Optional. This DD statement is needed when: • The HCL Workload Automation for Z load modules are not link listed, or the version in the link list is not the version you need. • To provide access to user load modules that might call EQQWAPL.
SYSPROC	Used only for EQQYXTOP. Use this DD statement to point to the library containing the REXX entry point EQQYXTOP. This is normally delivered in the SEQQMISC library
SYSTSIN	Used only for EQQYXTOP. TSO environment terminal simulated input. Allocate to DUMMY.

Table 3. DD statements used by Workload Automation Programming Language (continued)

DD statement	Description
SYSTSPRT	Required (even if you defined an EQQYLOG*x* DD statement). The main message output stream.

Running Workload Automation Programming Language in batch

With APAR PI79321, the JCL EQQYXJPX procedure is provided to run the complete EXEC version of Workload Automation Programming Language in batch, by applying the new method of loading EQQREF, EQQLANG, and EQQFILE internally, and making EQQOPTS optional. Use EQQYXJPX instead of the EQQYXJCL procedure for all new Workload Automation Programming Language jobs; this will reduce startup I/O processing.

The simplest use of EQQYXJPX is a call to the procedure, with a SYSIN DD statement to enter the commands. The following example shows how to add an application to the plan:

```
//RUNWAPL  EXEC EQQYXJPX,
//         SUBSYS=WSMC
//SYSIN    DD *
ADD ADID(ADHOC)
```

You can use the following symbolic parameters with EQQYXJPX:

ARGS

Additional OPTIONS to be passed to Workload Automation Programming Language. They are run after any statements in EQQOPTS, but before any statements within the REXX stack, passed control block, or SYSIN.

CMD

REXX EXEC entry point to use (default is EQQYXTOP).

To run your own REXX EXEC, call EQQYXTOP and pass the name of your EXEC using the CMD symbolic. Ensure that your EXEC is included in the SYSPROC concatenation.

REG

REGION to allocate (default is 4M).

MOD

The Workload Automation Programming Language modification level. Modification level indicates a release of Workload Automation Programming Language between official versions. Specify the modification level in the format `+nnn`, for example MOD='+001'.

Unless you need to connect to or generate output for an earlier version of the product, do not set the **`MOD`** keyword to any value. The default is the modification level of Workload Automation Programming Language installed.

SUBSYS

The controller with which to communicate.

VER

The Workload Automation Programming Language version. You can specify the version in various formats, to allow the same symbolic to be used in data set names where appropriate (for example VER=950, VER=V950, VER=V9R5M0). For versions later than 9 and earlier than 16, you can also use the hexadecimal format (for example, VER=A10 for version 10.1).

Unless you need to connect to or generate output for an earlier version of the product, do not set the **VER** keyword to any value. The default is the version of Workload Automation Programming Language that is installed.

Use the **OUTPUT** and **LOADDEF** statements to specify additional DD statements to be used. By default, ILSON data is written to OUTDATA and batch loader data is written to OUTBL, but keywords in **LOADDEF** overrides original keyword values in the predefined **OUTPUT** statements.

The following example shows a Workload Automation Programming Language job to send ILSON data to MYDATA DD statement. Because no **LOADER** keyword is specified, Batch Loader data is sent to the OUTBL DD statement, according to the default behavior.

```
//RUNWAPL  EXEC EQQYXJPX,
//         SUBSYS=WSMC
//MYDATA   DD SYSOUT=*,LRECL=1024
//OUTBL    DD SYSOUT=*
//SYSIN    DD *
OPTIONS STRIP(Y) SHOWDFLT(N) SELECT(Y)
LOADDEF AD* DATA(MYDATA)
LIST ADCOM ADID(ADHOC) VALID(=)
```

Set the **ARGS** symbolic parameters to specify the **OPTIONS** keywords rather than including them in SYSIN, as shown in the following example. In this way, the **OPTIONS** are influenced directly in JCL, rather than within SYSIN data sets.

```
//RUNWAPL  EXEC EQQYXJPX,
//         ARGS='STRIP(Y) SHOWDFLT(N) SELECT(Y)',
//         SUBSYS=WSMC
//MYDATA   DD SYSOUT=*,LRECL=1024
//OUTBL    DD SYSOUT=*
//SYSIN    DD *
LOADDEF AD* DATA(MYDATA)
LIST ADCOM ADID(ADHOC) VALID(=)
```

If you reference a supplied variable or user field related to the controlling occurrence in the current plan, Workload Automation Programming Language batch jobs check if they are being controlled within the CP by searching for a started instance of the *job name* under which they are running, to find an operation with the same JES number. This works if the job is submitted outside of the controller or tracker but tracked in the current plan, or if the job is submitted from the controller or tracker.

For jobs that are submitted by the controller or tracker, you can be more efficient in identifying the controlling occurrence by passing the details directly from the supplied JCL tailoring variables through an EQQCPOP DD statement, as shown in the following example:

```
//RUNWAPL  EXEC EQQYXJPX,
//         SUBSYS=WSMC
```

```
//SYSIN    DD *
ADD ADID(ADHOC)
/*
//*%OPC SCAN
//EQQCPOP  DD *
&OADID. &OYMD1.&OHHMM. &OOPNO
```

To identify the jobs that are not submitted by HCL Workload Automation for Z and prevent unnecessary searching in the CP, set the EQQCPOP DD statement to a single hyphen (-) as follows:

```
//EQQCPOP DD *
-
```

Alternatively, you can set the `OPTIONS OPID(-)` command.

Running Workload Automation Programming Language as a load module

In addition to running Workload Automation Programming Language as a compiled REXX EXEC, you can also use the load module EQQWAPL.

Because EQQWAPL runs without a TSO environment, the following limitations apply:

- The **`MEMBER`** keyword of **`OISTART`** and **`JSSTART`** is not available. You can specify the text for these commands only by using the **`DLM`** keyword, or the respective **`OIT`** or **`JST`** statement.
- The **`INCLUDE`** command cannot be used in the format **`INCLUDE <DDNAME>(<MEMBER>)`**. You can use other formats, for example **`INCLUDE <DDNAME>`** and **`INCLUDE USER_FIELD(<name>)`**.
- The **`JCL`** keyword of **`ADDJOB`** is not available.
- You cannot use the TSO commands **`BULKDISC, DEFINE, DELETE, JSUACT`**, and **`REPRO`**. You can use the PIF version of **`DELETE`** to delete the database and plan objects.
- You cannot use the **`DYNLOG`** and **`XLATE`** keywords of **`OPTIONS`**.

Apart from the above limitations, the EQQWAPL load module is equivalent to EQQYXTOP. EQQWAPL is provided for scenarios where you want to call Workload Automation Programming Language from other load modules or exits; in all other cases, you can use EQQYXTOP.

To call the load module version from JCL, use the EQQYXJPL procedure. This procedure is similar to EQQYXJPX, but it is set to run the load module instead of a compiled EXEC. Other than the name of the procedure, running Workload Automation Programming Language as a load module is the same as running the EXEC. Within the procedure the differences are:

- IKJEFT01 is not run to establish a TSO environment.
- The CMD symbolic points directly to the load module to execute (default EQQWAPL).
- The SYSPROC and SYSTSPRT DD statements are omitted.

The following example shows a simple Workload Automation Programming Language step to add an application to the plan using the load module version of Workload Automation Programming Language:

```
//RUNWAPL  EXEC EQQYXJPL,
//         SUBSYS=WSMC
//SYSIN    DD *
```

```
ADDJOB ADID(ADHOC)
```

To call Workload Automation Programming Language from another load module in batch, use the EQQYXJPL procedure. The following example shows the Workload Automation Programming Language step to call Workload Automation Programming Language from another load module. There is no SYSIN DD statement, because the commands are passed to EQQWAPL through a control block.

```
//RUNWAPL  EXEC EQQYXJPL,
//         CMD=MYPROG,
//         SUBSYS=WSMC
```

You can pass commands to EQQWAPL through an ordinary SYSIN, but when EQQWAPL is called by another load module, commands are passed through a control block. Create the WPLI command control block as follows:

Table 4. WPLI command control block

Offset (dec)	Offset (hex)	Length	Type	Field	Description
00	00	4	Signed	WPLILINE	Number of 80 byte input lines.
04	04	4	Unsigned	WPLIOADR	Address of output data block. Must be initialized to zero.
08	08	4	Signed	WPLIOSIZ	Size of output block. Must be initialized to zero.
12	0C	4	Unsigned	WPLIMADR	Address of message data block. Must be initialized to zero.
16	10	4	Signed	WPLIMSIZ	Size of message data block. Must be initialized to zero.
20	14	80xWPLILINE	Character	WPLICMDS	Each line of WAPL commands.

Specify the commands to be passed to Workload Automation Programming Language in the **WPLICMDS** field. Each line of command text must be long 80 bytes, with normal continuation rules applying; you can use blanks to reach the required 80-byte length. The **WPLILINE** field must be set to represent the number of 80 byte lines, because it refers to the command text in **WPLICMDS**. The **WPLIOADR**, **WPLIOSIZ**, **WPLIMADR**, and **WPLIMSIZ** must be initialized to zero, because they will be set by Workload Automation Programming Language if any data is returned.

The following example shows a control block running the following command statements:

- `OPTIONS STRIP(Y) SHOWDFLT(N)`
- `LOADDEF AD* DATA(-) LOADER(*)`
- `SELECT AD ADID(DAILYPLANNING)`

```
WPLI:
    START ADDRESS: 0000CAA0   LENGTH: 0xFC
            00112233 44556677 8899AABB CCDDEEFF  -0123456789ABCDEF-
    000000  00000003 00000000 00000000 D6D7E3C9  *... ........OPTI*
    000010  D6D5E240 E2E3D9C9 D74DE85D 40E2C8D6  *ONS STRIP(Y) SHO*
    000020  E6C4C6D3 E34DD55D 40404040 40404040  *WDFLT(N)        *
    000030  40404040 40404040 40404040 40404040  *                *
    000040  40404040 40404040 40404040 40404040  *                *
    000050  40404040 40404040 40404040 D3D6C1C4  *            LOAD*
    000060  C4C5C640 C1C45C40 C4C1E3C1 4D605D40  *DEF AD* DATA(-) *
    000070  D3D6C1C4 C5D94D5C 5D404040 40404040  *LOADER(*)       *
    000080  40404040 40404040 40404040 40404040  *                *
    000090  40404040 40404040 40404040 40404040  *                *
    0000A0  40404040 40404040 40404040 E2C5D3C5  *            SELE*
    0000B0  C3E340C1 C440C1C4 C9C44DC4 C1C9D3E8  *CT AD ADID(DAILY*
    0000C0  D7D3C1D5 D5C9D5C7 5D404040 40404040  *PLANNING)       *
    0000D0  40404040 40404040 40404040 40404040  *                *
    0000E0  40404040 40404040 40404040 40404040  *                *
    0000F0  40404040 40404040 40404040
```

If any data is directed to the REXX stack within EQQWAPL, this data is returned to the calling program in a second control block.

After calling EQQWAPL, the calling load module checks the fields **WPLIOADR** and **WPLIOSIZ** in the WPLI control block. If these fields are set to a value different from zero, the data has been returned in a second control block WPLO. If messages have been returned, **WPLIMADR** and **WPLIMSIZ** are set to a value different from zero, and a second WPLO control block containing messages is created.

The WPLO control block has the following format:

Table 5. WPLO control block format

Offset (dec)	Offset (hex)	Length	Type	Field	Description
0	0	4	Signed	WPLOLINE	Number of WAPL output lines.
4	4	Variable	*	WPLORECS	Block of WAPL output records.

Though batch loader data output from Workload Automation Programming Language is set to 80 bytes, ILSON data output can be extremely variable in length, therefore the **WPLORECS** field is made of a repeating pair of fields that describe the length and text.

Table 6. WPLORECS field content

Offset (dec)	Offset (hex)	Length	Type	Field	Description
0	0	4	Unsigned	WPLRLEN	Length of WAPL output record.
4	4	Variable	Character	WPLRTEXT	WAPL output text.

Use **WPLIOADR** and **WPLIOSIZ** to get the control block data, then use **WPLOLINE** to know how many records to retrieve from within the data. The first 4 bytes of **WPLORECS** (the **WPLRLEN** field) describes how long the **WPLRTEXT** field is, and the offset to the next **WPLRLEN** field. Repeat this process for the number specified in the **WPLOLINE** field.

To call Workload Automation Programming Language from another load module and use the WPLI control block, you must include a parameter string when you call EQQWAPL. The parameter string must include the subsystem name and an **INPUT** keyword that includes the address of the WPLI control block. The address must be passed as an 8 byte hex address prefixed with 0x, for example `IWSC INPUT(0x0000CAA0)`.

Before calling EQQWAPL, the storage for the WPLI control block is obtained within the calling program code. The storage for the WPLO control block is obtained automatically during the running of EQQWAPL. After the control is returned to the calling program, the storage for the WPLI and WPLO control blocks is released within the calling program code.

Running Workload Automation Programming Language within an online TSO session

Workload Automation Programming Language is primarily designed for use in batch, but you can use it from within an online TSO session, either inside or outside of the HCL Workload Automation for Z dialogs.

To use Workload Automation Programming Language from within a TSO session, allocate the following files.

Note: Ensure that the appropriate level of the HCL Workload Automation for Z load library is in the execution path, either through the link list or a STEPLIB statement.

EQQMLIB

The HCL Workload Automation for Z message library. If Workload Automation Programming Language is being run from within HCL Workload Automation for Z dialogs, this file is already allocated.

EQQMLOG

The HCL Workload Automation for Z message log.

EQQOPTS

Optional file to provide environment default options. Allocate this file only if you want to set environment defaults before any commands that the process will run. For online performance reasons, to prevent additional dynamic allocations, it is more efficient to include any relevant statements in the main command stream.

To allocate the files, you can use one of the following ways:

- Within the TSO logon procedure.
- Within a startup CLIST or REXX, called as part of the TSO logon process.

- Within an application CLIST or REXX that starts the set of dialogs in which Workload Automation Programming Language is being called, using ALTLIB and LIBDEF for REXX and ISPF libraries. This process can also free the files as the set of dialogs is exited.
- In each individual REXX process that calls Workload Automation Programming Language, using ALTLIB and LIBDEF for REXX and ISPF libraries. This process can also free the files as the set of dialogs is exited.

Note: If you use mechanisms that free the files on exit, ensure that you do not design a process that would cause problems for other applications running in ISPF, in particular where split screen is needed.

Workload Automation Programming Language can then be called as an external EXEC from within calling REXX, with the following syntax:

```
CALL EQQYXTOP (<subsys> <options>)
waplRC = RESULT
```

When run within TSO, the INPUT stream is read from a file called INPUT. Workload Automation Programming Language does not read from a file called SYSIN to prevent the risk of stalling if SYSIN was left allocated to the terminal, which is the default position of SYSIN in the foreground. Alternatively, input can be placed on the REXX stack before calling Workload Automation Programming Language by passing an option of `INPUT(-OFF-)`; commands are read directly from the stack without having to allocate an input file to pass the commands.

The EQQJOBS process creates some examples in the EQQJOBS output file that you can use, and optionally customize, as a basis to run Workload Automation Programming Language online:

EQQWTSO1

Shows how to define a Workload Automation Programming Language environment, run a complete Workload Automation Programming Language process, and reset the environment in a single member.

EQQWTSO2

Shows how to run a complete Workload Automation Programming Language process in a single member, having the environment already set up.

EQQWTSO3

Shows a generic Workload Automation Programming Language execution member that defines a Workload Automation Programming Language environment, calls Workload Automation Programming Language with commands queued before this member is called, and then resets the environment without processing any output.

EQQWTSO4

Shows a generic Workload Automation Programming Language execution member that assumes that a Workload Automation Programming Language environment has already been defined, calls Workload Automation Programming Language with commands queued before this member is called, and then exits, leaving the calling member to process any output.

EQQWTSX3

Is an example of a process queueing commands to the stack before calling EQQWTSO3, which manages the Workload Automation Programming Language environment setup, and then processes the output when EQQWTSO3 completes.

EQQWTSX4

Is an example of a process queueing commands to the stack before calling EQQWTSO4, which assumes that the Workload Automation Programming Language environment is already set up, and then processes the output when EQQWTSO4 completes.

Running Workload Automation Programming Language on a started task workstation

You can run Workload Automation Programming Language on an HCL Workload Automation for Z started task workstation. In this way, you can code Workload Automation Programming Language commands within user fields, enabling commands to be issued without creating individual jobs.

To use this capability, you must create a started task workstation. For details about configuring a started task workstation, see *HCL Workload Scheduler for Z: Planning and Installation*.

The Started Task Workstation feature operates differently, according to the setting of the **OPCOPTS RCLEANUP** statement.

If you set **OPCOPTS RCLEANUP(YES)**:

- The started task JCL can be defined as a job that can call the started task.
- The installed **EQQYXJPX** procedure can be called directly from the library in which it is installed either by including it in the necessary concatenation, or by using a **JCLLIB** statement.
- Symbolic overrides can reference **GLOBAL** variables, allowing application tables to override them for individual jobs.

The following example shows an STC job for **RCLEANUP(YES)** (EQQJOBS member EQQWCMD1):

```
//EQQWCMD1 JOB CLASS=A,MSGCLASS=X
//*%OPC SCAN
//*********************************************************************
//* THIS SHOULD BE USED ON AN STC WORKSTATION TO RUN WAPL COMMANDS FOR
//* INSTALLATIONS WHERE OPCOPTS RCLEANUP(YES) IS USED.
//*
//* IT IS RECOMMENDED THAT THE JOBNAME BE PREFIXED BY THE SUBSYS NAME
//* E.G. TWSACMD1
//*
//* TO ENABLE USER WAPL MEMBERS TO BE INCLUDED FROM WITHIN USER FIELDS
//* ADD A DD STATEMENT POINTING TO YOUR OWN CODE LIBRARY E.G. USRCODE
//*
//*********************************************************************
//EQQYXJPX EXEC EQQYXJPX,
//         SUBSYS=TWSA
//EQQYLOG3 DD SYSOUT=*
//*USRCODE DD DISP=SHR,DSN=MY.USER.CODE
//SYSIN    DD DISP=SHR,DSN=TWS.V920.SEQQWAPL(USRSYSIN)
//EQQCPOP  DD *
```

```
&OADID. &OYMD1.&OHHMM. &OOPNO
```

If you set **OPCOPTS RCLEANUP(NO)**:

- Define the JCL as a procedure.
- Modify a copy of **EQQYXJPX**, with additional JCL statements appended for running as an STC operation.
- Specify **OPCOPTS VARPROC(YES)** to have the HCL Workload Automation for Z JCL variables resolved within a procedure.

Note: If VARPROC is not set to YES, be cautious when turning on this feature. Check all pre-existing JCL to search for any jobs that contain both //*%OPC SCAN and any instream procedures, because these might be affected by this change if the //*%OPC SCAN statement precedes the instream procedure, and symbolic parameters are coded within the procedure.

The following example shows additional statements to add to EQQYXJPX to create an STC job for **RCLEANUP(NO)** (EQQJOBS member EQQWCMD2):

```
//*********************************************************************
//* THIS SHOULD BE USED ON AN STC WORKSTATION TO RUN WAPL COMMANDS FOR
//* INSTALLATIONS WHERE OPCOPTS RCLEANUP(NO) IS USED.
//*
//* IT IS RECOMMENDED THAT THE JOBNAME BE PREFIXED BY THE SUBSYS NAME
//* E.G. TWSACMD1
//*
//* TO ENABLE USER WAPL MEMBERS TO BE INCLUDED FROM WITHIN USER FIELDS
//* ADD A DD STATEMENT POINTING TO YOUR OWN CODE LIBRARY E.G. USRCODE
//*
//*********************************************************************
//EQQYLOG3 DD SYSOUT=*
//*USRCODE DD DISP=SHR,DSN=MY.USER.CODE
//SYSIN    DD DISP=SHR,DSN=TWS.SEQQWAPL(USRSYSIN)
//*%OPC SCAN
//EQQCPOP  DD *
&OADID. &OYMD1.&OHHMM. &OOPNO
//         PEND
```

Regardless of how you set **RCLEANUP**:

- An EQQCPOP DD must be coded as shown to pass the occurrence ID to the process.
- An OPC SCAN statement is required to resolve the occurrence variables.
- The SYSIN can point to member USRSYSIN in the SEQQWAPL library, which contains command **INCLUDE USER_FIELD(EQQ-SYSIN-*)** that points Workload Automation Programming Language to the user fields of the submitting operation to find the commands to run.
- Consider adding an additional DD statement, with a name of your choice, that points to a library in which Workload Automation Programming Language INCLUDE members could be stored and referenced from within the commands that are run by this job. If you do this, ensure that either the library is used regularly, or an appropriate HSM

management class is assigned to avoid migration; an INCLUDE library that is not regularly used could become migrated and delay Workload Automation Programming Language commands.
- Consider adding an EQQYLOG3 DD statement, because `INCLUDE` processing means that the commands run from user fields will appear only at message level 3.

The commands can then be coded as user fields prefixed with EQQ-SYSIN- and are run in sort order.

There can be up to 100 user fields, some or all of which could be Workload Automation Programming Language statements, each up to 54 characters in length. Normal Workload Automation Programming Language continuation rules apply, therefore if a command does not fit within 54 characters it can continue in the next user field.

The following example shows commands coded in User Fields. This example in the first operation of an application will seek and HOLD or NOP any operations tagged with appropriate user fields and values.

```
 ---------------------------- OPERATION USER FIELDS ----------- Row 1 to 3 of 3
 Command ===>
Scroll ===> CSR
 Enter/Change data in the rows, and/or enter any of the following
 row commands:
 I(nn) - Insert, R(nn),RR(nn) - Repeat, D(nn),DD - Delete
 Application     : CMDDEMO                    Demonstrate CMD1 WS
 Operation       : CMD1 001
 Jobname         : WSLCCMD1
 Row  User Field Name  User Field Value
 cmd                   ----+----1----+----2----+----3----+----4----+----5----
 '''' EQQ-SYSIN-01     VARSUB SCAN(!)
 '''' EQQ-SYSIN-02     HOLD ADID(!OADID.) IA(!OYMD1.!OHHMM.) USRF(HOLD=YES)
 '''' EQQ-SYSIN-03     NOP ADID(!OADID.) IA(!OYMD1.!OHHMM.) USRF(NOP=YES)
 ****************************** Bottom of data ********************************
```

Because a started task is not limited to a single occurrence of the same name running simultaneously, you define only a single Workload Automation Programming Language started task, which can run many times in parallel. It is recommended that you use a special resource exclusively for any Workload Automation Programming Language started task operations, to limit the number of processes you want to allow in parallel.

Running Workload Automation Programming Language as a console command

You can create custom versions of the EQQYXJPX procedure, for each controller you have, that enable simple Workload Automation Programming Language commands to be run from the console or automation products.

You do this by using a special entry point value of CMD called EQQYXSTC, which takes the ARGS value and runs it as a command, allowing a single command to be entered at the console. The custom version would be similar to the normal EQQYXJPX, but would be specifically customized for the controller values in the PROC statements and the CMD symbolic would default to EQQYXSTC to call the special entry point.

The following example shows a custom procedure for a controller called WSLC to run a single command from the console. Ensure that:

- The data sets in the STEPLIB, SYSPROC, and EQQMLIB DD statements reference the appropriate controller version.
- You set a EQQYPARM DD statement, to ensure that the command reaches the controller regardless of the LPAR from which it is submitted.

```
//WSLCEXEC PROC @=,
//         ARGS='',
//         CMD=EQQYXSTC,
//         REG=4M,
//         SUBSYS=WSLC,
//         VER=V920,
//         #=
//EQQYXTOP EXEC PGM=IKJEFT01,
//         REGION=&REG,
//         PARM=&@'&CMD &SUBSYS-&VER &ARGS'&#
//STEPLIB  DD DISP=SHR,DSN=ZWS.DA.V9R5.SEQQLMD0
//SYSPROC  DD DISP=SHR,DSN=ZWS.DA.V9R5.SEQQMISC
//EQQMLIB  DD DISP=SHR,DSN=ZWS.DA.V9R5.SEQQMSG0
//EQQYPARM DD DISP=SHR,DSN=DEMO.&SUBSYS..CONFIG.PARM(YPARM)
//EQQMLOG  DD SYSOUT=*
//EQQDUMP  DD SYSOUT=*
//EQQSMTP  DD SYSOUT=(B,SMTP
//SYSTSPRT DD SYSOUT=*
//SYSTSIN  DD DUMMY
```

Provided that the customized procedure is located in the correct library for started tasks, you can run a console command such as `S WSLCEXEC,ARGS='ADD ADID(MYAPPL)'`.

Note: There is a restriction for this method that the commands to be issued cannot contain single quotation marks. Single quotation marks are needed only in limited circumstances with Workload Automation Programming Language, limiting the impact of this restriction.

For triggering multiple commands from the console, provided that they can be predefined, you can add a DD statement, for example MYCODE, within the custom procedure to contain predefined groups of statements, which can then be called as follows:

```
S WSLCEXEC,ARGS='INCLUDE MYCODE(WHATEVER)'
```

This would include and execute a member called WHATEVER from the MYCODE DD statement within the customer procedure.

With some automation products it is possible to define command rules to look for an automation-specific prefix to then translate a short form of the command into a started task call. For example, if you enter `!WSLC ADD ADID(MYAPPL)` automation rules translate it to `S WSLCEXEC,ARGS='ADD ADID(MYAPPL)'`.

Specifying the subsystem

To run Workload Automation Programming Language both batch and online, you must set the `SUBSYS` parameter to specify the subsystem used when the communication with HCL Workload Automation for Z was initialized.

When Workload Automation Programming Language finds a command that requires communication with the HCL Workload Automation for Z PIF, it automatically generates an `INIT` statement.

Note: Some **OPTIONS** keywords require communication with the HCL Workload Automation for Z PIF.

To communicate with more than one HCL Workload Automation for Z subsystem in a Workload Automation Programming Language session, you can initialize your own connection to HCL Workload Automation for Z by using an INIT statement before any other statement that requires the PIF.

SUBSYS is a required positional parameter, so even if you are specifying the subsystem explicitly using **INIT** statements, you must provide a value. The default in EQQYXJPX and EQQYXTSO is set to an asterisk (*) for this purpose.

You can also use the **SUBSYS** parameter to specify the default version to load at startup.

Using OUTPUT statements

Use the **OUTPUT** statements to define what information you want to extract from HCL Workload Automation for Z, where to send it, and in what format. Workload Automation Programming Language writes the output from **LIST** or **SELECT** requests *only* if you have run **OUTPUT** statements for the segments encountered in these commands.

Workload Automation Programming Language provides you with many predefined **OUTPUT** statements. They are available either in external members that you can include with the **INCLUDE** command, or through in-built statements that you can specify with the **LOADDEF** command.

The **OUTPUT** statements are located in the SEQQWAPL library and consist of the following types of members:

Segment members

There is a member for every HCL Workload Automation for Z segment supported by Workload Automation Programming Language. The member names match the segment names.

Record members

There is a member for every HCL Workload Automation for Z record supported by Workload Automation Programming Language, that contains each segment within that record. The member names match the record names.

Group members

There are members for large groupings of records such as all Database Records, all Current Plan records, and so on. The member names for these members begin with EQQGRP.

There are also some members in the SEQQWAPL library whose names begin with EQQFL*xxxx*. These members are designed to help you use Workload Automation Programming Language as a command line to HCL Workload Automation for Z, without having to consider in detail what **OUTPUT** statements you need. They are:

EQQFLALL

Includes definitions for all HCL Workload Automation for Z data obtained by Program Interface commands (for example, SELECT and LIST). This does *not* include all the tracking log records. EQQFLALL is a historical name chosen before Tracking Log records were considered; adding all tracking log records into EQQFLALL could result in slower start times for existing Workload Automation Programming Language jobs.

EQQFLDB

Includes all HCL Workload Automation for Z database records.

EQQFLCP

Includes all HCL Workload Automation for Z current plan records.

EQQFLLTP

Includes all HCL Workload Automation for Z long term plan records.

EQQFLPLN

Includes all HCL Workload Automation for Z plan records (CP and LTP).

EQQFLSYS

Includes all HCL Workload Automation for Z system records.

Many `EQQFLxxx` and `EQQGRPxxx` members contain the same **`OUTPUT`** definitions. The `EQQFLxxx` members contain also statements to escalate the severity of two messages that would indicate whether you have referenced a segment or field not supported by your current version of HCL Workload Automation for Z. By default, these messages are advisory and do not set a return code. The `FILE` members cause RC=4 if the **`OUTPUT`** statements following the content of the supplied member references something that is not supported. The two different approaches allow for unsupported fields to be ignored, for example, if running common code against current and earlier versions of HCL Workload Automation for Z.

Individual segment, record, or group members can still be used with error checking in place, either by adding the two **`SETSEV`** statements from the `EQQFLxxxx` members into your site defaults (which will turn on checking for ALL **`OUTPUT`** statements including the supplied ones) or by coding the statements in your SYSIN to issue only RC=4 for invalid references in the statements that you create.

The in-built **`OUTPUT`** definitions include every field, for every segment, that sends ILSON data to OUTDATA, and Batch Loader data to OUTBL. You can load these definitions by using the **`LOADDEF`** statement. For example, set **`LOADDEF AD*`** to load the in-built **`OUTPUT`** definitions for all the segments beginning with AD. If instead of exploiting the in-built versions, you want to load the prebuilt members from SEQQWAPL, use the **`INCLUDE`** statement.

The SEQQWAPL library must be allocated in the JCL, then an **`INCLUDE`** statement can load the relevant member. For example, set **`INCLUDE WAPLMAPS(EQQADW)`** to load the member that contains all of the **`OUTPUT`** statements for application description segments. Load the in-built **`OUTPUT`** statements only when you need all the fields for a particular segment, for example when extracting batch loader for database objects. If you do not need every field from each segment in your output, define your own **`OUTPUT`** statements. For a list of all available segment and field names, refer to the SEQQWAPL member EQQFLALL described in Alternative resource names on page 369

Workload Automation Programming Language commands syntax

The general syntax for all Workload Automation Programming Language commands is **`COMMAND [RESOURCE] KEYWORD(value) KEYWORD(value)`**

Consider that:

- Some commands require a resource which is either the type of item, or a single item upon which the command is issued. For these command,s this is always the second word of the command statement.
- Specify the values of a command within parentheses. Do no insert a blank between the keyword and the opening parenthesis, because this is interpreted as a separate keyword.
- Keywords and their values can be separated from other keywords and their values by commas, blanks, or both.
- If a keyword value requires parentheses, that value must be enclosed within single quotation marks inside the keyword parentheses. For example, `KEYWORD('my.value(member)')`. In all other cases, the containing quotation marks are optional.
- Containing quotation marks are not passed through to the underlying process.
- If containing quotation marks are used, you must specify two consecutive single quotation marks to represent a single quotation mark in the underlying process.
- A single quotation mark at the start of a value is considered to be a containing quotation mark, and a terminating quotation mark is required at the end of the value.
- Double quotation marks are always passed through to the underlying process.

Using comparators

When you need to specify comparators for a command (for example **`LIST`**) specify them in the form **`KEYWORD-COMPARATOR(value)`**. For example, **`VALFROM-GE(060124)`**

If no comparator is specified, **`EQ`** is assumed. If the PIF command does not support comparators, they are ignored.

The following list shows the valid comparators:

EQ

Equal to

NE

Not equal to

LT

Less than

GT

Greater than

LE

Less than or equal to

GE

Greater than or equal to

The native PIF convention for specifying comparators is to specify the comparator at the end of the field, and the comparison works counter intuitively. For example, **`VALFROM=060124<=`** means when 060124 is less than or equal to the value of **`VALFROM`** in the application in the database NOT when **`VALFROM`** in the database is less than or equal to 060124.

The Workload Automation Programming Language notation works the more intuitive way, therefore `VALFROM-GE(060124)` means list applications whose VALID FROM date is greater than or equal to 060124.

Hence, `VALFROM-GE(060124)` translates to `VALFROM=060124<=` in native PIF.

You can still use the native PIF approach to specify comparators, by appending the comparator onto the value inside the keyword parentheses, as follows:

```
VALFROM(060124<=)
```

Note: Comparators are valid only for native PIF commands `LIST`, `SELECT`, and `DELETE`. They are not valid for keywords of any other command.

Setting dates and times

Some keywords allow you to enter dates, times, or date and times. To use the current date, time, or datetime, you can enter the equal sign (=) as the *only* content of the keyword.

The following restrictions apply to the use of the equal sign (=) for dates and times:

- It can be used only for fields whose value is specifically a date, time, or datetime. You cannot use it for any other type of field, `DESCR(=)` is not a date or time short form, `DESCR(=)` will result in the description being set to a single equals sign.
- It cannot form a part of the data. For example, `IA(=)` is allowed, while `IA(060124=)` is not allowed.
- Do *not* use the equal sign (=) for field values in conjunction with native PIF comparators. For example, `IA-GE(=)` is allowed, while `IA(=<=)` is not allowed.

The value used must be appropriate to whatever the date and time was when Workload Automation Programming Language started. You can specify your own values to substitute for the equal sign (=) by using `OPTIONS DATE` and `OPTIONS TIME`.

You can also use the plus sign (+) or minus sign (–) in date or time fields to set dates or time relative to the Workload Automation Programming Language internal date and time. For example, `VALFROM(+7)` will be a date in 7 days time, `STARTTIME(+10)` will be in ten minutes time.

If you use the plus sign (+) or minus sign (–) in a time field, this might result in *carry over* to other fields if the addition or subtraction crosses a date boundary. Be aware that in some cases this can create an invalid value for some fields. The following fields perform automatic carryover:

- `OPTIONS TIME` will carry over into `OPTIONS DATE`
- Long Term Plan `IAT` will carry over into `IAD`
- Long Term Plan `PREIAT` will carry over into `PREIAD`
- Batch Loader `ADOP STARTTIME` will carry over into `STARTDAY`
- Batch Loader `ADOP DLTIME` will carry over into `DLDAY`

Note: For consistency, the equal sign (=) short form uses the date and time when you started your Workload Automation Programming Language session. This ensures that you can use the equal sign (=) throughout your command statements with consistency, preventing potential problems with dependencies being missed. The exception to this is the `@` function, which uses the input arrival of the occurrence in which the Workload Automation Programming Language job is running, if the job is under HCL Workload Automation for Z control.

For dates, the plus sign (+) or minus sign (–) adds or subtracts a number of calendar days to the date. To add or subtract work days, you can use the WD suffix. For example, `ADD ADID(MYAPPL) IADATE(+3WD)` adds the application to the plan 3 working days from the current date.

When the job is being controlled by HCL Workload Automation for Z, the calendar used by the calculation is the application calendar; otherwise the default calendar, that you can set with `OPTIONS CALENDAR`, is used.

The calendar can be specified within the keyword by following the date offset with a slash (/) and the calendar name, for example `ADD ADID(MYAPPL) IADATE(+3WD/MYCAL)`.

Termination, line numbers, and continuation

There is no termination character for a command statement. A command is terminated when a new command word is identified as the first word on an input line, or the end of an input stream is encountered. Continuation is automatic; no continuation character is needed.

Any line ending in a comma is continued onto the next line with no intervening blank, any other ending character results in an intervening blank being considered between the adjoined lines.

If a single keyword needs to be longer than an input line, then continuing that statement up to the last column of available input (for example, column 80 for instream SYSIN) results in the following line being directly abutted to the end of the preceding line.

If line numbers were included in the input statements, column 72 is considered as the last available input column. Workload Automation Programming Language considers only line numbers to be set if *every* record of the input stream has *only* numeric digits in columns 73 to 80. If any records have blanks or characters other than 0 to 9 in columns 73-80, the content of columns 73-80 is considered valid input for every record. If any record length other than 80 bytes is used for input, Workload Automation Programming Language does not consider line numbers to be set.

For compatibility with EQQYCAIN and EQQYLTOP batch loader format, you can set `OPTIONS SYNTAX(LEGACY)` , which will ignore anything beyond column 72 and consider 72 as the continuation column, connecting the following line from column 1.

Using termination characters such as semicolon (;) or period (.) is not accepted by Workload Automation Programming Language.

Individual sub-segments for Batch Loader statements do not have to start on a new line, only the `xxSTART` statement needs to start on a new line. For example, `ADSTART ADID(MYAPPL) ADOP OPNO(001) JOBN(MYJOB)`

With a long enough SYSIN record length it would be possible to contain the entire Batch Loader for a single HCL Workload Automation for Z object on a single line of input.

Inserting comments in a statement

You can specify comments by inserting lines that start with `/*` and terminate with `*/`

Comments can use multiple lines, or be included anywhere within a statement. Unless the comment is started at column 1, the comment start must be preceded by at least one blank character between it, and the text it follows. For example:

```
/* ---------------------------------------------+
 | This is a multi line comment                 |
 +---------------------------------------------*/
```

```
COMMAND RESOURCE KEYWORD(VAL) /* TAIL COMMENT */
```

```
COMMAND RESOURCE /* EMBEDDED COMMENT */ KEYWORD(/*)
```

Note: Because `/* in the KEYWORD(/*)` is not preceded by a space, it is treated as the value for the keyword not as the start of a comment.

Multi-line comments cannot span a text mode block of data, as in the following example:

```
JSSTART ADID('CRITBATCH       ') IA(0805121738) OPNO(001)
JSJCL DLM(-END-OF-INPUT-TEXT-) /* Stop JCL
//*>OPC SCAN
//*>OPC TABLE NAME=TESTVAR
//* Y                          */
-END-OF-INPUT-TEXT-
```

Because the DLM keyword indicates the following line is text it is contradictory to the open comment leading into the text. Text data in Workload Automation Programming Language is treated as unprocessed input, so any comments within the text are passed into the object it is defining. So in this case the open comment will lead to a syntax error.

To comment out the data entirely, the following example shows the correct form:

```
JSSTART ADID('CRITBATCH       ') IA(0805121738) OPNO(001)
JSJCL MEMBER(MYJOB) /* DLM(-END-OF-INPUT-TEXT-)
//*>OPC SCAN
//*>OPC TABLE NAME=TESTVAR
//* Y
-END-OF-INPUT-TEXT-          */
```

Note: The comment start characters override all other directives, so if you want to include `/*` within one of your statements with a preceding space, then you must alter the comment start character using the **`OPTIONS COMSTART`** statement and similarly with **`OPTIONS COMEND`** to use something other than `*/` to end a comment. Consider that changing the default comment characters can affect the use of predefined members such as EQQFLALL, which contains many comments in the `/* */` format.

The `/*` value *can* be included within a statement, provided that is *not* preceded by a blank character.

For example, the OK variable will work, the NOTOK variable will be treated as comment from column 33 onwards:

```
JCLVVAR VARNAME(OK) DEFAULT('/*') SETUP(N)
JCLVVAR VARNAME(NOTOK) DEFAULT(' /*') SETUP(N)
```

Special resource and user field names

With HCL Workload Automation for Z, the names of special resources and user fields can contain characters different from alphanumeric and national characters, but this would cause problems when parsing in Workload Automation Programming Language.

Consider that:

- Quotes, blanks, and parentheses `(` and `)` can cause problems to read the syntax of a command correctly.
- When using Workload Automation Programming Language with instream SYSIN, characters `&`, `%`, and `?` could be resolved by JCL tailoring unintentionally.
- The exclamation point (`!`) could be resolved by Workload Automation Programming Language variable substitution unintentionally.
- The percent sign (`%`) and asterisk (`*`) could cause problems to commands that need to use wildcards to search.
- `/*` and `*/` could be interpreted as a comment by Workload Automation Programming Language.

These characters can cause problems especially in object names that form searchable and key fields. You should avoid them when naming objects to be used with Workload Automation Programming Language.

Special characters @, !, and

The characters at sign (@), exclamation mark (!), and number sign (#) have special usage in the Workload Automation Programming Language syntax.

`at sign (@)`

Used to identify object variables and to generate email addresses that have no qualifying domain name.

`exclamation mark (!)`

The default value for variable prefixing when **`VARSUB SCAN`** is activated.

`number sign (#)`

Used in object variables to prefix count fields to indicate the number of segments of a particular type. It is also used as a count indicator for **`VARSET ENVATTR`**.

However, these characters can be displayed differently when different code pages for different countries are used. They appear exactly as documented for US English (CP 37) and United Kingdom (CP 285); instead, some or all of them might appear differently in code pages for other languages. To find out what the characters actually are for your code page, run the **`SHOW OPTIONS`** command.

The following example shows the output of the **`SHOW OPTIONS`** command using a non-English code page (CP 280):

```
03/07 23.10.25 EQQI200I SHOW OPTIONS
03/07 23.10.25 EQQI602A OPTIONS in effect ADOICHK(N) ADPFX(ADHOC#) ADSFX()
            ...EQQI602A ADVALFROM(A) ADVERS(Y) ADWS(CPU1) ARGUMENT()
            ...EQQI602A BLSTYLE(TWS) CALENDAR(DEFAULT) CCREMOVE(A) CHARAT(§)
            ...EQQI602A CHARBANG(é) CHARHASH(£) CHARMAIL(§) CHECK(Y) COMEND(*/)
            ...EQQI602A COMMIT(1000) COMPSUCC(WARNING) COMSTART(/*) CONINFO(047
            ...EQQI602A 059 062 063 064 129 130 136 144 185 252 254 255)
            ...EQQI602A CONNAME(ADCDMST1) CONTENTION(30,10) CONWAIT(2,1)
            ...EQQI602A CONWARN(138 341) CPDEPR(Y) CPFAIL(ABORT) DATA()
            ...EQQI602A DATE(150307) DBMODE(ADD) DECODE(ONLY) DELAY(0)
            ...EQQI602A DELAYCMD(DELETE EXECUTE INSERT REPLACE) DELETE(N)
            ...EQQI602A DELFILE(OUTDEL) DLM(-END-OF-INPUT-TEXT-) DROP()
            ...EQQI602A DUPAUTO(Y) DURSEC(Y) DURUNIT(SECONDS)
```

The following **OPTIONS** keywords are available to set the characters to the value you want to use in your code page:

CHARAT

Sets a character to replace the at sign (@) for object variables.

CHARBANG

Sets a character to replace the exclamation mark (!) for the default variable prefix.

CHARHASH

Sets a character to replace the number sign (#) for object and **ENVATTR** count values.

CHARMAIL

Sets a symbol to replace the at sign (@) used in email addresses.

CHARAT and **CHARMAIL** are separate keywords, because the at sign (@) for email addresses must match what is used by the SMTP service, which is external to Workload Automation Programming Language. The at sign (@) in object variables affects only Workload Automation Programming Language.

For these **OPTIONS** you cannot use standard upper or lower case alphabetic characters, numbers, minus signs (-), or periods (.). They must not be in conflict with any other **CHARxxxx** or **VARNAMES** keyword. Setting **OPTIONS CHARAT(@) CHARBANG(!) CHARHASH(#) CHARMAIL(@)** in your code page ensures that the special characters appear as documented in your system.

Note: These **OPTIONS** keywords change only these characters for the uses specified. When the same characters are used as part of the data in your system, or part of field names in **OUTPUT** statements or object variables, the characters are displayed according to your code page.

The impact of code pages on special characters

When code pages are used the core characters A to Z and 0 to 9 are displayed as they are, while special characters, for example the at sign (@), are mapped to characters that depend on the code page applied. This means that a program compiled and documented in a native code page will have some characters represented in a different way when it is used with another code page.

Table 7: Special characters in different code pages on page 53 shows how the at sign (@), exclamation mark (!), and number sign (#) are represented in different code pages. Hence when WAPL is used with the German code page 1141, a line of code referring to a variable would display as follows:

```
DISPLAY “Application ÜOADID”
```

Table 7. Special characters in different code pages

Character (US and UK code pages)	EBCDIC code (US and UK code pages)	OPTIONS keyword	German code page (1141)	Sweden code page (1143)	Italian code page (1144)	Spanish code page (1145)	French code page (1147)
At sign (@)	7C	CHARAT	§	Ö	§	@	à
Exclamation mark (!)	5A	CHARBANG	Ü	€	é	]	§
Number sign (#)	7B	CHARHASH	#	Ä	£	Ñ	£

To see how each character is displayed in your local code page, issue the **SHOW OPTIONS** command.

Regardless of the code page, you can reassign the symbol to use for the at sign (@), exclamation mark (!), and number sign (#) by using the commands **OPTIONS CHARAT**, **OPTIONS CHARBANG**, and **OPTIONS CHARHASH** respectively. These commands enable you to choose the character to display, sometimes even the same character as the one used in the native code page.

For example, issuing the command **OPTIONS CHARAT(@)** with the German code page 1141 would reassign the character code 5B (which in code page 1141 corresponds to @) to the character that WAPL uses as @ in its native code page.

Table 8: Characters whose symbol cannot be overridden on page 53 shows the characters for which the commands **OPTIONS CHARAT**, **OPTIONS CHARBANG**, and **OPTIONS CHARHASH** cannot be used, because these are reserved characters in the REXX compiler.

Table 8. Characters whose symbol cannot be overridden

Character	Code (US and UK code pages)	Purpose in REXX
space	40	Space, key text delimiter
(	4D	Open parenthesis
+	4E	Mathematical operator
|	4F	REXX string concatenation character and Boolean OR
&	50	REXX Boolean AND
$	5B	Used for internal purposes
)	5D	Close parenthesis
¬	5F	Boolean not

Table 8. Characters whose symbol cannot be overridden (continued)

Character	Code (US and UK code pages)	Purpose in REXX
_	6D	Used for internal purposes
'	7D	Single quote
=	7E	Mathematical operator
"	7F	Double quote
A to I	C1 to C9	Alphabetic characters
J to R	D1 to D9	Alphabetic characters
\	E0	Boolean NOT
S to Z	E2 to E9	Alphabetic characters
0 to 9	F0 to F9	Numeric characters

For example, in the German code page 1141 the exclamation mark (!) is assigned to EBCDIC code 4F which REXX already uses for both text string concatenation and the `OR` Boolean operator. Therefore, the command `OPTIONS CHARBANG(!)` cannot be issued with this code page.

Other symbols not listed in Table 8: Characters whose symbol cannot be overridden on page 53 might be incompatible with REXX expressions that are interpreted by WAPL statements. If you find these characters, consider to write the expression without them, or reassign the commands `OPTIONS CHARAT`, `OPTIONS CHARBANG`, and `OPTIONS CHARHASH` before the impacted line, to avoid the incompatibility.

Knowing resource names

You must know what resources you need, because the resource names are not always obvious from the modern name of the object. For example the resource needed to retrieve a job stream is AD, which is derived from the old name of Application Description.

In addition, sometimes you need use a different resource name when using `LIST` to the one you must use when using `SELECT`.

For example, you would `LIST ADCOM` to find a list of job streams, but then you would need to `SELECT AD` to retrieve each job stream completely.

Workload Automation Programming Language allows alternative, more obvious names to be used; therefore you can now `LIST JOBSTREAM` to get a list of job streams and then `SELECT JOBSTREAM` to retrieve a complete individual definition. To retrieve only the common segment of a job stream, you can still `SELECT ADCOM` just as before.

For a complete list of the available aliases for each PIF command, see Alternative resource names on page 369.

Variable substitution

Workload Automation Programming Language supports variable substitution within its control statements. In this way, the actual values required for each command run are provided (for example, the occurrence name or Input Arrival date).

To activate the variable substitution, use the command **VARSUB SCAN**. By default, the prefix to identify the variables within the command text is the exclamation mark (!). Optionally, you can use a period (.) to terminate the variable name. Because the exclamation mark can be encoded as different character numbers within some code pages, it is recommended that you use **VARSUB SCAN(!)** to guarantee the use of the exclamation mark that corresponds to your code page.

For example, Workload Automation Programming Language can update the operation text of the running job to indicate something about the processing that day, without having to code the SYSIN as instream data in the JCL to use HCL Workload Automation for Z variables:

```
VARSUB SCAN(!)
MODIFY CPOC ADID(!OADID.) IA(!OYMD1.!OHHMM.)
MODIFY CPOP OPNO(!OOPNO) DESC('NO DATA TODAY')
```

This also means that HCL Workload Automation for Z information becomes available to jobs tracked by HCL Workload Automation for Z that were not necessarily submitted by HCL Workload Automation for Z.

For more detailed information about variables, see Variable naming convention on page 323.

Using wildcards

Workload Automation Programming Language supports the use of wildcards in many commands, either partially or completely.

Wildcards are *completely* supported for the standard keywords of the Data Access commands based on PIF, and for the PIF related keywords of the Current Plan Operation commands. In both cases, you can use wildcards as in the HCL Workload Automation for Z PIF commands:

Percentage sign (%)

Represents any single character.

Asterisk (*)

Represents one or more characters.

You can use more than one wildcard in the command. For example:

ABC*

Matches values beginning with ABC.

***XYZ**

Matches values ending with XYZ.

ABC%%%XYZ

Matches values beginning with ABC, followed by 3 characters of any value and ending with XYZ.

`%%%%DEF*`

Matches any values starting with DEF at the fifth position in the string.

Wildcards are *partially* supported for non-PIF keywords within Workload Automation Programming Language where wildcard support is explicitly stated. In this case, you can use wildcards as in the HCL Workload Automation for Z PIF commands:

Percentage sign (%)

Represents any single character.

Asterisk (*)

Represents one or more characters.

Only the percentage sign (%) can be used multiple times. If the asterisk (*) is used more than once, any subsequent asterisk will be treated as an asterisk (*) and not a wildcard. For example:

`ABC*`

Matches values beginning with ABC.

`*XYZ`

Matches values ending with XYZ.

`ABC%%%XYZ`

Matches values beginning with ABC, followed by 3 characters of any value and ending with XYZ.

`%%%%DEF*`

Matches any values starting with DEF at the fifth position in the string.

Controlling the processing within Workload Automation Programming Language

Workload Automation Programming Language runs each command in sequence, running them from each of the input streams in the order that they are.

If a critical error occurs, no more commands are run except an automatic **`TERM`** command, if needed. A critical error is any statement that issues a return code of 12 or above. This limit can be changed with the **`OPTIONS STOPRC`** keyword.

The **`LISTSTAT`** command can issue a response code greater than 12 without causing a problem because the response code from **`LISTSTAT`** is only applied to the return code of the step immediately prior to termination of Workload Automation Programming Language. A response code is a special case of a return code to avoid **`STOPRC`** processing.

For conditional processing when comparing against a particular step return code, the highest value of the return code and response code is considered.

You can influence which statements are run within Workload Automation Programming Language by using a series of process control tags. These tags all begin with a colon and can be coded anywhere within a Workload Automation Programming Language statement following the initial command name.

Labeling Workload Automation Programming Language statements

To control the flow of the Workload Automation Programming Language statements, or make key statements easier to identify in the job output, you can label individual Workload Automation Programming Language statements.

You label the statement by inserting a user-defined label before the statement, followed by a colon:

```
CHKJOB: LISTSTAT CPOP JOBNAME(WLSCCPEX) IA(&OYMD1.1200)
```

The label can be up to 16 alphanumeric characters, excluded the underscore character. The label must begin with an alphabetic character to avoid clashing with internal Workload Automation Programming Language labels, which are 8-character numeric fields.

The label allows other conditional processing functions to reference this statement directly, for example, **`IF @CMD(CHKJOB.EQ.4) THEN`**.

The label also allows you to reference a command directly, as long as it is in scope, with the **`GOTO`** command or run it as a subroutine with the **`CALL`** command.

The label assigned to each statement is shown in the messages 200 and 299 that record the running of each command, and can also be displayed by issuing the **`SHOW RC`** command.

The label does not need to be on the same line as the command to which it belongs, but each label must always be followed by a command. You cannot have two consecutive labels without a command associated to each. If you need two labels to the same location, add the **`NOACT`** statement after the first label, then add the actual statement after the second label, as shown in the following example:

```
ENTRY_1: NOACT
ENTRY_2": DISPLAY "Good morning"
```

Defining subroutines

Define subroutines to allow chunks of code to be reused from different parts of the main program. Subroutines are called by the **`CALL SUB`** command.

You define subroutines by using the **`SUBROUTINE`** statement, which must be coded at the end of the program stream from which they are being called. **`SUBROUTINE`** statements must not be contained within **`INCLUDE`** statements, but subroutines can contain **`INCLUDE`** statements.

A subroutine must start with **`SUBROUTINE`** and end with **`RETURN`**, but a **`RETURN`** statement is assumed when the next **`SUBROUTINE`** statement or the end of the stream of code is encountered. You can use **`IF`**-**`THEN`**-**`ELSE`** to issue a **`RETURN`** statement to conditionally exit a subroutine.

Protecting against PIF failure

Because Workload Automation Programming Language uses the HCL Workload Automation for Z programming interface (PIF), it is vulnerable to a few of types of problem against which you can protect.

The controller must be available for any job to use the PIF. If you run a Workload Automation Programming Language job while the controller is down, any PIF processing fails. Because occasionally the controller might fail and cause a Workload Automation Programming Language job to fail, for planned controller downtime you can protect against failure with the following procedure:

1. Ensure that all your production Workload Automation Programming Language jobs are controlled by HCL Workload Automation for Z. For Workload Automation Programming Language jobs that are submitted externally, ensure that they are submitted with TYPRUN=HOLD and have a corresponding ETT rule to trigger an application to control them. This ensures that no Workload Automation Programming Language job starts running while the controller is unavailable. Any job submitted while the controller is down will wait, and, providing that Event Data sets are adequately sized, will be released automatically after the controller is ready. Any Workload Automation Programming Language jobs that fail due to unexpected failure of the controller are seen in error when the controller is restarted. This method requires the EWTROPTS option HOLDJOB keyword setting to USER.
2. Create a special resource specifically for use of the programming interface. Add this resource to every Workload Automation Programming Language job, planned and event-triggered, with a usage of SHR. For an unplanned but controlled shutdown of the controller, you can then make this resource unavailable and monitor the In-Use list (5.7) for this resource to determine when it is safe to take the controller down. On restart of the controller, after the resource is made available again, Workload Automation Programming Language processing can continue safely.

Some HCL Workload Automation for Z planning functions make heavy use of the current plan and long-term plan. These can cause failure due to contention, or logical failure with elements being changed around the Workload Automation Programming Language job. It is recommended that you avoid running Workload Automation Programming Language jobs against the Current Plan Extend, Replan, Long-Term Plan Extend, and Modify. This can be done by adding the programming interface resource to these planning jobs with Exclusive usage.

Overriding Workload Automation Programming Language defaults

You can change the Workload Automation Programming Language defaults for many elements by setting the **`OPTIONS`** statement.

If you want to set different values as default, create entries in your own site default member and store it in one of your libraries, then reference it in the **`EQQOPTS`** statement within the **`EQQYXJPX`** and **`EQQYXJPL`** procedures. By default, the **`EQQOPTS`** statement is provided commented out, so nothing is read.

Any statements defined in the **`EQQOPTS`** statement are run every time Workload Automation Programming Language starts, before the **`ARGS`** statement is processed and before any SYSIN or REXX stack commands are run.

You can also create subsystem-specific **`OPTIONS`** by using `IF` statements and the **`ZSUBSYS`** variable.

Including a site default member in **`EQQOPTS`** means that it is run every time Workload Automation Programming Language starts. To prevent unnecessary storage consumption, include only the statements that are actually required every time Workload Automation Programming Language runs. In alternative to creating a site default member, you can create members that set site defaults and variables for key types of functions. Then **`INCLUDE`** only the Workload Automation

Programming Language jobs that need those setup instructions. For example, create an include member called TRACKERS as follows:

```
OPTIONS TRACKERS(WSMC.*.WSMT)
```

Then, include the TRACKERS member in any job that issues TSO commands:

```
INCLUDE MYWAPL(TRACKERS)
SRSTAT 'MY.RESOURCE' AVAIL(Y)
```

Message log

Workload Automation Programming Language produces a message log, with different levels of detail, according to the **OPTIONS MSGLEVEL** setting.

If a DD statement called **EQQYLOG** is specified in the JCL, Workload Automation Programming Language messages are directed there, with a default **MSGLEVEL** of 0. If a numeric suffix is added to the **EQQYLOG**, this determines the **MSGLEVEL** setting, if a number greater than 5 is used the **MSGLEVEL** are set to 5. For example, **//EQQYLOG1 DD SYSOUT=*** diverts all Workload Automation Programming Language messages to this DD statement with a **MSGLEVEL** of 1.

EQQYLOG* DD statements that define data sets not valid for output are ignored. If more than one valid variant of **EQQYLOG*** is specified, the last encountered in the JCL is used, allowing an **EQQYLOG*** DD statement to be coded in the **EQQYXJPX** and **EQQYXJPL** procedures, but still be overridden in the executing JCL, even if using a different **MSGLEVEL** suffix.

DUMMY DD statements are ignored. If no valid **EQQYLOG*** data sets are found in the JCL, **SYSTSPRT** is used. If writing to an **EQQYLOG*** data set fails, **SYSTSPRT** is used for the remainder of messages.

Output from the **DISPLAY** command is always directed to **SYSTSPRT**.

Chapter 3. Core programming commands

The following sections describe the Workload Automation Programming Language commands that control the behavior and logic of the program. Use the commands to set options, return codes, determine an output, modify the flow, change variables, or make decisions.

CALL – Execute external program, subroutine, or variable

Use the **CALL** command to call routines that might be in another REXX routine or variable.

You can specify:

```
CALL EXT(<external REXX>) [STACK(IN|OUT|BOTH|NONE)]
```

or

```
CALL SUB(<subroutine name>)
```

or

```
CALL VAR(<variable name>)
```

where:

EXT

Specifies an external REXX routine to execute, including arguments if required. If the external command sets a return code, this will be passed to the return code of the **CALL** command.

STACK

Defines what to do with the REXX stack around the running of an external program:

IN

Default. Leaves the stack intact for the external program to use. If the external program does not pull entries from the stack they are left in place for the remainder of the calling program to access or add to.

OUT

Preserves the stack before calling the external program, then once the external program completes, processes any new stack entries on the stack as Workload Automation Programming Language commands, before restoring the original contents of the stack. Workload Automation Programming Language commands from the stack is displayed at **MSGLEVEL(2)**.

BOTH

Passes the stack to the external program to use and then processes the entire content of the stack of Workload Automation Programming Language commands when the external program completes. If the input stack was not completely processed by the external command, any remainder is executed on exit. Workload Automation Programming Language commands from the stack is displayed at **MSGLEVEL(2)**.

NONE

Keeps the stack before calling the external program and then restores the stack ahead of any new entries added by the external program.

SUB

Specifies a subroutine to run immediately following the **CALL** command. The **CALL** command fails if the subroutine does not exist.

VAR

Specifies a variable that contains a complete Workload Automation Programming Language command to run immediately following the **CALL** command. For batch loader this must be the full construct for all segments. The **VAR** keyword causes the command contained in the variable to be added to the execution stream to execute following this command. The return code of the **CALL** command indicates only the success of finding the variable and adding the command to be executed. After the command is run, it sets the appropriate return code. Commands queued by the **VAR** keyword are displayed at **MSGLEVEL(2)**.

DISPLAY – Echo information to SYSTPRT

Use the **DISPLAY** command to echo information to the SYSTSPRT output.

DISPLAY <expression>

Where *expression* can be any valid REXX expression. For more details about REXX expressions and available functions, see *TSO/E REXX Reference*.

Examples:

```
DISPLAY “Hello World”
DISPLAY @LOG() @V(OBJ-ADID)
```

DO and END – Block and Loop commands

Use the **DO** and **END** commands to designate a block of code to run together. You can either use them to group multiple commands for **IF**-**THEN**-**ELSE** or to repeat blocks of code.

The following list shows the variants of the **DO** commands:

Block

A single run block of code: **DO**

Repeat

A simple repeating block: **DO x**

Iterative

A block that increments or decrements a variable: **DO v = x TO y BY z**

While

A block that runs only while a condition is true: **DO WHILE expression**

Until

A block that runs until a condition is true: **DO UNTIL expression**

Forever

A block that runs until **LEAVE** or **EXIT** is run.

Block DO

A block **DO** runs a block of code once, and has no arguments.

For example:

```
DO
   VARSET COLLECT = “Y”
   DISPLAY “COLLECT” !COLLECT
END
```

You can use it together with **IF-THEN-ELSE** to run more than one command, when the expression is true, as in the following example:

```
IF “!TAG” = “–JOBNAME”
   THEN DOVARSET COLLECT = “Y”
   DISPLAY “COLLECT” !COLLECT
END
ELSE DO
   VARSET COLLECT = “N”
END
```

Repeat DO loop

A repeat **DO** runs a block of code for the specified number of times, with a simple repeat count as the argument.

For example, the DISPLAY command is run 3 times:

```
DO 3
DISPLAY “HELLO”
END
```

Use the **ITERATE** command at any point in the block to start the next iteration of the currently active loop from the **DO** statement again. Use the **LEAVE** command to exit the currently active loop and resume processing following the corresponding **END** statement.

The **ITERATE** and **LEAVE** commands can be used only to exit the currently active block.

Iterative DO loop

Use the iterative **DO** to increment or decrement a loop counter. The loop is processed for each possible value of the variable within the limits specified. By default, the value is increased by one each time; you can modify this setting with the **BY** keyword.

```
DO <variable> = <start> TO <end> [BY <increment>] [FOR <limit>]
```

where:

<variable>

Name of the variable to be incremented or decremented. Do not include the prefix variable.

<start>

Starting value of the variable. Must be an integer.

<end>

Ending value of the variable. Must be an integer.

<increment>

Value by which the variable is to be incremented or decremented. Must be a positive or negative integer. The default is 1.

<limit>

The maximum number of times that the loop is to be run. Must be an integer. The default is the absolute difference between **<start>** and **<end>** + 1.

For example:

```
DISPLAY “Testing... Testing”
DO X = 1 TO 3
   DISPLAY !X
END
```

Use the **ITERATE** command at any point in the block to start the next iteration of the currently active loop from the **DO** statement again. Use the **LEAVE** command to exit the currently active loop and resume processing following the corresponding **END** statement.

The **ITERATE** and **LEAVE** commands can be used only to exit the currently active block.

DO While loop

Use the **DO WHILE** statement to run the block of code while the expression is true. If the expression is untrue when the **DO WHILE** statement is processed for the first time, the contents of the **DO|END** block are not run, and processing continues from the command after the **END** statement.

The condition is evaluated again at the end of each iteration of the **DO|END** block, and if still true the block is run again.

```
DO WHILE <expression>
```

where:

<expression>

A REXX-based expression. The expression can be any valid REXX expression that results in 1 or 0.

For more details about the REXX expressions and available functions, see the *TSO/E REXX Reference* manual.

For example:

```
VARSET X = 1
DO WHILE “!X” < 10
   DISPLAY !X
   VARSET X DELTA(1)
END
```

Use the **ITERATE** command at any point in the block to start the next iteration of the currently active loop from the **DO** statement again. Use the **LEAVE** command to exit the currently active loop and resume processing following the corresponding **END** statement. The **ITERATE** and **LEAVE** commands can be used only to exit the currently active block.

Loops of this type are protected from running forever by accident by the global loop limit **OPTIONS LIMIT**, whose default is 100.

DO Until loop

Use the **DO UNTIL** statement to run the block of code until the expression is true. The contents of a **DO UNTIL** block will always run at least once, even if the expression is true the first time the **DO** statement is processed. Every time the **END** statement is processed the expression is evaluated again; if not true, processing returns to the **DO** statement for another pass through the block of code.

The condition is evaluated again at the end of each iteration of the **DO**/**END** block; if it is still not true, the block is run again.

DO UNTIL <expression>

where:

<expression>

A REXX-based expression. The expression can be any valid REXX expression that results in 1 or 0.

For more details on REXX expressions and available functions, see *TSO/E REXX Reference*.

For example:

```
VARSET X = 1
DO UNTIL “!X” = 10
   DISPLAY !X
   VARSET X DELTA(1)
END
```

The **ITERATE** command can be used at any point in the block to start the next iteration of the currently active loop from the **DO** statement again. The **LEAVE** command can be used to exit the currently active loop and resume processing following the corresponding **END** statement. **ITERATE** and **LEAVE** can only exit the currently active block.

Loops of this type are protected from running forever by accident by the global loop limit. This is set by **OPTIONS LIMIT**, which defaults to 100.

DO Forever loop

Use the **DO FOREVER** command to repeat a block until the global loop limit is reached or a **LEAVE** or **EXIT** command is found.

For example:

```
VARSET X = 1
DO FOREVER
   DISPLAY !X
   VARSET X DELTA(1)
IF “!X” > 20 THEN LEAVE
END
```

Use the **ITERATE** command at any point in the block to start the next iteration of the currently active loop from the **DO** statement again. Use the **LEAVE** command to exit the currently active loop and resume processing following the corresponding **END** statement. **ITERATE** and **LEAVE** can exit only the currently active block.

The **ITERATE** and **LEAVE** commands can be used only to exit the currently active block.

Loops of this type are protected from running forever by accident by the global loop limit **OPTIONS LIMIT**, whose default is 100.

Note:

- The **VARSCAN** command has the ability to perform an **ITERATE** or **LEAVE** action in the event of not finding any match by using the **ACTION** keyword. This negates the need for a separate **IF/THEN/ITERATE** or **IF/THEN/LEAVE** statement.
- By default, every statement in a **DO** block is listed in the log every time it is run, for each iteration of the block. This can result in extremely large logs and might not be the desired result. To reduce unwanted log traffic, set **OPTIONS MSGLEVEL(0)** before the **END** statement of the loop and **OPTIONS MSGLEVEL(1)** after the **END** statement. This results in all commands within the loop being listed once for the first full iteration of the loop, and from that point only statements which end in a non-zero return code are listed. The complete list of all commands will be resumed after the loop has been exited.

  For example:

  ```
  VARSET X = 1
  DO FOREVER
     DISPLAY !X
     VARSET X DELTA(1)
     IF “!X” > 20 THEN LEAVE
     OPTIONS MSGLEVEL(0)
  END
  ```

```
OPTIONS MSGLEVEL(1)
```

DROP – Drop elements from memory

Use the `DROP` command to drop saved structures from memory.

```
DROP <type>(<item>) [,<type>(<item>)...]
```

where:

`<type>`

The type of element being dropped from memory, this can be `SAVELIST` or `OBJECT`.

`<item>`

The name of the element being dropped.

The `DROP` command drops saved structures from memory. For large elements this might be required to reduce storage consumption, especially if multiple `SAVELIST` elements are merged using the `MERGE` command, or many large `OBJECT` elements are extracted from the database.

You can drop more than one element at once by repeating the keyword. For example, `DROP OBJECT(AD1) OBJECT(AD2)`.

Where a `LIST` request has generated multiple objects, dropping the `LIST` object also drops all of the generated objects.

Note: Before Workload Automation Programming Language version 3.4, only one kind of object could be dropped therefore the object type was not required. For backwards compatibility, if a `SAVELIST` name is provided without a keyword, it is assumed to be a `SAVELIST` and dropped.

For example, `DROP MYLIST` is considered equivalent to `DROP SAVELIST(MYLIST)`. `DROP SAVELIST` is considered equivalent to `DROP SAVELIST(SAVELIST)`.

EXIT – Terminate processing

Use the `EXIT` statement to terminate processing at that point.

When an `EXIT` statement is processed, no further user commands are run; but any automatic `EXECUTE` and `TERM` statements are executed before final termination.

```
EXIT [<rc>]
```

where `<rc>` is the return code (optional).

If you set the return code, it overrides any highest return code or any response codes from `LISTSTAT`.

An `EXIT` statement is assumed when the end of the program is reached or a `SUBROUTINE` statement is found.

FILTER – Post process selected records to reduce output

Use the **FILTER** command to post process what was returned to Workload Automation Programming Language from HCL Workload Automation for Z, to selectively reduce the output actually returned by Workload Automation Programming Language.

FILTER record [segment1-comparator(count)],[segment*n*-comparator(count)]

or

FILTER segment [field1-comparator(value)],[field*n*-comparator(value)]

or

FILTER record|segment OFF

Filtering can be performed at two levels:

Record level

When a record is selected, the **FILTER** command can include or exclude segments based on the number of segments available of each type. For example, **FILTER AD ADRUN-GT(0)** returns output only for AD records that contain run cycles.

Segment level

When a segment is being processed, fields within it can be checked that they match the **FILTER** argument.

For example, **FILTER ADOP ADOPWSID(CPU1)** returns only operations that use workstation CPU1. When a segment is excluded by a filter, all of its child segments are also excluded.

The **FILTER** command must be issued before any **LIST** or **SELECT** command. Any number of valid segment or field names can be included on a single **FILTER** statement, all must be true for the record or segment to be selected for output.

For example, **FILTER AD ADRUN-EQ(0) ADOP-GT(1)** outputs only applications with no run cycles and more than one operation.

To stop filtering for any subsequent commands use a **FILTER** command specifying only the record or segment name and the keyword **OFF**. For example, **FILTER ADOP OFF** turns off filtering of **ADOP** segments.

Combining FILTER with other commands and options can produce some complex processing, without having to write any specific REXX code. In the following example:

```
INCLUDE EQQFILE(ADCOM,ADRUN)
      OPTIONS RUNSTAT(SUSPEND) POSTPROC(Y) LOADER(*) DBMODE(UPDATE) DATA(-)
      FILTER AD ADRUN-GT(0)
      LIST ADCOM ADID(AB*) SELECT(Y)
```

the following actions are taken:

- The **INCLUDE** statement ensures that *only* the common segment and run cycles are output.
- **OPTIONS RUNSTAT(SUSPEND)** ensures that the output Batch Loader statements for the run cycle will include In and Out of Effect dates of LOWDATE.

- **FILTER** ensures that *only* applications with run cycles are selected for output.
- **LIST** causes every application beginning with AB to be selected for output, assuming it has at least one run cycle.
- **OPTIONS LOADER(*)** sends the batch loader statements to the REXX stack.
- **OPTIONS DATA(-)** ensures no ILSON data is written.
- **OPTIONS POSTPROC(Y)** tells Workload Automation Programming Language when it has finished processing the commands in SYSIN to process the command on the REXX stack.
- **OPTIONS DBMODE(UPDATE)** tells Workload Automation Programming Language to process the generated Batch Loader statements in **UPDATE** mode.

The result is that these four commands change only applications with names beginning with AB that have run cycles, to set them out of effect; thereby rendering these applications ineligible for planning, but still available to be manually added to the plan. Without the **FILTER** command, this job would attempt to update applications that have no run cycles, thereby performing a lot of unnecessary processing.

To create batch loader to perform the “suspension” of the applications in a later job, removing **POSTPROC(Y)**, **LOADER(*)** and **DBMODE(UPDATE)** from the **OPTIONS** statement would cause the statements to be written to OUTBL. They could then be processed by a later job with OPTIONS **DBMODE(UPDATE)** coded to set the job into **UPDATE** mode.

GOTO – Continue execution from an in scope label

Use the **GOTO** command to allow the process to skip to a named line of code, as long as it is in scope.

```
GOTO <label>
```

Where **<label>** is the name of the line to which the process is to skip (the colon sign must not be included).

Workload Automation Programming Language labels can be up to 16 characters in length, contain alpha, numeric or the underscore character. A label is indicated by the colon sign (:) at the end (for example, **MYLABEL:**). The colon is not included in the length.

The label to which you want to jump must be in scope. This means that:

- You can jump only to a label in the same input stream. For example, you cannot jump from a statement in EQQOPTS to one in SYSIN.
- You can jump forward or backward.
- You cannot jump into a loop or other **DO|END** structure.
- You can jump out of a loop or other **DO|END** structure, including jumping out into a **DO|END** structure that contains the current one.
- You cannot jump into a SUBROUTINE.
- You cannot jump out of a SUBROUTINE.

Using **GOTO** could cause an uncontrolled loop. The **OPTIONS LIMIT** keyword sets the number of times that any single **GOTO** statement can run; when the limit is reached RC=12 is issued. The default limit is 100.

Because the **GOTO**command runs in an uncontrolled way, consider first the following alternatives:

- **ITERATE** escapes the running of the current loop and continues from the top of the loop on the next iteration.
- **LEAVE** exits the current loop and resumes processing after the **END** statement at the bottom of the loop.
- **CALL SUB** allows a subroutine to be called and returns processing to the line following the **CALL** when the end of the subroutine is reached.

The following example shows a **GOTO** command used within SYSIN, loops, and subroutines:

```
VARSUB SCAN
PART1: DISPLAY "PART1"
VARSET COUNT = 0
PART2: DISPLAY "PART2"
GOTO PART9
PART3: DISPLAY "PART3"
VARSET COUNT DELTA(1)
PART4: DISPLAY "PART4"
PART5: DISPLAY "PART"
DO 5
   PART5A: DISPLAY "PART5A"
   GOTO PART5E
   PART5B: DISPLAY "PART5B"
   PART5C: DISPLAY "PART5C"
   PART5D: DISPLAY "PART5D"
   PART5E: DISPLAY "PART5E"
END
CALL SUB(BLK_SUBROUTINE1)
PART6: DISPLAY "PART6"
PART7: DISPLAY "PART7"
PART8: DISPLAY "PART8"
PART9: DISPLAY "PART9"
IF !COUNT < 4 THEN DO
   GOTO PART3:
END
PART10: DISPLAY "PART10"
EXIT

BLK_SUBROUTINE1: SUBROUTINE
SUB2: DISPLAY "SUB2"
GOTO SUB8
SUB3: DISPLAY "SUB3"
SUB4: DISPLAY "SUB4"
SUB5: DISPLAY "SUB5"
SUB6: DISPLAY "SUB6"
SUB7: DISPLAY "SUB7"
SUB8: DISPLAY "SUB8"
SUB9: RETURN

Example of use of GOTO within SYSIN, loops and subroutines
```

IF-THEN-ELSE – Conditional execution

Use the **IF** and **THEN** construct to conditionally execute a Workload Automation Programming Language statement, or block of statements if true. Use the **ELSE** statement to allow an alternative statement or block to run, if the condition is not true. These commands use underlying REXX processing to evaluate the expression.

The basic syntax is:

```
IF <expression> THEN
     <command1>
ELSE
     <command2>
```

The expression can be any valid REXX expression that results in 1 or 0. For more details about REXX expressions and available functions, see *TSO/E REXX Reference*.

Although Workload Automation Programming Language uses the REXX interpreter to evaluate the expression, there are some considerations to be made for Workload Automation Programming Language use:

- Workload Automation Programming Language variables are resolved *before* the statement is run, rather than evaluated by the REXX interpreter. This means that any Workload Automation Programming Language variable that includes special characters, such as blanks, must be enclosed within double quotation marks:

  ```
  IF "!TAG" = "-JOBNAME" THEN DO
  ```

- If the `THEN` keyword is surrounded by blanks, it must not appear within the `IF` or `THEN` expression, even if within double quotation marks. Therefore, the following example is valid because `THEN` does not have blanks around it:

  ```
  IF “!MYVAR” = “THEN” THEN DO
  ```

  When `THEN` needs blanks around it, you must specify it in a variable, as follows:

  ```
  VARSET THEN VALUE(THEN)
  IF “!MYVAR” = “THIS !THEN THAT” THEN DO
  ```

- Workload Automation Programming Language functions can be used within `IF`, `THEN`, and `ELSE` constructs, but cannot contain any REXX functions within the bounds of the Workload Automation Programming Language function. Only literal text or variables can be contained within the arguments of Workload Automation Programming Language functions. Workload Automation Programming Language functions are prefixed by the at sign (@).

  The following example is *not* valid:

  ```
  IF (@JCL(RIGHT(!LINE,8)..EQ.0)) & (LEFT(“!JOB”,1) = “P”) THEN
  ```

  but must be handled by resolving the REXX function in a previous statement:

  ```
  VARSET STEP = RIGHT(!LINE,8)
  IF (@JCL(!STEP..EQ.0)) & (LEFT(“!JOB”,1) = “P”) THEN
  ```

The conditional commands `<command1>` and `<command2>` can be included in the same line as `THEN` and `ELSE`, respectively. For example:

```
IF “!TAG” = “-JOBNAME” THEN VARSET COLLECT = “YES”
ELSE VARSET COLLECT = “NO”
```

To run multiple commands conditionally, use a `DO`, `END` block:

```
IF “!TAG” = “-JOBNAME” THEN DO
   VARSET COLLECT = “Y”
   ITERATE
END
```

You can use **AND** or **OR** within parentheses and the ampersand (**&**) and vertical bar (**|**):

```
IF (“!DAY” = “WED”) & (“!MONTH” = “JAN”) THEN
IF (“!DAY” = “MON”) | (“!DAY” = “TUE”) THEN
```

INCLUDE – Include code from other data sets or members to be run

Use the **INCLUDE** statement to include Workload Automation Programming Language statements from other data sets or members in the Workload Automation Programming Language command stream. With this command, you can also access user fields belonging to the operation that controls the job to use their values as SYSIN.

```
INCLUDE ddname-1(member-1,member-2,...,member-n) ddname-2,...,ddname-n
```

or

```
INCLUDE USER_FIELD(mask)
```

The statements can be read from DD statements allocated to the step running Workload Automation Programming Language.

In a single **INCLUDE** statement you can include multiple members from one DD statement and multiple DD statements. Workload Automation Programming Language reads and runs the statements in the order specified.

You can specify a DD statement without a member name if the DD statement is allocated to a sequential file, a partitioned data set with a member name specified, or a concatenation of both.

You can specify a DD statement with a member name if the DD statement is allocated to either a single partitioned data set without a member name specified, or a concatenation of partitioned data sets. You can also **INCLUDE** a member name when the DD statement is allocated to a partitioned data set with a different member allocated. If you specify a member when either the partitioned data set has no member name specified at allocation, or the member name allocated is different from the one requested, Workload Automation Programming Language allocates the partitioned data set, in which the member is found, temporarily to DD statements EQQTEMP to allow the content to be read.

Note:

- To search for a member name, Workload Automation Programming Language requires that all files in the concatenation are cataloged. If you want to **INCLUDE** a member from an uncataloged data set, you must allocate the member explicitly in the Workload Automation Programming Language step.
- If you use the **INCLUDE** statement with the EQQWAPL load module, you cannot specify member names. You can set only DD statements and user fields.

In the following example, you load the in-built **OUTPUT** definitions **ADCOM** and **ADRUN**. You have activated variable substitution and set two variables that will be referenced in Batch Loader contained within the skeletons in members **DAILY** and **WEEKLY** of the **MYSKELS** file:

```
//WAPLSTEP EXEC EQQYXJPX,
//         SUBSYS=IWSA
//MYSKELS DD DISP=SHR,DSN=MY.SKELS.LIB
//OUTDATA DD SYSOUT=*,LRECL=4096
//OUTBL DD SYSOUT=*
//SYSIN DD *
LOADDEF ADCOM
LOADDEF ADRUN
LIST ADCOM ADID(MYAPPLS*) SELECT(Y)
VARSUB SCAN
VARSET LOB=PY
VARSET PHASE=P
INCLUDE MYSKELS(DAILY,WEEKLY)
```

By default, the contents of any statements loaded by an **INCLUDE** statement are run at **MSGLEVEL(3)**, in the same way as whatever is included by being referenced by the **FILESPEC** symbolic parameter. This means that ordinarily if you use an **INCLUDE** statement the content is not displayed in the **SYSTSPRT** output of Workload Automation Programming Language, unless a command fails.

User fields can be included by using a special DD name of USER_FIELD. The member name is the name of an individual User Field, or it can be a mask, using the wildcard characters percent sign (%) and asterisk (*) to include values from many matching user fields.

The comparison with the field name is not case sensitive, and each field is used in alphabetical order.

ITERATE – Proceed to the next iteration of current loop

Use the **ITERATE** command to exit the current iteration of a **DO** block, and resume processing at the **DO** statement.

ITERATE

If this is a repeat or iterative loop, the loop counter is incremented. If this is within a Block **DO** structure, the block is exited, and the **ITERATE** applies to the nearest loop **DO** structure above.

The **ITERATE** command applies only to the currently active **DO** loop, you cannot **ITERATE** a higher-level nested loop structure this way.

LEAVE – Exit the current loop

Use the **LEAVE** command to exit the current iteration of a **DO** block, and resume processing following the **END** statement.

LEAVE

If this is within a Block **DO** structure, the block is exited, and the **LEAVE** applies to the nearest loop **DO** structure above.

The **LEAVE** command applies only to the currently active **DO** loop, you cannot **LEAVE** a higher level nested loop structure this way.

LOG – Echo information to the log

Use the **LOG** command to echo information to the log output.

```
LOG <expression>
```

where **<expression>** can be any valid REXX expression. For more details about REXX expressions and available functions, see *TSO/E REXX Reference*.

For example:

```
VARSET STEP ENVATTR(JCL,STEP,0)
LOG MSGX000I @V(OJOBNAME) @V(STEP) @V(OJJESNO)
LOG MSGX001I @V(OADID) @V(OYMD1)||@V(OHHMM) @V(OOPNO)
```

MERGE – Merge SAVELIST output

Use the **MERGE** command to merge the contents of two lists into a single list with no duplicates.

```
MERGE <list1> AND|OR|XOR|NOT <list2> INTO <list3> [BY <fields>]
```

where:

<list1>

The name of the first **SAVELIST** to merge.

<list2>

The name of the second **SAVELIST** to merge.

<list3>

The name of the **SAVELIST** in which to store the results.

<fields>

Optional. The name of the fields to merge by.

The following list shows the different kinds of **MERGE**:

AND

Only output entries that are in both lists.

OR

Output the combined content of both lists.

XOR

Only output entries that are not in either list.

NOT

Only output entries that are in the first list but not in the second list.

The **MERGE** command combines the contents of two lists into a single list with no duplicates, based on the criteria. Ordinarily it uses the entire text of each **SAVELIST** entry to perform the **MERGE**, but you can use **BY** to restrict the **MERGE** criteria to named keywords in the **SAVELIST** text.

Note:

- The **MERGE** technique relies upon the two lists being ordered in the sort sequence of the merging criteria. If you select criteria that do not match the sort sequence the results might not be as expected.
- If an entry exists in both lists, with matching criteria, the entry from the first list is selected for output.
- When merging **LIST GENDAYS** output it is recommended to code **BY DATE IAT** as criteria, because the **GENDAYS SAVELIST** also contains **FLAGS** that might often not match between entries for the same date.

NOACT – Peform no action

The main purpose of the **NOACT** command is to be a null action for complex **IF** expressions where the action you really want is the **ELSE** condition.

The **NOACT** command has no arguments, does nothing and always ends with return code zero.

For example:

```
IF (@CMD(CHK1.EQ.0)) & (@CMD(CHK2.EQ.0)) THEN NOACT
ELSE SENDMSG T(Something failed) U(ADCDMST)
```

Note: The **NOACT** statement performs no action at all, but accepts further text following the command. Therefore, you can use **NOACT** as a testing or diagnostic aid to see the value of variables being resolved at that point:

```
08/21 14.48.03 EQQI200I NOACT !_LSTAT1_CPOPERR
08/21 14.48.03 EQQI201I NOACT SB37
08/21 14.48.03 EQQI299I Statement completed - RC=0 (00000012)
```

OPTIONS – Define run time options and PIF requests

Use the **OPTIONS** commands to set both Workload Automation Programming Language and HCL Workload Automation for Z options.

OPTIONS <arguments>

The **OPTIONS** statement has no resource. For Workload Automation Programming Language it can be used to set both Workload Automation Programming Language options and HCL Workload Automation for Z options. It can be used also by ILSON and WAX, although not all keywords are available in each utility. To determine which keywords are available for each utility, use the **SHOW OPTIONS** command.

For a complete list of the available **OPTIONS**, see OPTIONS keywords on page 410.

OUTPUT – Define output record

```
OUTPUT <segment> [KEYS(field1,field2,...)]
                 [FIELDS(field3,field4,...)]
                 [LABEL(YES|NO|NOFIELD|NOSEGMENT)]
                   [DATA(*|=|<dd-statement>)]
                   [LOADER(*|=|<dd-statement>)]
```

The **KEYS** keyword specifies a list of key fields to be output for the record, the **FIELDS** keyword specifies a list of non-key fields for the record. Workload Automation Programming Language makes no distinction between fields specified in **KEYS** or **LIST** when creating the output record; the distinction is made only if you are going to load the output file into ISPF using the undocumented EQQILSON utility.

The **OUTPUT** statement must precede any **LIST** or **SELECT** statements for which you want to specify the output characteristics.

You can define segment names as follows:

```
<segment>[=<alias>|*]
```

where:

segment

Required. The name of the HCL Workload Automation for Z segment for which you want to define output.

alias

Optional. An alternative name to use as the record label in the output.

*****

Optional. A special case of alias that suppresses the segment label in the output.

You can define fields as follows:

```
[+|-][<iws-segment>.]<iws-field>[=<alias>|*|<length>]
```

where:

+ or –

Indicates that this field is to be used as the sort sequence, if the record is loaded into ISPF using EQQILSON (optional).

If the plus sign (+) or minus sign (–) is specified for use with EQQYXTOP, it does not impact the output, therefore the same FILESPEC member can be used by EQQYXTOP and EQQILSON to ensure that data is written from HCL Workload Automation for Z in the same way as it is loaded into ISPF.

iws-segment

Optional, needed only to reference a field from a parent segment. Reference only segments that are parents to the current segment; values from other segments might not be available.

iws-field

Required. The HCL Workload Automation for Z field name as described in Appendix A of *Developer's Guide: Driving HCL Workload Automation for Z*.

alias

Optional. An alternative field label to use in the output record.

*****

Optional. A special case of alias that suppresses the field label in the output.

length

If a numeric length is specified instead of an alias, it suppresses the field label and generates a fixed width field.

For valid segment names, see OUTPUT field definition reference on page 370.

The **LABEL** keyword determines what happens with field and segment labels, as follows:

YES

Default. Labels appear at Field and Segment level, unless an asterisk (*) is specified.

NO

No labels appear.

NOFIELD

No field labels appear, but the segment is labeled.

NOSEGMENT

No segment label appears, but the fields are labeled.

The **DATA** keyword specifies the Output Destination for Data records generated for this segment. If no **DATA** Output Destination is specified for this segment and **OPTIONS DATA** is not specified, no Data records are written for this segment. If **OPTIONS DATA** is specified, the **DATA** keyword of **OUTPUT** is ignored and all Data Records are sent to the destination specified in **DATA**.

The **LOADER** keyword specifies the Output Destination for Batch Loader generated for this segment. If no **LOADER** Output Destination is specified for this segment and **OPTIONS LOADER** is not specified, no Batch Loader statement is written for this segment. If **OPTIONS LOADER** is specified, the **LOADER** keyword of **OUTPUT** is ignored and all Batch Loader is sent to the destination specified by **LOADER**.

Note: You can have multiple **OUTPUT** statements for the same segment, only the keywords specified are effective. If a keyword is not specified in an **OUTPUT** statement, the keywords of a previous **OUTPUT** statement for the same segment

are applied. Therefore, you can specify fields and output destination in one statement and then divert subsequent output to an alternative destination with a later statement without having to specify all the fields again.

The following example shows multiple **OUTPUT** statements for the same segment. The ADCOM file contains fields ADID and ADDESC and is sent to MYOUT. This technique allows you to have standard file definitions and override the output destination without specifying all the fields again.

```
OUTPUT ADCOM FIELDS(ADID,ADDESC) DATA(OUTBL)
OUTPUT ADCOM DATA(MYOUT)
```

For each segment there is an in-built **OUTPUT** definition that includes all fields, sends ILSON data to OUTDATA, and sends batch loader data to OUTBL. To load all the fields for a segment, use the **LOADDEF** command to use these in-built definitions instead of coding your own **OUTPUT** statements. The **LOADDEF** command overrides the keywords in the **OUTPUT** statement, in a similar way to the use of subsequent **OUTPUT** statements for the same segment.

Specifying output destinations

Specify output destinations to tell Workload Automation Programming Language where to send output records.

The destinations can be either a DD statement or an asterisk (*). Specify an asterisk (*) to write the output to the external data queue. If you specify the equal sign (=) as the output destination, the output is written to the same destination as its parent record.

Many different segments can be written to the same output destination.

If you specify a DD statement that is not allocated, the first command requiring the DD statement issues a warning message (setting return code 4), and the output destination is suppressed.

Setting additional fields

Additional fields are available for each **OUTPUT** segment.

KEY

The fully qualified key of the segment.

TYPE

The type of segment, for example ADOP.

PARENT_KEY

The fully qualified key of the parent segment.

PARENT_TYPE

The type of parent segment, for example ADCOM.

TAG

The data passed into the **LIST** or **SELECT** statement in the **TAG** keyword. This allows output from multiple **LIST** or **SELECT** commands to be correlated back to the originating command by tagging each output record.

One single level of a key is formed from the segment type followed by a hex 00 and then each key field separated by hex 00. Therefore, a single level of a key for an application called MYAPPL with a status of Active that is valid until 31 December 2071 would have a single level key of `ADCOM 00x MYAPPL 00x A 00x 711231`.

A fully qualified key is a sequence of single keys separated by hex 01, to uniquely identify a segment within an object within the database. Therefore, operation 010 within the previously described MYAPPL would be `ADCOM 00x MYAPPL 00x A 00x 711231 01x ADOP 00x 010`.

READ – Read an external file or the external data queue

Use the **`READ`** command to load external data sources into **`OBJECT`** variables.

```
READ <file>[+|-] | * OBJECT(<object>)
[ADID(<application>)]
[CLIP(<n>|<-n>)]
[COUNT(0|<num-of-records>)]
[EXCLUDE|INCLUDE([a]n,<comparator>,<string>)]
[FROM|FROMX(<string>)]
[IA(<iput-arrival>)]
[JOB(<job-name>)]
[JOBID(<jes-number>)]
[MATCHTYP(EXA|END|INC|PFX|SFX|<n>)]
[OPNO(<operation-number>)]
[POSITION(UNIQUE|EARLIEST|LATEST)]
[PREDECESSOR(Y|N)]
[SKIP(0|<records>)]
[SCOPE(ALL|ACTIVE|HELD|OUTPUT)]
[SOURCE(DATASTORE|DSN|FILE|SYSOUT)]
[TEMPFILE(<dd-name>)]
[TO|TOX(<string>)]
```

where:

`<file>`

Name of an input DD statement, data set, or a SYSOUT identifier from which to read. If the **`SOURCE`** is SYSOUT, the format is **`PROCSTEP.JOBSTEP.DDNAME`**; you can omit **`PROCSTEP`** if not appropriate.

`[+|-]`

Specifies whether the output file can be written by further commands or is to be closed, allowing the subsequent commands in the same step to read it. The plus sing (+) allows additional writing; the minus sign (-) closes the output file. The plus or minus sign must be appended immediately after the end of the DD statement. The default is the plus sign (+), meaning that a DD statement without a suffix allows subsequent writing to the file.

`*`

Valid only for **`SOURCE(FILE)`**. Indicates that the external data queue is to be used as input.

`OBJECT(<object>)`

Valid only for **`SOURCE(DATASTORE)`**. Name of the application that contains the job for which you want to extract the job log.

`ADID(<application>)`

Name of the object variable into which to read the data.

`CLIP(<n>|<-n>)`

Deletes *n* characters from the beginning of each record of the input file. This is useful to delete control characters from the beginning of a SYSOUT file, so that the column positions match the original input data rather than the printed version. If you specify a positive number, the record is cut before the **`FROM|TO`** and **`INCLUDE|EXCLUDE`** processing is performed. If you use a negative number, the **`FROM|TO`** and **`INCLUDE|EXCLUDE`** processing is performed before cutting the absolute number of characters.

For example, if you specify **`CLIP(-1)`** the first character is deleted after **`FROM|TO`** and **`INCLUDE|EXCLUDE`** comparisons are performed.

`COUNT(0|<num-of-records>)`

When `SOURCE(FILE)`

The **`COUNT`** keyword allows large files to be read in small blocks to manage memory usage. If you set a number of records with the **`COUNT`** keyword, the command reads that number of records in the file. A following **`READ`** command starts from that point in the file, rather than from the beginning. If **`COUNT`** is set to a number higher than the number of remaining records, all the remaining records are read and a message is issued, setting RC=4.

If you set **`COUNT(0)`** the file is closed after reading, therefore subsequent **`READ`** commands start again from record 1.

When `SOURCE(DSN)`, `SOURCE(SYSOUT)`, or `SOURCE(DATASTORE)`

The **`COUNT`** keyword limits the number of records stored in the **`OBJECT`**, but subsequent **`READ`** requests to the same input will always start from the beginning.

`EXCLUDE|INCLUDE([a]n,<comparator>,<string>)`

The filter by which to include or exclude a record. You can define multiple filters for each **`READ`** command, but you cannot mix **`INCLUDE`** and **`EXCLUDE`** keywords. By setting the **`INCLUDE`** filter, only records that match the filter will be output. By setting the **`EXCLUDE`** filter, all records except those that match the filter will be output. Filtering is performed after the **`FROM|TO`** processing.

You can set the following values:

`[a]`*`n`*

The presence of a single alpha character prefix indicates that the filter is grouped (AND) with all adjacent filters with the same group character. The absence of a single alpha character indicates a standalone (OR) filter, where only one filter needs to be true to trigger the **`INCLUDE`** or **`EXCLUDE`** processing. ***`n`*** specifies the column at which to do the comparison.

For example, the following command returns rows that begin with A if they have C in column 3, all records beginning with C, and rows with C in column 1 and E in column 3.

```
INCLUDE(A1,EQ,A) INCLUDE(A3,EQ,C) INCLUDE(1,EQ,B) INCLUDE(C1,EQ,C) INCLUDE(C3,EQ,E)
```

<comparator>

The following list shows the allowed comparators:

- **EQ**, Equal to
- **NE**, Not equal to
- **LT**, Less than
- **GT**, Greater than
- **LE**, Less than or equal to
- **GE**, Greater that or equal to
- **BL**, Blank at that point
- **NB**, Not blank at that point

<string>

The string to be compared. In the case of comparator **NB** or **BL**, this argument can be the number to indicate the length of the non-blank or blank character. If omitted, only a single character is compared.

FROM|FROMX(<string>)

A range filter that does not output any records until a match is found. **FROM** starts the output from the row of the match, **FROMX** starts the output from the following row. The **MATCHTYP** keyword affects how the match is made. If you do not specify **FROM**n or **FROMX**, the output starts from the beginning of file.

IA(<input-arrival>)

Valid only for **SOURCE(DATASTORE)**. The input arrival (in the format *YYMMDDHHMM*) of the occurrence that contains the job for which you want to extract the job log.

JOB(<job-name>)

Valid only for **SOURCE(DATASTORE)** and **SOURCE(SYSOUT)**. Name of the job for which you want to extract the job log or SYSOUT.

JOBID(<jes-number>)

Valid only for **SOURCE(SYSOUT)**. JES number of the job for which you want to extract the job log. The format corresponds to the format of the JES numbers for your system.

MATCHTYP(EXA|END|INC|PFX|SFX|<n>)

Determines how the **FROM** and **TO** strings are matched:

- **EXA** will make an exact match for the entire row, trailing blanks are ignored. This is the default.
- **END** will compare with the end of the record. Trailing blanks are compared.
- **INC** will match if the string is found anywhere in the row.

- **PFX** will match if the record begins with the string.
- **SFX** will match if the non-blank portion of the record matches the string. Trailing blanks are ignored.
- **<n>** will match if the string appears at that specific column in the record.

OPNO(<operation-number>)

Valid only for **SOURCE(DATASTORE)**. Operation number of the job for which you want to extract the job log.

POSITION(UNIQUE|EARLIEST|LATEST)

Valid only for **SOURCE(DATASTORE)** and **SOURCE(SYSOUT)**. The position in the queue of the matching elements to select for processing. Allowed values are:

- **UNIQUE**, meaning that your selection criteria must identify a single job, otherwise the command fails. This is the default.
- **EARLIEST**, to select the first entry on the list.
- **LATEST**, to select the last entry on the list.

If you use **OPTIONS TRACE(1)** before the **READ** command, all identified entries in the list are shown.

PREDECESSOR(Y|N)

Valid only for **SOURCE(DATASTORE)**, and only for jobs being run from within the current plan. Restricts the search for the job for which you want to extract the job log to its direct predecessors. Making the job for which you want to extract the job log a predecessor is highly recommended, because the **PREDECESSOR(Y)** option avoids searching the entire current plan, consuming less resources.

This keyword can be shortened to **PRED** without specifying a value, as long as it is not coded the first keyword to the **READ** command. Default is **N**.

SCOPE(ALL|ACTIVE|HELD|OUTPUT)

Valid only for **SOURCE(DATASTORE)**, and only for jobs being run from within the current plan. Restricts the search for the job for which you want to extract the job log to its direct predecessors. Making the job for which you want to extract the job log a predecessor is highly recommended, because the **PREDECESSOR(Y)** option avoids searching the entire current plan, consuming less resources.

This keyword can be shortened to **PRED** without specifying a value, as long as it is not coded the first keyword to the **READ** command. Default is **N**.

SKIP(0<records>)

Number of records to skip before loading the input into the **OBJECT**. Default is **0**.

SOURCE(DATASTORE|DSN|FILE|SYSOUT)

The type of data being read. Allowed values are:

- **DATASTORE**, to fetch a job log from the HCL Workload Automation for Z data store, if the data store feature has been configured.
- **DSN**, to dynamically allocate a data set to read, using the name specified as the first argument, and then free it afterwards. The data set is allocated to the DD statement specified by the **TEMPFILE** keyword, if any; otherwise it uses the name set by **OPTIONS TEMPFILE**.
- **FILE**, to read from the DD statement named as the first argument. This is the default data source, unless you specify an asterisk (*) as the file name, in which case it reads from the External Data Queue (REXX stack).
- **SYSOUT**, to read SYSOUT from the JES spool, using the name specified as the first argument. This option is valid only if SDSF is installed.

TO|TOX(<string>)

Range filter that stops the output when a match is found. **TO** stops the output after the row of the match, **TOX** does not output the line of the match or any following lines. The **MATCHTYP** keyword affects how the match is made.

The **READ** command allows data from an external source to be loaded into an **OBJECT** variable, allowing the contents to be interrogated either with bespoke code in a **DO** loop, or with commands such as **VARSCAN**.

For example:

- To read data from the DD statement **MYFILE** and store it in object **MYOBJ**, issue the following command:

```
READ MYFILE OBJECT(MYOBJ)
```

- To read the first three records from the DD statement **MYFILE** and store them in object **MYOBJ**, issue the following command. Subsequent **READ** commands will start from record 1 due to minus sign (–) DD suffix:

```
READ MYFILE- OBJECT(MYOBJ) COUNT(3)
```

- To read data from SYSOUT file **SYSUT2** in step **IEBGENR** in the current job, issue the following command:

```
READ IEBGENR.SYSUT2 SOURCE(SYSOUT) OBJECT(MYOBJ)
```

- Issue the following command to read lines 2 to 4 from SYSOUT file **SYSUT2** in step **IEBGENR** in the current job, then look for the text **JAM** in the text, which if found will cause return code 16 to be set.

```
VARSUB SCAN
READ IEBGENR.SYSUT2 SOURCE(SYSOUT) OBJECT(MYOBJ)
                    SKIP(1) COUNT(3)
DO I = 1 TO !@MYOBJ
   VARSET THISLINE @V(@MYOBJ-!I)
   IF POS("JAM","!THISLINE") > 0 THEN DO
      WRITE OUTDATA "JAM DETECTED"
      WRITE OUTDATA "!THISLINE"
      EXIT 16
   END
END
```

- To read the JES message log from the last instance of **DEANWPL1**, issue the following command:

Note: If the job running this command is also named **DEANWPL1**, it is not classed at the latest job, because active jobs are listed before completed jobs for **SCOPE(ALL)**. To get the currently running job, use **SCOPE(ACTIVE)** or **POSITION(EARLIEST)**.

- To read the job log from the immediate predecessor to this job, issue the following command. If the job has more than one predecessor, you must specify additional keywords to restrict the selection to only one of the potential predecessors.

```
READ SOURCE(DATASTORE) OBJECT(MYOBJ) PRED
```

- To read the job log from the immediate predecessor to this job, and process only the **JESMSGLG** portion of the job log, issue the following command:

```
READ SOURCE(DATASTORE) OBJECT(MYOBJ) PRED
     FROMX(££EQQFSWWU-START-JESMSG)
     TOX(££EQQFSWWU-START-JESJCL)
```

- Issue the following command to read a report from file and output pages 8 to 19, where the page marker is at column 113 on each page before clipping, then to remove the first character:

```
READ LTP SOURCE(FILE) OBJECT(MYOBJ) CLIP(-1)
         MATCHTYP(113) FROM(PAGE 0008) TOX(PAGE 0020)
```

- Issue the following command to read a report from the first occurrence of **LONG TERM PLAN FOR DATE** until the line before **GRAND TOTAL WORKLOAD FOR PERIOD**. Note that depending on where the report header appears on the page, you might lose some header lines from the first page, and gain redundant lines from the start of the next section.

```
READ LTP SOURCE(FILE) OBJECT(MYOBJ) CLIP(-1)
         MATCHTYP(INC)
         FROM(LONG TERM PLAN FOR DATE)
         TOX(GRAND TOTAL WORKLOAD FOR PERIOD)
```

- Issue the following command to extract a job log from the data store. By using **FROM** and **TO** filters, you restrict the processing to the JES message log portion of the output. The **INCLUDE** filters are set to pick up the **-JOBNAME** header as a standalone filter. The minus sign (–) and the job name are combined in a group filter to find the job specific rows.

  The **CONTENTION** keyword reduces the waiting time between the recall of the job log and its subsequent reprocessing.

```
VARSUB SCAN
OPTIONS CONTENTION(5,60)
VARSET JOB = "TWSXCPEX"
READ SOURCE(DATASTORE) OBJECT(MYOBJ)
     ADID(DH#PLANNING) JOB(!JOB) POSITION(LATEST)
     FROMX(££EQQFSWWU-START-JESMSG)
     INCLUDE(20,EQ,-JOBNAME)
     INCLUDE(J20,EQ,-)
     INCLUDE(J21,EQ,!JOB)
     TOX(££EQQFSWWU-START-JESJCL)
```

  Output example:

```
12.04.45 JOB08895  -JOBNAME  STEPNAME PROCSTEP    RC   EXCP
12.04.45 JOB08895  -TWSXCPEX          DNTOP       00   1241
12.04.48 JOB08895  -TWSXCPEX          SORT        00     65
12.04.51 JOB08895  -TWSXCPEX          DPREPORT    00    363
12.04.51 JOB08895  -TWSXCPEX          RCLDUMMY    00      2
12.04.51 JOB08895  -TWSXCPEX ENDED.  NAME-
```

- Issue the following command to read from the data store. By using **FROMX** you restrict processing to the JES message section of the job log, before using **INCLUDE** to extract all the **IGD104I** messages that also have **EQQ** at column 72.

```
READ SOURCE(DATASTORE) OBJECT(MYOBJ)
     ADID(DH#PLANNING) JOB(!JOB) POSITION(LATEST)
     FROMX(££EQQFSWWU-START-JESMSG)
     INCLUDE(A1,EQ,IGD104I)
     INCLUDE(A72,EQ,EQQ)
```

Note:

- **POSITION(EARLIEST|LATEST)** refers to the elements found and their order in the list. It does not refer to the scheduled start time or any other time based field.
- If not selecting the element you expect, using **OPTIONS TRACE(1)** causes Workload Automation Programming Language to list all the potential elements that match your criteria.
- **FROM|TO** processing is not performed on skipped records.
- **FROM|TO** and **INCLUDE|EXCLUDE** processing is performed on data after it is read from the data source. If you know the range of records in which the required data occurs, you can make the process more efficient by also using **SKIP** and **COUNT**, to reduce the number of records being filtered.
- For **INCLUDE** and **EXCLUDE** as each separate AND group is aggregated together with other filter keywords by OR, only a single AND group, or OR filter needs to be true for the filter to be true.
- To use **SOURCE(SYSOUT)**, ensure that the SDSF product is installed and that you are authorized to use it and access the requested data.
- By setting **SOURCE(DATASTORE)**, the first time the job log is accessed a recall command is issued; then Workload Automation Programming Language retries later to check if the recall was successful. **OPTIONS CONTENTION** determines the delay between the recall and retry, and the number of retry attempts. For example, by setting **OPTIONS CONTENTION(5,60)**Workload Automation Programming Language waits 5 seconds and retries up to 60 times. It is recommended that you check your site default for the wait parameter, because the product default is to wait for 30 seconds, which could be longer than needed for a data store recall.

RETURN – Exit the subroutine

Use the **RETURN** command to exit the subroutine and return the processing to the instruction that follows the **CALL** command.

When multiple subroutines are defined, a **RETURN** statement is automatically assumed when the next **SUBROUTINE** statement is encountered. When the last statement in SYSIN is reached, a **RETURN** is assumed if a subroutine is being defined at that point.

A **RETURN** statement encountered outside a subroutine is treated like an **EXIT** statement.

SETMAX – Manipulate the maximum return code

Use the **SETMAX** command to modify the maximum return code and maximum response code, at any point in the Workload Automation Programming Language command sequence.

```
SETMAX POLICY(<in_rc1>=<out_rc1>,<in_rc2>=<out_rc2>,...,
               <in_rcn>=<out_rcn>,<catch_all>)
      [SET(MAX_RC|MAX_RESP|BOTH)]
```

The command takes the current maximum return code, or maximum response code, or both, and use the **POLICY** keyword, looking for a match on the left side of each expression (`in_rc`). If a match is found, it sets the new maximum to the right side of the expression (`out_rc`).

If no match is found it, it sets the return code to the specified `catch_all` return code. If you omit to supply a catch all return code, and there is no match, the return code remains unchanged.

This command can influence maximum values: the maximum return code and maximum response code. The maximum response code is a special case of return code that is set by commands such as **LISTSTAT**, which sets the return codes to communicate the status of an HCL Workload Automation for Z object to other steps outside Workload Automation Programming Language. However, the number of variants of return code to do this usually exceeds the critical return code (set by **OPTIONS STOPRC**), which would flush the remaining Workload Automation Programming Language processing. To avoid this, commands of this type set a response code, which is returned by Workload Automation Programming Language when it completes, if it is higher than the maximum return code.

To specify which maximum code the **SETMAX** command will affect, use the **SET** keyword with one of the following arguments:

MAX_RC

Default. Compares the **POLICY** against the current maximum return code, and modifies the maximum return code if a match is made within the **POLICY**. It also sets the return code of the **SETMAX** command to the same value.

MAX_RESP

Compares the **POLICY** against the current maximum response code, and modifies the maximum response code if a match is made within the **POLICY**. It does not set the return code of the **SETMAX** command.

BOTH

Compares with the higher of the current maximum return code and maximum response code. It modifies both if a match is found within the **POLICY**. It also sets the return code of the **SETMAX** command to the same value.

In the following example, the result of a **LIST** statement is reversed. If something is found, typically you would get RC=0 and if not found you would get RC=4. The **SETMAX** statement reverses that, so you can have a situation where the “good outcome” is for something to not be there (RC=0) and the “bad outcome” is if it is found (RC=4).

```
05/28 12.55.11 EQQI200I LIST CPOPCOM ADID(TWSCDAILYPLAN) OPNO(010)
                        IA(0805281300)
05/28 12.55.12 EQQI299I Statement completed - RC=4
```

```
05/28 12.55.13 EQQI200I SETMAX POLICY(0=4,4=0)
05/28 12.55.13 EQQI122A Maximum return code 4 changed to 0
05/28 12.55.13 EQQI299I Statement completed - RC=0
```

Another use of this would be to consolidate unexpected return codes. Workload Automation Programming Language returns only 0, 4, 8, 12, and 16, but the HCL Workload Automation for Z PIF could return other higher return codes (for example, RC=700 for an uninitialized session). In the following example, return codes 0, 4, 8, 12, and 16 remain unchanged, but if any other return code is found it is set to 20.

```
SETMAX POLICY(0=0,4=4,8=8,12=12,16=16,20)
```

The following example shows how you can also influence a response code set by `LISTSTAT`. This would set the maximum response code from any previous `LISTSTAT` commands to zero, but not alter the maximum return code as set by other processes:

```
SETMAX POLICY(0) SET(MAX_RESP)
```

Note: The default value of the `SETMAX SET` keyword can be influenced by `OPTIONS SETMAX`.

SETSEV – Set message severity

Use the `SETSEV` command to change the severity of an individual message in Workload Automation Programming Language to allow different return code processing to take place.

```
SETSEV <msgID>
```

Enter the message ID whose severity you want to change.

For example, `SETSEV EQQI114W` changes the severity of EQQI114E to the value W.

The EQQI prefix can be omitted to simply specify the message number and new severity, as follows:

```
SETSEV 114W
```

In this way, the severity of EQQI114E is changed to W.

Note: `SETSEV` is available only for Workload Automation Programming Language messages prefixed with EQQI, although it also allows the use of the prefix EQQB to be compatible with Workload Automation Programming Language versions 3.3, or earlier.

SHOW – Show diagnostic information

Use the `SHOW` command to show diagnostic information, to help you understand with what information Workload Automation Programming Language is operating.

```
SHOW FILES|OBJECT|OPTIONS|RC|SPE|SUBSYSTEM|SYSINFO|USRF|VARIABLES
```

Note:

1. The **SHOW** command is available within the Workload Automation Programming Language ILSON and WAX utilities, but not all functions are available in both.
2. **USRF** and **VARIABLES** keywords cause the job to attempt to find itself in the Current Plan if it has not already done so.
3. You can specify multiple keywords in a single **SHOW** statement, for example **SHOW USRF VARIABLES**.

SHOW FILES – Display files allocated to Workload Automation Programming Language

Use the **SHOW FILES** command to list each file in turn that it detected as being allocated to the Workload Automation Programming Language step when Workload Automation Programming Language started.

For each file, the **SHOW FILES** command lists:

DD Name:Position

The position shows which file number this is in a concatenation. In most cases it is 1.

DSN(name)

Data set name for this file.

MEM(name)

Member name for this file, if one was set at allocation time.

NOINPUT

Indicates that Workload Automation Programming Language does not consider this file eligible for INPUT.

NOOUTPUT

Indicates that Workload Automation Programming Language does not consider this file eligible for OUTPUT processing.

NOINCMEM

Indicates that Workload Automation Programming Language does not consider this file eligible for member INCLUDE.

LRECL(num)

The record length Workload Automation Programming Language uses to calculate output line breaks.

RECFM(xxx)

Indicates the record format of the file. If set to a question mark (?), it means that record format has not been set explicitly in either the JCL or when the file was opened.

TYPE(NORMAL|SYSIN|SYSOUT|DUMMY)

The type of file.

TMP

Indicates that Workload Automation Programming Language considers this a temporary data set.

CAT

Indicates that Workload Automation Programming Language considers this a cataloged data set.

CC

Indicates that the data set is using printer control characters.

The following example shows the output of **SHOW FILES**:

```
03/15 14.42.42 EQQI600A STEPLIB:1 - DSN(TWS.V920.SEQQLMD0) LRECL(72)
            ...EQQI600A TYPE(NORMAL) CAT
03/15 14.42.42 EQQI600A SYSPROC:1 - DSN(MY.USER.REXX)
LRECL(80)
            ...EQQI600A TYPE(NORMAL) CAT
03/15 14.42.42 EQQI600A SYSPROC:2 - DSN(TWS.V920.SEQQMISC)
LRECL(72) TYPE(NORMAL)
            ...EQQI600A CAT
03/15 14.42.42 EQQI600A EQQMLIB:1 - DSN(TWS.V920.SEQQMSG0) LRECL(72)
            ...EQQI600A TYPE(NORMAL) CAT
03/15 14.42.42 EQQI600A EQQMLOG:1 - DSN(ADCDMST.ADCDMSTS.JOB01870.D0000102.?)
            ...EQQI600A NOINPUT NOINCMEM LRECL(72) TYPE(SYSOUT) TMP
03/15 14.42.42 EQQI600A EQQDUMP:1 - DSN(ADCDMST.ADCDMSTS.JOB01870.D0000103.?)
            ...EQQI600A NOINPUT NOINCMEM LRECL(72) TYPE(SYSOUT) TMP
```

SHOW OBJECT – Display the object structure of an HCL Workload Automation for Z record

Various commands allow you to create object variables. The format of the object variables reflects the structure of the record being retrieved. Use the **SHOW OBJECT** command to list the structure of a specific record.

The syntax is **SHOW OBJECT(<record>)**

Where **<record>** can be one of the following values:

AD

Application Description (database)

AWSCL

All workstations closed (database)

CL

Calendar (database)

CPCOND

Condition (current plan)

CPOC

Occurrence (current plan)

CPOP

Operation (current plan)

CPOPSRU

Special resource usage (current plan)

CPUSRF

User field (current plan)

CRITPATH

Critical path (current plan)

CSR

Special resource (current plan)

CPST

Status (current plan)

CPWS

Workstation (current plan)

CPWSV

Workstation destination (current plan)

ETT

Event trigger (database)

GENDAYS

Rule date list

JCLV

JCL variable table (database)

JL

Job log (current plan)

JS

JCL (current plan)

JCLPREP

JCL (current plan)

LTOC

Occurrence (current plan)

OI

Operator instruction (database)

PR

Period (database)

RG

Run cycle group (database)

SR

Special resource (database)

WS

Workstation (database)

WSV

Workstation destination (database)

XENV

Execution environment

In the following example, the **SHOW OBJECT(CL)** command shows all the available object variables for a calendar with **-n-** showing where sequence numbers fit into the syntax:

```
08/22 10.47.39 EQQI200I SHOW OBJECT(CL)
08/22 10.47.39 EQQI601A Object: @OBJ-CLNAME
08/22 10.47.39 EQQI601A Object: @OBJ-CLDAYS
08/22 10.47.39 EQQI601A Object: @OBJ-CLSHIFT
08/22 10.47.39 EQQI601A Object: @OBJ-CLDESC
08/22 10.47.39 EQQI601A Object: @OBJ-CLVERS
08/22 10.47.39 EQQI601A Object: @OBJ-CLLDATE
08/22 10.47.39 EQQI601A Object: @OBJ-CLLTIME
08/22 10.47.39 EQQI601A Object: @OBJ-CLLUSER
08/22 10.47.39 EQQI601A Object: @OBJ-CLLUTS
08/22 10.47.39 EQQI601A Object: @OBJ-#CLSD
08/22 10.47.39 EQQI601A Object: @OBJ-CLSD-n-CLSDDATE
08/22 10.47.39 EQQI601A Object: @OBJ-CLSD-n-CLSDSTAT
08/22 10.47.39 EQQI601A Object: @OBJ-CLSD-n-CLSDDESC
08/22 10.47.39 EQQI601A Object: @OBJ-#CLWD
08/22 10.47.39 EQQI601A Object: @OBJ-CLWD-n-CLWDDAY
08/22 10.47.39 EQQI601A Object: @OBJ-CLWD-n-CLWDNUM
08/22 10.47.39 EQQI601A Object: @OBJ-CLWD-n-CLWDSTAT
08/22 10.47.39 EQQI601A Object: @OBJ-CLWD-n-CLWDDESC
08/22 10.47.39 EQQI299I Statement completed - RC=0 (00000014)
```

SHOW OPTIONS – Display Workload Automation Programming Language OPTIONS currently effective

Use the **SHOW OPTIONS** command to show each Workload Automation Programming Language option, with its current setting. Although some PIF options are ghosted in Workload Automation Programming Language, not all of them are; only the options with default settings in Workload Automation Programming Language that are different from native PIF are shown. This command is designed to show only the Workload Automation Programming Language settings.

The following example shows the output of the **SHOW OPTIONS** command:

```
03/15 14.42.42 EQQI602A OPTIONS in effect ADVALFROM(A) ARGUMENT()
            ...EQQI602A CALENDAR(CALENDAR) CCREMOVE(A) CHECK(Y) COMMIT(1000)
            ...EQQI602A COMEND(*/) COMSTART(/*) CPDEPR(N) CONTENTION(30,10)
            ...EQQI602A DATA() DATE(110315) DBMODE(ADD) DECODE(ONLY) DELAY(0)
```

```
            ...EQQI602A DELAYCMD(DELETE EXECUTE INSERT REPLACE) DELETE(N)
            ...EQQI602A DELFILE(OUTDEL) DLM(-END-OF-INPUT-TEXT-)
            ...EQQI602A DURUNIT(SECONDS) EXECUTE(AUTO) EXIT() EXITUSE(N)
            ...EQQI602A EXPAND(N) FIELDSEP( ) FIELDSPEC() FIRST(1)
            ...EQQI602A GTABLE(GLOBAL) HIGHRC(0) INCLEVEL(3) INPUT(SYSIN)
```

SHOW RC – Display return codes

Use the `SHOW RC` to list the return codes of each previous job step and Workload Automation Programming Language command.

The following example shows the output of the `SHOW RC` command:

```
08/08 23.48.49 EQQI200I SHOW RC
08/08 23.48.49 EQQI603A JOBNAME  JES NUM  STEPNAME PROCSTEP    RC PROGRAM
08/08 23.48.49 EQQI605A ADCDMSTT JOB05387          STEP0010  0004 EQQRETWM
08/08 23.48.49 EQQI605A ADCDMSTT JOB05387          STEP0020  0008 EQQRETWM
08/08 23.48.49 EQQI605A ADCDMSTT JOB05387          STEP0030 FLUSH EQQRETWM
08/08 23.48.49 EQQI605A ADCDMSTT JOB05387          STEP0040  SB37 EQQRETWM
08/08 23.48.49 EQQI605A ADCDMSTT JOB05387          STEP0050 U4095 EQQRETWM
08/08 23.48.49 EQQI605A ADCDMSTT JOB05387          STEP0060  SD37 EQQRETWM
08/08 23.48.49 EQQI605A ADCDMSTT JOB05387          STEP0070  S0C1 EQQRETWM
08/08 23.48.49 EQQI605A ADCDMSTT JOB05387          STEP0080  0006 EQQRETWM
08/08 23.48.49 EQQI605A ADCDMSTT JOB05387          STEP0090 FLUSH EQQRETWM
08/08 23.48.49 EQQI605A ADCDMSTT JOB05387 RUNWAPL  EQQYXTOP  EXEC IKJEFT01
08/08 23.48.49 EQQI605A
08/08 23.48.49 EQQI604A LABEL       RC LVL COMMAND
08/08 23.48.49 EQQI605A 00000007  0000  1  SHOW
08/08 23.48.49 EQQI605A LISTSTAT  0099  1  LISTSTAT
08/08 23.48.49 EQQI605A SETMAX1  FLUSH  1  SETMAX
08/08 23.48.49 EQQI605A SETMAX2  FLUSH  1  SETMAX
08/08 23.48.49 EQQI605A SETMAX3  FLUSH  1  SETMAX
08/08 23.48.49 EQQI605A SETMAX4   0004  1  SETMAX
```

SHOW SAVELIST – Display the contents of a SAVELIST

Use the `SHOW SAVELIST` command to list the contents of a saved list of results.

The following example shows the output of the `SHOW SAVELIST` command:

```
05/25 16.31.55 EQQI606A MYLIST has 12 entries.  Content follows -
05/25 16.31.55 EQQI607A GENDAYS DATE('120101') IAT('1000') FLAGS('NNYNNNN')
05/25 16.31.55 EQQI607A GENDAYS DATE('120201') IAT('1000') FLAGS('NNNNNNN')
05/25 16.31.55 EQQI607A GENDAYS DATE('120301') IAT('1000') FLAGS('NNNNNNN')
05/25 16.31.55 EQQI607A GENDAYS DATE('120401') IAT('1000') FLAGS('NNYNNNN')
05/25 16.31.55 EQQI607A GENDAYS DATE('120501') IAT('1000') FLAGS('NNNNNNN')
05/25 16.31.55 EQQI607A GENDAYS DATE('120601') IAT('1000') FLAGS('NNNNNNN')
05/25 16.31.55 EQQI607A GENDAYS DATE('120701') IAT('1000') FLAGS('NNYNNNN')
05/25 16.31.55 EQQI607A GENDAYS DATE('120801') IAT('1000') FLAGS('NNNNNNN')
05/25 16.31.55 EQQI607A GENDAYS DATE('120901') IAT('1000') FLAGS('NNYNNNN')
05/25 16.31.55 EQQI607A GENDAYS DATE('121001') IAT('1000') FLAGS('NNNNNNN')
05/25 16.31.55 EQQI607A GENDAYS DATE('121101') IAT('1000') FLAGS('NNNNNNN')
05/25 16.31.55 EQQI607A GENDAYS DATE('121201') IAT('1000') FLAGS('NNYNNNN')
```

SHOW SPE – Display active Small Product Enhancements

The **SHOW SPE** command lists all Small Product Enhancements that can affect how Workload Automation Programming Language processes and identifies which are active.

The following example shows the output of the **SHOW SPE** command. For each SPE, the command shows the version of the SPE when it becomes available, the SPE name, whether it is active (Y or N), and a simple description of the SPE itself:

```
03/15 14.42.42 EQQI608A 8.2   SPE  WLM=N - Workload Manager integration (SCHENV)
03/15 14.42.42 EQQI608A 8.2   SPE PEND=N - Allow ADSTAT within Batch Loader
03/15 14.42.42 EQQI608A 8.2   SPE   SA=N - System Automation integration
03/15 14.42.42 EQQI608A 8.3   SPE JCLV=N - JCL variable improvements
03/15 14.42.42 EQQI608A 8.3   SPE VIWS=N - Virtual Workstation support
03/15 14.42.42 EQQI608A 8.3   SPE   IP=N - TCP/IP Server Connect
03/15 14.42.42 EQQI608A 8.5.1 SPE USRF=Y - Operation User Fields
```

Note: After the version at which functionality is available has been reached, that functionality is automatically available for all later versions. Hence, if the previous example was output from a system V8.5.1, the function associated with the SPEs WLM, PEND, SA, JCLV, VIWS, and IP is automatically available, even if the individual SPEs have not been activated.

SHOW SYSINFO – Display information about the LPAR

Use the **SHOW SYSINFO** command to show information about the LPAR where the Workload Automation Programming Language job is running. This is useful to determine the correct values to be set for **OPTIONS TRACKERS**.

The following example shows the output of the **SHOW SYSINFO** command:

```
03/15 14.42.42 EQQI609A System information SYSNAME(ADCD) JESNODE(N1)
           ...EQQI609A SMFID(SYS1) SYSPLEX(ADCDPL) SYSID()
```

where:

SYSNAME

Current system name

JESNODE

Current JES Node name

SMFID

Current SMFID

SYSPLEX

Current Sysplex Name

SYSID

Name of the system field used to determine the tracker subsystem name

SHOW SUBSYSTEM – Display controller information

Use the **SHOW SUBSYSTEM** command to show information about the connected HCL Workload Automation for Z controller.

The following example shows the output of the **SHOW SUBSYSTEM** command:

```
03/15 14.42.42 EQQI610A Subsystem information  NAME(WSIC) VER(8.5.1)
            ...EQQI610A FMID(HWSZ510) CWBASE(00) HIGHDATE(711231)
            ...EQQI610A LOWDATE(720101) CALENDAR() CONLEVEL(00000001)
            ...EQQI610A FUNCLEVEL(00000006) ADDBCS(N) OWDBCS(N)
```

where:

NAME

Subsystem name of the controller.

VER

External version of the HCL Workload Automation for Z software.

FMID

Internal version.

CWBASE

Century window base.

HIGHDATE

Highest date HCL Workload Automation for Z considers part of this century.

LOWDATE

Lowest date HCL Workload Automation for Z considers part of last century.

CALENDAR

Default calendar name.

CONLEVEL

Connector software level.

FUNCLEVEL

Connector function level.

ADDBCS

Double byte character support in Application ID.

OWDBCS

Double byte character support in Owner ID.

SHOW USRF – Display Operation User Fields for this job

Use the **SHOW USRF** command to show the names and values of all Operation User Fields attached to the job.

The following example shows the output for the **SHOW USRF** command:

```
03/17 01.18.10 EQQI200I SHOW USRF
03/17 01.18.11 EQQI013A Job ADCDMSTU,JOB01991 in UFTEST 1409071119 CPU1_010
03/17 01.18.11 EQQI612A User field AUTHGROUP = ADCD
```

```
03/17 01.18.11 EQQI612A User field CATEGORY = C
03/17 01.18.11 EQQI299I Statement completed - RC=0
```

SHOW VARIABLES – Display current variable values

Use the **SHOW VARIABLE** command to show the current values of the Workload Automation Programming Language variables and the table to which they belong.

The following example shows the output for the **SHOW VARIABLES** command:

```
03/17 15.36.38 EQQI613A Variables BATRACHOLOGY101.MYCOUNT(2) MYFIRST(A)
            ...EQQI613A MYLAST(F) BATRACHOLOGY101.MYLEFT(HELLO)
            ...EQQI613A BATRACHOLOGY101.MYRIGHT(WORLD)
            ...EQQI613A BATRACHOLOGY101.MYVAR(HELLO WORLD) MY5TH(E)
            ...EQQI613A OJJESNO(JOB02038) OJJSTEP(RUNWAPL) OJOBNAME(ADCDMSTS)
            ...EQQI613A OJPSTEP(EQQYXTOP) OJWSA(BELOW)
```

The variables are listed in alphabetical order by name, unless the variable is assigned to a table. In this case, the table name is prefixed to the variable name. For example, `MYCOUNT` belongs to the table `BATRACHOLOGY101`, while variable `MY5TH` is not assigned to a table.

SUBROUTINE – Indicate the start of a subroutine

Use the **SUBROUTINE** statement to indicate the start of a new subroutine, which can be run by the **CALL SUB** command.

```
LABEL: SUBROUTINE
```

A label must precede a **SUBROUTINE** statement.

Subroutines must be coded at the end of the SYSIN. The **SUBROUTINE** statements themselves cannot be contained within an **INCLUDE** statement, but subroutines can contain **INCLUDE** statements.

If an **EXIT** statement is not coded at the end of the main program, the first **SUBROUTINE** statement will indicate an implicit **EXIT**.

TRANSLATE – Define rules for life-cycle translation

Use the **TRANSLATE** command to set up a series of rules to translate values of specific fields, or types of fields in the Batch Loader output generated by Workload Automation Programming Language. This command affects only Batch Loader output.

```
TRANSLATE       [[<type>|<field>] OLD(<name>) NEW(<name>)]
                [[<type>|<field>] LIST(<member>) [DLM(=|<char>)]]
                [[<type>|<field>] FILTER(<mask>) OVERLAY(<mask>)]
                [[<type>|<field>] RULES(<member>) [DLM(=|<char>)]]
                [OFF|ON]
```

A single **TRANSLATE** command can generate a single rule, or reference a set of rules, for an individual field or type of field. The following list shows the available types of field:

AD

Application name

CL

Calendar name

JS

Job name

OW

Owner ID

PR

Period name (includes Run Cycle Group Name)

SR

Special Resource Name

Note:

1. When processing **TRANSLATE** rules, Workload Automation Programming Language first checks if an **OLD**/**NEW** pair of keywords is defined for the field. A **FILTER**/**OVERLAY** pair is processed only if there is no matching pair of **OLD**/**NEW** keywords.
2. The content of the **ETTNAME** field is automatically considered to be a field of type **SR** when **ETTTYPE** is set to **R** and **JS** when **ETTYPE** is set to **J**.
3. The content of the **ADRPER** field is automatically considered to be a field type of PR when **ADRTYPE** is set to **N** or **X**, otherwise the field type is not set.
4. Periods contained within rule text are translated for run cycles in Applications and Run Cycle Groups.

The following restrictions apply to the **TRANSLATE** command:

- **TRANSLATE** rules can act only upon the value of the field being translated. You cannot define a rule that relies upon the value of other fields.
- The **ETTNAME** field can contain both job names and application names, if you define a rule specifically for the field **ETTNAME** you must consider this in the design of your rules, because the automatic detection of the content type will be overridden by your rule.
- The **ADRPER** field can contain both period names or just run cycle names, if you define a rule specifically for the field **ADRPER** you must consider this in the design of your rules, because the automatic detection of the content type will be overridden by your rule.

The following types of rule exist:

- Absolute rules look for an **OLD** value and exchange it for a **NEW** value. An entire set of absolute rules can be defined at the same time by using the **LIST** keyword to refer to an external member containing a line for each **OLD**/**NEW** pair separated by an equal sign (=).
- Filter rules look for entries matching the **FILTER** and **OVERLAY** a mask on the first match. A whole set of filter rules can be defined at the same time by using the **RULES** keyword to refer to an external member containing a line for each **FILTER**/**MASK** pair separated by the equal sign (=).
- To provide multiple absolute or filter rules for the same field type, you can set multiple **OLD/NEW** and **FILTER/OVERLAY** keyword pairs in the same **TRANSLATE** statement.

Note: For fields that contain an equal sign (=), you can remap the equal sign delimiter to another character by using the DLM keyword.

The **TRANSLATE OFF** command can be used to stop the existing rules being processed in any commands that follows. Then issue a **TRANSLATE ON** command to turn processing back on at a later point in the SYSIN. Defining any new rule with the **TRANSLATE** command will also turn translation processing back on, if it had been turned off.

When a field is being processed for translation, the following actions are taken:

1. The field value is checked for an entry in the absolute list. If one is found, the new value is substituted at this point, and the new value is passed on for filter processing.
2. Each filter value is checked, in the order they were defined, for a match. If a match is found then no more filter rules are evaluated.
3. If a match was made the **OVERLAY** value is used to modify the value.

The **FILTER** and **OVERLAY** keywords can be a combination of characters and the following wildcard characters:

- Percent sign (%) equates to any single character.
- Asterisk (*) equates to any number of characters.

For example:

`FILTER(Z*)` looks for anything beginning with Z.

`FILTER(%%P*)` looks for anything with P in the third position.

`FILTER(%%P*) OVERLAY(%%T*)` replaces P in the third character position with T.

The order of filter rules is important, you can use some rules to exclude certain values from processing, by giving the **FILTER** and the **OVERLAY** keywords the same value.

For example, you used names beginning with Z for all your dummy operations on a non-reporting workstation. These jobs never need to be modified. Their batch jobs begin with characters different from Z, and have the phase of the life-cycle as the third character in the job name. The following rules will convert test jobs to production jobs, without changing the names of the dummy jobs:

```
TRANSLATE JS FILTER(Z*) OVERLAY(Z*)
         FILTER(%%T*) OVERLAY(%%P*)
```

WAIT – Delay before continuing with the next command

Use the **WAIT** command to swap out the job for a specified amount of time.

WAIT hh:mm:ss

You can specify the time in the format *hh:mm:ss*, *mmm:ss*, or *sss*. If required, use a period (.) delimiter instead of a colon (:).

The time cannot exceed 9999 seconds in total.

WRITE – Echo information to a file or the external data queue

Use the **WRITE** command to echo information to a file or to the external data queue.

WRITE <file>[+|-] | * <expression>

where:

<file>

Any valid output DD statement.

[+|-]

Specifies whether the output file can be written by further commands or is to be closed, allowing the subsequent commands in the same step to read it. The plus sing (+) allows additional writing; the minus sign (-) closes the output file. The plus or minus sign must be appended immediately after the end of the DD statement. The default is the plus sign (+), meaning that a DD statement without a suffix allows subsequent writing to the file.

*****

Indicates write to the external data queue.

<expression>

Can be any valid REXX expression.

For more details about REXX expressions and available functions, see *TSO/E REXX Reference*.

For example:

- To write the text `Hello World` into the file referenced by the OUTDATA statement:

  ```
  WRITE OUTDATA “Hello World”
  ```

- To write `A B C` to the external data queue which would allow the data to be processed by a subsequent **READ** statement, **OPTIONS POSTPROC(Y)**, or a REXX program that called Workload Automation Programming Language:

  ```
  WRITE * “A B C”
  ```

- To write the contents of the **FILE** object variable **@MYOBJ** to the OUTDATA DD statement:

```
DO I = 1 TO !@MYOBJ
   VARSET THISLINE @V(@MYOBJ-!I)
   WRITE OUTDATA "!THISLINE"
END
```

- To use the REXX variable notation to avoid resolution issues:

```
DO FOREVER
   VARSCAN MYOBJ TARGET(Oh) CURSOR(RX,CX) DISTINCT(Y)
                CASE(N) COLS(001,080) ACTION(LEAVE)
   VARSET MYLINE VARIABLE(@MYOBJ-!RX)
   WRITE OUTDATA com.VAR.MYLINE
END
```

Note: Because the **WRITE** command exploits the REXX interpreter, it is sometimes better to use the REXX variable notation to avoid resolution errors. Because **OBJECT** variables are not accessible in REXX mode, use the **VARSET** command to transfer the value to a REXX variable without resolution issues.

Consider that:

- If a **WRITE** command is followed by a **READ** command to the same file, the output file is closed so that the **READ** command can read the data.
- When an output file is closed, either explicitly or by a subsequent READ command, if you write to it again, all previous output is lost and output is resumed at record 1. The only exception is when the file is a SYSOUT file, in which case all output data is kept.
- When an output file is closed, all buffered data for that file is committed to disk.

Chapter 4. Workload Automation Programming Language functions

Some commands in Workload Automation Programming Language operate in REXX mode. These commands are **CONSOLE**, **DISPLAY**, **DO**, **IF**, **LOG**, **SETVAR** when the equal sign (=) is used, and **WRITE**.

REXX mode allows Workload Automation Programming Language to take advantage of REXX functions and expressions, meaning that you can take advantage of both Workload Automation Programming Language and REXX functions. For more information about REXX expressions and functions, see the REXX reference manual.

Workload Automation Programming Language provides you with some functions, prefixed with the at sign (@), that you can use as part of an expression. However, you cannot use any REXX functions or expressions within the functions with the @ prefix. Like Workload Automation Programming Language variables, the functions with the @ prefix are resolved within quotes, before the command is run.

Care should be taken with Workload Automation Programming Language variables when passing to functions. If they contain blanks or mathematical expressions, this might cause problems. It is recommended that you enclose Workload Automation Programming Language variables within double quotes when they are part of functions or expressions, as in the following example:

```
VARSET MYVAR2 = SUBSTR(“!MYVAR1”,2,1
```

If the command resolves to a string longer than 250 characters, which would cause the REXX interpreter to consider the variable names as not valid, reference the Workload Automation Programming Language variables as REXX variables by using the **com.VAR.** prefix, as in the following example:

```
VARSET MYVAR2 = SUBSTR(com.VAR.MYVAR1,2,1)
```

The REXX versions of Workload Automation Programming Language variables are not resolved before running, and must follow REXX variable rules (for example, not being included within quotes). If a variable was not created or referenced by Workload Automation Programming Language ahead of using the **com.VAR** syntax, it will not search for the variable and return nothing.

As well as the functions with the @ prefix, Workload Automation Programming Language also provides additional functions that you can use together with REXX variables and expressions, with full nesting permitted.

Extending Workload Automation Programming Language by writing your own functions

For commands able to exploit the REXX interpreter, you can write your own functions in REXX to extend the capabilities of Workload Automation Programming Language. You do this by writing a REXX function routine, saving it in a member with the same name as the function, and including it in the SYSPROC concatenation.

The function can perform only REXX functions, it cannot reference any Workload Automation Programming Language commands or features. Such functions can perform string manipulation, or mathematical calculations and return a value back to Workload Automation Programming Language.

The following example shows a function named **MYFUNC** to add the **<** and **>** signs to either end of the string that is passed to it:

```
/* REXX */
PARSE ARG in_String
RETURN "<"||in_String||">"
```

You can reference it in `VARSET` in the same way as any REXX function:

```
11/21 07.25.28 EQQI200I VARSET FC = MYFUNC("WORLD")
11/21 07.25.28 EQQI033A Variable FC set to "<WORLD>"
11/21 07.25.28 EQQI299I Statement completed - RC=0
```

Note:

- The REXX function can be compiled or interpreted.
- The REXX function is loaded from the library every time it is called in a single running, therefore ensure good performance for the library if it is to be called many times.

@ - Date logic function

The @ function returns a boolean true or false (1 or 0), based on whether the input string is true for the day that is being evaluated.

The following string allows `IF` statements and other REXX based commands to make decisions, based on the characteristics of the date being checked.

`@("<input-string>",[<date>],["<additional-keywords>"])`

For example, the string `IF @(FRI) & @(WORKDAY) THEN` would only be true if the date being checked was both a Friday and a workday.

The following list shows the valid values for `<input-string>`:

- Day of the week: `MON TUE WED THU FRI SAT SUN`
- Part of the week: `WEEKDAY` or `WEEKEND`
- Day type: `WORKDAY` or `FREEDAY` (`W`, `WORK`, `F` and `FREE` are also valid)
- Day of the month: `D01` – `D31`
- Month of the year: `M01` – `M12`
- Name of the month: `JAN FEB MAR APR MAY JUN JUL AUG SEP OCT NOV DEC`
- Year: `Y00` – `Y99`
- *yymmdd*: a specific date
- A valid `ADRULE` string, for example `ONLY LAST(1) DAY(FRIDAY) MONTH`. For details about how you specify the ADRULE statement, see *Managing the Workload*.

Note: The `<input-string>` should be contained within single or double quotes if it contains `(` or `)`, otherwise quotes are optional.

By default, the date being checked is either the input arrival date for jobs controlled by HCL Workload Automation for Z, or the current date for jobs not controlled by HCL Workload Automation for Z. This date can be overridden by specifying a date as the second argument of the function. The date can either be a specific date in the format *yymmdd* or a relative date, using plus or minus to offset from the default date for this function.

For example, the string `IF @(FRI) & @(FREEDAY,+1) THEN` is true if the scheduled date for the job is a Friday and the following day is a Free day.

When checking `WORKDAY` and `FREEDAY`, the calendar to be used is determined as follows:

- If the job is controlled by HCL Workload Automation for Z, the calendar associated with the controlling occurrence is used.
- If the job is run outside of HCL Workload Automation for Z, the calendar specified by `OPTIONS CALENDAR` or `CALENDAR` set within EQQYPARM is used.
- If no calendar is set, the calendar named `DEFAULT` is used.

The `CALENDAR` can be overridden by specifying the `CALENDAR` keyword within the third argument. For example, the string `IF @(WORK,,"CALENDAR(NATIONAL)") & @(FREE,,"CALENDAR(LOCAL)") THEN` is true if the date being checked is a Workday in the National calendar and a Freeday in the Local calendar.

Note: The Work Day End Time is not considered for `WORKDAY` or `FREEDAY`. To exploit Workday End Time, you can use the `ADRULE` statement. Where you use an `ADRULE` statement, the rule is checked to see if it would generate a match for the date being checked.

The following list shows the additional keywords that you can specify as the third argument to influence how the rule is evaluated:

`CALENDAR`

Sets the calendar to use. If not specified, the calendar is decided using the same method as with `WORKDAY` and `FREEDAY` input strings.

`FDAYRULE(1|2|3|4|E)`

Sets the free day rule. By default, this is set to 3 which does not adjust the date.

`IAT(hhmm)`

Sets the input arrival time to evaluate against the rule, for Calendar Work Day End Time processing. By default, this will use the input arrival time of the job running the command, if it is controller by HCL Workload Automation for Z, otherwise it will use 0000 for a job outside of HCL Workload Automation for Z control.

For example:

- **@("EVERY DAY(FREEDAY) WEEK",,"CALENDAR(MYCAL) IAT(0300)")** evaluates whether 3am on the date being checked is a Freeday, taking into account the Work Day End Time of calendar **MYCAL**.
- **@("ONLY LAST(1) DAY(FRIDAY) MONTH",-3)** evaluates whether it was the last Friday of the month 3 days ago.
- **@("ONLY LAST(1) DAY(FRIDAY) MONTH",,"FDAYRULE(1)")** evaluates whether this is the last working day before or on the last Friday of the month.

Note: The **<additional-keywords>** must be specified within single or double quotes, because it contains opening and closing parenthesis.

The date logic expressions can also be used in the **CRITERIA** user fields for the **ALTIF** and **RUNIF** commands.

@CMD and @JCL – Check RC of previous command or JCL step

This function compares the return code of a previous Workload Automation Programming Language command, or a previous JCL step in the same job, with a specified value.

The command returns 1 if the comparison is true, or 0 if the comparison is false.

@CMD([<label>.]EQ|NE|LT|LE|GT|GE.]<value>)

@JCL([[<label>.]EQ|NE|LT|LE|GT|GE|EM|NM.]<value>)

where:

<label>

Workload Automation Programming Language label of the command or step name within the job currently running.

EQ

Is true if the return code is equal to **<value>**.

NE

Is true if the return code is not equal to **<value>**.

LT

Is true if the return code is less than **<value>**.

LE

Is true if the return code is less than equal to **<value>**.

GT

Is true if the return code is greater than **<value>**.

GE

Is true if the return code is greater than or equal to **<value>**.

EM

Is true if the abend matches the mask specified as **<value>** (**@JCL** only).

NM

Is true if the abend does not match the mask specified as **<value>** (**@JCL only**).

<value>

The value with which the return code or abend is being compared.

For **@JCL**, the **<label>** can be *STEPNAME.PROCSTEP* or *PROCSTEP*, where *PROCSTEP* is the actual step name running the program (whether it is in a job or a procedure), and *STEPNAME* is the name of the step calling the procedure where the step runs. If the same *PROCSTEP* occurs more than once, the return code of the latest step is returned if only the *PROCSTEP* is specified. If more than one combination of *STEPNAME.PROCSTEP* exists within the job, the latest is used.

Examples:

- **@JCL(STEP0040.EQ.0)** allows a command to run when STEP0040 has a return code of 0.
- **@JCLL(RUNWAPL.EQQYXTOP.EQ.4)** allows a command to run when a step called RUNWAPL calling the Workload Automation Programming Language procstep EQQYXTOP ends with return code of 4.
- **@JCL(EQQYXTOP.EQ.4)** allows a step to run when the latest step using the name EQQYXTOP occurs.

For both **@JCL** and **@CMD**, a positive number can be used for **<label>** to reference an absolute step or command number, for example 1 for the first step, 3 for the third command. Negative numbers can be used for relative steps or command, for example -1 for the immediately previous step, -2 for the command before the last one.

Note:

- Workload Automation Programming Language automatically runs commands on your behalf, and some commands may internally run other commands to achieve their goal. Use **SHOW RC(5)** to see all the commands that have run up to that point. Using **SHOW RC** without specifying a message level, lists all commands up to and including the current value of **OPTIONS MSGLEVEL**.
- The **SHOW RC** command lists the names of all steps in the job, including the Workload Automation Programming Language step currently running. This step, however, is not used when looking up steps by name, because it is not a preceding step. You can reference the step currently running by the relative address of 0, as follows: **VARSET THISSTEP ENVATTR(JCL,STEP,0)**, and then use **VARSET** to find out details of the running step.

To receive step information, you can use the following special values for **<label>** in **@CMD**, **@JCL** and **VARSET**:

LAST_RC

Refers to the previous JCL step or Workload Automation Programming Language command.

LAST_XRC

Refers to the preceding JCL step or Workload Automation Programming Language command that was actually run.

MAX_RC

Refers to the step that set the maximum return code.

MAX_RESP

Refers to the step that set the maximum response code (for **LISTSTAT**).

For **@JCL**, the **MAX_RC** label points to the step issuing the highest return code, unless an ABEND has occurred, when this will be the last abended step, to allow for situations where **ONLY** and **EVEN** is used. For **MAX_RC** and **MAX_RESP**, if **SETMAX** is used to lower the maximum return code, it will point to the step issuing the highest return code from that point, which could be the **SETMAX** statement itself.

If the **<label>** is not specified, the last executed step or command is used (**LAST_XRC**). This default can be altered by **OPTIONS IFJCL** or **OPTIONS IFCMD** as appropriate. For example, **@CMD(EQ.0)** would allow only a command to run if the last command that was not flushed ended with a return code of 0.

If both **<label>** and the comparator are not specified, **<label>** defaults to **LAST_XRC** and the comparator defaults to **EQ**. For example, **@CMD(0)** would allow only a command to run if the last executed command ended with return code 0.

@LOG – Return the date and time in EQQYLOG format

This function provides the current date and time in the same format used in the EQQYLOG. It is a function with no arguments, but the parentheses must be coded.

For example, **DISPLAY @LOG() “Hello world”**

08/24 23.06.55 Hello World

@V – Return the value of a named variable

This function returns the value of a named variable.

@V(<variable-name>)

where **<variable-name>** is the name of the variable to return. The **VARSUB** prefix (for example, the exclamation mark) must be coded only if you want to return the value of a variable named within a variable.

Variable substitution does not need to be active for **@V** to work. Variable substitution can be used as a mechanism to subscript variables, such as object variables.

For example, the command:

```
VARSUB SCAN(!)
SELECT AD ADID(DAILYPLANNING) OBJECT(FREDDO)
DO X = 1 TO !@FREDDO-#ADOP
```

```
   DISPLAY @V(@FREDDO-ADOP-!X.-ADOPWSID) @V(@FREDDO-ADOP-!X.-ADOPJN)
END
```

produces output displaying workstation and job name for each operation in the object:

```
NONR ZFIRST
CPU1 WSLCLTEX
CPU1 WSLCCPEX
NONR ZLAST
```

ADDWORD – Add words to a list

This function adds one or more words to a list, if they are not already existing in the list.

```
ADDWORD(<list>,<new-items>)
```

Where:

`<list>`

The current list of words.

`<new-items>`

The new words to be added to the list, separated by blanks.

For each word defined in `<new-items>`, a check is performed to see if it already exists with the same format in the `<list>`. The function returns the original list, with any`<new-items>` not already contained within it appended to the list.

For example:

```
VARSUB SCAN
VARSET IR = "SCOTT VIRGIL ALAN GORDON JOHN"
VARSET IR = ADDWORD("!IR","BRAINS TIN TIN") variable IR set to
GORDON JOHN BRAINS "SCOTT VIRGIL ALAN TIN"
VARSET VARSET IR = IR = ADDWORD("!IR","ALAN GEOFF")  variable IR set to
"SCOTT VIRGIL ALAN GORDON JOHN BRAINS TIN GEOFF”
```

CONTAINS – Search for certain characters in a string

This function returns the position of the first character, from those defined in a list, that is found in a string.

```
ADDWORD(<string>,<characters>)
```

Where:

`<string>`

The string to search.

`<characters>`

The list of characters to be searched for, within the string.

The following command returns **8**:

```
CONTAINS(“THE QUICK BROWN FOX”,”ABC”)
```

The following command returns **1**:

```
CONTAINS(“Spectrum Is Green”,”SIG”)
```

The following command returns **0**:

```
CONTAINS(“ABC DEF”,”GHI”)
```

DATEADD – Add or subtract days to or from a date

This function adds or subtracts a number of days to or from a date.

```
DATEADD(<date>,<n>, [<informat>],[<outformat>])
```

Where:

`<date>`

The date from which you want to add or subtract days.

`<n>`

The number of days to be added. Use a negative integer to subtract.

`<informat>`

The format of the input date. The default is *YYMMDD*.

`<outformat>`

The format of the output date that is created. The default is the same format of the input date.

The format strings can be made up as follows:

YYYY

Year, in the complete format.

YY

Last two digits of the year.

CC

Century part of the year.

MM

Month

MMM

First three characters of the month, in English and upper-case format. For example, `JAN` for January.

Month_name

Complete name of the month. Valid only for the output date.

W

Numeric day of the week, starting from 1 for Monday.

WW

First two characters of the day, in English and upper-case format. For example, `MO` for Monday.

WWW

First three characters of the day, in English and upper-case format. For example, `MON` for Monday.

DD

Day.

Day_name

Complete name of the day of the week. Valid only for the output date.

Any other characters is considered literally in the date format. For example, the string `DD/MM/YY` generates a date like `31/10/21`.

The following command returns 200229:

```
DATEADD(“200228”,”1”)
```

The following command returns 210301:

```
DATEADD(“210228”,”1”)
```

The following command returns 01/01/22:

```
DATEADD("25/12/21",7,"DD/MM/YY")
```

The following command returns 01-02-2022:

```
DATEADD("25/12/21",8,"DD/MM/YY","MM-DD-YYYY")
```

Consider that:

- If there is no date part in the input format, but it is in the output format, they will be set to their format placeholder for each field. For example, *MM*.
- The equal sign (=) can be used in place of a date to indicate the current date.
- ***+n*** can be used in place of a date to indicate *n* days after the current date.
- ***-n*** can be used in place of a date to indicate *n* days before the current date.
- In all cases the relative dates must be contained in quotes for function calls.
- To use textual months in another language, use the `OPTIONS MMMM` keyword to set months in an alternative language.
- If the text version of a month or weekday does not match any known month or weekday, their format placeholder will be returned as question marks in the output. For example `??`

- If the month or weekday placeholders are in lower case in the output format, the month or day will be output in mixed case. For example, `Mon`.
- The day of the week is always calculated from the date part of the input date, regardless of what day could be in the input string.

DATEDIFF – Calculate the difference between two dates

This function returns the difference in days between two dates.

```
DATEDIFF(<date1>,<date2>)
```

Where:

`<date1>`

The date from which to subtract.

`<date2>`

The date to be subtracted.

`<date2>` is subtracted from `<date1>`. If `<date2>` is greater than `<date1>`, the result will be negative.

Both dates must be in the format *YYMMDD*. Use the `DATEFORM` function to convert the format, if needed.

The following command returns 7992:

```
DATEDIFF("211118","000101")
```

The following command returns the number of days since day 1 of January 2000:

```
DATEDIFF("=","000101")
```

Consider that:

- The equal sign (=) can be used in place of a date to indicate the current date.
- ***+n*** can be used in place of a date to indicate *n* days after the current date.
- ***-n*** can be used in place of a date to indicate *n* days before the current date.
- In all cases the relative dates must be contained in quotes for function calls.

DATEFORM – Change the format of a date variable

This function changes a date and/or a time to a new format.

```
DATEFORM(<date>,<informat>,<outformat>)
```

Where:

<date>

The date and/or the time whose format is to be changed.

<informat>

The format of the input date.

<outformat>

The format of the output date that is to be created.

The format strings can be made up as follows:

YYYY

Year, in the complete format.

YY

Last two digits of the year.

CC

Century part of the year.

MM

Month

MMM

First three characters of the month, in English and upper-case format. For example, **JAN** for January.

Month_name

Complete name of the month. Valid only for the output date.

W

Numeric day of the week, starting from 1 for Monday.

WW

First two characters of the day, in English and upper-case format. For example, **MO** for Monday.

WWW

First three characters of the day, in English and upper-case format. For example, **MON** for Monday.

DD

Day.

HH

Hour.

NN

Minutes.

SS

Seconds.

XX

Hundredth of seconds.

Consider that:

- If there is no time part in the input format, but it is in the output format, the values will be set to 00 for each field.
- If there is no date part in the input format, but it is in the output format, they will be set to their format placeholder for each field. For example, `MM`.
- The equal sign (=) can be used in place of a date to indicate the current date.
- ***+n*** can be used in place of a date to indicate *n* days after the current date.
- ***-n*** can be used in place of a date to indicate *n* days before the current date.
- In all cases, the relative dates must be contained in quotes for function calls.
- To use textual months in another language, use the `OPTIONS MMMM` keyword to set months in an alternative language.
- If the text version of a month or weekday does not match any known month or weekday, their format placeholder will be returned as question marks in the output. For example `??`
- If the month or weekday placeholders are in lower case in the output format, the month or day will be output in mixed case. For example, `Mon`.
- The day of the week is always calculated from the date part of the input date, regardless of what day could be in the input string.

DROPWORD – Delete a single word from a string

This function deletes a word from a space delimited string, if it exists.

```
DROPWORD(<needle>,<haystack>)
```

Where:

`<needle>`

The word for which to search.

`<haystack>`

The string where to search.

If the function finds `<needle>` as a space delimited word within `<haystack>`, it returns `<haystack>` with that word deleted. Otherwise it returns `<haystack>` unchanged. The command is case sensitive.

If `<needle>` occurs more than once, only the first occurrence is deleted.

ENVATTR – Environment attributes

This function returns environment attributes, for example return codes from previous steps or data set names attached to DD statements.

This function works in the same way as the `ENVATTR` keyword of the `VARSET` command. For more details, see VARSET – Set a Workload Automation Programming Language variable on page 349

Where:

FLAG – Check if flag is set

This function checks if binary flags are set within a byte, but performed in decimal number form, to avoid having to convert values to binary. The function returns **1** if the bits in `<check>` are also set in`<source>`. On the contrary, it returns **0**.

```
FLAG(<source>,<check>)
```

Where:

`<source>`

The string to search.

`<check>`

The list of characters to be searched for, within the string.

For example, 7 is binary 00000111, 4 is binary 00000100 and 3 is binary 00000011. The following command returns **0**:

```
FLAG(7,4)
```

The following command returns `1`:

```
FLAG(7,3)
```

The following command returns `0`:

```
FLAG(4,3)
```

GETHHMM – Convert minutes to hours and minutes

This function converts a number of minutes into the *HHMM* format.

```
GETHHMM(<minutes>)
```

Where `<minutes>` is the number of minutes to convert.

For example, `GETHHMM(72)` returns **0112**.

GETMINS – Convert minutes to hours and minutes

This function converts hours and minutes expressed in the *HHMM* format into the number minutes.

```
GETMINS(<hhmm>)
```

Where `<hhmm>` is the number of hours and minutes.

For example, `GETMINS(72)` returns **1438**.

IS – Validate input type

This function confirms if the input data belongs to the specified data type. It returns **1** if the answer is affirmative. On the contrary, it returns **0**.

```
IS(<type>,<value>)
```

Where:

`<type>`

The type of field. Allowed values are:

DATE

Date in the format *YYMMDD*

IA

Input arrival in the format *YYMMDDHHMM*

LABEL

Workload Automation Programming Language label

OBJECT

Workload Automation Programming Language object name

TIME

Input arrival in the format *HHMM*

VARNAME

Workload Automation Programming Language variable name

`<value>`

The data for whose type you want to determine.

For example, the following command returns `0`:

```
IS(“DATE”,211324)
```

The following command returns `1`:

```
IS(“TIME”,2359)
```

While `DATE` and `TIME` rely on a specific input format, you can use `DATEFORM` to convert the input for testing. For example:

The following command returns `0`:

```
IS("DATE",DATEFORM(311321,"DDMMYY","YYMMDD"))
```

The following command returns `1`:

```
IS("DATE",DATEFORM("31/10/21","DD/MM/YY","YYMMDD"))
```

The following command returns `1`:

```
IS("TIME",DATEFORM("21:12","HH:NN","HHNN"))
```

ISDSN – Confirms whether a data set and/or member exists

This function returns **1** if a data set exists, or **0** if it does not.

```
ISDSN(<dsname>)
```

Where `<dsname>` is the name of the data set to check.

For example, `ISDSN(“MY.DATASET”)` returns **1** if `MY.DATASET` exists, otherwise returns **0**.

LOWER – Confirms whether a data set and/or member exists

This function converts the non-accented uppercase characters found in a string to lowercase format.

```
LOWER(<string>)
```

Where `<string>` is the string to convert.

For example, the following command returns `"fab"`:

```
LOWER(“FAB”)
```

The following command returns `"scott virgil alan gordon john"`:

```
LOWER(“Scott Virgil Alan Gordon John”)
```

NODE – Return a node from a string

This function returns a node from a string, using a specified delimiter.

```
NODE(<string>,<delimiter>,<node-number>)
```

Where:

<string>

String from which to extract the data.

<delimiter>

String that delimits each node.

<node-number>

Node number.

Unlike the functions **WORD** and **WORDX**, which ignore leading and double spaces, the **NODE** function counts every delimiter. For example, the following command returns **"FOUR"**:

```
NODE(“FIVE!FOUR!THREE!TWO!ONE”,”!”,2)
```

The following command returns **"A"**:

```
NODE(“.A.B..C.D”,”.”,2)
```

The following command returns **"C"**.

```
NODE(“.A.B..C.D”,”.”,5)
```

NODES – Return the number of nodes in a string

This function returns the number of nodes within a string, using a specified delimiter.

```
NODE(<string>,<delimiter>)
```

Where:

<string>

String from which to extract the data.

<delimiter>

String that delimits each node.

Unlike the functions **WORD** and **WORDX**, which ignore leading and double spaces, the **NODES** function counts every delimiter. For example, the following command returns **5**:

```
NODES(“FIVE!FOUR!THREE!TWO!ONE”,”!”,2)
```

The following command returns **6**:

```
NODES(“.A.B..C.D”,”.”,2)
```

OP – Format an operation number

This function formats operation numbers with leading zeros, according to the maximum width allowed for the operation field.

```
OP(<oper>)
```

Where `<oper>` is the operation number.

For example, the command `OP(1)` returns `"001"`, the command `OP(22)` returns `"022"`.

REMAIN – Return the remainder from division

This function performs division on two numbers, and returns the remainder.

```
REMAIN(<numerator>,<divider>)
```

Where:

`<numerator>`

The number on which to perform the division.

`<divider>`

The number by which to divide the `<numerator>`.

For example, the command `REMAIN(15,6)` returns `3`, the command `REMAIN(24,12)` returns `0`.

REMOVE – Remove characters from a string

This function deletes the specified characters from a string and returns the modified input string.

.

```
REMOVE(<string>,<characters>)
```

Where:

`<string>`

The string to be modified.

`<characters>`

The character or characters to be deleted.

For example, the command `REMOVE(“24/01/2022”,”/”)` returns `"240122"`. The command `REMOVE(“Hello, is it me you’re looking for?”,”,?’”` returns `"Hello is it me youre looking for"`.

SEC100 – Convert hundredths of a second to *HHMMSSXX*

This function converts hundredths of a second to hours, minutes, seconds and hundredths of seconds.

```
SEC100(<hundredths>)
```

Where `<hundredths>` is time in hundredths of a second.

For example, the following command returns `"02464000"`:

```
SEC100(“1000000”)
```

UPPER – Format an operation number

This function converts the non-accented lowercase characters within a string into the uppercase format.

```
UPPER(<string>)
```

Where `<string>` is the string whose format is to be converted.

For example, the command `UPPER(“fab”)` returns `"FAB"`, the command `UPPER(“Scott Virgil Alan Gordon John”)` returns `"SCOTT VIRGIL ALAN GORDON JOHN”`.

WORDSX – Extend the REXX WORDS function

This function extends the REXX WORDS function to enable you delimit words with characters different from a blank. As with the `WORDS` function, the leading delimiters are ignored and consecutive delimiters treated as one, returning the number of delimited word. If the number is greater than the number of words, a null string is returned. The input string must not contain the “00”x character.

```
WORDSX(<string>,<delimiter>,<number>)
```

Where:

`<string>`

The string from which to extract the words.

`<delimiter>`

The delimiter with which to separate each word.

For example, the following command returns `5`:

```
WORDSX(“FIVE!FOUR!THREE!TWO!ONE”,”!”,2)
```

The following command returns `4`:

```
WORDSX(“.A.B..C.D”,”.”,2) returns 4
```

WORDX – Extend the REXX WORD function

This function extends the REXX WORD function to enable you delimit words with characters different from a blank. As with the `WORD` function, the leading delimiters are ignored and consecutive delimiters treated as one, returning the number of

delimited word. If the number is greater than the number of words, a null string is returned. The input string must not contain the “00”x character.

```
WORDX(<string>,<delimiter>,<number>)
```

Where:

`<string>`

The string from which to extract the words.

`<delimiter>`

The delimiter with which to separate each word.

`<number>`

The word number to return,

For example, the following command returns `"FOUR"`:

```
WORDX(“FIVE!FOUR!THREE!TWO!ONE”,”!”,2)
```

The following command returns `"B"`:

```
WORDX(“.A.B..C.D”,”.”,2)
```

The following command returns `""`:

```
WORDX(“.A.B..C.D”,”.”,5) returns “”
```

Chapter 5. Data Access commands based on PIF

The Data Access commands supported by Workload Automation Programming Language are low level commands based on the HCL Workload Automation for Z programming interface (PIF).

PIF provides access to the HCL Workload Automation for Z database and plans at a record and segment level. Because Workload Automation Programming Language is cross version compliant, see *Developer's Guide: Driving HCL Workload Automation for Z* for notes, restrictions, or limitations that apply to the version you are using.

DELETE – Delete object from database or plan

Use the **DELETE** request to delete a record or record segment.

```
DELETE <resource> <arguments>
```

If you are deleting a record, the arguments identify the specific record to be deleted. To delete only some information within an occurrence (for example, one of its operations), you must first use a **MODIFY** request to identify the occurrence, then a **DELETE** request to delete the operation.

To delete a special resource specification for an operation, you must first use a **MODIFY** request to identify the occurrence, then a **MODIFY** request to identify the operation, and finally a **DELETE** request to delete the special resource.

To delete an interval of a current plan workstation, you must precede the **DELETE IVL** with a **MODIFY CPWS** to identify the workstation.

To delete the extended name of an operation, you must use the **MODIFY** request.

If the **DELETE** request was used to modify information in the current plan, a later **EXECUTE** request must be issued for the modification to take effect: Workload Automation Programming Language will do this automatically if **OPTIONS EXECUTE(AUTO)** is set (this is the default).

Return codes have the following meaning:

0

The request was successful.

4

The record is currently being updated by another user. The record is not deleted.

8

The request was unsuccessful. An error message is issued to the message log data set.

DELETE AD – Application Definition

Table 9. DELETE AD – Application Definition

Argument Name	Description
ADID	Application description ID
GROUP	Authority group name
GROUPDEF	Group definition ID
OWNER	Owner ID
PRIORITY	Priority
STATUS	Status: • P=Pending • A=Active
VALFROM	Valid-from date

Note: HCL Workload Automation for Z assumes application type A, if you do not specify the `TYPE` argument name.

DELETE AWSCL – All Workstations Closed

Table 10. DELETE AWSCL – All Workstations Closed

Argument Name	Description
DATE	Date workstations are closed *YYMMDD*

DELETE CL – Calendar

Table 11. DELETE CL – Calendar

Argument Name	Description
CALENDAR	Calendar ID

DELETE CPCOND – Condition

Table 12. DELETE CPCOND – Condition

Argument Name	Description
CONDID	Condition ID (1-999)

Note: Resource `CPCOND` is valid only for the `DELETE` request starting from HCL Workload Automation for Z version 8.5, or later.

DELETE CPLAT – Operation user-defined late information

There are no arguments for the DELETE CPLAT request.

DELETE CPOC – Current Plan Occurrence

Table 13. DELETE CPOC – Current Plan Occurrence

Argument Name	Description
ADID	Application description ID
IA	Input arrival date and time *YYMMDDHHMM*

DELETE CPOCPRE – Current Plan Occurrence Predecessor

Table 14. DELETE CPOCPRE – Current Plan Occurrence Predecessor

Argument Name	Description
PREADID	Predecessor application ID.
PREIA	Predecessor input arrival date and time *YYMMDDHHMM*.
PREOPNO	Predecessor operation number. If the predecessor is an occurrence, you do not need to set this argument.

DELETE CPOCSUC – Current Plan Occurrence Successor

Table 15. DELETE CPOCSUC – Current Plan Occurrence Successor

Argument Name	Description
SUCADID	Successor application ID.
SUCIA	Successor input arrival date and time *YYMMDDHHMM*.
SUCOPNO	Successor operation number. If the successor is an occurrence, you do not need to set this argument.

DELETE CPOP – Current Plan Operation

Table 16. DELETE CPOP – Current Plan Operation

Argument Name	Description
OPNO	Operation number

DELETE CPPRE – Current Plan Predecessor

Table 17. DELETE CPPRE – Current Plan Predecessor

Argument Name	Description
PREADID	Predecessor application ID.
PREIA	Predecessor input arrival date and time *YYMMDDHHMM*.
PREMAND	Whether the predecessor is required. The allowed values are: Y or N (default).
PREOPNO	Predecessor operation number. If the predecessor is an occurrence, you do not need to set this argument.

Note: When deleting an internal predecessor, specify only `PREOPNO`. To delete an external predecessor, specify all the arguments (omit PREMAND if the predecessor is not required).

DELETE CPSIMP – Conditional predecessor

Table 18. DELETE CPSIMP – Conditional predecessor

Argument Name	Description
PREADID	Predecessor application name.
PREIA	Predecessor input arrival date and time *YYMMDDHHMM*.
PREOPNO	Predecessor operation number.
PROCSTEP	Use it to define a step level dependency. If the step is not in a procedure, this parameter identifies the job step name, otherwise it identifies the step name in the JCL procedure. It must correspond to a step specifying the EXEC PGM= statement.
STEPNAME	Use it in conjunction with PROCSTEP when defining a step level dependency, only if the step is in a procedure, to identify the procedure invocation step name.
TYPE	Condition type: **RC** To check predecessor return code **ST** To check predecessor status

Table 18. DELETE CPSIMP – Conditional predecessor (continued)

Argument Name	Description
LOG	Logical operator: **GE** Greater than or equal to (TYPE=RC only) **GT** Greater than (TYPE=RC only) **LE** Less than or equal to (TYPE=RC only) **LT** Less than (TYPE=RC only) **EQ** Equal to **NE** Not equal to **RG** Range (TYPE=RC only)
VALRC	Return code value, or lower limit of a return code value range when TYPE=RC & LOG=RG
VALRC2	Upper limit of a return code value range when TYPE=RC & LOG=RG
VALST	Status (TYPE=ST only)

Note:

1. Resource **CPSIMP** is valid only for the **DELETE** request starting from HCL Workload Automation for Z version 8.5, or later.
2. Before a **DELETE CPSIMP** request, always identify the occurrence, operation, or condition by using:
   - An **INSERT** or **MODIFY CPOC** request
   - An **INSERT** or **MODIFY CPOP** request
   - An **INSERT** or **MODIFY CPCOND** request

DELETE CPSR – Current Plan Operation Special Resource

Table 19. DELETE CPSR – Current Plan Operation Special Resource

Argument Name	Description
RESNAME	Special Resource name

DELETE CPSUC – Current Plan Successor

Table 20. DELETE CPSUC – Current Plan Successor

Argument Name	Description
SUCADID	Successor application ID.
SUCIA	Successor input arrival date and time *YYMMDDHHMM*.
SUCOPNO	Successor operation number. If the successor is an occurrence, you do not need to set this argument.

Note: To delete an internal successor, specify only `SUCOPNO`. To delete an external successor, specify all the arguments.

DELETE CPUSRF – User Field

Table 21. DELETE CPUSRF – User Field

Argument Name	Description
UFNAME	User Field Name.

Note: Resource `CPUSRF` is valid only for the `DELETE` request.

DELETE ETT – Event Trigger

Table 22. DELETE ETT – Event Trigger

Argument Name	Description
ADID	Associated application ID
ETTNAME	Name of trigger
ETTTYPE	Type of trigger: **J** Job

Table 22. DELETE ETT – Event Trigger (continued)

Argument Name	Description
	R Special resource

DELETE IVL – Current Plan Workstation Interval

If an interval contains information originating from the workstation description, the indicator CPIVLDP in segment CPIVL is set to Y otherwise it is set to N. If an interval is changed or created by using the dialog or program interface, the indicator CPIVLMOD in CPIVL is set to Y, otherwise is set to N. **`DELETE IVL`** affects only modifications. Intervals with CPIVLDP=Y remain after a **`DELETE`**, the interval is reset to the daily planning values and CPIVLMOD is set to N. Intervals with CPIVLDP=N are completely deleted.

Table 23. DELETE IVL – Current Plan Workstation Interval

Argument Name	Description
FROM	Interval start date and time *YYMMDDHHMM*

DELETE JCLV – JCL Variable Table

Table 24. DELETE JCLV – JCL Variable Table

Argument Name	Description
JCLVTAB	JCL variable table ID

DELETE JL – Job Log

Table 25. DELETE JL – Job Log

Argument Name	Description
ADID	Application ID
IA	Input arrival date and time *YYMMDDHHMM*
JOBNAME	z/OS Job name
OPNO	Operation number
WSNAME	Workstation name

DELETE JS – Current Plan JCL

Table 26. DELETE JS – Current Plan JCL

Argument Name	Description
ADID	Application ID
IA	Input arrival date and time *YYMMDDHHMM*
JOBNAME	z/OS job name
OPNO	Operation number
WSNAME	Workstation name

DELETE LTOC – Long-Term Plan Occurrence

Table 27. DELETE LTOC – Long-Term Plan Occurrence

Argument Name	Description
ADID	Application description ID
IAD	Input arrival date *YYMMDD*
IAT	Input arrival time *HHMM*

DELETE LTCPRE – LTP Conditional Predecessor

Table 28. DELETE LTCPRE – LTP Conditional Predecessor

Argument Name	Description
ADID	Application description ID
IAD	Input arrival date *YYMMDD*
IAT	Input arrival time *HHMM*
PREADID	Conditional predecessor application ID
PREIAD	Conditional predecessor input arrival date *YYMMDD*
PREIAT	Conditional predecessor input arrival time *HHMM*

Note: Resource `LTCPRE` is valid only for the `DELETE` request starting from HCL Workload Automation for Z version 8.5, or later.

DELETE LTPRE – Long-Term Plan Predecessor

Table 29. DELETE LTPRE – Long-Term Plan Predecessor

Argument Name	Description
ADID	Application description ID
IAD	Input arrival date *YYMMDD*
IAT	Input arrival time *HHMM*
PREADID	Predecessor application ID
PREIAD	Predecessor input arrival date *YYMMDD*
PREIAT	Predecessor input arrival time *HHMM*

Note: `DELETE LTPRE` is used only to delete external predecessors. No support is provided in the long-term plan for internal dependencies.

DELETE OI – Operator Instruction

Table 30. DELETE OI – Operator Instruction

Argument Name	Description
ADID	Application description ID
OPNO	Operation number

DELETE PR – Period

Table 31. DELETE PR – Period

Argument Name	Description
PERIOD	Period name
PRTYPE	Period type

DELETE RGCOM – Run Cycle Group

Table 32. DELETE RGCOM – Run Cycle Group

Argument Name	Description
RGID	Run cycle group ID
RGOWNER	Run cycle group owner
RGCALEND	Run cycle group calendar

Table 32. DELETE RGCOM – Run Cycle Group (continued)

Argument Name	Description
RGVARTAB	Run cycle group variable table
RUNNAME	Run cycle name
RUNCAL	Run cycle calendar
RUNVTAB	Run cycle variable table
RUNSETID	Run cycle subset ID

DELETE SR – Special Resource

Table 33. DELETE SR – Special Resource

Argument Name	Description
RESGROUP	Special resource group
RESHIPER	DLF resource indicator
RESNAME	Special resource name

DELETE VIVL – CP Virtual Workstation Interval

If an interval contains information originating from the Virtual Workstation Destination description, the indicator CPVIVLDP in segment CPVIVL is set to Y, otherwise it is set to N. If an interval is changed or created by using the dialog or the program interface, the indicator CPVIVLMOD in segment CPVIVL is set to Y, otherwise it is set to N.

`DELETE VIVL` affects only modifications. Intervals with CPVIVLDP=Y remain after a `DELETE`, the interval is reset to the daily planning values and CPVIVLMOD is set to N. Intervals with CPVIVLDP=N are completely deleted.

Table 34. DELETE VIVL – CP Virtual Workstation Interval

Argument Name	Description
FROM	Interval start date and time *YYMMDDHHMM*

Note: Resource `VIVL` is valid only for the `DELETE` request starting from HCL Workload Automation for Z version 8.5, or later.

DELETE WS – Workstation

Table 35. DELETE WS – Workstation

Argument Name	Description
WSNAME	Workstation name
WSREP	Workstation reporting attribute
WSRETYPE	Remote engine type: **D** HCL Workload Automation **Z** HCL Workload Automation for Z **blank**
WSTYPE	Workstation type
WSWAIT	WAIT workstation, Y or N

DELETE WSV – Virtual workstation destination

Table 36. DELETE WSV – Virtual workstation destination

Argument Name	Description
WSNAME	Virtual workstation name.
WSDEST	Virtual workstation destination. For using the controller itself as a destination enter a value of `********`.

Note: Resource `WSV` is valid only for the `DELETE` request starting from HCL Workload Automation for Z version 8.5, or later.

EXECUTE – Commit updates to the Current Plan

Use the `EXECUTE` request to update the current plan after one or more modify, insert, or delete current plan requests are completed.

Note: If you issue Workload Automation Programming Language commands through the server by using REST APIs or Zowe™ commands and you also issue additional REST API or ZOWE commands, ensure that you set both `OPTIONS SERVER(Y)` and `EXECUTE` to commit updates to the CP.

```
EXECUTE [MCPBLK]
```

If you are changing more than one current plan occurrence or current plan workstation before an **EXECUTE** request, you must complete all changes to one occurrence or workstation before changing the other. If you do not complete all changes to one occurrence or workstation, a message is issued and all the changes made since the last **EXECUTE** request are reset.

For changes to current plan resources, CSR, no **EXECUTE** is required.

An **EXECUTE** request is performed automatically by Workload Automation Programming Language before termination, if updates to the current plan were detected and **OPTIONS EXECUTE(AUTO)** was specified (this is the default).

Note: The resource can be only **MCPBLK**, therefore is optional. Any keywords you specify on an **EXECUTE** statement other than **MCPBLK** are ignored.

Return codes are:

0

The request was successful.

8

The request was not successful. An error message is issued to the message log data set.

INIT – Initialize communication with HCL Workload Automation for Z

Use the **INIT** request to identify the required HCL Workload Automation for Z subsystem and initialize the communication session between the subsystem and Workload Automation Programming Language.

```
INIT <resource> <arguments>
```

Through the parameter file EQQYPARM, you can override the subsystem name specified in the INIT request, and set a LU name, a TRACE level, and the DATINT flag.

The parameter file can be a sequential file, or a PDS allocated as **//EQQYPARM DD DISP=SHR,DSN=OPCESA.SYS1.CNTL(YPARM)**

If you do not specify any **INIT** request, Workload Automation Programming Language automatically executes one before any other PIF requests. It also generates a **TERM** request ahead of your **INIT** request, if you make an **INIT** request to a new subsystem while a session is already established.

An automatic **INIT** request specifies v the subsystem to which to connect, if you code an **INIT** request yourself you can specify additional arguments.

Return codes are:

0

The request was successful. A programming interface session is successfully started. The address of the communication block is placed in the parameter list.

8

The request was not successful. For details about the error, see the message log, SYSLOG, and EQQDUMP data sets.

INIT subsystem

Table 37. INIT subsystem

Argument Name	Description
LUNAME	A server or controller LU name for the program interface session to communicate through.
MLOGDDN	Message log where to messages are to be stored, instead of the default (EQQMLOG). Each **INIT** request requires its own message log. If you make more than one **INIT** request before a **TERM** request, or if PIF is invoked by a program or started task that is already using EQQMLOG, specify MLOGDDN for each additional **INIT** request. If MLOGDDN is not specified and EQQMLOG is already in use, message EQQZ038E is stored in the SYSLOG and the **INIT** request fails.
REMHOST	Server host name for the program interface TCP/IP session. **REMHOST** and **LUNAME** are mutually exclusive.
REMPORT	Server port number for the program interface TCP/IP session. **REMPORT** and **LUNAME** are mutually exclusive.

INSERT – Add objects into the plan

In Workload Automation Programming Language, use the **INSERT** request to add occurrences to the Current Plan and Long-Term Plan. The PIF facility to update the database is handled by Batch Loader within Workload Automation Programming Language.

```
INSERT <resource> <arguments> ALIAS(<newname>)
```

When inserting a new occurrence, the input arrival date and time and deadline date and time can be provided in the arguments. If the input arrival is not provided when inserting a current plan occurrence, the current date and time is used (that is, the date and time at which the occurrence is inserted). However, if an occurrence already exists with this application ID and input arrival date and time, the next available minute in which no occurrence of this application exists will be used. You must supply an input arrival date and time if you are inserting an occurrence in the LTP.

If arguments are not provided for the deadline, these defaults are observed by HCL Workload Automation for Z:

- If the occurrence is being added to the current plan and the input arrival is provided, the deadline from the first run cycle is used if a run cycle exists. If there are no run cycles or the input arrival is not provided, the deadline is set to the input arrival time plus 8 hours.
- When the occurrence is being added to the long-term plan, the deadline is set to the input arrival plus 8 hours.

By default, external dependencies of the occurrence are not resolved when it is added to the LTP or CP. If resolution of external dependencies is required, an **OPTIONS LTDEPR** or **CPDEPR** request must be used to specify this.

- To insert the extended name of an operation, use the **MODIFY** request.
- To insert new information into an existing LTP or CP occurrence, specify the required arguments. For example, you can insert a new operation into an existing current plan occurrence, but you must have first identified the actual occurrence to which the information is to be added with a previous **MODIFY** or **INSERT** request. Similarly, you can insert new information for an existing current plan operation only if you first identified the operation. This means that you must first use a **MODIFY** request to identify the occurrence, then you use a **MODIFY** request to identify the operation, before inserting a predecessor (CPPRE), successor (CPSUC), or special resource (CPSR).

  When identified, Workload Automation Programming Language maintains a *current occurrence* and *current operation*.

  To insert a new interval into a current plan workstation, first identify the workstation with a **MODIFY CPWS** request.

If the **INSERT** request was used to modify information in the current plan, a later **EXECUTE** request must be made for the modification to actually take effect; Workload Automation Programming Language does this for you automatically if you set **OPTIONS EXECUTE(AUTO)** (this is the default).

The return codes are:

0

The request was successful.

4

One or more of the dependencies, specified by the application description of the **INSERT LTOC** request, was not set up because no applicable predecessor occurrence exists. This return code could also result from an **INSERT** request for any of LTPRE, CPOP, CPOC, CPPRE, and CPSR, if the dependency was not set up.

8

The request was not successful. An error message is issued to the message log data set.

INSERT CPCOND – Current Plan Condition

Table 38. INSERT CPCOND – Current Plan Condition

Header	Header
CONDID	Condition ID (1-999)
COUNT	Condition Counter. Used to determine how many conditional predecessors must be met for the condition to be true.

Table 38. INSERT CPCOND – Current Plan Condition (continued)

Header	Header
	If set to zero ALL conditional predecessors must be true for the condition to be satisfied. If greater than zero then the condition will be satisfied if at least that many conditional predecessors are met.
DESC	Descriptive text (16 characters)

Note: Resource `CPCOND` is valid only for the `INSERT` request starting from HCL Workload Automation for Z version 8.5, or later.

INSERT CPLAT – Operation User-defined Late Information

Table 39. INSERT CPLAT – Operation User-defined Late Information

Argument Name	Description
LATACT	The action taken if the operation has not yet started when the specified day and time is reached: `A` = Only an alert message is issued. `C` = The operation is set to Complete, if its status allows it. Otherwise, it is NOPed. `E` = The operation is set to Error with OLAT, if its status allows it. Otherwise, this setting is postponed at the time when the status allows it. `N` = The operation and all its internal successors are NOPed, if their status allows NOPing. Otherwise, it is ignored.
LATACTDT	Date and time by which the operation must start. If not, an action is issued. *YYMMDDHHMM*.
LATALEDT	Date and time by which the operation must start. If not, an alert is taken. *YYMMDDHHMM*.

INSERT CPOC – Current Plan Occurrence

Table 40. INSERT CPOC – Current Plan Occurrence

Argument Name	Description
ADID	Application description ID as stored in the database
ALIAS	Application description ID to use on the Current Plan
DEADLINE	Deadline date and time *YYMMDDHHMM*

Table 40. INSERT CPOC – Current Plan Occurrence (continued)

Argument Name	Description
DESC	Descriptive text
ERRCODE	Error code
GROUP	Authority group
GROUPDEF	Group definition ID
IA	Input arrival date and time *YYMMDDHHMM*
JCLVTAB	JCL variable table
ODESC	Descriptive text of owner
OWNER	Owner ID
PRIORITY	Priority

Note:

1. A **DEADLINE** argument is accepted also when no **IA** argument is specified. If the **IA** selected by HCL Workload Automation for Z is later than the **DEADLINE** argument value, the argument value is ignored. The default, **IA** plus 8 hours, is used.
2. If you specify 24.00 as the **IA** time, it is converted to 00.00 of the following day. Valid input arrival times are from 00.00 to 23.59.
3. If you specify 00.00 as deadline, it is converted to 24.00 of the previous day. Valid deadline times are from 00.01 to 24.00.
4. If you use **ALIAS**, unique occurrences could be created in the plan; but if another application with the same name exists, this is never run and the JCL for these occurrences remains in the JS file indefinitely. Consider using the HCL Workload Automation for Z sample EQQPIFJX to maintain the JCL repository.

INSERT CPOCPRE – Current Plan Occurrence Predecessor

Table 41. INSERT CPOCPRE – Current Plan Occurrence Predecessor

Argument Name	Description
PREADID	Predecessor application ID.
PREIA	Predecessor input arrival date and time *YYMMDDHHMM*.
PREOPNO	Predecessor operation number. If the predecessor is an occurrence, you do not need to set this argument.

INSERT CPOCSUC – Current Plan Occurrence Successor

Table 42. INSERT CPOCSUC – Current Plan Occurrence Successor

Argument Name	Description
SUCADID	Successor application ID.
SUCIA	Successor input arrival date and time *YYMMDDHHMM*.
SUCOPNO	Operation number. If the successor is an occurrence, you do not need to set this argument.

INSERT CPOP – Current Plan Operation

Table 43. INSERT CPOP – Current Plan Operation

Argument Name	Description
AEC	Automatic error completion
AJR	Automatic job hold/release
ASUB	Automatic job submission
CLATE	Cancel if late
CLNTYPE	Data Set cleanup type
CONDRJOB	Conditional recovery job
DEADWTO	Issue deadline WTO
DESC	Descriptive text
DURATION	Estimated duration in 100th of a second (mutually exclusive with EDUR)
EDUR	Estimated duration *HHMM* (mutually exclusive with DURATION)
EXPJCL	Expanded JCL option
FORM	Form number or blanks
HRC	Highest successful return code
JCLASS	Job class
JOBCRT	Critical job: **N** Not eligible for WLM assistance **P** Critical path target

Table 43. INSERT CPOP – Current Plan Operation (continued)

Argument Name	Description
	W Eligible for WLM assistance
JOBNAME	Job name
JOBPOL	Workload monitor late job policy: **C** Conditional mode **D** Deadline **L** Long duration **S** Latest start time **‘ ’ (blank)** Use default
MONITOR	Operation monitored by an external product, Y or N
OPDL	Operation deadline date and time *YYMMDDHHMM* or blank
OPDLACT	The action taken if the operation does not complete at its deadline: **‘ ’ (blank)** Default. No action is taken. **A** Only an alert message is issued. **C** The operation is set to Complete, if its status allows it. Otherwise, it is NOPed. **E** The operation is set to Error with ODEA, if its status allows it. Otherwise, this setting is postponed at the time when the status allows it.

Table 43. INSERT CPOP – Current Plan Operation (continued)

Argument Name	Description
	N The operation and all its internal successors are NOPed, if their status allows NOPing. Otherwise, it is ignored.
OPIA	Operation input arrival date and time *YYMMDDHHMM* or blank
OPNO	Operation number
PSUSE	Parallel servers required
R1USE	Resource 1 required
R2USE	Resource 2 required
RERUT	Reroutable operation
RESTA	Restartable operation
STATUS	Operation status
TIMEDEP	Time-dependent job
USERDATA	Data stored in operation user field
USRSYS	User sysout support
WSNAME	Workstation name
WLMSCLS	WLM service class

INSERT CPPRE – Current Plan Predecessor

Table 44. INSERT CPPRE – Current Plan Predecessor

Argument Name	Description
PREADID	Predecessor application ID.
PREIA	Predecessor input arrival date and time *YYMMDDHHMM*.
PREOPNO	Predecessor operation number. If the predecessor is an occurrence, you do not need to set this argument.
TRPTTIME	Transport time *HHMM*.

Note: When you use `CPPRE` to insert an internal dependency, `PREADID` nor `PREIA` must not be set.

INSERT CPSAI – Current Plan System Automation Info

Table 45. INSERT CPSAI – Current Plan System Automation Info

Argument Name	Description
AUTFUNC	System Automation automated function (for operation). It must be an alphanumeric value, uppercase format. The first character cannot be numeric.
COMMTEXT	System Automation command text. It is required.
COMPINFO	System Automation completion information.
SECELEM	System Automation security element.

Note:

1. The occurrence and operation to which the system automation information refers are identified, respectively, by `INSERT/MODIFY CPOC` and `INSERT/MODIFY CPOP` sequences
2. You can only use `INSERT CPSAI` for an operation that runs on an automation workstation.
3. Resource `CPSAI` is valid only for the `INSERT` request starting from HCL Workload Automation for Z version 8.3, or later.

INSERT CPSIMP – Current Plan Conditional Predecessor

Table 46. INSERT CPSIMP – Current Plan Conditional Predecessor

Argument Name	Description
PREADID	Predecessor application name.
PREIA	Predecessor input arrival date and time *YYMMDDHHMM*.
PREOPNO	Predecessor operation number.
PROCSTEP	Use it to define a step level dependency. If the step is not in a procedure, this parameter identifies the job step name, otherwise it identifies the step name in the JCL procedure. It must correspond to a step specifying the EXEC PGM= statement.
STEPNAME	Use it in conjunction with PROCSTEP when defining a step level dependency, only if the step is in a procedure, to identify the procedure invocation step name.
TYPE	Condition type: **RC** To check predecessor return code **ST** To check predecessor status

Table 46. INSERT CPSIMP – Current Plan Conditional Predecessor (continued)

Argument Name	Description
LOG	Logical operator: **GE** Greater than or equal to (TYPE=RC only) **GT** Greater than (TYPE=RC only) **LE** Less than or equal to (TYPE=RC only) **LT** Less than (TYPE=RC only) **EQ** Equal to **NE** Not equal to **RG** Range (TYPE=RC only)
VALRC	Return code value, or lower limit of a return code value range when TYPE=RC & LOG=RG.
VALRC2	Upper limit of a return code value range when TYPE=RC & LOG=RG.
VALST	Status (TYPE=ST only).

Note:

1. To create an internal dependency, do not specify either **PREADID** or **PREIA**.
2. Resource **CPSIMP** is valid only for the **INSERT** request starting from HCL Workload Automation for Z version 8.5, or later.

INSERT CPSR – Current Plan Operation Special Resource

Table 47. INSERT CPSR – Current Plan Operation Special Resource

Argument Name	Description
ONCOMPL	Availability to set on complete:

Table 47. INSERT CPSR – Current Plan Operation Special Resource (continued)

Argument Name	Description
	Y Available **N** Unavailable **R** Reset availability to default
ONERROR	Keep on error, Y or N.
QUANTITY	Quantity required. Specify 0 to allocate the total quantity of the special resource. The value 0 is the same as blank in the dialogs.
RESNAME	Special resource name.
RESUSAGE	Special resource usage: S or X.

INSERT CPSUC – Current Plan Successor

Table 48. INSERT CPSUC – Current Plan Successor

Argument Name	Description
SUCADID	Successor application ID.
SUCIA	Successor input arrival date and time *YYMMDDHHMM*.
SUCOPNO	Operation number. If the successor is an occurrence, you do not need to set this argument.

Note: When you use `CPSUC` to insert an internal dependency, `SUCADID` nor `SUCIA` must not be set.

INSERT CPUSRF – User Field

Table 49. INSERT CPUSRF – User Field

Argument Name	Description
UFNAME	User Field Name
UFVALUE	User Field Value

Note:

- Always identify an operation with an **INSERT CPOP** or **MODIFY CPOP** request before using an **INSERT CPUSRF** request.
- Resource **CPUSRF** is valid only for the **INSERT** request.

INSERT IVL – Current Plan Workstation Interval

If an interval contains information originating from the workstation description, indicator CPIVLDP in segment CPIVL is set to Y otherwise it is set to N. If an interval is changed through the dialog or program interface, the indicator CPIVLMOD is set to Y, or otherwise it is set to N. **INSERT IVL** can insert an interval spanning existing intervals with CPIVLMOD=N.

The inserted interval is converted to several intervals as required by daily planning. Other requests following the **INSERT** must take this possible split into account; each request is handled fully before the next request.

Table 50. INSERT IVL – Current Plan Workstation Interval

Argument Name	Description
FROM	Interval start date/time *YYMMDDHHMM*
PSCAP	Parallel server capacity
R1CAP	Resource 1 capacity
R2CAP	Resource 2 capacity
TO	Interval end date and time *YYMMDDHHMM*

INSERT JCLPREP – JCL Preparation

Table 51. INSERT JCLPREP – JCL Preparation

Argument Name	Description
ADID	Application ID
IA	Input arrival date and time *YYMMDDHHMM*
OPNO	Integer Operation number

INSERT LTOC – Long Term Plan Occurrence

Table 52. INSERT LTOC – Long Term Plan Occurrence

Argument Name	Description
ADID	Application ID
DEADLINE	Deadline date and time *YYMMDDHHMM*
ERRCODE	Error code

Table 52. INSERT LTOC – Long Term Plan Occurrence (continued)

Argument Name	Description
GROUPDEF	Group definition ID
IAD	Run date *YYMMDD*
IAT	Input arrival time *HHMM*
JCLVTAB	JCL variable table
PRIORITY	Priority

INSERT LTPRE – Long Term Plan Predecessor

Table 53. INSERT LTPRE – Long Term Plan Predecessor

Argument Name	Description
ADID	Application ID
IAD	Run date *YYMMDD*
IAT	Input arrival time *HHMM*
PREADID	Predecessor Application ID
PREIAD	Predecessor Run date *YYMMDD*
PREIAT	Predecessor Input arrival time *HHMM*

Note: `INSERT LTPRE` is used only to insert external predecessors. No support is provided in the long-term plan for internal dependencies.

INSERT VIVL – CP Virtual Workstation Interval

An interval can have information originating from the workstation description, indicator CPVIVLDP in segment CPVIVL is set to Y, or otherwise to N. If an interval is changed via the dialog or the program interface then the indicator CPVIVLMOD is Y, or otherwise N. `INSERT VIVL` can insert an interval spanning existing intervals with CPVIVLMOD=N.

The inserted interval is converted to several intervals as required by daily planning. Other requests following the `INSERT` must take this possible split into account; each request is handled fully before the next request.

Table 54. INSERT VIVL – CP Virtual Workstation Interval

Argument Name	Description
FROM	Interval start date/time *YYMMDDHHMM*
PSCAP	Parallel server capacity

Table 54. INSERT VIVL – CP Virtual Workstation Interval (continued)

Argument Name	Description
R1CAP	Resource 1 capacity
R2CAP	Resource 2 capacity
TO	Interval end date and time *YYMMDDHHMM*

Note: Resource `VIVL` is valid only for the `INSERT` request.

Note:

LIST – Find objects in the Database and Plans

Use the `LIST` request to retrieve a list of records of a selected type.

```
LIST <resource> <arguments> VALID(<date>) SAVELIST(<listname>)
```

When you use `LIST`, the resulting list includes only the common segments of the records. By default, no other segments is retrieved and Batch Loader statements are not generated. To do this, you must `SELECT` a record by adding the keyword `SELECT(Y)` to the LIST statements. To make this the default behaviour, specify `OPTIONS SELECT(Y)` for all LIST statements. For more details, see Automatic SELECT and DELETE on page 145.

For retrieving current plan occurrences and operations, the default is to retrieve all matching objects except those in deleted status. When you specify the argument `STATUS`, it overrides the default processing.

Argument names specify field names of the record to be tested to determine if the record should be included in the list.

Note:

1. Because the first blank or comparison-operator symbol ends the argument value, you cannot search for fields that contain imbedded blanks or comparison-operator symbols.
2. The wildcard search arguments asterisk (*) and percent sign (%), cannot be used in the year part (*YY*) of date arguments.
3. To use a comparison operator (such as <, >, or ≠) in an argument that contains an IA value including a date and time, specify the complete value as the argument. The comparison operator can follow this value.
4. The values of PIF arguments as dates depend on the PIF base year, which is defined by the PIFCWB keyword on the INTFOPTS statement, or the CWBASE keyword of the INIT statement. The value of the VALTO

argument for default high date depends on the PIFHD keyword of the INTFOPTS statement or the HIGHDATE keyword of the INIT statement.

5. The HCL Workload Automation for Z programming interface requires that you use the Common Segment in `LIST` requests. WAPL allows the record to be specified instead, and translates this internally to the common segment before passing the request to the PIF.

The return codes are:

0

The request was successful.

4

The request was not successful, for one of the following reasons:

1. The requestor is not authorized to read the records.
2. No records meet the criteria specified by the arguments.

8

The request was not successful. An error message is issued to the message log data set.

OBJECT

Use the `OBJECT` keyword to create an object variable for the `LISSTAT` process and the record identified.

`OBJECT(<name>)`

where `<name>` is the name of the object variable to create.

The primary object for the `LISSTAT` command is a simple object variable that contains the number of records found by the `LISTSTAT` command. This is either 1 or 0, because `LISSTAT` is designed to identify only one record.

Int the following example, the object for the identified record is the object name suffixed by 1:

```
LISTSTAT CPOPCOM ADID(TWSCDAILYPLAN) OPNO(010) IA(0805281200)
         POLICY(C=0,?=0,20) OBJECT(CHK)
```

Therefore `!@CHK` will contain 1 if the operation is found, and 0 if not. `!@CHK1-CPOPJES` will contain the JES number of the job being checked.

MATCHTYP Argument

Use the `MATCHTYP` argument for `LIST` requests with searchable fields that might contain wildcards or spaces.

When you specify the argument `MATCHTYP`, the asterisk (*) and percent sign (%) represent normal characters instead of wildcards, and blank represents a normal character instead of ending the selection value.

`MATCHTYP` `EXA`, `PFX`, and `SFX` affect:

- The STATUS argument of the CPOPCOM segment
- The ETTNAME argument of the ETT segment
- The RESNAME argument of the SRCOM and CSRCOM segments.

The **MATCHTYP** argument can have the following values:

EXA

Exact match

PFX

Treat as a prefix match

SFX

Treat as a suffix match

Note:

1. If MATCHTYP has the EXA value specified, a record is selected only if the value in the record is exactly the same as the argument value.
2. If MATCHTYP has the PFX value specified, a record is selected only if the start value in the record is the same as the argument value.
3. If MATCHTYP has the SFX value specified, a record is selected only if the end value in the record is the same as the argument value.

SAVELIST Argument

Use the **SAVELIST** argument to save the list of objects that were found for input to Batch Loader commands to identify the objects to update.

In the following example, the command finds all the applications with the owner ID starting with ABC and pass that list into Batch Loader command ADSTART to change the owner ID to start with XYZ:

```
OPTIONS OUTMASK(Y)
LIST ADCOM OWNER(ABC*) SAVELIST(ABCOWNED)
ADSTART SAVELIST(ABC*) OWNER(XYZ*)
```

Note: Do not use **SAVELIST** for names beginning with an underscore (_), because this is the prefix that Workload Automation Programming Language uses for any **SAVELIST** commands that is generated internally. Names prefixed with an underscore (_) are considered temporary and might be automatically dropped by commands.

TAG Argument

Use the **TAG** argument in a **LIST** command to create an additional output field called TAG, which will be available in any segment generated by the command. This allows for the output from multiple **LIST** commands to be correlated back to the originating command.

In the following example, the command performs 2 **LIST** requests: one for the applications whose names begin with ABC, one for the applications whose owner ID begins with ABC. The returned records lists the **TAG** and the **ADID**. By checking the **TAG**, you can determine from which **LIST** request each record comes.

```
OUTPUT ADCOM FIELDS(TAG,ADID)
      LIST ADCOM ADID(ABC*) TAG(ABCAPPS)
      LIST ADCOM OWNER(ABC*) TAG(ABCOWNED)
```

Note: Any **SELECT** statements generated from a **LIST** statement using **OPTIONS SELECT(Y)** will automatically be passed the same **TAG** argument.

Automatic SELECT and DELETE

Both the **SELECT** and **DELETE** commands require keywords that identify a specific record. In many cases, you will want to **SELECT** or **DELETE** a set of records based on various criteria, rather than individually create a statement for each record.

The **LIST** statement can identify sets of records and can be used to automatically generate and execute **SELECT** and **DELETE** statements for each record found by adding **SELECT** and **DELETE** keywords to the **LIST** statement.

For example, **LIST ADCOM ADID(ABC*) VALID(=) SELECT(Y)** generates and executes **SELECT** statements for each application definition beginning with ABC that is valid the day of execution.

The **SELECT** keyword can have values **Y** or **N**. If **Y** is set, every application found by the **LIST** statement will also be subsequently have a **SELECT** command executed for it. If **SELECT** is not specified as a keyword **N** is assumed.

Note: When you use **SELECT** with **LIST CPOPCOM**, you can also specify **OP,JS,USRF** and **ALL** (for details, see LIST CPOPCOM – Current Plan Operation on page 148).

It is recommended that the **DELETE** keyword is used in conjunction with the **SELECT** keyword so the record is selected before it is deleted. This gives the opportunity for batch loader to be generated for each object before it is deleted, assuming the relevant OUTPUT statements are in play. It is recommended that **FILESPEC=EQQFLALL** is used to ensure that it is possible to recover the deleted records.

For example, **LIST ADCOM ADID(ABC*) VALID(=) SELECT(Y) DELETE(Y)** generates and executes **SELECT** and **DELETE** statements for each application definition beginning with ABC that is valid the day of execution.

The **DELETE** keyword can have values **Y**, **N** or **D**. If **Y** is set, every application found by the **LIST** statement will also be subsequently have a **DELETE** command executed for it. If **D** is set, then deletion of every application is deferred. This results in **DELETE** statements being generated for each object and written to an output file for later execution (see **OPTIONS DELFILE**). If **DELETE** is not specified as a keyword **N** is assumed.

The **SELECT** and **DELETE** statements executed by this method are set to message level 2. This means that by default you will not see these statements in the job output, unless they fail. To see these statements even if they are successful set **OPTIONS MSGLEVEL(2)**.

Defaults for the **LIST SELECT** and **LIST DELETE** keywords can be set by **OPTIONS SELECT** and **OPTIONS DELETE**.

LIST ADCOM, LIST ADKEY – Application ID, Application Key

Table 55. LIST ADCOM, LIST ADKEY – Application ID, Application Key

Argument Name	Description
ADID	Application description ID.
GROUP	Authority group name.
GROUPDEF	Group definition ID.
MONITOR	**Y** Application with at least one operation monitored by an external product. **N** Application with no operation monitored by an external product.
OWNER	Owner ID
PRIORITY	Priority
STATUS	Status: **P** Pending. **A** Active.
TYPE	Application type: **A** Application (default, if TYPE not specified). **G** Group.
VALID	Valid-on date *YYMMDD*. The Valid-on date is used to find an application valid on a specific date. Workload Automation Programming Language uses this to generate the correct combination of `VALFROM` and `VALTO`. For example, `VALID(081125)` finds the version of an application valid on the 25th of November 2008.
VALFROM	Valid-from date *YYMMDD*.
VALTO	Valid-to date *YYMMDD*.

LIST AWSCL – All Workstations Closed

Table 56. LIST AWSCL – All Workstations Closed

Argument Name	Description
DATE	Date *YYMMDD*

LIST CLCOM - Calendar

Table 57. LIST CLCOM - Calendar

Argument Name	Description
CALENDAR	Calendar ID

LIST CPCONDCO – Current Plan Condition (Common)

Table 58. LIST CPCONDCO – Current Plan Condition (Common)

Argument Name	Description
ADID	Application description
IA	Input arrival date and time (*YYMMDDHHMM*)
OPNO	Operation number
CONDID	Condition ID (1-999)
CONDVAL	Condition status: **F** Flase **T** True **U** Undefined

Note: Resource `CPCONDCO` is valid only for the `LIST` request starting from HCL Workload Automation for Z version 8.5, or later.

LIST CPOC – Current Plan Occurrence

Table 59. LIST CPOC – Current Plan Occurrence

Argument Name	Description
ADID	Application description ID

Table 59. LIST CPOC – Current Plan Occurrence (continued)

Argument Name	Description
GROUP	Authority group
GROUPDEF	Group definition ID
IA	Input arrival date and time *YYMMDDHHMM*
MCPADDED	Add by Modify Current Plan operation, Y or N
MONITOR	**Y** Occurrence with at least one operation monitored by an external product. **N** Occurrence with no operation monitored by an external product.
OWNER	Owner ID
PRIORITY	Priority
RERUN	Rerun requested, Y or N
STATUS	Occurrence status

Note: By default, occurrences in deleted status are not retrieved when the `STATUS` argument is not supplied. If you do not provide the `STATUS` argument, the request is processed as `STATUS-NE(D)`.

LIST CPOPCOM – Current Plan Operation

Table 60. LIST CPOPCOM – Current Plan Operation

Argument Name	Description
ADID	Application description ID
AUTFUNC	System Automation automated function (for operation)
CLNSTAT	Data Set cleanup status
CLNTYPE	Data Set cleanup type
COMMTEXT	System Automation command text
COMPINFO	System Automation completion information
CONDRJOB	Conditional recovery job
DPREM	Removable by daily planning
ERRCODE	Error code

Table 60. LIST CPOPCOM – Current Plan Operation (continued)

Argument Name	Description
EXECDEST	Execution destination (******** represents the controller)
EXPJCL	Expanded JCL option
EXTNAME	Operation extended name
EXTSE	Scheduling Environment name
GROUP	Authority group
IA	Input arrival date and time of the occurrence *YYMMDDHHMM*
JOBCRT	Critical job: **N** Not eligible for WLM assistance **P** Critical path target **W** Eligible for WLM assistance
JOBNAME	Job name
JOBPOL	Workload monitor late job policy: **C** Conditional mode **D** Deadline **L** Long duration **S** Latest start time **blank** Default
LATEE	Operations that are either late on their latest start time, or late on the settings for Not Started Alert or Not Started Action. Y or N.
LATEL	Operations that are late on their latest start time. Y or N.

Table 60. LIST CPOPCOM – Current Plan Operation (continued)

Argument Name	Description
LATEN	Operations that are late on the settings for Not Started Alert or Not Started Action. Y or N.
MONITOR	**Y** Operation monitored by an external product **N** Operation not monitored by an external product
OPNO	Operation number
OWNER	Owner ID
PRIORITY	Priority
SECELEM	System Automation security element
SHADOWJ	Shadow job (Y/N)
STATUS	Operation status
USRSYS	User sysout support
UFNAME	User Field Name
UFVALUE	User Field Value
UNEXPRC	An Unexpected RC was encountered (Y/N)
VIRTDEST	Submission destination (******** represents the controller)
WAITSE	Waiting for Scheduling Environment, N or Y
WLMSCLS	WLM service class
WSNAME	Workstation name
WAITFORW	Started on WAIT workstation, Y or N
WMPRED	Waiting for mandatory pending predcessors, Y or N
WPMPRED	Waiting for either mandatory pending or pending predecessors, Y or N
WPPRED	Waiting for pending predecessors, Y or N

 **Note:**

1. By default, operations in deleted status are not retrieved when the **STATUS** argument is not supplied. If you do not provide the **STATUS** argument, the request is processed as **STATUS-NE(D)**.
2. **UFNAME**, **UFVALUE** and **UNEXPRC** are only available as keywords from version 8.5.1 with **SPE(USRF)** applied and beyond.
3. Using **SELECT(Y)** with LIST **CPOP** will **SELECT** the **CPOP** record. In addition the **SELECT** keyword for **LIST CPOP** has some additional values – **SELECT(OP)** will select the **CPOP** record (equivalent to **Y**), **SELECT(JL)** will select the Job Log, **SELECT(JS)** will select the **JS** file entry, **SELECT(USRF)** will **SELECT** the **CPUSRF** record for the Operation and **SELECT(ALL)** will **SELECT** the **CPOP**, **JL**, **JS** and **CPUSRF** records.
4. **WMPRED**, **WMPPRED** and **WPPRED** are available as keywords starting from HCL Workload Automation for Z version 9.2.

LIST CPOPSRU – Current Plan Operation SR Usage

Table 61. LIST CPOPSRU – Current Plan Operation SR Usage

Argument Name	Description
LISTTYPE	INUSE or WAITQ
RESNAME	Special resource name

Note:

1. Both arguments are required. The argument value specified for **RESNAME** is the name of the special resource for which the In-Use list or Wait Queue is to be retrieved.
2. Generic characters are not supported. It is processed as if **MATCHTYP(EXA)** was specified; exact match is required for record selection. The argument **MATCHTYP** is not supported.

LIST CPWSCOM – Current Plan Workstation

Table 62. LIST CPWSCOM – Current Plan Workstation

Argument Name	Description
WSAUTO	Automation Workstation, Y or N
WSNAME	Workstation name
WSREP	Workstation reporting attribute
WSRETYPE	Remote engine type: **D** Distributed

Table 62. LIST CPWSCOM – Current Plan Workstation (continued)

Argument Name	Description
	Z z/OS **blank**
WSTYPE	Workstation type
WSWAIT	WAIT Workstation, Y or N
WSZCENTR	z-Centric workstation, Y or N

LIST CPWSVCOM – CP Virtual workstation destination

Table 63. LIST CPWSVCOM – CP Virtual workstation destination

Argument Name	Description
WSNAME	Virtual workstation name
WSDEST	Virtual workstation destination (******** represents the controller)

Note: Resource `CPWSVCOM` is valid only for the `LIST` request.

LIST CRITSUCS – Critical Successors

Table 64. LIST CRITSUCS – Critical Successors

Argument Name	Description
ADID	Application ID of the job whose critical successors you want to list
OPNO	Operation number of the job whose critical successors you want to list
IA	Occurrence input arrival of the job whose critical successors you want to list (*YYMMDDHHMM*)

Note: The Workload Service Assurance process requires at least 1 operation in the current plan that is marked as CRITICAL=P to have been processed by the latest daily planning job. If there are no such operations, any operation dynamically added will not be considered critical and will not be returned by ISPF option 6.7 nor by the LIST CRITSUCS request. To prevent this issue, after you dynamically add a critical operation, it is required that you run a REPLAN daily planning job.

This problem can also be avoided by ensuring that there is at least 1 critical operation in the current plan. The simplest way to do this is to mark the daily planning jobs EXTEND and REPLAN as CRITICAL=P, because one of these jobs are always included in the current plan and they are critical to the operations of HCL Workload Automation for Z.

LIST CSRCOM – Current Plan Special Resource

Table 65. LIST CSRCOM – Current Plan Special Resource

Argument Name	Description
RESALCS	If any operation is currently allocating the resource shared, Y or N
RESAVAIL	Whether or not the resource is available, Y or N
RESGROUP	Resource group name
RESHIPER	Whether or not it is a DLF control resource, Y or N
RESNAME	Resource name
RESWAIT	Whether or not any operation is waiting for the resource

Note:

1. All the arguments are optional. The argument `MATCHTYP` is supported.
2. Fields `CSRXUSE`, `CSRSUSE`, `CSRXALL`, `CSRSALL`, `CSRWAITQ` and `CSRCIDATE` are only set for a `LIST` request. If you have used `SELECT(Y)` on the `LIST` request or `OPTIONS SELECT(Y)` is in effect, these fields will not be set, or return zero for numeric fields.

LIST ETT – Event Triggers

Table 66. LIST ETT – Event Triggers

Argument Name	Description
ADID	Associated application ID
ETTNAME	Name of trigger
ETTTYPE	Type of trigger

LIST GENDAYS – Generate dates from a rule

Note:

1. Specifying **FROMDATE** or **TODATE** outside of the allowable range will result in error message EQQH310E being issued from EQQYCOM and an S0C9 abend.
2. Resource **GENDAYS** is only valid for the **LIST** request starting from HCL Workload Automation for Z version 8.6 SPE, or later.

Table 67. LIST GENDAYS – Generate dates from a rule

Argument Name	Description
CALENDAR	The name of the calendar to use, if not specified the default calendar is used.
FDAYRULE	The Free Day Rule, to determine what to do with dates that are Free Days. This is a required keyword. 1. Move to nearest Work Day preceding 2. Move to nearest Work Day following 3. Keep the date on the Free Day 4. Cancel the date and do not output it E. Free days excluded.
FROMDATE	The date from which dates are generated from the rule in the format ***YYMMDD***. The earliest possible value for **FROMDATE** is the first day of the current month four years previous to the current year. For example, on 8 February 2012 the earliest possible value for **FROMDATE** is 1 February 2008. The latest possible value for **FROMDATE** is the 1st of January seven years after to the current year. For example, on 8 February 2012 the latest possible value for **FROMDATE** is 1 January 2019. If **FROMDATE** is not specified, the current date is used. **FROMDATE** must always be earlier than or equal to **TODATE**.
IAT	The input arrival time in the format ***HHMM***. It is used to determine the logical day for calendars with a Work Day End Time other than 00.00 (Default 0000).
RULEDEF	The run cycle rule, using the same keywords as the Batch Loader **ADRULE** statement.
SCOPE	Determines the scope of what dates are returned: **NORMAL** Return only run dates within the **FROMDATE** and **TODATE** limits (this is the default). **EXTENDED** Return run dates, including any that may have been shifted outside of the **FROMDATE** and **TODATE** limits due to the Free Day Rule.

Table 67. LIST GENDAYS – Generate dates from a rule (continued)

Argument Name	Description
	ALL Return all dates generated by the rule, including any free dates that have been moved or cancelled. **SKIPPED** Return only dates generated by the rule that landed on free days and were either moved to other dates or cancelled. **FREEDAY** Return only free days generated by the rule, including those moved or cancelled.
TODATE	The date to which dates will be generated from the rule in the format ***YYMMDD***. The latest possible value for **TODATE** is 31 December of the year seven years after the current date. For example, on 8 February 2012 the latest possible value for **TODATE** is 31 December 2019. If **TODATE** is not specified, it is set to **FROMDATE** +90 days. **TODATE** must always be later than or equal to **FROMDATE**.

LIST JCLVCOM – JCL Variable tables

Table 68. LIST JCLVCOM – JCL Variable tables

Argument Name	Description
JCLVTAB	JCL Variable Table ID

LIST JSCOM – Current Plan JCL

Table 69. LIST JSCOM – Current Plan JCL

Argument Name	Description
ADID	Application ID
IA	Input arrival date and time *YYMMDDHHMM*
JOBNAME	z/OS job name
OPNO	Operation number
WSNAME	Workstation name

Note: The resource code JSCOM retrieves JCL records from the JCL repository data set (JS file) and not from a JCL library. But a `SELECT` request tries to get JCL records from a JCL library if they are not found in the JCL repository data set.

LIST LTOCCOM – Long Term Plan Occurrence

Table 70. LIST LTOCCOM – Long Term Plan Occurrence

Argument Name	Description
ADID	Application ID
GROUP	Authority group
GROUPDEF	Group definition
IAD	Run date *YYMMDD*
IAT	Input arrival time *HHMM*
OWNER	Owner ID

LIST OICOM – Operator Instructions

Table 71. LIST OICOM – Operator Instructions

Argument Name	Description
ADID	Application ID
OPNO	Operation number
VALTO	Valid to Date and time for temporary operator instructions in the format *yymmddhhmm*. If **`VALTO`** is not specified, all instructions for the operation are listed, both temporary and permanent. If only the permanent instructions are required use **`VALTO(EMPTY)`**.

Note: **`VALTO`** is an undocumented PIF keyword for **`LIST OICOM`**. **`VALTO(EMPTY)`** is not part of PIF, it is implemented only by Workload Automation Programming Language.

LIST PRCOM – Period

Table 72. LIST PRCOM – Period

Argument Name	Description
PERIOD	Period name
PRTYPE	Period type

LIST RGCOM – Run Cycle Group

Table 73. LIST RGCOM – Run Cycle Group

Argument Name	Description
RGID	Run cycle group ID

LIST SRCOM – Special Resource

Table 74. LIST SRCOM – Special Resource

Argument Name	Description
RESGROUP	Special resource group ID
RESHIPER	DLF resource indicator
RESNAME	Special resource name

LIST WSCOM – Workstation

Table 75. LIST WSCOM – Workstation

Argument Name	Description
WSAUTO	Automation workstation, Y or N
WSNAME	Workstation name
WSREP	Workstation reporting attribute
WSRETYPE	Remote engine type: **D** Distributed **Z** z/OS **blank**
WSTYPE	Workstation type

Table 75. LIST WSCOM – Workstation (continued)

Argument Name	Description
WSVIRT	Virtual workstation, Y or N
WSWAIT	WAIT Workstation, Y or N
WSZCENTR	z-Centric workstation, Y or N

LIST WSVCOM – Virtual workstation destination

Table 76. LIST WSVCOM – Virtual workstation destination

Argument Name	Description
WSNAME	Virtual workstation name.
WSDEST	Virtual workstation destination. For using the controller as a destination, enter the value `********`.

Note: Resource `WSVCOM` is valid only for the `LIST` request.

MODIFY – Modify objects in the plans

Use the `MODIFY` request to modify one or more fields in an LTP or CP record.

```
MODIFY <resource> <arguments>
```

The arguments can be used both to identify the record to be modified, and to provide new values for this record. Or, the arguments can be used just to identify a record, and later requests can be used to perform particular actions. For example, with a `MODIFY` request, you can identify a particular current plan occurrence record. Then, with later `INSERT` requests, you can insert new operation records for that occurrence.

The `MODIFY` request can be used to modify information in the current plan. Requests that cause a modification of the current plan, except CSR requests, require a later `EXECUTE` request for the modification to actually take effect.

With the arguments described here, you specify the names and values of fields, either to identify a particular record, or provide updated information for a record.

Note: The values of PIF arguments as dates depend on the PIF base year, which is defined by the PIFCWB keyword on the INTFOPTS statement, or the CWBASE keyword of the INIT statement. The value of the VALTO argument for default high date depends on the PIFHD keyword of the INTFOPTS statement or the HIGHDATE keyword of the INIT statement.

The return codes are:

0

The request was successful.

4

The `MODIFY CPOP` request might end with return code 4 if the operation input arrival value specified in the request is earlier than the occurrence. If this happens, run the `EXECUTE` request for the modification to be enforced.

8

The request was not successful. An error message is issued to the message log data set.

MODIFY CPCOND – CP Condition

When you are modifying an existing current plan condition, the CONDID argument is required to identify the condition to be modified. All remaining arguments are optional and provide the information used to modify the condition.

Note:

1. Before specifying a `MODIFY CPCOND` request, you must always identify an operation with an `INSERT` or `MODIFY CPOP` request.
2. Resource `CPCOND` is valid only for the `MODIFY` request.

Table 77. MODIFY CPCOND – CP Condition

Argument Name	Description
CONDID	Condition ID (1-999)
COUNT	Condition Counter. Used to determine how many conditional predecessors must be met for the condition to be true. If set to zero ALL conditional predecessors must be true for the condition to be satisfied. If greater than zero then the condition will be satisfied if at least that many conditional predecessors are met.
DESC	Descriptive text (up to 16 characters).

MODIFY CPEXT – CP Extended Operation Info

Note: Resource `CPEXT` is valid only for the `MODIFY` request.

Table 78. MODIFY CPEXT – CP Extended Operation Info

Argument Name	Description

Table 78. MODIFY CPEXT – CP Extended Operation Info (continued)

Argument Name	Description
EXTNAME	Operation extended name. To delete the operation extended name, enter blank.
EXTSE	Scheduling Environment name. Special characters are allowed. To delete the SE name, enter blank.

MODIFY CPOC – Current Plan Occurrence

When you are modifying an existing current plan occurrence, the **ADID** and **IA** arguments identify the occurrence to be modified. All the remaining arguments provide the information used to modify the occurrence. The valid values for the **STATUS** argument are `w` (Waiting) and `c` (Complete).

Table 79. MODIFY CPOC – Current Plan Occurrence

Argument Name	Description
ADID	Application description ID
ALLMON	**Y** All operations of occurrence monitored by an external product **N** All operations of occurrence not monitored by an external product
DEADLINE	Deadline date and time *YYMMDDHHMM*
ERRCODE	Error code
GROUPDEF	Group definition ID
IA	Input arrival date and time *YYMMDDHHMM*
IANEW	New input arrival date and time *YYMMDDHHMM*
JCLVTAB	JCL variable table
PRIORITY	Priority
STATUS	Occurrence status

MODIFY CPOP – Current Plan Operation

When you are modifying an existing current plan operation, the **OPNO** argument is required to identify the operation to be modified. All remaining arguments are optional and provide the information used to modify the operation. If you are inserting, modifying, or deleting a predecessor connection or special resource specification for the operation, the **MODIFY CPOP** request is required only to identify the operation that will be referred to in the following **INSERT**, **MODIFY**, or **DELETE** request. Then, only the **OPNO** argument is required.

Note: Before using a `MODIFY CPOP` request, you must always identify an occurrence with a `MODIFY CPOC` request.

Table 80. MODIFY CPOP – Current Plan Operation

Argument Name	Description
AEC	Automatic error completion
AJR	Automatic job hold/release
ASUB	Automatic job submission
CLATE	Cancel if late
CLNTYPE	Data Set cleanup type
CONDRJOB	Conditional recovery job
DEADWTO	Issue deadline WTO
DESC	Operation descriptive text
DURATION	Estimated duration in 100th of second
EDUR	Estimated duration *HHMM*
ERRCODE	Error code
EXPJCL	Expanded JCL option
FORM	Form number or blanks
HRC	Highest successful return code
JCLASS	Job class
JOBCRT	Critical job: **N** Not eligible for WLM assistance **P** Critical path target **W** Eligible for WLM assistance
JOBNAME	Job name
JOBPOL	Workload monitor late job policy: **C** Conditional mode

Table 80. MODIFY CPOP – Current Plan Operation (continued)

Argument Name	Description
	D Deadline **L** Long duration **S** Latest start time **blank** This is the default
MONITOR	**Y** Operation monitored by an external product **N** Operation not monitored by an external product
OPCMD	Operation command: **BD** Bind shadow job **EX** Execute operation **KJ** Kill operation **MH** Hold operation **MR** Release operation **NP** NOP operation **PN** Prompt reply no **PY** Prompt reply yes

Table 80. MODIFY CPOP – Current Plan Operation (continued)

Argument Name	Description
	UN Un-NOP operation
OPDL	Operation deadline date and time or blank *YYMMDDHHMM*
OPDLACT	The action taken if the operation does not complete at its deadline: **' ' (blank)** Default. No action is taken. **A** Only an alert message is issued. **C** The operation is set to Complete, if its status allows it. Otherwise, it is NOPed. **E** The operation is set to Error with ODEA, if its status allows it. Otherwise, this setting is postponed at the time when the status allows it. **N** The operation and all its internal successors are NOPed, if their status allows NOPing. Otherwise, it is ignored.
OPIA	Operation input arrival date and time or blank *YYMMDDHHMM*
OPNO	Operation number
PSUSE	Parallel servers required
R1USE	Resource 1 required
R2USE	Resource 2 required
RERUT	Reroutable operation
RESTA	Restartable operation
STATUS	Operation status
TIMEDEP	Time-dependent job
USERDATA	Data stored in operation user field
USRSYS	User sysout support
WLMSCLS	WLM service class

Table 80. MODIFY CPOP – Current Plan Operation (continued)

Argument Name	Description
WSNAME	Workstation name

MODIFY CPREND – Distributed remote job info

Note:

1. The occurrence and operation to which the remote job information refers are identified, respectively, by the `INSERT` or `MODIFY CPOC ADID IA` and `INSERT` or `MODIFY CPOP OPNO` sequences.
2. You can use `MODIFY CPREND` only if the operation runs on a remote engine workstation.
3. When you run `MODIFY CPOP` to modify the workstation from one that is a remote engine type to any other type, the remote job info related to the operation is automatically deleted.

Table 81. MODIFY CPREND – Distributed remote job info

Argument Name	Description
COMPBNDF	Complete if bind fails (Y|N)
REJOBNM	Remote job name
REJSNM	Remote job stream name
REJSWS	Remote job stream workstation

MODIFY CPRENZ – z/OS remote job info

Note:

1. The occurrence and operation to which the remote job information refers are identified, respectively, by the `INSERT` or `MODIFY CPOC ADID IA` and `INSERT` or `MODIFY CPOP OPNO` sequences.
2. You can use `MODIFY CPRENZ` only if the operation runs on a remote engine workstation.
3. When you run `MODIFY CPOP` to modify the workstation from one that is a remote engine type to any other type, the remote job info related to the operation is automatically deleted.

Table 82. MODIFY CPRENZ – z/OS remote job info

Argument Name	Description
COMPBNDF	Complete if bind fails (Y|N)
READID	Remote application name

Table 82. MODIFY CPRENZ – z/OS remote job info (continued)

Argument Name	Description
REOPNO	Remote operation number

MODIFY CPSAI – Current Plan System Automation Info

Table 83. MODIFY CPSAI – Current Plan System Automation Info

Argument Name	Description
AUTFUNC	Automation automated function (for operation)
COMMTEXT	System Automation command text
COMPINFO	Automation completion information
SECELEM	System Automation security element

Note:

1. The occurrence and operation to which the system automation information refers are identified, respectively, by the **INSERT** or **MODIFY CPOC ADID IA** and **INSERT** or **MODIFY CPOP OPNO** sequences.
2. You can use **MODIFY CPSAI** only if the operation runs on an automation workstation.
3. Resource **CPSAI** is valid only for the **MODIFY** request.

MODIFY CPUSRF – User Field

Before using an `INSERT CPOP` request, you must always identify an operation with an `INSERT CPOP` or `MODIFY CPOP` request.

Note: Resource `CPUSRF` is valid only for the `MODIFY` request.

Table 84. MODIFY CPUSRF – User Field

Argument Name	Description
UFNAME	User Field Name
UFVALUE	User Field Value

MODIFY CPWS – Current Plan Workstation

When you are modifying a current plan workstation, the `WSNAME` argument is required. The remaining arguments contain the modified information.

Table 85. MODIFY CPWS – Current Plan Workstation

Argument Name	Description
ALTWS	When the workstation is set to failed or offline then another workstation can be specified for rerouting. Specify ALTWS if operations should be rerouted; if ALTWS is not supplied, no rerouting takes place.
PSC	Control on parallel server.
R1C	Control on resource 1.
R2C	Control on resource 2.
STARTACT	Action to be taken on current plan operations that have a status of started when the workstation status is set to failed or offline. Values are restart (R), set to error (E), or leave operation as is (L). **Note:** If the STARTACT argument is omitted when a workstation is set to failed or offline then no action is performed on the operations, as though STARTACT L was specified.
STATUS	New status of active (A), failed (F), or offline (O).
WSNAME	Workstation name.
WSREP	Workstation reporting attribute. **Note:** For Virtual Workstations this arguments is ignored.

MODIFY CPWSV – CP Virtual Workstation Destination

When you are modifying a current plan virtual workstation, the `WSNAME` and `WSDEST` arguments are required. The remaining arguments contain the modified information.

Note: Resource `CPWSV` is valid only for the `MODIFY` request.

Table 86. MODIFY CPWSV – CP Virtual Workstation Destination

Argument Name	Description
WSNAME	Virtual workstation name.
WSDEST	Destination (******** represents the controller).
PSC	Control on parallel server.
R1C	Control on resource 1.

Table 86. MODIFY CPWSV – CP Virtual Workstation Destination (continued)

Argument Name	Description
R2C	Control on resource 2.
STARTACT	Action to be taken on current plan operations that have a status of started when the workstation status is set to failed or offline. Values are restart (R), set to error (E), or leave operation as is (L). **Note:** If the STARTACT argument is omitted when a workstation is set to failed or offline then no action is performed on the operations, as though STARTACT L was specified.
STATUS	New status of active (A), failed (F), or offline (O).

MODIFY CSR – Current Plan Special Resource

`MODIFY CSR` takes as selection argument the resource name in the `RESNAME` argument. This argument is required. The resource name must be padded to the full length of 44 characters. It is processed as if `MATCHTYP(EXA)` was specified and an exact match is required for record selection. Alternatively, the common segment `CSRCOM` can be given as the selection argument. The remaining arguments are optional and contain modifications.

Note: MATCHTYP is not supported.

Table 87. MODIFY CSR – Current Plan Special Resource

Argument Name	Description
DEFAVAIL	Default availability, N or Y
DEFQTY	Default quantity, 1 to 999999
MAXLIMIT	Maximum usage limit. From 0 (no limit) to 999999
MAXTYPE	Type of action when maximum usage limit is reached: Y|N|R
ONCOMPL	Action on complete Y|N|R
ONERROR	Action on error, F, FX, FS, K, or blank
QUANTITY	Override quantity, numeric 1 to 999999, or 0 to indicate that there is no overriding quantity
RESAVAIL	Override availability, N, Y, or blank to indicate there is no overriding availability
RESDEVIA	Deviation, -999999 to 999999
RESNAME	Resource name

Table 87. MODIFY CSR – Current Plan Special Resource (continued)

Argument Name	Description
USEDFOR	Used for C, P, B, or N

MODIFY IVL – Current Plan Workstation Interval

When you are modifying a workstation open interval, the `FROM` argument is required. The remaining arguments are optional and provide the information used to modify the open interval.

Note: Before using a `MODIFY IVL` request, you must always identify a workstation with a `MODIFY CPWS` request.

Table 88. MODIFY IVL – Current Plan Workstation Interval

Argument Name	Description
ALTWS	Workstation to take over if this one fails or is set offline
FROM	Interval start date and time *YYMMDDHHMM*
PSCAP	Parallel server capacity
R1CAP	Resource 1 capacity
R2CAP	Resource 2 capacity

MODIFY LTOC – Long Term Plan Occurrence

When you are modifying an existing LTP occurrence, the `ADID`, `IAD`, and `IAT` arguments identify the occurrence to be modified. The remaining arguments provide the information used to modify the occurrence.

Table 89. MODIFY LTOC – Long Term Plan Occurrence

Argument Name	Description
ADID	Application description ID
DEADLINE	Deadline date and time *YYMMDDHHMM*
ERRCODE	Error code
GROUPDEF	Group definition ID
IAD	Input arrival date *YYMMDD*
IAT	Input arrival time *HHMM*
JCLVTAB	JCL variable table
PRIORITY	Priority

MODIFY VIVL – CP Virtual workstation interval

When you are modifying a virtual workstation open interval, the **FROM** argument is required. The remaining arguments are optional and provide the information used to modify the open interval.

Note:

1. Before using a **MODIFY VIVL** request, you must always identify a workstation with a **MODIFY CPWSV** request.
2. Resource **VIVL** is valid only for the **MODIFY** request.

Table 90. MODIFY VIVL – CP Virtual workstation interval

Argument Name	Description
FROM	Interval start date and time *YYMMDDHHMM*
PSCAP	Parallel server capacity
R1CAP	Resource 1 capacity
R2CAP	Resource 2 capacity

REPLACE

The **REPLACE** request is performed by Workload Automation Programming Language internally using the Batch Loader functionality to build records before writing them to the database.

Since the **REPLACE** request needs a fully formed record to be built in storage, it does not have a direct command line equivalent in Workload Automation Programming Language.

RESET – Resets pending changes to the plan

Use the **RESET** request to delete the current Modify Current Plan (MCP) block.

```
RESET [MCPBLK]
```

If performed before an **EXECUTE** request, the **RESET** request deletes a series of **MODIFY** current plan requests that have been collected in an MCP block.

The return codes are:

0

The request was successful.

8

The request was not successful. An error message is issued to the message log data set.

SELECT – Retrieve a record or common segment

Use the **SELECT** request to retrieve a record by specifying field names and values that identify the record you want to retrieve.

```
SELECT <resource> <arguments>
```

The **SELECT** statement is used to retrieve an individual record from the HCL Workload Automation for Z database or plans. You must set enough arguments to identify one record only; if the arguments apply to more than one record, the **SELECT** fails with RC=8 and the following message:

```
EQQY708E A SELECT REQUEST WITH MORE THAN ONE RECORD SELECTED,
         RESOURCE IS AD
```

The **SELECT** and **DELETE** statements can be automatically generated from **LIST** requests (for details, see LIST – Find objects in the Database and Plans on page 142).

LIST statements can be automatically generated for other objects referred to by the object retrieved by the **SELECT** statement, or objects that may refer to the **SELECT** statement by using **OPTIONS EXPAND(Y)**.

In the following example, the command retrieves the application MYAPPL and then **LIST** any event rules that may point to it and any workstation definitions, special resources, periods, referenced within it. **SELECT(Y)** causes any items found by the **LIST** processes to also have a **SELECT** statement run for them:

```
OPTIONS EXPAND(Y) SELECT(Y)
SELECT ADID(MYAPPL)
```

The end result being, if you have the relevant **OUTPUT** statements in place, that you obtain batch loader for the entire object and any other objects needed by it.

When you retrieve a record by using **SELECT**, you can get the complete record rather than just the common segment that is available from a **LIST** request. For example, **SELECT AD** retrieves the complete AD record, while **SELECT ADCOM** retrieves only the common segment.

Note:

1. The **SELECT JS** and **SELECT JSCOM** requests try to retrieve JCL from the JCL repository. If no JCL is found, it is retrieved from the JCL library or through the job-library-read exit, EQQUX002. The full key is required, that is, the application ID, the input arrival time, and the operation number. You might need to precede the **SELECT JS** request by a **LIST CPOPCOM** request to get the key values.
2. **LIST JSCOM** requests try to retrieve JCL only from the JCL repository.
3. **SELECT CPOPSRU** can be issued for list elements only, from a list created by **LIST CPOPSRU**.
4. The values of PIF arguments as dates depend on the PIF base year, which is defined by the **PIFCWB** keyword on the **INTFOPTS** statement, or the **CWBASE** keyword of the **INIT** statement. The value of the **VALTO** argument for

default high date depends on the **PIFHD** keyword of the **INTFOPTS** statement or the **HIGHDATE** keyword of the **INIT** statement.

5. **CPST** (current plan status) is only one record; therefore, select arguments are not required.

The return codes are:

0

The request was successful.

4

The request was not successful. No records meet the criteria specified by the arguments.

6

You are not authorized to read the record. You specified a unique key in the SELECT request; the record exists, but you do not have authority to read it.

8

The request was not successful. An error message has been written to the message log data set. This can occur if more than one record in the database satisfies the field values specified by your arguments. For example, you want to select an application description record with the ID APPL1, and there are two such application descriptions in the database with different validity dates. Your arguments must specify both the application ID and the valid-from date to uniquely identify the record.

OBJECT Argument

Use the **OBJECT** argument to create an object variable for the record retrieved by the **SELECT** command.

```
OBJECT(<name>)
```

where **<name>** is the name of the object variable to be created.

The record that is found creates a complete set of object variables containing all the record information, using the object name as the prefix.

For example, the following request creates a set of object variables named **@DAILY**, which will contain the data for the record that is retrieved. Therefore, **!@DAILY-ADID** would resolve to the application name of the retrieved record.

```
SELECT AD ADID(DLYAPPL) OBJECT(DAILY)
```

TAG Argument

Use the **TAG** argument in a **SELECT** command to create an additional output field named **TAG** that will be available in any segment generated by the command. This allows for the output from multiple **SELECT** commands to be correlated back to the originating command.

In the following example, the returned records are marked with a **TAG** value of TODAY or TOMORROW, depending from which **LIST** statement they come.

```
OUTPUT ADCOM FIELDS(TAG,ADID,ADFROM,ADSTAT)
SELECT ADCOM ADID(ABC123) VALID(=) TAG(TODAY)
SELECT ADCOM ADID(ABC123) VALID(+1) TAG(TOMORROW)
```

Note: Any `SELECT` statements generated from a `LIST` statement using `OPTIONS SELECT(Y)` is automatically passed the same `TAG` argument.

SELECT AD/ADCOM – Application Description

Table 91. SELECT AD/ADCOM – Application Description

Argument Name	Description
ADID	Application description ID
GROUP	Authority group name
GROUPDEF	Group definition ID
MONITOR	**Y** Application with at least one operation monitored by an external product **N** Application with no operation monitored by an external product
OWNER	Owner ID
PRIORITY	Priority
STATUS	Status: **A** Active **P** Pending
TYPE	Application type: **A** Application (default) **G** Group
VALFROM	Valid-from date *YYMMDD*
VALTO	Valid-to date *YYMMDD*

Note: If you do not specify the AD argument name `TYPE`, HCL Workload Automation for Z assumes application of type `A`.

SELECT AWSCL – All Workstations Closed

Table 92. SELECT AWSCL – All Workstations Closed

Argument Name	Description
DATE	Date *YYMMDD*

SELECT CL/CLCOM - Calendar

Note: If the name of the default calendar is specified in the EQQYPARM INIT statement, `SELECT CL` without the `CALENDAR` argument will return the default calendar. Otherwise, `CALENDAR` is a required argument.

Table 93. SELECT CL/CLCOM - Calendar

Argument Name	Description
CALENDAR	Calendar ID

SELECT CPCOND/CPCONDCO – CP Condition

Table 94. SELECT CPCOND/CPCONDCO – CP Condition

Argument Name	Description
ADID	Application description
IA	Input arrival date and time (*YYMMDDHHMM*)
OPNO	Operation number
CONDID	Condition ID (1-999)
CONDVAL	Condition status: **F** False **T** True **U** Undefined

Note: Resources `CPCOND` and `CPCONDCO` are valid only for the `SELECT` request.

SELECT CPOC – Current Plan Occurrence

Table 95. SELECT CPOC – Current Plan Occurrence

Argument Name	Description
ADID	Application description
GROUP	Authority group
GROUPDEF	Group definition ID
IA	Input arrival date and time *YYMMDDHHMM*
MCPADDED	Manually added to the Current Plan, Y or N
MONITOR	**Y** Occurrence with at least one operation monitored by an external product **N** Occurrence with no operation monitored by an external product
OWNER	Owner ID
PRIORITY	Priority
RERUN	Rerun requested, Y or N
STATUS	Occurrence status

SELECT CPOP/CPOPCOM – Current Plan Operation

Table 96. SELECT CPOP/CPOPCOM – Current Plan Operation

Argument Name	Description
ADID	Application description ID
CLNSTAT	Data Set cleanup status
CLNTYPE	Data Set cleanup type
CONDRJOB	Conditional recovery job
DPREM	Removable by Daily Planning
ERRCODE	Error code
EXECDEST	Execution destination (******** represents the controller)

Table 96. SELECT CPOP/CPOPCOM – Current Plan Operation (continued)

Argument Name	Description
EXPJCL	Expanded JCL option
EXTNAME	Operation extended name
EXTSE	Scheduling Environment name
GROUP	Authority group
IA	Input arrival date and time *YYMMDDHHMM*
JOBCRT	Critical job: **N** Not eligible for WLM assistance **P** Critical path target **W** Eligible for WLM assistance
JOBNAME	Job name
JOBPOL	Workload monitor late job policy: **C** Conditional mode **D** Deadline **L** Long duration **S** Latest start time **blank** Default
MONITOR	**Y** Operation monitored by an external product **N** Operation not monitored by an external product
OPNO	Operation number

Table 96. SELECT CPOP/CPOPCOM – Current Plan Operation (continued)

Argument Name	Description
OWNER	Owner ID
PRIORITY	Priority
SHADOWJ	Shadow job
STATUS	Operation status
USRSYS	User sysout support
VIRTDEST	Submission destination (********represents the controller)
WAITNAME	Waiting for Scheduling Environment, Y or N
WLMSCLS	WLM service class
WSNAME	Workstation name
WAITFORW	Started on WAIT workstation, Y or N
WMPRED	Waiting for mandatory pending predecessors, Y or N
WPMPRED	Waiting for either mandatory pending or pending predecessors, Y or N
WPPRED	Waiting for pending predecessors, Y or N

Note:

1. `SELECT CPOP` does not return `CPCOND` or `CPUSRF` information. For these resources, you must use separate `SELECT` statements.

SELECT CPST – Current Plan Status

This request has no arguments.

SELECT CPUSRF – Current Plan Operation User Fields

Use the `SELECT CPUSRF` command to generate a record containing a `CPUSRF` segment for each user field belonging to the operation.

Table 97. SELECT CPUSRF – Current Plan Operation User Fields

Argument Name	Description
ADID	Application description ID
IA	Input arrival date and time (*YYMMDDHHMM*)
OPNO	Operation number

Note: Resource `CPUSRF` is valid only for the `SELECT` request.

SELECT CPWS/CPWSCOM – Current Plan Workstation

Table 98. SELECT CPWS/CPWSCOM – Current Plan Workstation

Argument Name	Description
WSAUTO	Automation workstation, Y or N
WSNAME	Workstation name
WSREP	Workstation reporting attribute
WSRETYPE	Remote engine type: **D** Distributed **Z** z/OS **blank**
WSTYPE	Workstation type
WSVIRT	Virtual workstation, Y or N
WSWAIT	WAIT Workstation, Y or N
WSZCENTR	z-Centric workstation, Y or N

SELECT CPWSV/CPWSVCOM – CP Virtual workstation destination

Table 99. SELECT CPWSV/CPWSVCOM – CP Virtual workstation destination

Argument Name	Description
WSNAME	Virtual workstation name
WSDEST	Destination name (******** represents the controller)

Note: Resources `CPWSV` and `CPWSVCOM` are valid only for the `SELECT` request.

SELECT CRITPATH – Critical Path

Table 100. SELECT CRITPATH – Critical Path

Argument Name	Description
ADID	Application ID of the critical job
OPNO	Operation number of the critical job
IA	Input arrival of the critical job (*YYMMDDHHMM*)

Note:

1. Arguments, `ADID`, `OPNO`, and `IA` must be specified together.
2. The operation identified by the arguments must be a critical job (`P`).
3. This Workload Automation Programming Language command exploits an undocumented PIF request designed for the Dynamic Workload Console.

SELECT CSR/CSRCOM – Current Plan Special Resource

Table 101. SELECT CSR/CSRCOM – Current Plan Special Resource

Argument Name	Description
RESALCS	Whether or not any operation is currently allocating the resource shared, Y or N.
RESAVAIL	Whether or not the resource is available, Y or N.
RESGROUP	Resource group name.
RESHIPER	Whether or not it is a DLF control resource, Y or N.
RESNAME	Resource name.
RESWAIT	Fields **`CSRXUSE`**, **`CSRSUSE`**, **`CSRXALL`**, **`CSRSALL`**, **`CSRWAITQ`**, and **`CSRCIDATE`** are set only for a **`LIST`** request. If you have set **`SELECT(Y)`** on the **`LIST`** request or if **`OPTIONS SELECT(Y)`** is in effect, then these fields will not be set, or return zero for numeric fields. Whether or not any operation is waiting for the resource.

Note: Fields **`CSRXUSE`**, **`CSRSUSE`**, **`CSRXALL`**, **`CSRSALL`**, **`CSRWAITQ`**, and **`CSRCIDATE`** are set only for a **`LIST`** request. For a **`SELECT`** request, these fields are not set or return zero for numeric fields.

SELECT ETT – Event Trigger

Table 102. SELECT ETT – Event Trigger

Argument Name	Description

Table 102. SELECT ETT – Event Trigger (continued)

Argument Name	Description
ADID	Associated application ID
ETTNAME	Name of trigger
ETTTYPE	Type of trigger: **J** Job **R** Special Resource

SELECT JCLPREP – JCL Preparation

Table 103. SELECT JCLPREP – JCL Preparation

Argument Name	Description
ADID	Application ID
IA	Input arrival date and time *YYMMDDHHMM*
OPNO	Operation number

SELECT JCLPREPA – JCL Preparation simulation

Table 104. SELECT JCLPREPA – JCL Preparation simulation

Argument Name	Description
ADID	Application ID.
IA	Input arrival date and time *YYMMDDHHMM*.
OPNO	Operation number.
SIMTIME	Simulated time *CCYYMMDDHHMM*. *CCYY* can have a value from 1984 to 2071.
SIMTYPE	Simulation type: **FULL** or **PARTIAL** (default).

SELECT JCLV/JCLVCOM – JCL Variable Table

Table 105. SELECT JCLV/JCLVCOM – JCL Variable Table

Argument Name	Description
JCLVTAB	JCL Variable Table ID

SELECT JL/JLCOM – Job Log

Table 106. SELECT JL/JLCOM – Job Log

Argument Name	Description
ADID	Application ID
IA	Input arrival date and time *YYMMDDHHMM*
JOBNAME	z/OS Job name
OPNO	Operation number
WSNAME	Workstation name

Note:

1. `SELECT JL` will cause the job log of the selected operation to be output. If the job log has not already been retrieved then this will trigger a retrieval action and Workload Automation Programming Language will attempt the SELECT command again in accordance with the retry policy set by OPTIONS CONTENTION.
2. The `SELECT JL` command exploits an undocumented PIF request designed for the Dynamic Workload Console.

SELECT JS/JSCOM – Current Plan JCL

Table 107. SELECT JS/JSCOM – Current Plan JCL

Argument Name	Description
ADID	Application ID
IA	Input arrival date and time *YYMMDDHHMM*
JOBNAME	z/OS Job name
OPNO	Operation number
WSNAME	Workstation name

SELECT LTOC/LTOCCOM – Long Term Plan Occurrence

Table 108. SELECT LTOC/LTOCCOM – Long Term Plan Occurrence

Argument Name	Description
ADID	Application ID
GROUP	Authority group
GROUPDEF	Group definition ID
IAD	Input arrival date *YYMMDD*
IAT	Input arrival time *HHMM*
OWNER	Owner ID

SELECT OI/OICOM – Operator Instructions

Table 109. SELECT OI/OICOM – Operator Instructions

Argument Name	Description
ADID	Application ID
OPNO	Operation number
VALTO	Valid-to date and time *YYMMDDHHMM*

SELECT PR/PRCOM - Period

Table 110. SELECT PR/PRCOM - Period

Argument Name	Description
PERIOD	Period name
PRTYPE	Period type

SELECT RGCOM – Run Cycle Group

Table 111. SELECT RGCOM – Run Cycle Group

Argument Name	Description
RGID	Run cycle group ID
RGOWNER	Run cycle group owner
RGCALEND	Run cycle group calendar
RGVARTAB	Run cycle group variable table
RUNNAME	Run cycle name

Table 111. SELECT RGCOM – Run Cycle Group (continued)

Argument Name	Description
RUNCAL	Run cycle calendar
RUNVTAB	Run cycle variable table
RUNSETID	Run cycle subset ID

SELECT SR/SRCOM – Special Resource

Table 112. SELECT SR/SRCOM – Special Resource

Argument Name	Description
RESGROUP	Special resource group
RESHIPER	DLF resource indicator
RESNAME	Special resource name

SELECT WS/WSCOM – Workstation

Table 113. SELECT WS/WSCOM – Workstation

Argument Name	Description
WSAUTO	Automation workstation, Y or N
WSNAME	Workstation name
WSREP	Workstation reporting attribute
WSRETYPE	Remote engine type: **D** Distributed **Z** z/OS **blank**
WSTYPE	Workstation type
WSVIRT	Virtual workstation, Y or N
WSWAIT	WAIT workstation, Y or N
WSZCENTR	z-centric workstation, Y or N

SELECT WSV/WSVCOM – Virtual workstation destination

Table 114. SELECT WSV/WSVCOM – Virtual workstation destination

Argument Name	Description
WSNAME	Virtual workstation name.
WSDEST	Destination name. For using the controller itself as a destination enter a value of `********`.

Note: Resources `WSV` and `WSVCOM` are valid only for the `MODIFY` request.

SETSTAT – Sets a Condition status

The `SETSTAT` request changes the condition status from undecided to true or false, if the original status was undecided because of missing step-end information.

```
SETSTAT <resource> <arguments>
```

It produces the same result as the `T` and `F` commands available from the MCP dialog.

Return codes are:

0

The request was successful.

8

The request was not successful. An error message is issued to the message log data set.

SETSTAT CPSIMP – Condition dependency

Table 115. SETSTAT CPSIMP – Condition dependency

Argument Name	Description
PREADID	Predecessor application name.
PREIA	Predecessor input arrival date and time *YYMMDDHHMM*.
PREOPNO	Predecessor operation number.
PROCSTEP	Use it to define a step level dependency. If the step is not in a procedure, this parameter identifies the job step name, otherwise it identifies the step name in the JCL procedure. It must correspond to a step specifying the EXEC PGM= statement.
STEPNAME	Use it together with PROCSTEP when defining a step level dependency, only if the step is in a procedure, to identify the procedure invocation step name.

Table 115. SETSTAT CPSIMP – Condition dependency (continued)

Argument Name	Description
NEWSTAT	Requested status: **F** False **T** True

Note:

1. PREADID, PREIA, PREOPNO, PROCSTEP, and STEPNAME are used to identify the dependency to update.
2. The `SETSTAT` request is available only starting from HCL Workload Automation for Z version 8.5, or later.

TERM – Terminate HCL Workload Automation for Z session

Use the TERM request to terminate the programming interface session and perform the cleanup processing.

`TERM`

The cleanup processing performs the following actions:

- FREEMAIN of storage
- Closes data sets
- Detaches subtasks
- Terminates the HCL Workload Automation for Z session.

Before Workload Automation Programming Language terminates, it automatically performs any required `TERM` requests.

To communicate with more than one subsystem during a Workload Automation Programming Language program, you must terminate a session before initializing a new session. Use the `TERM` command before the `INIT` command to another subsystem. However, Workload Automation Programming Language automatically generates a `TERM` session if a new session is required before performing a new `INIT`.

Chapter 6. Current Plan Operation commands

Use the Current Plan Operation commands to perform specific functions against operations in the current plan, using a common set of keywords to identify the operation you want.

These commands are designed to simplify the process of finding and updating operations in the current plan.

Common syntax

Use the current plan operation commands to work with operations, by using a common syntax with specific sets of keywords.

The current plan operation commands use the following sets of keywords:

Identification keywords

To identify the potential operations to update, using combinations of Application ID, Job name, Workstation, Input Arrival, Operation, and Status.

Filter keywords

To choose one or more operation from the list of identified operations.

Data keywords

To manage the data related to the list of identified operations.

Command keywords

Specific to the action to be performed.

You can use the special keyword `UPDATE(NO|YES)` to verify the process before it is performed. The keyword determines if the commands actually performs the updates: if you specify `NO`, the command reports what it would do to each operation, but does not perform any updates. The default value is `YES`, but this default can be influenced by `OPTIONS UPDATE`.

Each command can have its own specific optional keywords, followed by common keywords to identify and select the operations:

```
<command> <arguments>    ADID(<adid>) IA(<iadatetime>)| ==
                         JOBNAME(<jobn>) OPNO(<opno>) + other identification keywords
                         WSNAME(<wsid>) DATE(<iadate>) TIME(<iatime>)
                         STATUS(<status>) USRF(<name>=<value>)
                         RANGE(<range>) POSITION(EARLIEST|LATEST)
                         COUNT(<count>)
                         OBJECT(<object>) SAVELIST(<list>) USELIST(<list>)
                         [FAIL(SKIP|STOP)]
```

The `MATCHTYP` keyword must be used if the `<value>` for `USRF` contains an asterisk (*) or percent sign (%) that is to be treated as non-wildcard character. `MATCHTYP` is not needed for `STATUS(*)`, because it is always treated as “Ready with none-reporting predecessor”.

Identification keywords

Use the following keywords to identify the potential operations to update. You can use any combination of the following keywords.

`ADID(<adid>)`

Searches for operations within applications with name **`<adid>`** for the operations to update. You can use the wildcards asterisk (*) and percent sign (%).

`CLNSTAT(C|E)`

Searches for operations by cleanup result.

`CLNTYPE(A|I|M|N)`

Searches for operations by cleanup type.

`CONDRJOB(Y|N)`

Searches for operations depending on whether they are conditional recover jobs.

`DPREM(Y|N)`

Searches for jobs depending on whether they are marked as removable from the current plan by planning processes.

`ERRCODE(`*`xxxx`*`)`

Searches for operations with the specified error code.

`EXECDEST(`*`xxxxxxxx`*`)`

Searches for jobs that executed on the specified destination.

`EXPJCL(Y|N)`

Searches for operations based on the expanded JCL setting.

`EXTNAME(<extended-jobname>)`

Searches for operations by extended job name.

`EXTSE(`*`xxxxxxxxxxxxxxx`*`)`

Searches for operations by scheduling environment.

`GROUP(`*`xxxxxxxx`*`)`

Searches for operations by authority group.

`IA(<iadatetime>)`

Alternative way to set the **`DATE`** and **`TIME`** keywords in a single keyword. Use the equal sign (=) to set the current date and time; use the plus sign (+) and minus sign (–) to set a relative date, the time portion will be set to asterisk (*). If you specify only a date, the time will be set to asterisk (*).

==

Special short form for a Workload Automation Programming Language job that is being controlled by HCL Workload Automation for Z. If you specify **==**, the **ADID**, **DATE**, and **TIME** values are set to the same values as the controlling occurrence, constraining the command to search only for operations in the same occurrence.

JOBCRT(P|W|N)

Searches for operations by critical job type.

JOBNAME(<jobname>)

Searches for jobs named **<jobname>**. You can use wildcards asterisk (*) and percent sign (%).

JOBPOL(|L|D|S|C)

Searches by late job policy.

MONITOR(Y|N)

Searches for operations by their external monitor setting.

OPNO(<opno>)

Searches for an operation numbered **<opno>**. You cannot use wildcards.

OWNER(*xxxxxxxxxxxxxxxx*)

Searches for operations by owner ID.

PRIORITY(*n*)

Searches for operations by priority.

SHADOWJ(Y|N)

Allows shadow jobs to be identified.

UNEXPRC(Y|N)

Allows conditional jobs with unexpected return codes to be identified.

USRSYS(Y|N)

Allows jobs that store user sysout to be identified.

VIRTDEST(*xxxxxxxx*)

Searches for jobs submitted on the specified destination.

WAITFORW(Y|N)

Allows wait operations to be identified.

WAITSE(Y|N)

Allows operations waiting for a scheduling environment to be identified.

WLMSCLS(*xxxxxxxx*)

Searches for operations by WLM service class.

WMPRED(Y|N)

Allows operations waiting for mandatory pending predecessors to be identified.

WPMPRED(Y|N)

Allows operations waiting for either mandatory pending or pending predecessors to be identified.

WPPPRED(Y|N)

Allows operations waiting for pending predecessors to be identified.

WSNAME(<wsname>)

Searches for operations scheduled on the **<wsname>** workstation. You can use wildcards asterisk (*) and percent sign (%).

STATUS(<status>)

If not specified, the command uses the status default values, as described hereafter. You can use the **NE** modifier to specify all statuses except one. For example, **STATUS-NE(C)** finds all statuses except Complete (C). **STATUS(*)** selects only the status of * (Ready with none-reporting predecessor). To find more than one status, you can list all the statuses in the same keyword; for example **STATUS(AR*)** lists the statuses Arriving, Ready and Ready with none-reporting predecessor. To list all possible values for status, use the percent sign (%), like in **STATUS(%)**.

DATE(<*yymmdd*>) (or **IADATE**)

Searches for operations scheduled with a particular application input arrival date in the format *YYMMDD*. You cannot use wildcards, but you can use relative dates:

=

The current date

+n

The current date + *n* days

-n

The current date - *n* days

TIME(<hhmm>) (or **IATIME**)

Searches for operations scheduled at a particular application input arrival time in the format ***HHMM***. **TIME** cannot be used without **DATE** and shows as an invalid value if **TIME** is set without **DATE**. You can use only the wildcard asterisk (*) for the complete time field, but you can also use relative times:

=

The current time

+n

The current time + *n* minutes

-n

The current time - *n* minutes

USRF(<name>=<value>)

Searches for operations with the specified user field. The equal sign (=) is used to separate the field name from the field value, but you can also use modifiers. For example, **USRF-GE(MYFIELD=100)** looks for operations where user field MYFIELD is set to a value of 100 or greater.

You can use comparators can be used, for example**DATE-LE**, on all identification keywords except **TIME**. The **DATE** and **TIME** fields are combined to form an Input Arrival time search, therefore the comparator specified against **DATE** is used as the comparator for the combined Input Arrival.

Depending on the command, the default value for **STATUS** is:

FORCE

AR*

KILL

S

REPLY

E

For all the other commands, the default value is ARW*.

Filter keywords

Use the filter keywords to select the operations to be modified from the list found by the identification keywords.

The following list describes the filter keywords:

RANGE(<yymmddhhmm> TO <yymmddhhmm>)

Limits the selection to the operations within the specified application input arrival range. The format is ***YYMMDDHHMM*** but if you omit it, the time 0000 is used as the “from” time and 2359 is used as the “to” time. You can specify open ended ranges; for example, **RANGE(TO 091231)** or **RANGE(090101 TO)**, **RANGE(090101)** corresponds to **RANGE(0901010000 TO 0901012359)**. You cannot use wildcards, but you can use relative dates when time the is not specified:

=

Current date

+n

Current date ***+n*** days

-n

Current date ***-n*** days

POSITION(EARLIEST|LATEST)

Determines where to start in the list of identified operations for processing. **EARLIEST** starts at the operation with the earliest Input Arrival date, **LATEST** starts with the latest. The number of operations that will be modified depends on the **COUNT** keyword. The default value is **EARLIEST**.

COUNT

Determines the number of operations to select for processing. The default is **1**, therefore **POSITION(EARLIEST)** selects only the earliest operation it found within the **RANGE**, while **POSITION(LATEST)** only selects the latest. Any positive number selects that number of operations from the starting position included; if you specify a number higher than the number or records found matching the criteria, all matching operations are selected. A value of **COUNT(0)** selects however many operation are found matching the criteria.

A negative value selects however many operations are found, minus the number in count. For **POSITION(EARLIEST)**, a negative count drops the latest entries off the list, for **POSITION(LATEST)** a negative count drops the earliest entries off the list. For example, **POSITION(EARLIEST) COUNT(-1)** selects all except the latest operation found; **POSITION(LATEST) COUNT(-1)** selects all except the earliest operation found.

Note:

1. **RANGE** keywords are processed before **POSITION** and **COUNT**

   For example, you could find 20 operations, but the **RANGE** might filter that down to 10. **POSITION** will start at the earliest or latest operation within the 10, so at most you will be able to select 10 operations.

2. **MATCHTYP** is not compatible with these commands due to the use of not equals comparators for handling multiple dependencies. This means that for user fields and special resource names, the asterisk (*) and percent sign (%) are considered wildcards if used in keywords, and use of comparator characters, such as =, ¬=, <, >, <= and >= at the end of a keyword, are considered as field comparators and not part of the field or resource name.

Data keywords

Use the data keywords to specify what to do with the data identified by the command.

The following list describes the data keywords:

DISPLAY(YES|NO)

Causes a simple line display of each operation found. The display includes the Application ID, Input Arrival, Workstation, Operation number, and Job name. It is followed by a description of the status, and key information relating to that status.

- For A, R, and * – SUB=N shows when SUBMIT is set to No.
- For W – PR(n/max) shows the number of predecessors when n=complete and max=total number of predecessors.

- For W – CPR(n) shows the number of conditional predecessors.
- For A, R, *, and W – SR(n) shows the number of special resources.
- For E – The error code is shown.
- For C and E – The Job ID is shown.
- For A, R, and * with Time Dependencies – The time dependency is shown.
- For S – The start time is shown.
- For C, D, E, and X – The completion time is shown.

FORMAT(FIXED|VARIABLE)

Sets the format for the **DISPLAY**: **FIXED** keeps a fixed width for all fields, **VARIABLE** strips trailing spaces and leading zeroes (default).

OBJECT(<object>)

Specifies the prefix of a set of object variables to store the details of each record identified by the **LIST** command used as part of the Current Plan Operation command. The base object will contain the number of ALL of the records identified by the **LIST** statement, even ones later discarded by the filter arguments. Each record is contained in a numerically suffixed object variable, the list of records that have passed the filter criteria may be found in the **@FILTER** attribute of the object.

SAVELIST(<list>)

Saves a list of the selected operations for use by other commands.

USELIST(<list>)

Uses a previously saved list of operations to drive the actions for the command. Lists from other Current Plan Operation commands and **LIST CPOPCOM** can be used. The **USELIST** keyword is mutually exclusive with other Identification, Filter, and Data arguments.

FAIL(SKIP|STOP)

When **COUNT** is set to 0 or a number higher than 1, or **USELIST** is specified, one of the actions might fail, leaving the other operations in the list to be processed. Typically this occurs at occurrence level, such as an occurrence being locked in the current plan. The **FAIL** keyword determines what to do with the rest of the list:

SKIP

Skips the updates to the failing occurrence and continues from the next occurrence in the list. When **SKIP** is specified, **OPTIONS CPFAIL** is temporarily considered as ERROR, therefore an ABORT condition is not generated. If errors are found when **FAIL(SKIP)** is specified, the command ends with RC=8.

STOP

If an error is found the command stops, and further operations are not processed. The command ends with a return code in accordance with the setting of **OPTIONS CPFAIL**, therefore **CPFAIL(ERROR)** ends with RC=8, and **CPFAIL(ABORT)** fails with RC=12.

For example, the following command:

```
FIND JOBNAME(WT0005) OBJECT(CPO) POSITION(LATEST) COUNT(2)
DISPLAY "Number of objects found" @V(@CPO)
DISPLAY "Filtered list" @V(@CPO-@FILTER)
```

returns the following output:

```
Number of objects found 5
Filtered list 4 5
```

You can then identify the filtered records by using a loop to extract the record numbers from the `@FILTER` attribute:

```
VARSET LOOPMAX = WORDS(@V(@CPO-@FILTER))
DO X = 1 TO !LOOPMAX
  VARSET Y = WORD(@V(@CPO-@FILTER),@V(X))
  DISPLAY "IA="||@V(@CPO!Y.-CPOPIA)
END
```

which would return the following output:

```
IA=1408251241
IA=1408251242
```

Performance considerations

The Identification keywords run a query against the HCL Workload Automation for Z current plan, and extract key information for every operation that matches. The Filter keywords filter down this information before taking action. The more precise you are in the Identification keywords, the faster the process will complete. Therefore, consider how many matches you might expect to find on the current plan at any one time using the Identification keywords when considering the estimated duration of the command.

Note: If you use `COUNT(0)`, these commands can produce many updates to the current plan, across many different occurrences. The command automatically performs a `PIF EXECUTE MCPBLK` command after every change of occurrence, to ensure that no more than 255 operations are modified in a single transaction.

Relative date and variables

To produce relative dates, can use the equal sign (=), plus sign (+), or minus sign (-) in the `DATE`, `TIME`, and `RANGE` keywords.

By default, the date and time is the current date and time when the command is processed. However, you can change this value to be relative to the Input Arrival time of the job running the commands by using the `OPTIONS` statement and set the Workload Automation Programming Language internal `DATE` and `TIME` to match the variables containing the running job's input arrival details.

For example, to `NOP` a job the day before the input arrival date of the running job use the following command:

```
VARSUB SCAN
OPTIONS DATE(!OYMD1.) TIME(!OHHMM)
NOP JOB(MYJOB) WSNAME(CPU1) DATE(-1)
```

Note: `DATE`, `TIME`, and `RANGE` refer to the Application Input Arrival time of the operation being targeted, not to the Operation Input Arrival time.

Automatic detection of current state of operation

The Current Plan Operation commands can be targeted to perform commands that might already have been actioned in some way, either by manual operator action or some other automation.

For example, you could run a command that is trying to HOLD an operation that is already held, or add a dependency that is already present. In most cases, the HCL Workload Automation for Z PIF would fail with RC 8, but the Current Plan Operation commands check the current state before attempting to perform the command. Therefore, if you attempt to HOLD an operation that is already held, or QUEUE for an operation that is already a predecessor, Workload Automation Programming Language issues message EQQI162A and the command ends with RC 0.

If you require notification when such conflicts occur, use the SETSEV command to alter the severity of message EQQI162A to issue a return code. For example:

```
07/02 12.08.22 EQQI200I SETSEV 162W
07/02 12.08.22 EQQI007A Message 162 changed from A to W
07/02 12.08.22 EQQI299I Statement completed - RC=0
07/02 12.08.22 EQQI200I QUEUE SUCC1
07/02 12.08.23 EQQI013A Job QUEUEJOB,JOB01725 in QUEUE 1407011812 CPU1_005
07/02 12.08.24 EQQI161A SELECTED: MODOPTEST1 1406281807 MANC_020 SUCC1 W
07/02 12.08.24 EQQI162W No action - already in correct state
07/02 12.08.24 EQQI299I Statement completed - RC=4
```

SAVELIST and USELIST

Use these commands to save a list of the records that they process into a `SAVELIST` format. This allows a series of commands to be guaranteed to run across the same set of operations.

Use the `FIND` command to find a set of operations and save the list without performing any action. You can use other commands to perform the needed actions. For example:

```
FIND MYJOB SAVELIST(MYLIST)
HOLD USELIST(MYLIST)
ALTER USELIST(MYLIST) DROPPRED(EXTERNAL)
ALTER USELIST(MYLIST) NEW_STATUS(C)
RELEASE USELIST(MYLIST)
```

Relationship to the EQQWXMOD WAPLEXEC

The `EQQWXMOD` WAPLEXEC program was the prototype for many of these commands and they have been transferred into the base language of Workload Automation Programming Language. There are a few amendments in syntax to avoid conflicts with existing Workload Automation Programming Language commands and some commands have been combined.

- The **MODIFY** command within **EQQWXMOD** becomes the **ALTER** command within base Workload Automation Programming Language.
- The **MOVETO** command is now also part of the **ALTER** command with a new **NEW_WSNAME** keyword specifying the workstation to move to.
- The **RELEASE** keyword of **MOVETO** has now become the **DROPSUCC** keyword of the **ALTER** command.
- The **UPDATE** keyword has been moved from the PARM= route into being a keyword in each command but can still be passed via the PARM= route as this will set **OPTIONS UPDATE**.
- The **STATUS** keyword now allows multiple status values to be specified in a single command. This was a restriction for **EQQWXMOD** but Workload Automation Programming Language V3.3 lifted this with changes in the core engine.
- New commands **BIND**, **KILL** and **REPLY** added to this set of commands that were not part of the **EQQWXMOD** set.
- The **NOW** keyword is not supported in base Workload Automation Programming Language, because you can use **OPTIONS DATE** and **OPTIONS TIME** instead.

The **EQQWXMOD** WAPLEXEC is now being deprecated and is not developed further to accommodate any new features of HCL Workload Automation for Z. Use the equivalent commands within Workload Automation Programming Language, instead.

ALTER

Use the **ALTER** command to modify the attributes on an operation.

```
ALTER <identification> <filter> <data> <modify-cpop-keywords>
      [NEW_WSNAME(<wsname>)] [NEW_JOBNAME(<jobname>)] [NEW_STATUS(<status>)]
      [UFNAME(<usrf_name>)] [UFVALUE(<usrf_value>)]
      [DROPPRED(INTERNAL|EXTERNAL|ALL|<pred>)]
      [DROPSUCC(INTERNAL|EXTERNAL|ALL|<succ>)]
      [DROPUSRF(<ufname>)]
      [DROPSR(<special-resource>)]
```

where:

<identification>

Keywords to identify the operations to update. For details, see Identification keywords on page 186.

<filter>

Keywords to select the operations from the list. For details, see Filter keywords on page 189.

<data>

Keywords to store data from the command. For details, see Data keywords on page 190.

<modify-cpop-keywords>

Additional keywords to specify the changes you want to make to the operation. The following keywords are available (for detailed information, see MODIFY CPOP – Current Plan Operation on page 160):

- **ASUB(Y|N)**
- **CLATE(Y|N)**
- **DEADWTO(Y|N)**
- **DESC(*up to 24 characters*)**

- **DURATION(*100th of seconds*)**
- **EDUR(*HHMM*)**
- **FORM(*form number*)**
- **HRC(0-4095)**
- **JCLASS(*job class*)**
- **NEW_CLNTYPE(A|I|M|N)**
- **NEW_CONDRJOB(Y|N)**
- **NEW_ERRCODE(*errcode*)**
- **NEW_EXPJCL(Y|N)**
- **NEW_JOBCRT(N|P|W)**
- **NEW_JOBNAME(*job name*)**
- **NEW_JOBPOL(C|D|L|S)**
- **NEW_MONITOR(Y|N)**
- **NEW_STATUS(A|C|E|I|R|S|U|X|*)**
- **NEW_USRSYS(Y|N)**
- **NEW_WLMSCLS(*WLM service class*)**
- **NEW_WSNAME(*workstation name*)**
- **OPDL(*YYMMDDHHMM*)**
- **OPIA(*YYMMDDHHMM*)**
- **PSUSE(1-99)**
- **R1USE(1-99)**
- **R2USE(1-99)**
- **RERUT(Y|N|blank)**
- **RESTA(Y|N|blank)**
- **TIMEDEP(Y|N)**
- **USERDATA(operation user field)**

UFNAME

Optional keyword that specifies the name of a user field to be updated for the selected job. If **UFNAME** is specified and **UFVALUE** is not, the field is set to blanks.

UFVALUE

Optional keyword that specifies the value to set the user field named by **UFNAME** for the selected job. If **UFVALUE** is specified, **UFNAME** must also be specified.

DROPPRED and **DROPSUCC**

Optional keywords that can cause the operation's predecessor or successor links to be deleted. The valid values are:

INTERNAL

Release only internal successors.

EXTERNAL

Release only external successors.

ALL

Release all internal and external successors.

The definition of an individual dependency

In the form **[admask][/opno|jobmask][/iatime]**. For example, **ABC*/255/!OYMD1.*** would match operation 255 in any dependencies within applications beginning with ABC with an input arrival date matching the operation being modified. Or ***/XYZ*** would match any dependency to a job beginning with XYZ, regardless of the application or input arrival.

DROPSR

Optional keyword to drop a special resource, or resources. The value can be an absolute name, or use wildcards. **MATCHTYP** cannot be used with this command, therefore do not use special resources whose names contain wildcards.

DROPUSRF

Optional keyword to drop user fields from an operation. You can specify wildcards. **DROPUSRF(*)** drops all the user fields from an operation.

Managing split or inconsistent occurrences

In HCL Workload Automation for Z, an occurrence must always be consistent. This means that you must ensure that every operation is connected directly or indirectly.

To remove internal successors, the application must be designed in such a way that the removal of the dependencies does not create an inconsistent application. By default, if the result of removing internal dependencies would result in some operations becoming entirely separate from the rest of the occurrence then the command will fail. When you use**DROPPRED** or **DROPSUCC**, ensure that the application is designed specifically to manage such eventualities, in preference to **OPTIONS DROP**.

The **OPTIONS DROP** keyword provides 2 mechanisms that prevents occurrence inconsistencies:

- Making both sides of the broken dependencies a successor to a named operation number, providing that the named operation number has the status Complete.
- Adding an operation using a named workstation and job name, or reuse a matching pre-existing one that has no successors, and make this a success to both sides of the broken dependency.

The syntax of the **OPTIONS DROP** keyword is **DROP(<predop>,<succws>,<succjob>,<succtext>)**, where:

<predop>

A named operation number, for example 001, to be used as a predecessor to both sides of a dropped dependency, but only if **<predop>** is in Complete status. It does not add **<predop>**, if it does not already exist.

<succws>

Name of the workstation that is used to become a successor to both sides of the dropped dependency. This must be a non-reporting general workstation.

<succjob>

Name of the job that is used to become a successor to both sides of the dropped dependency. The default is **ZRELINK**.

<succtext>

Operation description that is used if a successor operation is inserted. The default is **Relink dropped deps**.

Figure 1: Example of two automatic relinking scenarios on page 198 shows the following scenarios:

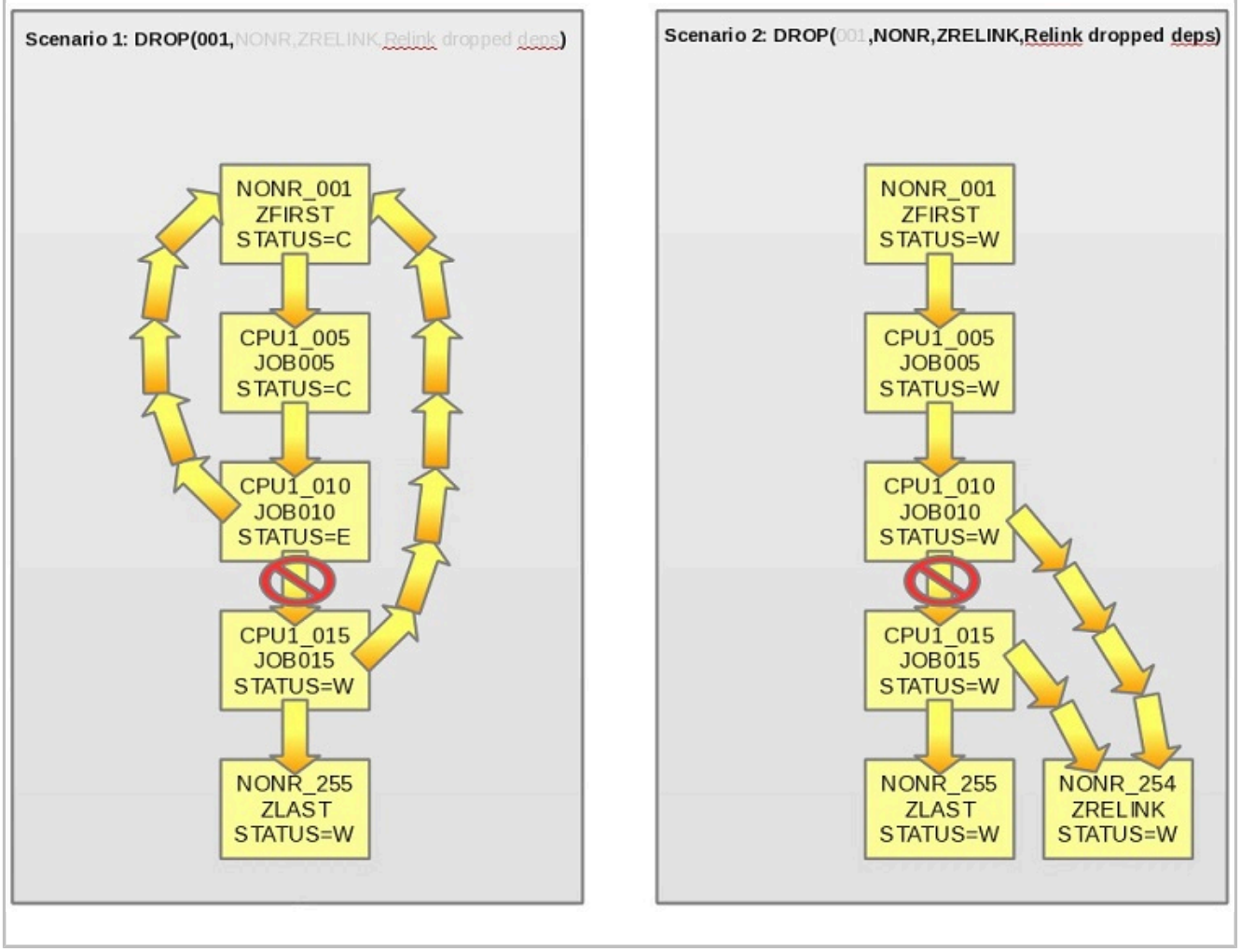
Scenario 1

When the dependency between operations 010 and 015 is dropped, the **<predop>** operation 001 that was set by **OPTIONS DROP** is in status C. This allows it to be connected as a predecessor to operations 010 and 015, preventing an occurrence split.

Scenario 2

When the dependency between operations 010 and 015 is dropped, the **<predop>** operation 001 that was set with **OPTIONS DROP** is in status W. This prevents the relink processing to occur, therefore the operation 254 is added using the **<succws>** and **<succjob>** keywords of **OPTIONS DROP**. This makes operation 254 a successor to operations 010 and 015, preventing in this way an occurrence split.

Figure 1. Example of two automatic relinking scenarios



When a dependencies is being dropped by this command, the following processing takes place in this order:

1. If the dropping of the dependency does not cause an occurrence split, no additional dependencies are added.
2. If `<predop>` is specified in `OPTIONS DROP` and the identified operation is in status Complete, both sides of the dropped dependency become successors to `<predop>`.
3. If `<predop>` is not specified or it has a status different from Complete, if `<succws>` is specified it will look for an operation on the `<succws>` workstation with the `<succjob>` job name with no successors. If a match is found, this operation becomes a successor to both sides of the dropped dependency.
4. If `<succws>` is specified, but no match is found within the occurrence, the highest unused operation number is identified and added to the occurrence to become a successor to both sides of the dropped dependency. If there are no unused operation numbers, an operation is not added.
5. If none of the preceding processes are specified or criteria are not met, the command fails to drop the dependencies due to the occurrence split.

Note:

1. If there is already a dependency in place to a valid **<predop>** or **<succws>** operation, the dependency is not dropped, even if identified by the **ALTER** command as being a dependency to drop.
2. Specifying **OPTIONS DROP** causes batch to carry on processing from the point of the dropped dependency, in contradiction to the original design of the application. If **OPTIONS DROP** is coded in a default **OPTIONS** member, this could be unknown to some of your user base, and result in unexpected behavior. **OPTIONS DROP** must be set only in the jobs where it is to be applied.

BIND

Use the **BIND** command to trigger an operation on a shadow workstation, to attempt to bind to its counterpart on the remote engine.

```
BIND <identification> <filter> <data>
```

where:

<identification>

Keywords to identify the operations to update. For details, see Identification keywords on page 186.

<filter>

Keywords to select the operations from the list. For details, see Filter keywords on page 189.

<data>

Keywords to store data from the command. For details, see Data keywords on page 190.

FIND

Use **FIND** to find an operation, or a set of operations, matching the input criteria. It performs no action against the operation, therefore you can use it to create **OBJECT** variables or **SAVELIST**s to be later processed by other commands.

```
FIND <identification> <filter> <data>
```

where:

<identification>

Keywords to identify the operations to update. For details, see Identification keywords on page 186.

<filter>

Keywords to select the operations from the list. For details, see Filter keywords on page 189.

<data>

Keywords to store data from the command. For details, see Data keywords on page 190.

FORCE

Use **FORCE** to submit an operation that could otherwise wait for special resources, workstation shutdown or that has job submission disabled. If the operation has predecessors outstanding, it cannot be forced. This command is equivalent to the **EX** command in ISPF panels.

```
EXECUTE <identification> <filter> <data>
```

where:

<identification>

Keywords to identify the operations to update. For details, see Identification keywords on page 186.

<filter>

Keywords to select the operations from the list. For details, see Filter keywords on page 189.

<data>

Keywords to store data from the command. For details, see Data keywords on page 190.

HOLD

Use **HOLD** to add a Manual Hold condition to an operation. This means the operation waits to be manually released after all of its prerequisites have been met.

```
HOLD <identification> <filter> <data>
```

where:

<identification>

Keywords to identify the operations to update. For details, see Identification keywords on page 186.

<filter>

Keywords to select the operations from the list. For details, see Filter keywords on page 189.

<data>

Keywords to store data from the command. For details, see Data keywords on page 190.

Note: If you try to hold an operation that is already held, the command fails with RC=8.

KILL

Use **KILL** to kill an operation on a z-centric workstation.

```
KILL <identification> <filter> <data> [TYPE(JOB|RECOVERY)]
```

where:

<identification>

Keywords to identify the operations to update. For details, see Identification keywords on page 186.

<filter>

Keywords to select the operations from the list. For details, see Filter keywords on page 189.

<data>

Keywords to store data from the command. For details, see Data keywords on page 190.

NOP

Use **NOP** to specify a No Process condition against an operation. This means that when the operation's prerequisites have been met, HCL Workload Automation for Z sets the operation complete.

```
NOP <identification> <data> <filter>
```

where:

<identification>

Keywords to identify the operations to update. For details, see Identification keywords on page 186.

<filter>

Keywords to select the operations from the list. For details, see Filter keywords on page 189.

<data>

Keywords to store data from the command. For details, see Data keywords on page 190.

QUEUE_BEHIND

Use **QUEUE_BEHIND** to make become the running job, or other jobs in the same occurrence as the running job, a successor behind the identified operations. The **QUEUE_BEHIND** command can be abbreviated to **QUEUE**.

```
QUEUE_BEHIND <identification> <filter> <data> [CONNECT(THIS|SUCC)]
             [INSERT(CONNECTED|ENDPOINT|NONE)]
```

where:

<identification>

Keywords to identify the operations to update. For details, see Identification keywords on page 186.

<filter>

Keywords to select the operations from the list. For details, see Filter keywords on page 189.

<data>

Keywords to store data from the command. For details, see Data keywords on page 190.

CONNECT

Specifies the job to connect to the operations identified by the command. If set to **THIS**, the job currently running the command becomes a successor to the identified jobs. If set to **SUCC**, the internal successors of the job currently running the command becomes a successor to the identified jobs. If set to **THIS**, the job returns to the Waiting status until the newly applied predecessors have run. After the predecessors have run, the job is rerun and might find new predecessors. If **CONNECT(SUCC)** is specified and the job running the command has no successors, the command fails.

INSERT

Specifies whether to insert the connected job, or occurrence after the identified job, so that it becomes a predecessor to the successors of the identified job. Specifying **CONNECTED**, makes the jobs identified by the **CONNECT** keyword a predecessor to the successors of the operations identified by the command. Specifying **ENDPOINT** will find the end points of the occurrence and make them a predecessor to the successors of the identified job. Specifying **NONE** will not insert the running occurrence before the successors of the identified job (this is the default). If anything other than **INSERT(NONE)** is specified and the job identified by the **QUEUE_BEHIND** command has no successors, the command fails.

Note: The **QUEUE_BEHIND** command is designed to influence the occurrence where the job running the command is contained. If the command is run by a job not being controlled by HCL Workload Automation for Z, it fails.

RELEASE

Use **RELEASE** to remove a Manual Hold condition from an operation. This means that the operation is ready to be run after all of its prerequisites have been met.

```
RELEASE <identification> <filter> <data>
```

where:

<identification>

Keywords to identify the operations to update. For details, see Identification keywords on page 186.

<filter>

Keywords to select the operations from the list. For details, see Filter keywords on page 189.

<data>

Keywords to store data from the command. For details, see Data keywords on page 190.

Note: If you try to release an operation that is not already held, the command fails with RC=8.

UNNOP

Use **UNNOP** to remove a No Process condition against an operation.

Using the **UNNOP** command means that when the operation's prerequisites are met, HCL Workload Automation for Z will no longer mark the operation complete.

```
UNNOP <identification> <filter> <data>
```

where:

<identification>

Keywords to identify the operations to update. For details, see Identification keywords on page 186.

<filter>

Keywords to select the operations from the list. For details, see Filter keywords on page 189.

<data>

Keywords to store data from the command. For details, see Data keywords on page 190.

Chapter 7. Current Plan Occurrence commands

Use the Current Plan Occurrence commands to perform specific functions against all operations within a single occurrence in the current plan, using a common set of keywords to identify the occurrence.

These commands are designed to simplify the process of updating operations within a single occurrence in the current plan. They are primarily designed to be run from the initial operation in an occurrence to influence the behaviour of the occurrence.

Common keywords

Use the Current Plan Occurrence commands to run commands against a specific occurrence by using a common syntax.

Two primary keywords are **`ADID(<application-name>)`** and **`IA(YYMMDDHHMM)`** to identify the occurrence with which you want to work. These keywords are needed only if you want to work with an occurrence from a job not running within the occurrence itself. If the occurrence to be modified is the one where the command is running, these keywords are not required.

You can use the special keyword **`UPDATE(NO|YES)`** to verify the process before it is performed. The keyword determines if the commands actually performs the updates: if you specify **`NO`**, the command reports what it would do to each operation, but does not perform any updates. The default value is **`YES`**, but this default can be influenced by **`OPTIONS UPDATE`**.

ALTIF – Alter operations if specific criteria are true

The **`ALTIF`** can be run as the first operation in an occurrence to conditionally **`ALTER`** other operations within the same occurrence.

```
ALTIF CRITERIA(<field-prefix>) [<common-keywords>]
```

where **`CRITERIA`** names a user field prefix to use for specifying date logic expressions against. The prefix is used to identify **`IF`**, **`DO`**, and **`EL`** user fields to define the date logic and a corresponding set of attributes to alter.

The command searches for matching user fields in each operation within the same occurrence. The fields must be coded with matching **`<field-prefix>`** and **`<tag>`** elements of the name with one **`IF`**, at least one **`DO`** user field, and optional **`EL`** fields, using the following convention:

`<field-prefix>-<tag>-IF`

: The user field with date logic.

`<field-prefix>-<tag>-DO`

: The user fields containing the **`ALTER`** keywords to act upon when the corresponding **`IF`** is true. At least one **`DO`** keyword must be coded for each **`IF`**.

`EL*`

: The user fields containing the **`ALTER`** keywords to act upon when the corresponding **`IF`** is false.

In the following example, **`ABC-01-IF`** is true on Monday, therefore **`ABC-01-DO`** is processed on Monday. **`ABC-02-IF`** is true only on Tuesday, therefore **`ABC-02-DO1`** and **`ABC-02-DO2`** are processed on Tuesday. On any other day, that operation is left unaltered, according to the day of the week.

```
ABC-01-IF = @(MON)
ABC-01-DO = OPIA(!OYMD1.1900)
ABC-02-IF = @(TUE)
ABC-02-DO1 = OPIA(!OYMD1.2100)
ABC-02-DO2 = OPIA(!OYMD1.2300)
```

In the following example, user field **ABC-03-IF** checks to see if the day is a workday: if true, it runs **ABC-03-DO**, if it is not a workday, it runs **ABC-03-EL1** and **ABC-03-EL2**.

```
ABC-03-IF = @(WORKDAY)
ABC-03-DO = TIMEDEP(Y)
ABC-03-EL1 = TIMEDEP(N)
ABC-03-EL2 = DROPSR('DONT.RUN.WITH.CICS')
```

The **IF** user field can contain any valid REXX Expression, containing date logic variables, Workload Automation Programming Language variables or HCL Workload Automation for Z JCL variables (for details, see IF-THEN-ELSE – Conditional execution on page 69). The variables are evaluated by the operation running the **ALTIF** command, hence their values are set in relation to that operation.

The **DO** and **EL** user fields can contain any action keywords from the **ALTER** command. The identification, filter, and data keywords are managed internally by the **ALTIF** command; you must use only the keywords that cause operation modification. Each individual **DO** or **EL** user field causes an individual **ALTER** statement to be run. The **ALTER** statements are run in the sort sequence of the user field names. For details, see ALTER on page 194.

Note:

- If you set an **-IF** User Field without at least one corresponding **-DO** user field, the **ALTIF** command fails with RC=8.
- When **UPDATE(N)** is set, to see the **ALTER** commands that would have been issued use **OPTIONS MSGLEVEL(2)**. To see the low level PIF commands that would have been issued, use **OPTIONS TRACE(1)**.

RUNIF – Run operations only if specific criteria are true

The **RUNIF** command can be run as the first operation in an occurrence to conditionally decide which operations are allowed to run, by moving the unwanted operations to a non-reporting workstation and optionally removing special resources and time dependencies.

```
RUNIF [USRF-EQ|NE(field=value)] [CRITERIA(<field-prefix>)]
        [DROPSR(YES|NO)] [DROPTIME(YES|NO)] [WSNAME(<workstation>)]
        [<common-keywords>]
```

where:

USRF

Specifies a user field and value to decide which operations to run. You can use the keyword comparators **EQ** and **NE**. When **EQ** is used, it runs only operations marked with the combination of field name and value. When **NE** is used, it runs only operations not marked with the combination of field name and value.

CRITERIA

Names a user field prefix to use for specifying date logic expressions against. The prefix is used to identify positive and negative rule user fields. For example, **CRITERIA(RUNME)** looks for user fields with the format **RUNME-POS-nn** and **RUNME-NEG-nn** to contain the date logic rules for each operation.

DROPSR(YES|NO)

Instructs the **RUNIF** command to remove special resources from operations identified as being ineligible to run. If resources are not removed, the operations could wait for special resources, even if they were moved to a non-reporting workstation. This might even affect the availability of resources if the use OnComplete. The default is **YES**.

DROPTIME(YES|NO)

Instructs the **RUNIF** command to remove any time dependency from any operations identified as being ineligible to run. If time dependencies are not removed the operation will wait until the time dependency has been satisfied, even if the operation was moved to a non-reporting workstation.

WSNAME

Specifies the workstation where the operations that are not required are moved by the **RUNIF** process. This must be a general non-reporting workstation.

All fields are optional, but you must specify at least one **USRF** and one **CRITERIA** . If both **USRF** and **CRITERIA** are specified, the **USRF** keyword is processed first, and only the operations that pass the **USRF** test have their **CRITERIA** fields processed.

When **CRITERIA** is used, the following considerations are made:

- If an operation has no user fields matching the **prefix-POS-*nn*** or **prefix-NEG-nn** format, the operation is considered eligible to run.
- If an operation has a user field matching the **prefix-POS-*nn*** format, the operation is considered eligible to run only if the expression contained within the user field is TRUE.
- If an operation has a user field matching the **prefix-NEG-*nn*** format, the operation is considered ineligible to run if the expression contained within the user field is TRUE.

nn is expected to be a two-digit numeric value, to ensure that the fields are evaluated in the correct order, because updates to the current plan might inadvertently change the sequence of user fields. It does not matter if ***nn*** is not numeric, but it is important to understand that the sort sequence of each individual field name decides the order in which the **CRITERIA** rules are combined. Multiple **prefix-POS-*nn*** or **prefix-NEG-*nn*** fields are not processed individually, but they are evaluated as a single positive or a single negative rule, using normal continuation rules to combine each user field value.

The following example shows a basic **RUNIF** usage, where:

- Operation 001 runs Monday to Friday
- Operation 005 runs only Monday
- Operation 010 runs Monday to Friday
- Operation 015 runs only on Mondays during October

- Operation 020 runs Monday, Tuesday, Thursday and Friday
- Operation 255 runs Monday to Friday

```
An application runs Monday to Friday
Operation 001 – Command –
EQQ-SYSIN-01 = RUNIF CRITERIA(RUNME) WSNAME(NONR)
Operation 005 – User Field RUNME-POS-01 = @(MON)
Operation 010 – No user fields
Operation 015 – User Field RUNME-POS-01 = @(MON) & @(OCT)
Operation 020 – User Field RUNME-NEG-01 = @(WED)
Operation 255
```

The contents of the **CRITERIA** user fields can contain Workload Automation Programming Language variables that can be set in the same operation as the **RUNIF** command. For substitution to take place, variable substitution must be activated in the same job as the **RUNIF** command.

The following example shows a basic **RUNIF** usage with variables as criteria, where:

- Operation 001 runs Monday to Friday
- Operation 005 runs Monday and Tuesday
- Operation 010 runs Monday and Tuesday
- Operation 015 runs Wednesday and Thursday
- Operation 020 runs Wednesday and Thursday
- Operation 255 runs Monday to Friday

```
An application runs Monday to Friday
Operation 001 – Command –
EQQ-SYSIN-01 = VARSUB SCAN
EQQ-SYSIN-02 = VARSETRULE1 = “@(MON) | @(TUE)”
EQQ-SYSIN-03 = VARSETRULE2 = “@(WEDE) | @(THU)”
EQQ-SYSIN-04 = RUNIF CRITERIA(RUNME) WSNAME(NONR)
Operation 005 – User Field RUNME-POS-01 = !RULE1
Operation 010 – User Field RUNME-POS-01 = !RULE1
Operation 015 – User Field RUNME-POS-01 = !RULE2
Operation 020 – User Field RUNME-POS-01 = !RULE2
Operation 255
```

The **USRF** keyword can be used to provide different execution routes through an application depending on the contents of a variable in a JCL variable table.

Varying the name of the user field to be used allows the same operation to be permissible on multiple routes through the application.

Both the variable value and presence of the **RUNIF** command can be varied at submission time if the application is added to the current plan dynamically.

Using the same user field name allows an application to be divided into sub applications, allowing the application to run as a whole if no **RUNIF** command is set, or only running a sub set of the operations if **RUNIF** is executed specifying a particular value for the user field.

The following example shows a basic **RUNIF** usage with variables to select the route, where:

When variable ROUTE in table MYTABLE is set to ROUTE1

- Operation 001 runs
- Operation 005 runs
- Operation 010 runs
- Operation 015 does not run
- Operation 020 runs
- Operation 025 does not run
- Operation 255 does not run

When variable ROUTE in table MYTABLE is set to ROUTE2

- Operation 001 runs
- Operation 005 runs
- Operation 010 does not run
- Operation 015 runs
- Operation 020 does not run
- Operation 025 runs
- Operation 255 does not run

```
An application runs Monday to Friday
Operation 001 – Command –
EQQ-SYSIN-01 = VARSUB SCAN(!) TABLE(MYTABLE)
EQQ-SYSIN-02 = RUNIF USRF(!ROUTE=Y) WSNAME(NONR)
Operation 005 – User Fields ROUTE1 = Y and ROUTE2 = Y
Operation 010 – User Fields ROUTE1 = Y and ROUTE2 = N
Operation 015 – User Fields ROUTE1 = N and ROUTE2 = Y
Operation 020 – User Fields ROUTE1 = Y and ROUTE2 = N
Operation 025 – User Fields ROUTE1 = N and ROUTE2 = Y
Operation 255 – No user fields, but already on workstation NONR
```

The following example shows a basic **RUNIF** usage with the **RUNIF** command being varied at submission time, where:

- EQQ-SYSIN-02 contains a commented template of the **RUNIF** command to be replaced at submission time by a real version of the **RUNIF** command. When submitted unmodified, all operations will run.
- When an occurrence is submitted as follows:

```
INSERT CPOC ADID(MYAPPL)
MODIFY CPOP OPNO(001)
MODIFY CPUSRF UFNAME(EQQ-SYSIN-02)
              UFVALUE('RUNIF USRF(SUBAPPL=GRPA) WSNAME(NONR)')
```

    - Operation 001 runs
    - Operation 005 runs
    - Operation 010 runs
    - Operation 015 runs
    - Operation 020 does not run
    - Operation 025 does not run
    - Operation 255 does not run
- When an occurrence is submitted as follows:

```
INSERT CPOC ADID(MYAPPL)
MODIFY CPOP OPNO(001)
MODIFY CPUSRF UFNAME(EQQ-SYSIN-02)
              UFVALUE('RUNIF USRF(SUBAPPL=GRPB) WSNAME(NONR)')
```

    - Operation 001 runs
    - Operation 005 does not run
    - Operation 010 does not run
    - Operation 015 does not run
    - Operation 020 runs
    - Operation 025 runs
    - Operation 255 does not run

```
Operation 001 - Commands -
EQQ-SYSIN-01 = VARSUB SCAN
EQQ-SYSIN-02 = /* RUNIF USRF(SUBAPPL=GRPx) WSNAME(NONR) */
Operation 005 - User Field SUBAPPL = GRPA
Operation 010 - User Field SUBAPPL = GRPA
Operation 015 - User Field SUBAPPL = GRPA
Operation 020 - User Field SUBAPPL = GRPB
Operation 025 - User Field SUBAPPL = GRPB
Operation 255 - No user fields, but already on workstation NONR
```

Note: This command changes the workstation where operations run, alters the time dependency attribute, and removes special resources from operations. If the `RUNIF` criteria were specified incorrectly and a rerun is required, you need to restore the occurrence to its original state, either by submitting a new occurrence in its place, or manually restoring the workstation names, time dependency attributes, and special resources, according to the database definition of the application.

Chapter 8. Function Based commands

The Function Based commands combine various low level PIF commands. Use the Function Based commands to perform specific scheduling or operational functions.

ADD – Add applications or groups to the current plan

Use the **ADD** command to add applications or groups of applications to the current plan. You can add a single occurrence (ONCE mode) or resubmit or restart the currently running occurrence (REPEAT mode).

When using this command, consider that:

- If the Input Arrival time is later than the end of the current plan, the **ADD** command adds the occurrence to the long-term plan, not to the current plan.
- When adding to the long-term plan:
  - If **DEPRES** is set to **Y**, **LTDRES** is set to **Y**, and some dependencies cannot be resolved, the **ADD** command adds the occurrences, resolve the dependencies that can be resolved, and end with RC=4. If
  - If **DEPRES** is set to **Y**, **LTDRES** is set to **N**, and some dependencies cannot be resolved, the **ADD** command fails when the application or operation related to a dependency is not defined in the Application Description.

```
ADD ADID(<adid>)|GROUPDEF(<groupdef>) [JCLVTAB(<table>)]
      [DEPRES(Y|N|P|S)] [LTDRES(YES|NO)]
      [IA(yymmddhhmm) | IADATE(yymmdd) IATIME(hhmm)] [FINDIA(Y|N)]
      [FROM(hhmm)] [UNTIL(hhmm)] [EVERY(hhmm)] [COUNT(nnnn)]
      [ORIGIN(START|END|IA)] [RESOLVE(NEXT|GAP|BOTH|hhmm)]
      [RESTART(opnum[,FLUSH])]
      [EARLY(ABORT|CONTINUE)] [NOTIFY(N|W|E)]
      [DLDAY(n)] [DLTIME(hhmm)] [LINK(YES|NO)]
      [HOLD(NO|START|ALL)] [UPDATE(YES|NO)]
```

The **ADD** command uses the same keywords used with other PIF and Workload Automation Programming Language commands.

For concistency with OCL, you can use **APPL** as an alternative to **ADID**, **GROUP** as an alternative to **GROUPDEF**, **VARTAB** as an alternative to **JCLVTAB**, and **CPDEPR** and **GDEPRES** as an alternative to **DEPRES**.

For consistency with repeating run cycles in Batch Loader, you can use **IATIME** as an alternative to **FROM**, **RPTEND** as an alternative to **UNTIL**, and **RPTEVRY** as an alternative to **EVERY**.

If you specified the **UNTIL** or **EVERY** keyword, the **ADD** command is used in REPEAT mode.

The following list shows the general keywords:

ADID|GROUPDEF

Indicates an application or a group to submit. If you specify **GROUPDEF**, the **ADD** command submits every application included in that application group. When operating in REPEAT mode, neither **ADID** or **GROUPDEF** are required, but you specify them they must match the running application or group.

JCLVTAB

Indicates a JCL variable table that can be attached to the occurrence.

DEPRES(Y|N|P|S)

Determines whether external dependencies are resolved when the occurrences are added to the current plan. This also affects the setting of **OPTIONS CPDEPR** or **LTDEPR** for the remainder of the Workload Automation Programming Language session, depending on the plan that is used. The default is Y. For additions to the long-term plan, **P** or **S** are translated to **Y**.

LTDRES(Y|N|P|S)

Determines whether, when resolving LTP external dependencies according to **DEPRES(Y)**, the **ADD** command fails or not if the application specified in the dependency is not defined in the Application Description.

When you set **LTDRES(N)** (default), the **ADD** command fails with RC=8. When you set **LTDRES(Y)** and the occurrence is to be added to the LTP, if some dependencies cannot be resolved because the application is not defined in the AD, the **ADD** command adds the occurrence, resolves the dependencies that can be resolved, and ends with RC=4.

UPDATE(YES|NO)

Determines whether the job actually performs the updates to the current plan or long-term plan. The default behaviour is to perform the updates, but you can modify it with **OPTIONS UPDATE**. If **UPDATE** is set to **NO**, the command processes everything and generate the commands to perform the update, but does not run them. If you specify **OPTIONS TRACE(1)**, messages EQQI906A are created to list the commands that would have been performed.

The following list shows keywords for running in ONCE mode:

IA(*yymmddhhmm*)

A fully qualified input arrival time. This keyword is mutually exclusive with **IADATE** and **IATIME**. You can use relative times; for example, **IA(+15)** creates an Input Arrival time 15 minutes later, and the date is adjusted if midnight is passed.

IADATE(*yymmdd*)

An input arrival date. If **IATIME** is not set, the current time is used in conjunction with the date specified. You can use relative dates; for example, **IADATE(+1)** uses tomorrow's date.

IATIME(*hhmm*)

An input arrival time to be set independently of the date. You can use relative times; for example, **IATIME(+15)** creates an Input Arrival time 15 minutes later, however if midnight is passed the date is not adjusted. Use the **IA** keyword only if you want to set a specific time.

FINDIA(Y|N)

If you specified **IA**, **IADATE**, or **IATIME**, the **ADD** command generates an Input Arrival time to use when adding the occurrence to the plan. If **FINDIA** is set to **N** and an occurrence already exists with the same Input Arrival, the submission fails. Setting **FINDIA** to **Y** causes the **ADD** command to find the next available Input Arrival time

following the one determined by the **ADD** command. If you specified **IA** or **IATIME**, the **FINDIA** keyword defaults to **N**; if you specified only **IADATE**, the **FINDIA** keyword defaults to **Y**. If you did not specify any **IA**, **IADATE**, or **IATIME**, the **ADD** command does not allocate an Input Arrival time, and is automatically assigned the next available time.

LINK(Y|N)

When you use the **ADD** command within a job controlled by HCL Workload Automation for Z, you can use **LINK** to make the newly added occurrence dependent on the submitting job, and become a predecessor to the successors of the submitting job. This ensures the dynamically added occurrence will complete before any successors to the submitting job are able to run. The process automatically finds the start and end points of the application being added. The **LINK** keyword can be used only when adding a single application.

LINK cannot be used with **GROUPDEF**.

HOLD(NO|START|ALL)

Use **HOLD** to hold the operations in the occurrence being added. Specifying **START** holds all operations with no predecessors. Specifying **ALL** holds all operations in the new occurrence. If there are operations within the occurrence that are on workstations that cannot be held, such as manual operations, message EQQI051W is issued with RC=4. This warning can be downgraded to information by running command **SETSEV 051I** prior to the **ADD** command in the same Workload Automation Programming Language step.

Note: If **FINDIA** appears to have skipped an Input Arrival minute, this is due to occurrences that have been manually deleted from the current plan. HCL Workload Automation for Z cannot reuse an input arrival time for an application, even if it was deleted.

The following list shows keywords for running in REPEAT mode:

Note: To select the REPEAT mode behavior, you must specify at least **UNTIL** or **EVERY**.

FROM(*hhmm*)

Sets the earliest Input Arrival time for a repeating cycle to start from. If **FROM** is specified earlier than **UNTIL**, **UNTIL** is considered to be after midnight. For values of **RESOLVE** other than **GAP**, the **FROM** time is used as the base to calculate multiples of **EVERY** to help chose the next Input Arrival time. If **FROM** is not specified, the current time is used for the first instance and passed to subsequent instances.

UNTIL(*hhmm*)

Sets the latest Input Arrival time for a repeating cycle to end. If **UNTIL** is not specified, one minute prior to the current time is used for the first instance and passed to subsequent instances.

EVERY(*hhmm*)

Sets the interval of hours and minutes to use to calculate the next Input Arrival time. If **EVERY** is not specified, 1 hour is used.

COUNT(*nnnn*)

Sets a limit to the number of instances of the occurrence is allowed to repeat.

ORIGIN

Specifies the time used to base the calculation for the next interval:

END

The current time (logical end point of the process). This is the default.

START

The start time of the occurrence (Occurrence Actual Arrival).

IA

The Input Arrival.

RESOLVE

Specifies the rules used to calculate the next interval:

NEXT

Uses the next multiple of the **EVERY** time after the **ORIGIN** time.

GAP

Adds the **EVERY** interval to the **ORIGIN** time.

BOTH

Adds the **EVERY** interval to the **ORIGIN** time and then take the next multiple of the **EVERY** time.

hhmm

Uses the next multiple of the **EVERY** time that is at least **hhmm** time from the **ORIGIN** time

RESTART(opnum[,FLUSH])

If the **FROM**, **UNTIL**, or **COUNT** keyword are not respected, this keyword restarts the current operation from the specified operation number at the time calculated according the **EVERY** keyword. To ensure that the JCL is resolved on restart, all the job stream records for this occurrence are deleted.

If the **FROM**, **UNTIL**, or **COUNT** keyword are respected, the **FLUSH** keyword sets all the remaining operations in the occurrence to **NOP**. This enables the batch program to repeat until a file is found, and process it.

In the following example, operation 005 searches for a file which, if not found, returns RC=4. The command restarts every 5 minutes, if the file is not found. If the file is found, the command continues to the remaining operations that process the file. However, if the repeat limit is reached without finding the requested file, the remaining operations are flushed. If you are expecting more than one file in a day, you might want to set another **ADD** command to restart the operations again and wait for the next file.

```
VARSUB SCAN
SELECT CPOPCOM ADID(!OADID) IA(!OYMD1!OHHMM)
            OPNO(005) OBJECT(CHECKOP)
IF '!@CHECKOP-CPOPORIGRC' = '0004' THEN DO
     ADD FROM(0000) UNTIL(2355) EVERY(005)
           RESTART(001,FLUSH)
```

NOTIFY(N|W|E)

Controls behavior when an **ADD** command has determined that no more occurrences are to be added:

N

Command ends with RC=0 for standard termination causes.

W

Command ends with RC=4 if next detected Input Arrival is outside of range, if the **COUNT** limit has been reached or an existing occurrence is detected between the **ORIGIN** time and the next Input Arrival.

E

Command ends with RC=4 if next detected Input Arrival is outside of range or if the **COUNT** limit has been reached. It ends with RC=8 if an existing occurrence is detected between the **ORIGIN** time and the next Input Arrival.

EARLY(ABORT|CONTINUE)

Controls behavior when the **ADD** command finds an **ORIGIN** time earlier than the Input Arrival time of the controlling occurrence. By default, the **ADD** command fails with RC=8 to protect against the eventuality that a Time Dependent flag has not been set in the Application Definition. If however the occurrence has been deliberately released early, restarting the **ADD** step with keyword **EARLY(CONTINUE)** will use the Application Input arrival time as the **ORIGIN** point if it is later than the chosen **ORIGIN**.

Note: **EARLY(CONTINUE)** must be used only as an option for restart. It must ever be coded in permanent SYSIN, because it prevents protection for missing Time Dependencies and might result in many instances of the occurrence running in close succession.

Deadline keywords are:

DLDAY(*n*)

Specifies the relative day to the current time to set the deadline for 0 being the same day, 1 being the following day, and so on. The **DLDAY** keyword counts Calendar days, it does not differentiate between Workdays and Freedays.

DLTIME(*hhmm*)

Specifies the time of the deadline.

The ONCE mode deadline policy is as follows:

- When **DLTIME** is set without **DLDAY**, Workload Automation Programming Language assumes the same “logical day”. If **DLTIME** is earlier than the current time, it is assumed to be post midnight.
- When **DLDAY** is set to 0 without **DLTIME**, HCL Workload Automation for Z sets the deadline by applying normal rules.
- When **DLDAY** is set > 0 without **DLTIME**, Workload Automation Programming Language assumes current time on relative day.

- If **DLDAY** is set to 0 and **DLTIME** is earlier than current time, the submission fails.
- In all other cases, HCL Workload Automation for Z sets the deadline by applying normal rules.

The REPEAT mode deadline policy is as follows:

- Deadline is always calculated relative to **FROM** and adjusted for current occurrence.
- When **DLTIME** is set without **DLDAY**, Workload Automation Programming Language assumes the same “logical day”. If **DLTIME** is earlier than the current time, it is assumed to be post midnight.
- When **DLDAY** is set to 0 without **DLTIME**, Workload Automation Programming Languageuses the **EVERY** value to calculate the deadline.
- When **DLDAY** is set > 0 without **DLTIME**, Workload Automation Programming Language assumes current time on relative day.
- When **DLDAY** and **DLTIME** are both set, the deadline is the specified time on the relative day.

  If **DLDAY** is explicitly set to 0 and **DLTIME** is earlier than the **FROM** time, the submission fails.
- In all other cases, the gap between the IA and deadline or the currently running occurrence will be used to calculate new deadline.

Usage notes for ONCE mode

ONCE mode is applied every time **UNTIL** or **EVERY** is not specified. It causes the **ADD** command to submit one occurrence of either a single application or every application in the Application Group.

The following keywords are applicable for ONCE mode:

```
ADD ADID(<adid>)|GROUPDEF(<groupdef>) [JCLVTAB(<table>)]
        [DEPRES(Y|N)]
        [IA(yymmddhhmm) | IADATE(yymmdd) IATIME(hhmm)]
        [FINDIA(Y|N)] [DLDAY(n)] [DLTIME(hhmm)]
```

The simplest example of the command is **ADD ADID(MYAPPL)**, which adds a single occurrence of **MYAPPL** using the current time, or nearest available minute as the Input Arrival and lets HCL Workload Automation for Z determine the deadline using normal rules.

More complex versions of the command can be used in conjunction with HCL Workload Automation for Z relative date and time processing. In the following example, you set the Input Arrival to be 15 minutes from the current time and the deadline to be 30 minutes from the current time, automatically adjusting if midnight is within those timescales. Use of relative times for the **IA** keyword allows occurrences to be submitted, with time dependencies to run at a relative point in the future.

```
ADD GROUPDEF(MYGROUP) IA(+15) DLTIME(+30)
```

For consistency with OCL syntax, **ADD GROUPDEF(MYGROUP) JCLVTAB(MYTABLE) DEPRES(Y)** is identical to **ADD GROUP(MYGROUP) VARTAB(MYTABLE) GDEPRES(Y)**.

Usage notes for REPEAT mode

REPEAT mode is applied every time you set either **UNTIL** or **EVERY**.

The following keywords are applicable for REPEAT mode:

```
ADD [JCLVTAB(<table>)] [DEPRES(Y|N)]
    [FROM(hhmm)] [UNTIL(hhmm)] [EVERY(hhmm)] [COUNT(nnnn)]
    [ORIGIN(START|END|IA)] [RESOLVE(NEXT|GAP|BOTH|hhmm)]
    [RESTART(opnum[,FLUSH])
    [EARLY(ABORT|CONTINUE)] [NOTIFY(N|W|E)]
    [DLDAY(n)] [DLTIME(hhmm)]
```

Use the REPEAT mode to dynamically schedule a repeating occurrence between set times, or for a certain number of occurrences. The maximum period of repeating is 24 hours, after which a new cycle must be initiated.

When you use the **ADD** command in REPEAT mode, without setting the RESTART keyword, it resubmits only the same occurrence or group. If it is included in an occurrence that is part of a group, the entire group is resubmitted, otherwise only the single controlling application is resubmitted. For this reason, the **ADID** or **GROUPDEF** keyword is not needed. However, if you specify any of them and it does not match the current occurrence or group, the **ADD** command fails.

The initial occurrence in the cycle can be added to the current plan by any of the following ways:

- Normal planning using a run cycle to add a single occurrence with an Input Arrival time matching the **FROM** time
- Adding manually by using 5.1
- Submitting by an **ADD** command running in ONCE mode
- Submitting by a PIF **INSERT** command
- An Event Triggered Tracking rule

The occurrence is then repeated by including a Workload Automation Programming Language step either within an existing job in the occurrence or group, or as a standalone job, scheduled at the point where you want the next occurrence to be added to the plan.

Note: To have the repeating mode run properly, ensure that:

- The WAPL job is the last job of the application, if the **RESTART** keyword is not set.
- The first operation of the application is set as time-dependent. If you have more than one first operation, create a time-dependent dummy operation as the internal predecessor of all your first operations.
- It is recommended that you set the **RESTART** keyword to reference the first operation.
- When you manually add the application to the plan, ensure that the application IA time is not earlier than the system current time. As the result, the application occurrence is kept is Waiting status until the IA time is reached. When the last operation, which is the WAPL job, has completed, the following application occurrence is added, and so on.

  To interrupt this process, delete or complete the occurrence that is in Waiting status; in this way, the WAPL job does not run and no other occurrences are added to the plan.

Modifying the **ORIGIN** and **RESOLVE** keywords will influence when the next occurrence is scheduled to start.

The default behaviour is **ORIGIN(END)** and **RESOLVE(NEXT)**. This means that the time when the **ADD** command runs is used to calculate the next run, which is the next multiple of **EVERY** following the time **ADD** runs.

For example, **ADD FROM(0900) UNTIL(2100) EVERY(0015)** repeats from 9am to 9pm every 15 minutes. If the occurrence starts at 1000 and ends at 1012, the next occurrence will be at 1015. If the occurrence starts at 1000 and runs until 1020, the next occurrence will be at 1030.

This mechanism effectively means that if a single occurrence runs longer than the **EVERY** interval, the intervals are skipped to catch up with the next time in the repeating cycle.

Using **RESOLVE(GAP)** ensures that the **EVERY** interval is used to make sure that each occurrence starts with the same interval between one finishing and the next starting. For example, **ADD FROM(0900) UNTIL(2100) EVERY(0015) RESOLVE(GAP)** repeats from 9am to 9pm every 15 minutes. If the occurrence starts at 1000 and ends at 1012, the next occurrence is at 1027. If the occurrence starts at 1000 and runs until 1020, the next occurrence is at 1035.

Using **RESOLVE(BOTH)** ensures that the **EVERY** interval is used to make sure that each occurrence starts with at least the specified **EVERY** interval between one finishing and the next starting, but also runs on perfect multiples of the **EVERY** time. For example, **ADD FROM(0900) UNTIL(2100) EVERY(0015) RESOLVE(BOTH)** repeats from 9am to 9pm every 15 minutes. If the occurrence starts at 1000 and ends at 1012, the next occurrence is at 1030. If the occurrence starts at 1000 and runs until 1020, the next occurrence is at 1045.

Using **RESOLVE(*hhmm*)** ensures that ***hhmm*** is used to make sure that there is a minimum time between one occurrence ending and the next one starting, but the occurrence starts on a perfect interval of the **EVERY** time. For example, **ADD FROM(0900) UNTIL(2100) EVERY(0015) RESOLVE(0005)** repeats from 9am to 9pm every 15 minutes. If the occurrence starts at 1000 and ends at 1012, the next occurrence is at 1030. If the occurrence starts at 1000 and runs until 1020, the next occurrence is at 1030.

Modifying the **ORIGIN** point can modify the characteristics of how the occurrences are calculated. Using **ORIGIN(IA)** with **RESOLVE(NEXT)** reproduces the behaviour that Repeat Every/Until achieves when using static run cycles. Every possible interval is scheduled. For example, **ADD FROM(0900) UNTIL(2100) EVERY(0015) ORIGIN(IA) RESOLVE(NEXT)** repeats from 9am to 9pm every 15 minutes. If the occurrence starts at 1000 and ends at 1012, the next occurrence is at 1015. If the occurrence starts at 1000 and runs until 1020, the next occurrence is at 1015.

COUNT can be used to limit the number of repeats, with or without **FROM | UNTIL** limits. For example, in the command **ADD EVERY(0015) COUNT(10)** the first instance could have been added by ETT, in which case the second instance is added a perfect multiple of 15 minutes from the initial triggering event. This will happen repeatedly until 10 individual runs have occurred.

For consistency between Batch Loader and the REPEAT mode of the **ADD** command, to add repeating run cycles **ADD FROM(0900) UNTIL(2100) EVERY(0015)** is identical to **ADD IATIME(0900) RPTEND(2100) RPTEVRY(0015)**.

Terminating repeating

If the next calculated Input Arrival time is later than the **UNTIL** time, or if the number of occurrences has reached the **COUNT** value, the **ADD** command does not submit any further occurrences.

The **ADD** command also stops if it detects another occurrence, submitted in other ways, with an Input Arrival time between the **ORIGIN** time and up to and including the calculated next Input Arrival. This prevents instances where multiple triggers may have added overlapping cycles to leap over each other, compromising the desired cycle interval.

Use of the **NOTIFY** keyword can be used to allow extra processing to be added into the repeating occurrence that only runs when the final occurrence of the cycle as run.

For example, **ADD FROM(0900) UNTIL(2100) EVERY(0015) NOTIFY(W)** means that the 2100 instance of the **ADD** job terminates with RC=4. Conditional dependencies can then be used to run jobs *only* when the **ADD** job terminates with RC=4, allowing final processes to be run at the end of the repeating cycle.

Conditional dependencies can also be used to terminate the repeating cycle early. If whatever process is being run by the repeating occurrence can detect it no longer needs to repeat, for example all the data for the day has arrived, then it too could issue a return code, that conditional dependencies could use to stop a further run of the **ADD** job.

Persistent data

If you do not specify either **FROM** or **UNTIL** time in REPEAT mode, these values default to the current time, and one minute before the current time of the first instance respectively. To allow this time to be used by subsequent instances, it must be passed from one occurrence to the next.

If you specify **COUNT**, this value needs to keep a running total of occurrences to pass from one occurrence to the next.

Until HCL Workload Automation for Z V8.5.1 included, there is no recognized mechanism to pass information from one occurrence to the other, encapsulated within the occurrence.

To do this, Workload Automation Programming Language uses the Owner Description field of the occurrence in the current plan. The Owner description is left unchanged in the database, so if needed the Application Definition can be referred to, only the current plan version is changed.

If you wish to keep the Owner Description unchanged in the current plan, you must always set values for **FROM** and **UNTIL**, and you must not set the **COUNT** keyword.

Considerations about interval processing performance

Repeating the same batch program many times is not the best use of system resources: the more frequent, the less efficient. When you schedule the intervals for repeating a command, consider the following information.

Schedule the commands to wait for the expected result rather than repeating the process over and over. This reduces the number of command runs and make more efficient use of system resources. For example, using the file monitor operations makes the current interval wait either until a file arrives, otherwise the operation times out. This heavily reduces the number of runs scheduled in a day.

Additionally, schedule the processing intervals to terminate the resubmission or restart after the expected data has been returned; this is especially useful when you set the **FROM** or **UNTIL** keyword.

By setting the **RESTART** keyword you use the existing occurrence, which in turn reduces the size of the current plan. This reduces the CPU consumption by any PIF, WAPL or other processes that read the current plan.

ADDJOB – Add job to the current plan

Use the **ADDJOB** command to add a job to the current plan by using attributes from the application where the job is defined.

```
ADDJOB <jobname> [ADID(<adname>)] [COMPSUCC(IGNORE|WARNING|ERROR]
                 [CPDEPR(YES|PREDECESSORS|SUCCESSORS|NO)]
                 [CRITERIA(CLOSEST|MANUAL|SAMEDAY)] [DEPTIME(yymmddhhmm)]
  [EXTLINK(APPL|JOB|WSJOB)] [GROUPDEF(<application-group>)]
                  [GROUP(<authority-group>)] [HOLD(YES|NO)
                 [IA(yymmddhhmm)]
  [IGNORE(=ALL,=NONE,=FIRST,=LAST,=<wstype>,<wsname>,<opno>)]
  [INTLINK(APPL|JOB|WSJOB)] [JCL('<library>(<member>)')
                 [MULTI(FAIL|FIRST|LAST)]
                 [NOTFOUND(FAIL|SUBMIT)] [OWNER(<owner>)]
                 [PFX(<prefix>)]
                 [SFX(<prefix>)] [VALFROM(yymmdd)] [VALID(yymmdd)]
                 [VALTO(yymmdd)] [WSNAME(<wsid>)]
```

where:

The first positional parameter

The job name to find.

ADID

A filter field to restrict the search to the application named in this keyword. Wildcards are allowed.

COMPSUCC

Determines the action to take when a successor is identified as already completed. A complete successor cannot be added as an external dependency. The equivalent **OPTIONS** keyword sets the default.

IGNORE

Do not add the dependency and issue an advisory message (RC=0).

WARNING

Do not add the dependency and issue an advisory message (RC=4). This is the default.

ERROR

Do not add the dependency and issue an advisory message (RC=8).

CPDEPR

Specifies whether to resolve dependencies:

YES

Resolve both predecessors and successors (default).

PREDECESSORS

Resolve only predecessors.

SUCCESSORS

Resolve only successors.

NO

Do not resolve any dependencies.

CRITERIA

How to resolve dependencies. The **ADDJOB** command ignores individual dependency resolution criteria in the database. Only the **CRITERIA** keyword determines the method to be used:

CLOSEST

Use the “Closest preceding” method.

SAMEDAY

Use the “Same day” method.

MANUAL

No dependencies as resolved, additional dependencies can be manually added.

DEPTIME

The time to be used to resolve dependencies from. If specified the job being added will resolve its dependencies as if the **DEPTIME** is it's operation input arrival. If **DEPTIME** is not specified, the value of the **IA** keyword is used. If neither **DEPTIME** or **IA** is specified, the current date and time are used.

EXTLINK

How to find the target external dependencies:

APPL

Use the application ID and operation number in the dependency definition (default).

JOB

Use the job name in the dependency definition.

JOBWS

Use the job name and workstation name in the dependency definition.

GROUPDEF

A filter field to restrict the search to the members of the application group named in this keyword. Wildcards are allowed.

GROUP

A filter field to restrict the search to the authority group named in this keyword. Wildcards are allowed.

HOLD(YES|NO)

Determines whether the added job is held or not when added to the plan (default is No).

IA

The Input Arrival to use when creating the occurrence. If the **IA** is already in use for an occurrence, the command fails.

IGNORE

Specifies a list of elements to define the internal dependencies to be ignored when adding the job to the current plan. The default is **=FIRST=LAST** to ignore internal dependencies to the first and last operations. The default is set by **OPTIONS IGNORE**.

The following elements can be combined:

=ALL

Ignore all internal dependencies.

=NONE

Do not ignore any internal dependencies.

=FIRST

Ignore the first operation (as defined by OPTIONS FIRST).

=LAST

Ignore the last operation (as defined by OPTIONS LAST).

<wstype>

Ignore operations that use a particular workstation type. Valid types are **AUTO**, **CPU**, **DUMMY**, **PRINT**, **REMOTE-D**, **REMOTE-Z**, **SETUP**, **STC**, **VIRTUAL**, **WAIT**, **WTO** or **ZCENTRIC**.

<wsname>

Ignore operations that use a particular workstation.

<opno>

Ignore operations on a particular operation number.

INTLINK

How to find the target internal dependencies:

JOBWSx(

Use the job name and workstation name in the dependency definition (default).

APPL

Use the application ID and operation number in the dependency definition.

JOB

Use the job name in the dependency definition.

JCL('<library>(<member>)')

Specifies the name of a library and member containing the JCL to be run for this job, instead of the information that would usually retrieved from the HCL Workload Automation for Z libraries. This keyword must point to a member of the specified library, it cannot be provided from a sequential file. The library and member name must be specified within single quotation marks.

MULTI

Specifies what to do if the job is found more than once. **FAIL** ends in error if more than one match is found (default). **FIRST** reports only the first place that is found. **LAST** reports only the last place that is found.

NOTFOUND

Specifies what to do if the job is not found in the database:

SUBMIT

Generates an occurrence to submit the job (default).

FAIL

Causes the job to fail with RC=8.

OWNER

A filter field to restrict the search to applications with the owner ID specified. Wildcards are allowed.

PFX(<prefix>)

The prefix to use in the generated occurrence name. The occurrence name is formed from the value of **PFX**, the job name, and **SFX** combined. The default for this value is set by **OPTIONS ADPFX**.

SFX(<suffix>)

The suffix to use in the generated occurrence name. The occurrence name is formed from the value of **PFX**, the job name, and **SFX** combined. The default for this value is set by **OPTIONS ADSFX**.

UPDATE(YES|NO)

Specifies whether the update to the current plan is made. When **UPDATE(NO)** is specified, the actions of forming the definition of the occurrence take place, but no updates are made to the current plan. You can create a list with the actions that would be performed by specifying **OPTIONS TRACE(1)**. The default is set by **OPTIONS UPDATE**.

VALFROM

A filter field to restrict the search to applications matching the valid from date specified. Wildcards are allowed.

VALID

A filter field to restrict the search to applications that are valid on the date specified. ildcards are allowed.

VALTO

A filter field to restrict the search to applications matching the valid to date specified. Wildcards are allowed.

WSNAME

A filter field to restrict the search to applications containing the workstation specified. Wildcards are allowed.

The **ADDJOB** command creates a dynamic occurrence in the current plan, using the attributes of the job found in the database. They include all operation attributes, extended information, automation attributes, special resources, and user fields. The command ignores predecessor selection criteria and conditional dependencies, because this is a dynamically added occurrence, therefore these elements can be dealt with by the process performing the add.

The dependencies defined to the job are processed by the **ADDJOB** command in accordance with the **CPDEPR**, **EXTLINK**, and **INTLINK** keywords. By default, to find external dependencies the application name and operation number are used, while to find the internal dependencies, the job name and workstation name are used.

The **CRITERIA** keyword specifies how to resolve the dependencies. The predecessor resolution is done by using the input arrival that is determined as follows:

- **DEPTIME** specifies the input arrival to use for dependency resolution.
- If **DEPTIME** is not set, the time specified in **IA** is used.
- If both **DEPTIME** and **IA** are not specified and the Workload Automation Programming Language job is controlled by HCL Workload Automation for Z, the input arrival of the controlling occurrence is used.
- In all other cases, the current date and time are used.

Predecessor resolution is performed using the criteria as if evaluated from the perspective of the job being added, using the input arrival time as derived above.

Successor resolution is performed using the criteria as if evaluated from the perspective of the successor to the job being added, using the input arrival time of the successor to the input arrival of the job being added as derived above.

If a matching job is not found, and **NOTFOUND** is set to **SUBMIT**, then a simple dynamic occurrence using default values for all job attributes will be used. If the **WSNAME** keyword was specified without using wildcards then it will be used as the workstation in the generated occurrence. Otherwise the value of **OPTIONS ADWS** will be used, which is CPU1 by default.

If the job name argument of the **ADDJOB** keyword contained wildcards, the submission fails if not matching job is found, regardless of the setting of **NOTFOUND**.

Note: If the combination of **PFX**, job name, and **SFX** exceeds 16 characters it will be truncated at 16 characters.

CONSOLE – Issue z/OS console commands

Use the **CONSOLE** command to issue commands to the z/OS console.

```
CONSOLE <command>
```

Where **<command>** can be any valid REXX expression that resolves to a valid console command. For more details about REXX expressions and available functions, see *TSO/E REXX Reference* manual.

Subscripted variables can be used to issue a series of commands in a loop. For example, to issue a set of commands 10 seconds apart:

```
VARSUB SCAN (!)
VARSET CMD1 = "F WSIC,STATUS"
```

```
VARSET CMD2 = "F WSJC,STATUS"
VARSET CMD3 = "F WSKC,STATUS"
VARSET CMD4 = "F WSLC,STATUS"
DO X = 1 TO 4
   CONSOLE @V(CMD!X)
   WAIT 10
END
```

The commands could be contained within a file and read into the process:

```
//RUNWAPL  EXEC EQQYXJPX,
//         SUBSYS=WSLC
//COMMANDS DD *
F WSIC,STATUS
F WSJC,STATUS
F WSKC,STATUS
F WSLC,STATUS
//SYSIN    DD *
VARSUB SCAN (!)
READ COMMANDS OBJECT(CMDS)
DO X = 1 TO !@CMDS
   CONSOLE @V(@CMDS-!X.)
END
```

To run the commands, ensure the you have the appropriate MCS console authority.

By default the console name will be the same as the job name. This can be altered by **OPTIONS CONNAME**.

After the command is issued, the process waits 2 seconds for the first messages to be issued, then waits up to 1 second for each subsequent message to be displayed. After one second passed with no new messages, the **CONSOLE** command completes. This can be altered by **OPTIONS CONWAIT**.

IEE*xxx* messages returned from issuing console commands are typically informational (suffix I) and indicate a failure. To ensure the errors are not missed, all IEE*xxx*I messages are considered as errors (suffix E). By setting **OPTIONS CONINFO**, the IEE*xxx*I messages are considered as informational messages. By setting **OPTIONS CONWARN**, the IEE*xxx*I messages are considered as warning messages. To verify the current settings of these **OPTIONS** commands, issue **SHOW OPTIONS**.

Any other messages retrieved will set return codes according to the message severity:

- I and A return 0
- W returns 4
- Anything else returns 8

GETDATES – Generate a list of run dates from a run cycle rule

Use the **GETDATES** command to generate the run dates for an entire set of run cycles within an application, or a particular input arrival subset. To produce the list of run dates, the command uses the **LIST GENDAYS** and **MERGE** commands internally.

```
GETDATES    ADID(<application>) [TYPE(A|G)]
            [STATUS(A|P)]
            [VALID(<valid-on>)] [IAT(hhmm]
            [FROMDATE(<from>)] [TODATE(<to>)] [SAVELIST(<pfx>)]
            [OUTPUT(ALL|COMBINED|INDIVIDUAL)]
```

where:

<application>

The name of the application for which to generate the run dates. By default, the command assumes the ACTIVE version of this, as an APPLICATION definition only, that is valid on the day the command runs. **TYPE** can be used to look for GROUP definitions and **STATUS** can be used to calculate dates for PENDING versions.

<valid-on>

The date used to find the version of the application. By default, it uses the current date.

hhmm

The input arrival time to select matching run cycles to calculate the run dates from. For applications that might have multiple runs within one day, this allows a specific single run to be calculated. If omitted, the run dates for all run cycles are calculated.

<from>

The date from which the run dates are calculated, in the format **YYMMDD**. The default is the current date.

<to>

The date until which the run dates are calculated, in the format **YYMMDD**. The default is the **FROMDATE** value plus 90 days.

<pfx>

The name of the **SAVELIST** to be produced, containing the combined run dates for all run cycles. This is also used as a prefix of internally generated **SAVELIST** for run cycle. It is required that **SAVELIST** entries are generated to enable internal use of the **MERGE** command to calculate the composite effect of all run cycles.

Output can be in the form of a **SAVELIST** and optionally output files if an **OUTPUT GNDAY** statement has been coded ahead of the command. The **SAVELIST** output can be used to generate interval dates within Periods using the **PRSTART DATELIST** keyword.

The amount of output can be influenced by the **OUTPUT** keyword, which can have the following values:

ALL

Generates **SAVELIST** and optionally file output for each individual run cycles and the combined result for all of the run cycles.

COMBINED

Only generate **SAVELIST** and optional file output for the combined result for all of the run cycles.

INDIVIDUAL

Only generate **SAVLIST** and optional file output for each individual run cycle. Do not generate any combined output.

Note:

1. For run cycles using the EVERY or UNTIL, only the initial Input Arrival time is used to select the run cycle with the **IAT** keyword.
2. Run cycle level **SAVELIST** output is named using the **SAVELIST** prefix followed by *_cccc*, where *cccc* is the run cycle sequence number.
3. For run cycles of type N and X, a rule definition is automatically generated internally, therefore the **LIST GENDAYS** command can calculate offsets of a period, for example **ONLY(1) DAY(DAY) PERIOD(MYDATES)**. Period based run cycles can generate more offsets than a rule definition can support (11 positive and 11 negative). In these instances, multiple rules are generated until all of the period offsets have been accounted for. The run cycle **SAVELIST** name will be suffixed with *_cccc_iiii*, where *cccc* is the run cycle sequence number and *iiii* is the instance number of the multiple rules generated for the single run cycle.
4. For the limitations related to **FROMDATE** and **TODATE**, see LIST GENDAYS – Generate dates from a rule on page 153.
5. The **LIST GENDAYS** PIF call uses the same internal HCL Workload Automation for Z code that is used to generate the dates for **GENDAYS** in ISPF.
6. The **GETDATES** command is available only starting from HCL Workload Automation for Z V8.6 SPE, or later.

LISTJOB – List job attributes from the database

Use **LISTJOB** to display the job attributes from the database. These can include the applications where the job is coded, the operation level details, including Special Resources and User Fields, and dependencies, including internal and external predecessors and successors.

```
LISTJOB <jobname> [ADID(<adname>)] [DETAIL(APPL|OPER|SUCC)]
                  [DISPLAY(YES|NO)]
                  [FLOW(n)] [GROUPDEF(<application-group>)]
                  [GROUP(<authority-group>)] [MULTI(ALL|FAIL|FIRST|LAST)]
                  [IGNORE(=ALL,=NONE,=FIRST,=LAST,=<wstype>,<wsname>,<opno>)]
                  [OBJECT(<object>)] [OPNO(nnn)] [OUTPUT(*|<ddname>)]
                  [OWNER(<owner>)][PRIORITY(n)] [SAVELIST(<savelist>)]
                  [STATUS(A|P|*)] [STYLE(TEXT|LOADER)]
                  [VALFROM(yymmdd)] [VALID(yymmdd)] [VALTO(yymmdd)]
                  [WSNAME(<wsid>)]
```

where:

The first positional parameter

The job name to find.

ADID

A filter field to restrict the search to the application specified. Wildcards are allowed.

DETAIL

How much detail to return. **APPL** lists only the applications in which the job is found. **OPER** also lists the operation attributes, extended information, automation information, special resources, dependencies, conditional dependencies, and user fields. **SUCC** lists external successors.

DISPLAY

Whether to generate the output to SYSTSPRT.

FLOW

The column width at which one line of output flows to another. If not specified, the data will flow appropriate to the length of the output stream.

GROUPDEF

A filter field to restrict the search to the members of the application group specified. Wildcards are allowed.

GROUP

A filter field to restrict the search to the authority group specified. Wildcards are allowed.

IGNORE

A list of elements that define the internal dependencies to be ignored when listing the dependencies of the job. The default is **=FIRST =LAST** to ignore the internal dependencies to the first and last operations. The default is set by **OPTIONS IGNORE**.

You can combine the following arguments:

=ALL

Ignore all internal dependencies.

=NONE

Do not ignore any internal dependencies.

=FIRST

Ignore the first operation (as defined by **OPTIONS FIRST**).

=LAST

Ignore the last operation (as defined by **OPTIONS LAST**).

<wstype>

Ignore the operations that use a specific workstation type. Valid types are: **AUTO**, **CPU**, **DUMMY**, **PRINT**, **REMOTE-D**, **REMOTE-Z**, **SETUP**, **STC**, **VIRTUAL**, **WAIT**, **WTO**, or **ZCENTRIC**.

<wsname>

Ignore the operations that use a specific workstation.

<opno>

Ignore the operations on a specific operation number.

MULTI

Specifies what to do if the job is found more than once:

ALL

Reports all places found.

FAIL

Ends with RC=8, if more than one match is found.

FIRST

Reports only the first place found.

LAST

Reports only the last place found.

SAVELIST and **OBJECT** always return all matches.

OBJECT

Sets an object variable in which to store all the attributes.

OPNO

A filter field to restrict the search to the operation number specified in this keyword.

OUTPUT

Sets an output destination for display style output, either * to output to the stack, or a DD statement.

OWNER

A filter field to restrict the search to applications with the specified owner ID. Wildcards are allowed.

SAVELIST

Saves the record selection criteria for each application containing the job to a specified list.

STATUS

A filter field to restrict the search to applications with a specific status:

A

Active applications.

P

Pending applications.

*

All applications (default).

STYLE

The output style:

TEXT

Uses descriptive text to identify each record.

LOADER

Uses batch loader keywords to identify each record.

VALFROM

A filter field to restrict the search to applications matching the valid from date specified. Wildcards are allowed.

VALID

A filter field to restrict the search to applications that are valid on the date specified. Wildcards are allowed.

VALTO

A filter field to restrict the search to applications matching the valid to date specified. Wildcards are allowed.

WSNAME

A filter field to restrict the search to applications containing the specified workstation. Wildcards are allowed.

For example, **LISTJOB JOB040** lists all the applications in which JOB040 is contained.

The following example is a simple list of applications containing JOB040:

```
Application: ADID(CMDDEMO3) ADVALFROM(141024) DESCR('Demonstrate CMD1 WS')
             OWNER(TWS)
Application: ADID(DEEPFROG1) ADVALFROM(141030) DESCR('Demonstrate CMD1 WS')
             OWNER(TWS)
Application: ADID(DROPTEST) ADVALFROM(140922) DESCR('Test DROP command')
             OWNER(TWS)
Application: ADID(DROPTEST2) ADVALFROM(140924) DESCR('Test DROP command')
             OWNER(TWS)
```

The command **LISTJOB JOB005 ADID(DEEPFROG6) DETAIL(SUCC)** lists all details of JOB005 as defined in application DEEPFROG6:

```
Application: ADID(DEEPFROG6) ADVALFROM(141102) DESCR('Demo RUNIF') OWNER(TWS)
  Operation: WSID(CPU1) OPNO(005) JOBN(JOB005) DURATION(1) HIGHRC(00000006)
             STARTTIME(2100) R1NUM(00000002) R2NUM(00000004) FORM(ABD00123)
             TIME(Y) CRITICAL(P)
    Predecessor-INT: WSLCCMD1 PREADID(DEEPFROG6) PREWSID(CMD1) PREOPNO(001)
    Predecessor-EXT: ZLAST  PREADID(DEEPFROG4) PREWSID(NONR) PREOPNO(255)
                     PRECSEL(R) PREMAND(C)
    Resource: RESOURCE(JOHN) USAGE(X)
    User field: UFNAME(EQQ-SYSIN-01) UFVALUE('VARSUB SCAN(!)')
    User field: UFNAME(EQQ-SYSIN-02) UFVALUE('SHO OPTIONS')
    Ext criteria: ADID(DEEPFROG4) WSID(NONR) OPNO(255) TYPE(R) TOWHEN(B)
                  TOHHH(000) TOMM(01)
    Successor-INT: JOB010 SUCADID(DEEPFROG6) SUCWSID(CPU1) SUCOPNO(010)
    Successor-INT: JOB020 SUCADID(DEEPFROG6) SUCWSID(CPU1) SUCOPNO(020)
    Successor-EXT: WSLCCMD1 SUCADID(DEEPFROG5) SUCWSID(CMD1) SUCOPNO(001)
```

The output from **LISTJOB** is similar to Batch Loader, though not directly executable as Batch Loader. The fields shown are the fields that are not set to their default values, and the values themselves are stripped of trailing spaces.

For the **DISPLAY** output, each segment type is shown as descriptive labels.

For the **OBJECT** output, the Batch Loader command name prefixes each record, with the exception of dependencies. The **OBJECT** structure includes a record counter contained in the high level object variable and a record for each segment of output. The following command:

```
VARSUB SCAN(!)
LISTJOB JOB005 DETAIL(SUCC) OBJECT(FREDDO) DISPLAY(N) MULTI(FAIL)
        ADID(DEEPFROG6)
DO X = 1 TO !@FREDDO
   DISPLAY @V(@FREDDO-!X.)
END
```

returns the following `OBJECT` output:

```
ADSTART ADID(DEEPFROG6) ADVALFROM(141102) DESCR('Demo RUNIF') OWNER(TWS)
ADOP WSID(CPU1) OPNO(005) JOBN(JOB005) DURATION(1) HIGHRC(00000006) STARTTIME(2100)
R1NUM(0000002) R2NUM(0000004) FORM(ABD0123) TIME(Y) CRITICAL(P) TIME(Y) CRITICAL(P)
ADPRE-INT WSLCCMD1 PREADID(DEEPFROG6) PREWSID(CMD1) PREOPNO(001)
ADPRE-EXT ZLAST  PREADID(DEEPFR4) PREWSID(NONR) PREOPNO(255) PRECSEL(R) PREMAND(C)
ADSR RESOURCE(JOHN) USAGE(X)
ADUSF UFNAME(EQQ-SYSIN-01) UFVALUE('VARSUB SCAN(!)')
ADUSF UFNAME(EQQ-SYSIN-02) UFVALUE('SHO OPTIONS')
ADXIV ADID(DEEPFROG4) WSID(NONR) OPNO(255) TYPE(R) TOWHEN(B) TOHHH(000) TOMM(01)
ADSUC-INT JOB010 SUCADID(DEEPFROG6) SUCWSID(CPU1) SUCOPNO(010)
ADSUC-INT JOB020 SUCADID(DEEPFROG6) SUCWSID(CPU1) SUCOPNO(020)
ADSUC-EXT WSLCCMD1 SUCADID(DEEPFROG5) SUCWSID(CMD1) SUCOPNO(001)
```

The exception to Batch Loader format is how dependencies are presented. The database works only with predecessors, but `LISTJOB` shows also successor relationships for each job. Instead of `ADDEP`, the dependencies are represented by `ADPRE` and `ADSUC` with `-INT` or `-EXT` appended to show whether the dependency is internal or external. The application name is shown, even for internal dependencies, so that the `ADDJOB` function can use this to resolve relationships. Also in an exception to the Batch Loader convention the job name is listed as the first word on any `ADPRE` or `ADSUC` record.

LISTSTAT – List Status of Current Plan Objects

Use `LISTSTAT` to `LIST` items and set a return code based on their status.

```
LISTSTAT <resource> <arguments> [POLICY(<RC-expression>)]
```

`LISTSTAT` performs a `LIST` request to find the relevant elements within HCL Workload Automation for Z. If the `LIST` request returns multiple records, the command ends with RC=8.

The arguments for `LISTSTAT` are the same as the arguments for the related `LIST` request for the same resource with the exception of `POLICY` which is used to set the return code policy for the request.

If you want to set your own Return Code policy you can add the `POLICY` keyword to the `LISTSTAT` command. The `POLICY` keyword allows you to set specific return codes for groups of status values, and set a return code for anything that does not match any of your listed statuses:

```
POLICY(<status_list1>=<rc1>,<status_list2>=<rc2>,...,
        <status_listn>=<rcn>,<catch_all>)
```

If you omit to supply a catch all return code then you will get RC=99 when no match occurs. The return code must be a non-negative whole number.

There is a special status of `?` that occurs if the `LISTSTAT` statement does not find an operation. This allows you to decide what return code to set for an operation not found.

For example, the following command:

```
LISTSTAT CPOPCOM ADID(TWSCDAILYPLAN) OPNO(010) IA(0805281200)
         POLICY(AR*=10,SW=20,C=30,?=90,40)
```

would return 10 if the status was A, R or *, it would return 20 if the status was S or W, it would return 30 if the status was C and return 40 if any other status was encountered. If the operation was not found in the current plan then 90 would be returned.

Another use of **POLICY** is to separate acceptable from unacceptable conditions, as in the following example:

```
LISTSTAT CPOPCOM ADID(TWSCDAILYPLAN) OPNO(010) IA(0805281200)
         POLICY(C=0,?=0,20)
```

This command ends with return code zero if the operation is complete or it is not in the current plan, but returns 20 if it is in any other state.

Note:

- Workload Automation Programming Language will return 4 if the item being listed is not found, 8 if more than one item is found, 12 if you have made a syntax error and 16 if an initialization error occurs. It is recommended, that you avoid using these return codes in your **POLICY** if you need to distinguish those errors from your return codes.
- If you run multiple **LISTSTAT** commands in one session, the highest return code is returned.
- If any command other than **LISTSTAT** breaches the **OPTIONS STOPRC** setting, the **LISTSTAT** return code is not returned.
- The **LISTSTAT** command itself will only ever complete with return codes 0, 4, 8 or 12. The actual return code for the status is returned as Workload Automation Programming Language terminates.
- The **SETMAX** command has no impact on the return code set by **LISTSTAT**.
- If the **LISTSTAT** command does not find an operation, a return code of 4 is set. However if you have a **POLICY** keyword and you have not added a specific entry for the special status of ?, then the “catch all” return code will be returned which may override the 4 if it is higher. This allows you to treat “not found” as a undesired status along with any others in the scope of the “catch all”.

The following list shows the supported resources for **LISTSTAT**:

CPOC

Occurrences in the Current Plan

CPOPCOM

Operations in the Current Plan

CPWSCOM

Workstations in the Current Plan

CSRCOM

Special Resources in the Current Plan

The following list shows the standard return codes are:

8

More than one record identified

12

Workload Automation Programming Language error

99

Unexpected status returned or no match in `POLICY`

The following list shows the return codes that are set when `POLICY` is not specified:

4

Resource not found or user ID has no RACF authorization to read the resource

31

Occurrence status C (completed)

32

Occurrence status D (deleted)

33

Occurrence status E (ended in error)

34

Occurrence status P (processor pending)

35

Occurrence status S (started)

36

Occurrence status U (undecided)

37

Occurrence status W (no started operations)

40

Operation status *

41

Operation status A (waiting for input to arrive)

42

Operation status R (ready)

43

Operation status S (started)

44

Operation status C (completed)

45

Operation status D (deleted)

46

Operation status I (interrupted)

47

Operation status E (ended in error)

48

Operation status W (waiting for a predecessor)

49

Operation status U (undecided)

50

Operations status X (excluded)

60

Workstation status A (active)

61

Workstation status O (offline)

62

Workstation status F (failed)

70

Special resource available

71

Special resource unavailable

OBJECT

Use the **`OBJECT`** keyword to create an object variable for the **`LISSTAT`** process and the record identified.

```
OBJECT(<name>)
```

where **`<name>`** is the name of the object variable to create.

The primary object for the **LISSTAT** command is a simple object variable that contains the number of records found by the **LISTSTAT** command. This is either 1 or 0, because **LISSTAT** is designed to identify only one record.

Int the following example, the object for the identified record is the object name suffixed by 1:

```
LISTSTAT CPOPCOM ADID(TWSCDAILYPLAN) OPNO(010) IA(0805281200)
         POLICY(C=0,?=0,20) OBJECT(CHK)
```

Therefore **!@CHK** will contain 1 if the operation is found, and 0 if not. **!@CHK1-CPOPJES** will contain the JES number of the job being checked.

Performing SRSTAT actions with LISTSTAT

As well as setting return codes, the **POLICY** statement can also perform **SRSTAT** actions based on the object status.

The complete syntax of the action to perform is as follows:

```
[rc/][[+|-|!]special-resource[/subsys]]
```

The plus sign (+), minus sign (–), or exclamation mark (!) used as prefix for special resources indicates whether to set the availability to YES (+), NO (-) or RESET (!). If neither the plus sign (+), minus sign (–), or exclamation mark (!) is specified, YES is assumed.

To direct the **SRSTAT** event to a tracker or controller different from the value specified in the SUBSYS= symbolic in combination with any setting of **OPTIONS TRACKERS**, you can append the subsystem to the end of the special resource name using the slash (/).

If a return code is not specified in front of a special resource name, RC=0 is assumed.

Note:

- Special Resource names containing the slash (/) or the parenthesis cannot be used with this command.
- If the special resource name begins with the plus sign (+) or minus sign (–), you must specify an additional plus sign or minus sign to explicitly set the availability.

In the following example, the job looks for operation 10 of an application called DAILYPLANNING that runs at 12:00 on the same input arrival date as the application running this Workload Automation Programming Language job.

- It ends with RC=0 if the operation is complete.
- It ends with RC=2 if the operation is not found in the current plan.
- It ends with RC=8 and generate an SRSTAT event to set resource MY.SR.TRIGGER to available, which will be sent to SUBSYS(WSIT).

```
//RUNWAPL EXEC EQQYXJPX,
//         SUBSYS=WSIC
//SYSIN    DD *
VARSUB SCAN
LISTSTAT CPOP ADID(DAILYPLANNING) OPNO(10) IA(!OYMD1.1200)
         POLICY(C=0,?=2,W=8/MY.SR.TRIGGER/WSIT)
```

In the following example, the job looks for MYJOB10 and MYJOB20 of an application called MYJOBS that arrives at 23:00 the day before input arrival date as the application running this Workload Automation Programming Language job.

- The command ends always with RC=0.
- If MYJOB10 is in error it sets resource JOB.FAIL.MYJOB10 to available.
- If MYJOB10 is waiting it sets resource JOB.WAIT.MYJOB10 to available.
- If MYJOB20 is in error it sets resource JOB.FAIL.MYJOB20 to available.
- If MYJOB20 is waiting it sets resource JOB.WAIT.MYJOB20 to available.
- `OPTIONS TRACKERS` is used to determine where to send the event. Ordinarily this would be specified in your site options member.

```
//RUNWAPL EXEC EQQYXJPX,
//         SUBSYS=WSIC
//SYSIN    DD *
OPTIONS TRACKERS(WSIC.*.WSIT)
VARSUB SCAN
VARDATE YESTERDAY BASE(!OYMD1.) OFFSET(-1)
LISTSTAT CPOP ADID(MYJOBS) JOBNAME(MYJOB10) IA(!YESTERDAY.2300)
         POLICY(E=JOB.FAIL.MYJOB10,W=JOB.WAIT.MYJOB10,0)
LISTSTAT CPOP ADID(MYJOBS) JOBNAME(MYJOB20) IA(!YESTERDAY.2300)
         POLICY(E=JOB.FAIL.MYJOB20,W=JOB.WAIT.MYJOB20,0)
```

These special resources could be used to trigger different escalation or notification applications, therefore a failed job could use a Special Resource ETT rule with `JOB.FAIL.*` to trigger an application to indicate in Problem Management records that the job was still in a failed state during the night's processing. In the same way, using a `JOB.WAIT.*` trigger you could run a job to automatically send an email to support staff notifying them of the delay.

If you set `//*%OPC SETVAR TJOB=SUBSTR(&OETEVNM,10,8)` in the triggered application, you can obtain the job name, that was the reason for triggering the event, for use in that notification or escalation process.

SENDMAIL – Send an email via z/OS SMTP

Use `SENDMAIL` to send a simple email to a single named recipient. The command relies upon SMTP having already been set up within the z/OS system on which the job runs. Workload Automation Programming Language requires that an email has both a subject and text.

```
SENDMAIL    [FROM(<address>)] [SERVER(<server>)] SUBJECT(<text>)
              [TEXTDD(<ddname>)] TO(<addresses>) [CC(<addresses>)]
              [BCC(<addresses>)]
            [TXT(line of text)]
            [TXT(line of text)]
```

Note: To enable Workload Automation Programming Language to send emails, the EQQSMTP DD statement must be added either to the Workload Automation Programming Language procedure being used, or the executing JCL of the step. The DD statement needs to contain a SYSOUT class pointing to the correct class and writer for SMTP email via z/OS. The name of the SMTP DD statement can be set by `OPTIONS MAILSMTP`.

The following keywords are available:

`FROM`

Allows the email sender's address to be specified. If no domain is specified the `SERVER` domain is appended to the address. If `FROM` is not specified the `OPTIONS` value for `MAILFROM` will be used. If no `FROM` address is provided either by the `FROM` keyword or `OPTIONS MAILFROM` the command will fail.

`SERVER`

Provides the domain name to be used as part of the HELLO handshake with the mail server. If the `SERVER` keyword is not specified the `OPTIONS` value for `MAILSERV` will be used. If no `SERVER` is provided either by the `SERVER` keyword or `OPTIONS MAILSERV` the command will fail.

`SUBJECT`

Specifies the subject line of the email.

`TEXTDD`

Refers to a DD statement in the JCL from which the text of the email will be read. The DD statement must refer to either a sequential file or an individual member within a partitioned dataset or library. The value of the `TEXTDD` keyword will default to `EQQEMAIL` if not specified. The default value can be altered by `OPTIONS MAILDD`.

`TO`

Specifies the primary recipients of the email. If no domain is specified the `SERVER` domain is appended to the address.

`CC`

Specifies the copy list of recipients.

`BCC`

Specifies the blind copy list of recipients.

`TXT`

Specifies an individual line of text for the email. Multiple `TXT` keywords can be coded.

The following example shows a `SENDMAIL` command:

```
OPTIONS MAILSERVER(frog1.frogpond.abc.com)
VARSUB SCAN (!)
SENDMAIL FROM(dean) TO(warren@nj.com) SUBJECT(Test SENDMAIL)
         CC(doug@pa.com,ann@pa.com) BCC(dean) :IFCMD(LISTSTAT.EQ.4)
TXT(Here is the text in the main command stream)
TXT(you can have as many TXT statements as you like)
TXT(and they can have variables like JOBNAME=!OJOBNAME.)
```

SENDMSG – Send a TSO message

Use **SENDMSG** to send a message from the job to a TSO user.

```
SENDMSG TEXT(<text>) [USER(<userid>)]
```

where:

<text>

The text of the message to send.

<userid>

The TSO user ID to whom to send the message.

If the **USER** keyword is omitted, the message is sent to the JES job log and can be displayed on SYSLOG and the console.

WSALTER – Alter intervals on a workstations in the current plan

Use **WSALTER** to set parallel server values across multiple workstations in the current plan.

```
WSALTER      [EXCLUDE(<wsid1>,<wsid2>,...)] FROM(yymmddhhmm)
             [PSCAP(n|RESET)]
             [R1CAP(n|RESET)] [R2CAP(n|RESET)] TO(yymmddhhmm)
             [UPDATE(Y|N)]
  [WSAUTO(Y|N)] [WSNAME(<wsid>)] [WSREP(A|S|C|M)]
             [WSRETYPE(D|Z)]
             [WSTYPE(G|C|P|R)] [WSWAIT(Y|N)] [WSZCENTR(Y|N)]
```

where:

EXCLUDE(<wsid1>,<wsid2>,...)

Allows a list of workstations to be specified to exclude from being processed by the command.

FROM(*yymmddhhmm*)

Specifies the start of the period to alter the parallel server capacity for.

PSCAP(*n*|RESET)

Specifies the number of parallel servers to set for this interval. **RESET** can be used to reset the parallel servers to the level planned for this interval. **RESET** can be used only if this interval has already been set by a previous command. If this keyword is not set, for a new interval HCL Workload Automation for Z will assume 0.

R1CAP(*n*|RESET)

Specifies the number of workstation resource 1 to set for this interval. **RESET** can be used to reset the parallel servers to the level planned for this interval. **RESET** can be used only if this interval has already been set by a previous command. If this keyword is not set, for a new interval HCL Workload Automation for Z will assume 0.

R2CAP(*n*|RESET)

Specifies the number of workstation resource 2 to set for this interval. **RESET** can be used to reset the parallel servers to the level planned for this interval. **RESET** can only be used if this interval has already been set by a previous command. If this keyword is not set, for a new interval HCL Workload Automation for Z will assume 0.

`TO(yymmddhhmm)`

Specifies the end of the period to alter parallel server capacity for.

`UPDATE(Y|N)`

Determines whether to perform the updates described by the command. If updates are requested to not be performed setting **`OPTIONS TRACE(1)`** will show what update commands would have been executed.

`WSAUTO(Y|N)`

Optional search criteria to search for workstations by the automation attribute.

`WSNAME (<wsid>)`

Optional search criteria to search for workstations to alter by name. Wildcards are allowed.

`WSREP(A|S|C|M)`

Optional search criteria to search for workstations by the reporting type.

`WSRETYPE(D|Z)`

Optional search criteria to search for workstations by remote engine type.

`WSTYPE(G|C|P|R)`

Optional search criteria to search for workstations by workstation type.

`WSWAIT(Y|N)`

Optional search criteria to search for workstations by the wait attribute.

`WSZCENTR(Y|N)`

Optional search criteria to search for workstations by the z-centric attribute.

Chapter 9. Using TSO commands within Workload Automation Programming Language

Some TSO commands are specific to HCL Workload Automation for Z; you can run them natively in TSO, through program EQQEVPGM, and through Workload Automation Programming Language.

Within Workload Automation Programming Language, you specify HCL Workload Automation for Z TSO commands almost in the same way as the normal HCL Workload Automation for Z TSO command with the same name. The only difference is that you are not required to specify **SUBSYS**.

If you omit **SUBSYS** from the command, Workload Automation Programming Language adds it by using the information that you set in **OPTIONS TRACKERS**, to set the correct tracker subsystem name for the controller with which you are communicating. If you did not specify any **OPTIONS TRACKERS**, the **SUBSYS** value that was passed to Workload Automation Programming Language at startup is used.

The **SUBSYS** keyword cannot be abbreviated within Workload Automation Programming Language.

Note:

- HCL Workload Automation for Z TSO commands are asynchronous: they generate an event to perform the action. Successful completion of the command indicates only that the event was generated, it does not mean that the event reached the target or that the keywords identified an object with the correct state for update.
- With the EQQWAPL load module, you cannot use the TSO commands **BULKDISC, DEFINE, DELETE, JSUACT**, and **REPRO**.

For detailed documentation and syntax about TSO commands, see *HCL Workload Automation for Z: Managing the Workload*.

BACKUP – Initiate JCL or CP backup

Use **BACKUP** to initiate a backup of either the current plan or JCL repository.

```
BACKUP <arguments>
```

BULKDISC – Initiate Bulk Discovery

Use **BULKDISC** to initiate a bulk discovery request.

```
BULKDISC <arguments>
```

Note: The `BULKDISC` request is available only starting from HCL Workload Automation for Z V8.3, or later.

JSUACT – Activate/Inactivate Job Submission

Use `JSUACT` to activate or inactivate job submission.

```
JSUACT <arguments>
```

OPINFO – Update Operation User field

Use `OPINFO` to update the User Field of an operation in the current plan.

```
OPINFO <arguments>
```

OPSTAT – Set operation status

Use `OPSTAT` to set the status of an operation in the current plan.

```
OPSTAT <arguments>
```

SRSTAT – Set special resource status

Use `SRSTAT` to set the status of a special resource in the current plan.

```
SRSTAT '<resource>' <arguments>
```

WSSTAT – Set workstation status

Use `WSSTAT` to set the status of a workstation in the current plan.

```
WSSTAT <arguments>
```

Other TSO commands

You can also run a small subset of TSO commands, including IDCAMS commands, directly from the Workload Automation Programming Language command stream.

The following commands are available:

`DEFINE`

IDCAMS DEFINE statement.

`DELETE`

IDCAMS DELETE statement. Note that there is also a PIF DELETE command accessible to Workload Automation Programming Language. If the first word of the command is a known HCL Workload Automation for Z record or segment, the delete command is directed to HCL Workload Automation for Z, otherwise it is considered to be an IDCAMS DELETE command. To ensure that you are issuing an IDCAMS DELETE, the first

word can be quoted fully qualified. Even if the fully qualified data set matches an HCL Workload Automation for Z record or segment name, the quotes will ensure the IDCAMS DELETE is used.

REPRO

IDCAMS REPRO statement.

Depending on the TSO PROFILE PREFIX setting for the user running the job, any data set names in TSO commands can be prefixed with the user ID. To avoid prefixing, you can specify data set names within single quotes.

Chapter 10. Batch loader commands

The batch loader function of Workload Automation Programming Language uses a set of statements to define or modify objects within the HCL Workload Automation for Z database.

Batch loader can be freely intermingled with other Workload Automation Programming Language commands, but no other Workload Automation Programming Language commands must intervene within a set of batch loader statements for a single HCL Workload Automation for Z database object.

Each HCL Workload Automation for Z object can be constructed using one or more batch loader statements, each statement relates to a component of the object. For example, an Application needs and **ADSTART** statement to define the core elements, such as the name and owner ID, an **ADRUN** statement for each run cycle and **ADOP** statements for each operation. The statements are hierarchical, so each **ADOP** statement can be followed by multiple **ADDEP** statements to define dependencies and **ADSR** statements to define Special Resource usage.

Every Batch Loader construct begins with a command that ends in **START**, such as **ADSTART** or **ETTSTART**. The end of the construct will be when either a new ***xxx*START** statement, or a statement not belonging to that object is encountered.

Modes of operation

The batch loader processing supports several modes of operation, which you can specify by setting either **OPTIONS DBMODE** or the **ACTION** keyword of individual batch loader statements. Behaviors are different at the different levels.

OPTIONS DBMODE

The following values are valid for the **OPTIONS DBMODE** statement.

ADD

The entire content of the object must be specified within Batch Loader statements and cannot exist already within the database (this is the default).

COPY

An object and its segments can be identified for updating using key fields (or **SAVELIST**), then *only* the fields that require changing need to be specified. If new values for key fields are given, a new object is created in the database and the original object is also kept in the database at completion. If an object with the same name already exists, the **COPY** fails.

EXPORT

An object is built from a combination of **TRANSLATE** and batch loader statements, then, instead of sending the object to the database, it is written out again as batch loader statements. In this way, input batch loader is translated to new batch loader, as part of life-cycle management, before applying to a database.

REPLACE

The entire content of the object must be specified within Batch Loader statements, but can already exist within the database and is replaced if an object with the same name and type already exists. If an object had been

selected for replacing but given a new name for output, the original object is deleted from the database at completion.

UPDATE

An object and its segments can be identified for updating using key fields (or **SAVELIST**), then *only* the fields that require changing need to be specified. If the object does not already exist, the **UPDATE** command acts like **ADD**. If new values for key fields are given, a new object is created in the database and the original object is deleted from the database at completion.

Note:

- With **OPTIONS FIRST** and **OPTIONS LAST**, you cannot use **COPY** and **UPDATE** in conjunction with **AUTOPRED**, **AUTOSUCC**, or **LINK**.
- Under EQQYLTOP, **OPTIONS ACTION** is the equivalent to **OPTIONS DBMODE**. Workload Automation Programming Language also recognizes **OPTIONS ACTION** for backwards compatibility. Use **OPTIONS DBMODE** to distinguish database between Record and Segment level actions.

Batch loader ACTION

The **ACTION** keyword is available on every batch loader statement and determines for what action the statement is to be used.

The default (and most used action) is **ADD**, meaning that the statement is used to define a segment.

The following values are valid for **ACTION** within batch loader statements:

ADD

The statement is defining an object or part of an object to be stored in the database.

DELETE

The segment is removed from the object. **DELETE** is not valid for primary batch loader statements (for example, ADSTART, CLSTART), objects themselves must be deleted by the **DELETE** command, not **ACTION(DELETE)** within batch loader statements. **DELETE** is valid only in conjunction with **OPTIONS DBMODE(COPY)** or **OPTIONS DBMODE(UPDATE)**.

EXPORT

The Batch Loader input is translated into ISPF loader output without being sent to the database. This is to allow Batch Loader to be sent into ISPF format, using ILSON, for manipulation before being sent into the database.

SUBMIT

When an Application is created using **ACTION(SUBMIT)**, instead of being stored in the Database it is inserted into the Current Plan without updating the database (only valid for **ADSTART**). Any occurrence created using this option will be Valid for all dates.

SETDEFAULT

If you specify **SETDEFAULT**, the remaining keyword values of the statement become default values for all the statements of the same type that follow. No database element is updated. Keywords that you do not specify are assigned their standard defaults. For more information about how **SETDEFAULT** works within Workload Automation Programming Language, see SETDEFAULT behaviour in Workload Automation Programming Language on page 248.

SETDEFAULT cannot be used for the following keywords:

- On **ADDEP** keywords **PREWSID**, **PREOPNO**, **PREJOBN** and **PREADID** can not be used with **SETDEFAULT**.
- On **ADOP** keywords **WSID**, **JOBN**, **PREWSID**, **PREOPNO** and **PREJOBN** can not be used with **SETDEFAULT**. Using **OPNO** with **SETDEFAULT** does not set the default operation number, instead it sets the interval to add to previous operation numbers when automatic operation numbering is used.
- On **ADOPEXTN** keyword **EXTNAME** can not be used with **SETDEFAULT**.
- On **ADRULE** no keywords can be used with **SETDEFAULT**.
- On **ADRUN** keywords **NAME** and **PERIOD** can not be used with **SETDEFAULT**.
- On **ADSR** keyword **RESOURCE** can not be used with **SETDEFAULT**.
- On **OISTART** keywords **ADID**, **JOBN**, **OPNO** and **MEMBER** can not be used with **SETDEFAULT**.

Output masking

Use output masking to update the content of existing fields using the percent sign (%) and asterisk (*).

Use the percent sign (%) as a wildcard for a single character, use the asterisk (*) as a wildcard for any number of characters at the end of the mask. Using the * anywhere but the end of the mask will cause the character to be treated as a literal asterisk.

For example, to modify a workstation NOVO with a description “New workstation”, issue the following command:

WSSTART WSNAME(NOVO) DESCR(“NEW *”) results in “NEW workstation”

WSSTART WSNAME(NOVO) DESCR(“%%% CPU”) results in “New CPU”

Note: By default, the output masking facility is turned off, to prevent accidental changes if you have the percent sign (%) or asterisk (*) in any of your database fields. To turn it on, use **OPTIONS OUTMASK(Y)**.

Batch loader syntax enhancements

By default, Workload Automation Programming Language supports the same format of batch loader supported by EQQYLTOP. However, the following syntax enhancements make the batch loader function easier to use.

Some enhancements are automatically available and do not require any **OPTIONS** to be set to exploit them or stop them.

- Quotation marks: Workload Automation Programming Language does not require you to specify the keyword values within quotation marks, but it supports them if used.
- Only primary batch loader statements (for example **ADSTART**, **CLSTART**) are required at the beginning of a new line.
- You can use as input alternative keywords that are shorter, more consistent, and meaningful.

Continuation rules have been amended to be able to use the full width of the input dataset, whatever the width may be, but to process batch loader generated under EQQYLTOP syntax rules you can set **OPTIONS SYNTAX(LEGACY)** to force EQQYLTOP compliance.

With **SYNTAX(LEGACY)**, Workload Automation Programming Language uses only keywords compatible with the legacy Batch Loader. With **SYNTAX(EXTENDED)**, the alternate syntax is listed as output.

The following list shows the alternative and extra keywords available:

- **ADSTART**
  - **ADSTAT** becomes **STATUS**
  - **ADTYPE** becomes **TYPE**
  - **ADVALFROM** becomes **VALFROM**
  - **GROUP** becomes **AUTHGRP**
  - **ADGROUPID** becomes **GROUPDEF**
- **ADCIV**
  - **ADCIVADID** becomes **ADID**
  - **ADCIVID** becomes **CONDID**
  - **ADCIVOPNO** becomes **OPNO**
  - **ADCIVTYPE** becomes **TYPE**
  - **ADCIVFWHE** becomes **FROMWHEN**
  - **ADCIVFHHH** becomes **FROMHHH**
  - **ADCIVFHH** becomes **FROMHH**
  - **ADCIVFMM** becomes **FROMMM**
  - **ADCIVFD** becomes **FROMDAYS**
  - **ADCIVTWHE** becomes **TOWHEN**
  - **ADCIVTHHH** becomes **TOHHH**
  - **ADCIVTHH** becomes **TOHH**
  - **ADCIVTMM** becomes **TOMM**
  - **ADCIVTD** becomes **TODAYS**
- **ADCNC**
  - **CONDDEPNO** is no longer required
  - **CONDCOUNT** becomes **COUNT**
  - **CONDDESCR** becomes **DESCR**

- **ADCNS**
  - **CONDDEPCONDID** is no longer required
  - **CONDDEPPREADID** becomes **PREADID**
  - **CONDDEPPRECSEL** becomes **PRECSEL**
  - **CONDDEPPREWSID** becomes **PREWSID**
  - **CONDDEPPREOPNO** becomes **PREOPNO**
  - **CONDDEPDEPTYP** is no longer required
  - **CONDDEPTYP** becomes **CHECK**
  - **CONDDEPLOG** becomes **LOGIC**
  - **CONDDEPVALRC** becomes **RC1**
  - **CONDDEPVALRC2** becomes **RC2**
  - **CONDDEPVALST** becomes **STATUS**
  - **CONDDEPPROCSTEP** becomes **PROCSTEP**
  - **CONDDEPSTEPNAME** becomes **STEPNAME**

- **ADDEP**
  - **PRINT** is a new keyword to allow the LTP Print option to be set

- **ADOP**
  - **ADOPWTO** becomes **WTO**
  - **ADOPCATM** becomes **CLEANUP**
  - **ADOPJOBCRT** becomes **CRITICAL**
  - **ADOPJOBPOL** becomes **POLICY**
  - **ADOPUSRSYS** becomes **USRSYS**
  - **ADOPEXPJCL** becomes **EXPJCL**
  - **ADOPWLMCLASS** becomes **WLMCLASS**

- **ADRUN**
  - **SEQ** is a new keyword to enable a specific run cycle to be identified for update, rather than having to specify every **ADRUN** statement again.
  - **ADRJTAB** becomes **JCLVTAB**

- **ADOPSAI**
  - **COMMTEXT** is no longer required.
  - **CT1** is a new keyword that maps to the first 64 characters of **COMMTEXT** in line with the way the command is displayed within the ISPF interface.
  - **CT2** is a new keyword that maps to the second 64 characters of **COMMTEXT**.
  - **CT3** is a new keyword that maps to the third 64 characters of **COMMTEXT**.
  - **CT4** is a new keyword that maps to the final 63 characters of **COMMTEXT**.
  - **COMPINFO** becomes **CI**.

The original syntax for **ADOPSAI** breaks keyword values across multiple lines. In the following example, the text is meaningless, used only to show the flow of complete keywords.

```
ADOPSAI
COMMTEXT('AAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAA
AABBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBCCCCCC
CCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCDDDDDDDDDDDDDD
DDDDDDDDDDDDDDDDDDDDDDDDDDDDDDDDDDDDDDDDDDDDDDDDDDDDDD')
AUTFUNC(AFAFAFAF) SECELEM(SESESESE)
COMPINFO('CICICICICICICICICICICICICICICICICICICICICICICICICICICICICICICICI
CI')
```

`SYNTAX(EXTENDED)` presents the same values in the following format, which is easier to read and follows the layout used to enter the information through the product dialogs.

```
ADOPSAI
   CT1(AAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAA)
   CT2(BBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBB)
   CT3(CCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCC)
   CT4(DDDDDDDDDDDDDDDDDDDDDDDDDDDDDDDDDDDDDDDDDDDDDDDDDDDDDDDDDDDDDDDDDDD)
   AUTFUNC(AFAFAFAF) SECELEM(SESESESE)
   CI(CICICICICICICICICICICICICICICICICICICICICICICICICICICICICICICICICICI)
```

- `ADXIV`
    - `ADXIVADID` becomes `ADID`
    - `ADXIVWSID` becomes `WSID`
    - `ADXIVOPNO` becomes `OPNO`
    - `ADXIVTYPE` becomes `TYPE`
    - `ADXIVFWHE` becomes `FROMWHEN`
    - `ADXIVFHHH` becomes `FROMHHH`
    - `ADXIVFHH` becomes `FROMHH`
    - `ADXIVFMM` becomes `FROMMM`
    - `ADXIVFD` becomes `FROMDAYS`
    - `ADXIVTWHE` becomes `TOWHEN`
    - `ADXIVTHHH` becomes `TOHHH`
    - `ADXIVTHH` becomes `TOHH`
    - `ADXIVTMM` becomes `TOMM`
    - `ADXIVTD` becomes `TODAYS`

- `RGSTART`
    - `RGNAME` becomes `RGID`
    - `RGIATIME` becomes `IATIME`
    - `RGJVTAB` becomes `JVTAB`
    - `RGCALEND` becomes `CALENDAR`
    - `RGDESCR` becomes `DESCR`
    - `RGOWNER` becomes `OWNER`
    - `RGDLDAY` becomes `DLDAY`
    - `RGDLTIME` becomes `DLTIME`

SETDEFAULT behaviour in Workload Automation Programming Language

An element of the EQQYLTOP syntax that is not compatible with Workload Automation Programming Language is the ability to `SETDEFAULT` for a segment without first providing a `SETDEFAULT` for the record itself.

For example, in EQQYLTOP the following commands set the default workstation to **CPU1** and use that default value in the remaining statements:

```
ADOP ACTION(SETDEFAULT) DURATION(10)
ADSTART ADID(NEWAPPL)
ADOP WSID(CPU1) OPNO(001) JOBN(JOB1)
ADOP WSID(CPU1) OPNO(002) JOBN(JOB2)
```

Because Workload Automation Programming Language uses object based structures throughout the syntax, to do the same thing you need to add an `ADSTART` statement before the `ADOP` statement to ensure that the structures are coherent:

```
ADSTART ACTION(SETDEFAULT)
ADOP ACTION(SETDEFAULT) DURATION(10)
ADSTART ADID(NEWAPPL)
ADOP WSID(CPU1) OPNO(001) JOBN(JOB1)
ADOP WSID(CPU1) OPNO(002) JOBN(JOB2)
```

Note: All the statements within a `SETDEFAULT` structure can specify only the `SETDEFAULT ACTION`. If `ACTION` is not specified for child batch loader statements, `SETDEFAULT` is assumed.

Keyword abbreviation

In Workload Automation Programming Language, you must specify all keywords completely, they cannot be abbreviated. This allows optimum performance for the Workload Automation Programming Language parsing engine. However, the `EXTENDED` syntax shortens many of the longer EQQYLTOP keywords.

Suffixing

Some Batch Loader statements support the `SUFFIX` keyword. This allows a variable value to be suffixed onto the name of the object.

`SUFFIX` supports the following variables:

`CDAY`
: Current day of the week as a number 1=Monday.

`CDD`
: Current day of the month.

`CDDD`
: Current day of the year.

`CDDMMYY`
: Current date in DDMMYY format.

CHH

Current hour.

CHHMM

Current time in HHMM format.

CHHMMSS

Current time in HHMMSS format.

CHHMMSSX

Current time in HHMMSSXX format.

CMM

Current month.

CMMYY

Current month and year in MMYY format.

CYMD

Current date in YYYYMMDD format.

CYY

Current year in YY format.

CYYDDD

Current Julian day in YYDDD format.

CYYMM

Current year and month in YYMM format.

CYYMMDD

Current date in YYMMDD format.

CYYYY

Current year in YYYY format.

CYYYYMM

Current year and month in YYYYMM format.

If **SUFFIX** is used to specify anything other than these variables, it is treated as a literal.

Once a particular variable has been referenced in an execution of Workload Automation Programming Language, any subsequent references return the same value. For example, **CHHMMSSX** referenced twice in the same Batch Loader, returns exactly the same value, despite the fact the second reference might be processed a few hundredths of a second later. This is to allow **SUFFIX** to be used to create an Application name, and then be able to reliably make dependencies to it.

The following example creates two dynamic applications using CHHMMSSX to create an Application name containing the time down to one hundredth of a second. Because the variables are static within Workload Automation Programming Language, both applications will have the same timestamp, but importantly the second Application can be made dependent on the first.

```
OPTIONS CPDEPR(Y)
ADSTART ACTION(SETDEFAULT) ADOP DURATION(1)
ADSTART ACTION(SUBMIT) ADID(DYNAMIC1) SUFFIX(CHHMMSSX)
        DESCR('SEQUENCE OF JOBS') OWNER(TWS)
ADOP WSID(NONR) OPNO(FIRST) AUTOPRED(PREV)
ADOP WSID(CPU1) JOBN(JOB1)
ADOP WSID(CPU1) JOBN(JOB2)
ADOP WSID(CPU1) JOBN(JOB3)
ADOP WSID(NONR) OPNO(LAST)
ADSTART ACTION(SUBMIT) ADID(DYNAMIC2) SUFFIX(CHHMMSSX)
        DESCR('SEQUENCE OF JOBS') OWNER(TWS)
ADOP WSID(NONR) OPNO(FIRST) AUTOPRED(PREV)
ADDEP PREADID(DYNAMIC1) SUFFIX(CHHMMSSX) PREOPNO(LAST) PREWSID(NONR)
ADOP WSID(CPU1) JOBN(JOB4)
ADOP WSID(CPU1) JOBN(JOB5)
ADOP WSID(CPU1) JOBN(JOB6)
ADOP WSID(NONR) OPNO(LAST)
```

Note: `OPTIONS SUFFIX` is used to control the behavior of the `SUFFIX` keyword, to manage situations where the addition of the suffix might exceed the maximum allowed length of the object name.

NEW_ keywords

With `OPTIONS DBMODE(UPDATE)` you can change the "name" of an object, by modifying its key fields. In the same way, with `OPTION DBMODE(COPY)` new key fields must be specified to save a copy to. To do this, the key fields of any HCL Workload Automation for Z object can have alternate names specified by using a `NEW_` prefix for any of the key fields you wish to alter.

For example:

```
OPTIONS DBMODE(COPY)
ADSTART ADID(MYAPPL1) NEW_ADID(MYAPPL1COPY)
```

The key fields can be identified by looking in the default `OUTPUT` definitions shipped with Workload Automation Programming Language in the SEQQWAPL library.

Note: This does not apply to temporary Operator Instructions. They cannot have their validity range changed by `NEW_` keywords. To modify the validity of temporary Operator Instructions they must be deleted and created again.

AD – Application definition record

The AD record is a multi segment record with the following structure.

```
ADCOM -+- Common segment (1 per appl)
       |
       += ADAPD =+= Application dependency(ies)
       |
```

```
      += ADRUN =+= Run Cycle(s)
      |         |
      |         +- ADRULE - Rule (1 per run cycle)
      |
      += ADOP =+= Operation(s)
               |
               += ADDEP = Dependency(ies)
               |
               += ADXIV = External dependency interval(s)
               |
               += ADSR = Special resource(s)
               |
               +- ADOPEXT - Extended name (1 per op)
               |
               +- ADOPSAI - System Automation (1 per op)
               |
               += ADCNC = Condition(s)
               |
               += ADCNS = Conditional dependency(ies)
               |
               += ADCIV = Conditional dependency interval(s)
               |
               += ADUSRF = User field(s)
               |
               += ADVDD  = Variable duration(s)
               |
               +- ADRE =- Remote job (1 per op)
```

Note: Some segments have Batch Loader statements with slightly different names:

- ADCOM has a Batch Loader statement of **ADSTART**
- ADOPEXT has a Batch Loader statement of **ADEXT**
- ADOPSAI has a Batch Loader statement of **ADSAI**
- ADUSRF has a Batch Loader statement of **ADUSF**

Automatic Operation numbering

The Operation number is the unique identifier of an Operation within an Application. It is specified by the **OPNO** argument on the **ADOP** statement which defines each operation.

If you omit the **OPNO** argument from an **ADOP** statement, Workload Automation Programming Language can automatically allocate operation numbers for you. Workload Automation Programming Language calculates the operation number by adding an interval (the default is 1) to the previous operation number. If the first operation number is omitted, the previous operation number is assumed to be zero, meaning the first operation number will be equal to the interval.

The interval can be set by using the **ACTION(SETDEFAULT)** process to set **OPNO**, the operation number you set using this method is not the default operation number, but becomes the interval between them when **OPNO** is omitted.

For example, the following command sets an interval of 5 between automatically allocated operation numbers:

```
ADSTART ACTIONS(SETDEFAULT)
      ADOP OPNO(005)
```

If the interval is larger than the previous operation number, the generated operation number is set to the value of the interval. In the following example, JOB1 is given an operation number of 005, with 010 and 015 for JOB2, and JOB3 respectively:

```
ADSTART ACTIONS(SETDEFAULT)
      ADOP OPNO(005) DURATION(1)
      ADSTART ADID(MYAPPL) OWNER(TWS)
      ADOP WSID(NONR) OPNO(001) JOBN(START) AUTOPRED(PREV)
      ADOP WSID(CPU1) JOBN(JOB1)
      ADOP WSID(CPU1) JOBN(JOB2)
      ADOP WSID(CPU1) JOBN(JOB3)
      ADOP WSID(NONR) OPNO(255) JOBN(LAST)
```

Automatically allocated operation numbers can be identified in the Workload Automation Programming Language output as the **OPNO** keyword is prefixed with a plus sign (+) on the listed **ADOP** statements:

```
EQQI200I ADSTART ADID(TESTSORT) OWNER(TWS)
EQQI203I ADOP WSID(DUMM) JOBN(JOB1) AUTOPRED(PREV) OPNO(1)
EQQI203I ADOP WSID(DUMM) JOBN(JOB2) +OPNO(005)
EQQI203I +ADDEP PREOPNO(1)
EQQI203I ADOP WSID(DUMM) JOBN(JOB3) +OPNO(010)
EQQI203I +ADDEP PREOPNO(5)
EQQI203I ADOP WSID(DUMM) JOBN(JOB4) +OPNO(015)
EQQI203I +ADDEP PREOPNO(10)
EQQI203I ADOP WSID(DUMM) JOBN(JOB5) +OPNO(020)
```

Automatic dependencies

Batch loader can create automatic dependencies within an application by using the **OPTIONS ADDEP** keyword and the **ADOP AUTOPRED** and **ADOP AUTOSUCC** statements, when adding or replacing an entire application.

The **AUTOPRED** and **AUTOSUCC** keywords are set in the **ADOP** statements and cause the automatic creation of dependencies based on the batch loader statements for any operations that follow.

The keywords can contain the following values:

nnn

Operation number. Predecessors or successors will be made to the specified operation number automatically.

FIRST

Predecessors or successors will be made to theWorkload Automation Programming Language designated first operation number automatically.

LAST

Predecessors or successors will be made to the Workload Automation Programming Language designated last operation number automatically.

OFF

The current level of **AUTOPRED** or **AUTOSUCC** will be stopped. If a previous **AUTOPRED** or **AUTOSUCC** was specified but not stopped by an **OFF** argument, then the previous setting will be resumed.

PREV

This is valid only for **AUTOPRED** and will make any subsequent operations automatically dependent on the operation identified by the preceding **ADOP** statement.

The **OPTIONS FIRST** and **OPTIONS LAST** keywords can also specify automatic dependencies to the **FIRST** and **LAST** operations by using **LINK** argument.

For example, **OPTIONS FIRST(1,LINK)** names operation 1 as the first operation and will make operation 1 an automatic predecessor to any operations specified in batch loader with no explicit predecessors.

By default, automatic dependencies are only made to operations without any explicit dependencies made from or to them in the batch loader.

For example, **AUTOPRED(FIRST)** will only make operations dependent on the first operation that do not have any explicitly specified predecessors, equally **AUTOSUCC(LAST)** will only make the last operation dependent on operations that have no explicit successors.

To make automatic dependencies take place regardless of whether explicit dependencies exist you can use the **ALL** argument.

For example, **AUTOPRED(010,ALL)** will make every operation dependent on operation 010, or **OPTIONS LAST(255,LINK,ALL)** will make operation 255 a successor to every operation in the application.

Precedence: When deciding whether an automatic dependency when **ALL** is not specified, the following dependencies are assessed in this order:

1. **ADOP** and **ADDEP** statements created explicitly by the user
2. **AUTOSUCC** keywords
3. **AUTOPRED** keywords
4. **OPTIONS LAST**
5. **OPTIONS FIRST**

Note:

1. Automatic dependencies can ONLY be addressed to specific operations by the operation number, not by using Jobname or Workstation name. When considering whether an operation already has a predecessor or successor Workload Automation Programming Language only acknowledges them from a statement that uses the operation number. It is therefore recommended that any manual dependencies made in conjunction with automatic dependencies should be performed using **PREOPNO** to ensure the desired result is achieved.
2. Automatic dependencies are not permitted when using **OPTIONS UPDATE** or **OPTIONS COPY**.
3. Automatic dependencies are for internal dependencies only.
4. Automatic dependencies rely on the order of the **ADOP** statements when resolving dependencies. The **ADOP** statements do not have to be specified in numerical order to build an application, as Workload Automation Programming Language will ensure the resulting application has operations in numeric sequence so it is

possible to exploit this, for example, to have 255 as your logical end point of an application, and still have higher numbered operations as predecessors to it, as long as you specify operation 255 last.

In the following example, operations 010 to 110 are all successors to 001 and predecessors to 255.

```
OPTIONS FIRST(1,LINK) LAST(255,LINK)
ADSTART ADID(MYAD) OWNER(TWS)
ADOP OPNO(001) WSID(NONR) DURATION(1)
ADOP OPNO(010) WSID(CPU1) JOBN(JOB01) DURATION(1)
ADOP OPNO(020) WSID(CPU1) JOBN(JOB02) DURATION(1)
ADOP OPNO(030) WSID(CPU1) JOBN(JOB03) DURATION(1)
ADOP OPNO(040) WSID(CPU1) JOBN(JOB04) DURATION(1)
ADOP OPNO(050) WSID(CPU1) JOBN(JOB05) DURATION(1)
ADOP OPNO(060) WSID(CPU1) JOBN(JOB06) DURATION(1)
ADOP OPNO(070) WSID(CPU1) JOBN(JOB07) DURATION(1)
ADOP OPNO(080) WSID(CPU1) JOBN(JOB08) DURATION(1)
ADOP OPNO(090) WSID(CPU1) JOBN(JOB09) DURATION(1)
ADOP OPNO(100) WSID(CPU1) JOBN(JOB10) DURATION(1)
ADOP OPNO(110) WSID(CPU1) JOBN(JOB11) DURATION(1)
ADOP OPNO(099) WSID(NONR) DURATION(1)
```

Submitting batch loader directly to the current plan

Within Workload Automation Programming Language, you can use batch loader statements to create an occurrence directly in the current plan without creating the application in the database. To do that, set `ACTION(SUBMIT)` in the `ADSTART` statement.

If `OPTIONS DBMODE(ADD)` or `OPTIONS DBMODE(REPLACE)` is used, only the statements in Batch Loader are used to build the occurrence in the plan. If `OPTIONS DBMODE(COPY)` or `OPTIONS DBMODE(UPDATE)` is used, any existing application with the same name in the database is used as a model with the batch loader statements making amendments to the version being added to the current plan, the database remains unchanged.

The `ADSTART` and subsequent segments perform the equivalent of an `INSERT CPOC`, so the batch loader must be followed by an `EXECUTE MCPBLK` to commit the occurrence to the plan. Unless you specify `OPTIONS EXECUTE(MANUAL)`, this happens automatically before Workload Automation Programming Language terminates your session with HCL Workload Automation for Z.

It is possible to follow the batch loader statements with `MODIFY` or `INSERT` statements to amend the newly created occurrence before it is committed to the current plan with an `EXECUTE` statement. This makes it possible to `INSERT` predecessors or successors between the occurrence being created and specific occurrences already in the plan.

For example:

```
ADSTART ACTION(SETDEFAULT)
ADOP DURATION(1)
ADSTART ACTION(SUBMIT) ADID(DYNAMAPPL) DESCR('DEMONSTRATE SUBMIT')
   OWNER(TWS) ODESCR('TWS INFRASTRUCTURE') PRIORITY(5)
ADOP WSID(CPU1) OPNO(1) JOBN(JOB1) DESCR('FIRST JOB')
ADOP WSID(CPU1) OPNO(10) JOBN(JOB2) DESCR('SECOND JOB')
ADOP WSID(CPU1) OPNO(20) JOBN(JOB3) DESCR('THIRD JOB')
ADOP WSID(CPU1) OPNO(30) JOBN(JOB4) DESCR('FOURTH JOB')
ADOP WSID(CPU1) OPNO(255) JOBN(JOB5) DESCR('LAST JOB')
```

```
MODIFY CPOP OPNO(1)
INSERT CPPRE PREADID(PLANNEDAPPL) PREIA(0803081200) PREOPNO(030)
EXECUTE MCPBLK
```

Note: The entire object must be specified in batch loader before any `INSERT` or `MODIFY` statements can be used to alter it. Batch loader statements (such as, for example, `ADSTART` and `ADOP`) cannot be interleaved with current plan commands (such as, for example, `MODIFY` and `INSERT`).

ADAPD - Application Dependency

Use the `ADAPD` statement to define a predecessor for an application.

Table 116. Keywords for ADAPD

Keyword	Description
`PREADID`	Name of the predecessor application.

`PRECSEL`	Specifies on which basis a matching predecessor is selected:
	`C`
	Closest preceding. The matching predecessor is the one with the nearest preceding input arrival time. This is the default.
	`S`
	Same scheduled date. The matching predecessor is the one with the nearest input arrival time within the same day of the operation (occurrence) under consideration. A matching predecessor is first searched before the IA time of the operation. Then, if not found, it is searched after the IA time of the operation.
	`A`
	Within an absolute interval. The matching predecessor is the one with the closest input arrival time in the specified interval. The interval boundaries are specified by a time and a number of days before or after the IA time of the operation (occurrence). The interval can be timed entirely before, entirely after, or across the IA time of the operation (occurrence).
	`R`
	Within a relative interval. The matching predecessor is the one with the closest input arrival time in the specified interval. The interval boundaries are calculated using an offset expressed in hours and minutes before or after the IA time of the operation (occurrence). The interval can be timed entirely before, entirely after, or across the IA time of the operation (occurrence).

Table 116. Keywords for ADAPD (continued)

Keyword	Description
PREOPNO	The operation number of a predecessor operation to this application. If the predecessor is an application, **PREOPNO** is 0.
PREWSID	The four-character workstation name of a predecessor operation to this application. If the predecessor is an application, **PREWSID** is blank.
FROMDAYS	The start of the absolute interval in days. The allowed range is 0-7.
FROMHH	The start of the absolute interval in the *HH* format. The allowed range is 00-24. To be specified together with **FROMMM**. For example, if the absolute interval starts at 10:30 of the day before the input arrival time of the successor, it is defined by **FROMHH(10) FROMMM(30) FROMDAYS(1) FROMWHEN(B)**
FROMHHH	The start of the relative interval in hours. The format is *HHH* and the allowed range is 0-167. To be specified together with **FROMMM**.
FROMMM	The minutes fraction of the start of the relative or absolute interval.
FROMWHEN	Specifies if the start of the relative or absolute interval is before (**B**) or after (**A**) the input arrival time of the successor. For relative intervals only, you can choose to make the interval start at an indefinite time in the plan (in this case the mechanism used is similar to that of the closest preceding predecessor). To do this, do not specify this parameter, nor any of the other **FROM...** ones.
TODAYS	The end of the absolute interval in days. The allowed range is 0-7.
TOHH	The end of the absolute interval in the *HH* format. The allowed range is 00-24. To be specified together with **TOMM**. For example, if the absolute interval ends at 12:30 two days after the input arrival time of the successor, it is defined by **TOHH(12) TOMM(30) TODAYS(2) TOWHEN(A)**
TOHHH	The end of the relative interval in hours. The format is *HHH* and the allowed range is 0-167. To be specified together with **TOMM**.
TOMM	The minutes fraction of the end of the relative or absolute interval.
TOWHEN	Specifies if the end of the relative or absolute interval is before (**B**) or after (**A**) the input arrival time of the successor.
TYPE	The interval type: **A** Absolute interval. Must be defined by the following parameters: **FROMWHEN, FROMHH, FROMMM, FROMDAYS,**

Table 116. Keywords for ADAPD (continued)

Keyword	Description

	`TOWHEN, TOHH, TOMM, TODAYS.`
	`R`
	Relative interval. Must be defined by the following parameters:
	`[FROMWHEN, FROMHHH, FROMMM, ]`
	`TOWHEN, TOHHH, TOMM.`

`DESCR`	A free-format description of the dependency. It can be up to 50 characters.

`PRINT`	Specifies the Long Term Plan print option:
	`A`
	Always
	`C`
	Conditional

ADCIV – External conditional dependency interval

Use the `ADCIV` statement to define the absolute or relative interval specified with the `A` or `R` value in the `ADCNS PRECSEL` parameter.

You can set only one `ADCIV` per `ADCNS`, but the same `ADCIV` can be used by more `ADCNS` statements if they refer to the same external predecessor application and operation. The statement must be nested within the `ADCNS` to which it refers.

Note: The `ADCIV` statement is available only starting from HCL Workload Automation for Z V9.1, or later.

Table 117. Keywords for ADCIV

Keyword	Description
`ADID`	The application name of the conditional external predecessor to which the interval applies.
`CONDID`	The condition ID of the conditional external predecessor to which the interval applies.
`FROMDAYS`	The start of the absolute interval in days. The allowed range is 0-7.
`FROMHH`	The start of the absolute interval in the *HH* format. The allowed range is 00-24. Goes together with `FROMMM`. For example, if the absolute interval starts at 10:30 of the day before the input arrival time of the successor, it is defined by `FROMHH(10) FROMMM(30) FROMDAYS(1) FROMWHEN(B)` .

Table 117. Keywords for ADCIV (continued)

Keyword	Description
FROMHHH	The start of the relative interval in hours. The format is *HHH* and the allowed range is 0-167. To be specified together with **FROMMM**.
FROMMM	The minutes fraction of the start of the relative or absolute interval.
FROMWHEN	Specifies if the start of the relative or absolute interval is before (**B**) or after (**A**) the input arrival time of the successor. For relative intervals only, you can choose to make the interval start at an indefinite time in the plan (in this case the mechanism used is similar to that of the closest preceding predecessor). To do this, do not specify this parameter, nor any of the other **FROM...** ones.
OPNO	The operation number of the conditional external predecessor to which the interval applies.
TODAYS	The end of the absolute interval in days. The allowed range is 0-7.
TOHH	The end of the absolute interval in the *HH* format. The allowed range is 00-24. To be specified together with **TOMM**. For example, if the absolute interval ends at 12:30 two days after the input arrival time of the successor, it is defined by **TOHH(12) TOMM(30) TODAYS(2) TOWHEN(A)**
TOHHH	The end of the relative interval in hours. The format is *HHH* and the allowed range is 0-167. To be specified together with **TOMM**.
TOMM	The minutes fraction of the end of the relative or absolute interval.
TOWHEN	Specifies if the end of the relative or absolute interval is before (**B**) or after (**A**) the input arrival time of the successor.
TYPE	The interval type: **A** Absolute interval. Must be defined by the following parameters: **FROMWHEN, FROMHH, FROMMM, FROMDAYS,** **TOWHEN, TOHH, TOMM, TODAYS.** **R** Relative interval. Must be defined by the following parameters: **[FROMWHEN, FROMHHH, FROMMM,]** **TOWHEN, TOHHH, TOMM.**

ADCNC – Condition

Use the **ADCNC** statement to define a condition for an operation that combines a set of following conditional dependencies (**ADCNS**).

Note:

1. Workload Automation Programming Language does not require an equivalent to the **CONDDEPNO** keyword, this is calculated automatically.
2. The **ADCNC** statement is available only starting from HCL Workload Automation for Z V8.5, or later.

Table 118. Keywords for ADCNC

Keyword	Description
CONDID	Condition identified (1 -999).
COUNT	Count of conditional dependencies that must be true for the condition to be met: **0** All conditions must be true (default). **Any number higher than zero** Minimum number of conditional dependencies that must be true for this condition to be met. Formerly **CONDCOUNT** with EQQYLTOP.
DESCR	A free format description of the condition (up to 16 characters). Formerly **CONDDESCR** with EQQYLTOP.

ADCNS – Conditional dependency

Use the **ADCNS** statement to define a conditional dependency for an operation. You must set at least one **ADCNS** statement for each **ADCNC** statement, matched by the **CONDID** keyword.

Note:

1. Workload Automation Programming Language does not require an equivalent to the following keywords:
   - **CONDDEPCONDID** is inherited from the preceding **ADCNC** statement.
   - **CONDDEPDEPTYPE** is determined from the setting of **PREADID**.
2. The **ADCNS** statement is available only starting from HCL Workload Automation for Z V8.5, or later.

Table 119. Keywords for ADCNS

Keyword	Description
CONDID	The **CONDID** keyword should be set to the same value as in the **ADCNC** statement to which this belongs.
PREADID	If the predecessor operation is in a different application from the one being built, or is in a different occurrence of the same application, you must identify the application ID with this keyword. If you use DBCS characters, you must enter them as a quoted string started by a shift-out and ended by a shift-in.
PRECSEL	Specifies on which basis a matching predecessor is selected: **C** Closest preceding. The matching predecessor is the one with the nearest preceding input arrival time. This is the default. **S** Same scheduled date. The matching predecessor is the one with the nearest input arrival time within the same day of the operation (occurrence) under consideration. A matching predecessor is first searched before the IA time of the operation. Then, if not found, it is searched after the IA time of the operation. **A** Within an absolute interval. The matching predecessor is the one with the closest input arrival time in the specified interval. The interval boundaries are specified by a time and a number of days before or after the IA time of the operation (occurrence). The interval can be timed entirely before, entirely after, or across the IA time of the operation (occurrence). If you select this option, the **ADCIV** statement must follow **ADCNS** with the specification of the interval boundaries. **R** Within a relative interval. The matching predecessor is the one with the closest input arrival time in the specified interval. The interval boundaries are calculated using an offset expressed in hours and

Table 119. Keywords for ADCNS (continued)

Keyword	Description
	minutes before or after the IA time of the operation (occurrence). The interval can be timed entirely before, entirely after, or across the IA time of the operation (occurrence). If you select this option, the **ADCIV** statement must follow **ADCNS** with the specification of the interval boundaries. **Note:** The **PRECSEL** keyword is available only starting from HCL Workload Automation for Z V9.1, or later.
PREOPNO	Operation number of conditional predecessor.
PROCSTEP	Proc Step name for a step dependency.
STEPNAME	Job Step name for a step dependency.
CHECK	What element to check: **ST** Check the status **RC** Check the return code
LOGIC	What logic to use when comparing values: **EQ** Equal to **GE** Greater than or equal to **GT** Greater than **LE** Less than or equal to **LT** Less than **NE** Not equal to

Table 119. Keywords for ADCNS (continued)

Keyword	Description
	RG Range
STATUS	The value of operation status to use in the comparison.
RC1	The return code to use in the comparison. When **LOGIC(RG)** is used, RC1 is the lower limit of the range.
RC2	When **LOGIC(RG)** is used, RC2 is the upper limit of the range.

ADDEP - Dependency

Use the **ADDEP** statement to define a predecessor for an operation.

Table 120. Keywords for ADDEP

Keyword	Description
PREADID	If the predecessor operation is in a different application from the one being built, or is in a different occurrence of the same application, you must identify the application ID with the **PREADID** keyword.
PRECSEL	Specifies on which basis a matching predecessor is selected: **C** Closest preceding. The matching predecessor is the one with the nearest preceding input arrival time. This is the default. **S** Same scheduled date. The matching predecessor is the one with the nearest input arrival time within the same day of the operation (occurrence) under consideration. A matching predecessor is first searched before the IA time of the operation. Then, if not found, it is searched after the IA time of the operation. **A** Within an absolute interval. The matching predecessor is the one with the closest input arrival time in the specified interval. The interval boundaries are specified by a time and a number of days before or after the IA time of the operation (occurrence). The interval can be timed entirely before, entirely after, or across the IA time of the operation (occurrence). If you select this option, the ADXIV statement must follow ADDEP with the specification of the interval boundaries.

Table 120. Keywords for ADDEP (continued)

Keyword	Description
	R Within a relative interval. The matching predecessor is the one with the closest input arrival time in the specified interval. The interval boundaries are calculated using an offset expressed in hours and minutes before or after the IA time of the operation (occurrence). The interval can be timed entirely before, entirely after, or across the IA time of the operation (occurrence). If you select this option, the ADXIV statement must follow ADDEP with the specification of the interval boundaries. **Note:** The **PRECSEL** keyword is available only starting from HCL Workload Automation for Z V9.1, or later.
PREMAND	Specifies if it is mandatory that the dependency be resolved before the operation can start: **N** The dependency is not mandatory. This means that, if the predecessor is not found, the dependency is considered resolved unless failure is required (within the dynamic addition of a dependency in the Modify Current Plan panel). This is the default value. **C** The dependency is mandatory at ad hoc add level. The predecessor is required, but might not be in the plan at the time the occurrence that includes the successor is added and might be made available later via ETT, PIF, or manual intervention. This means that if the predecessor is not found when an occurrence is added to the current plan, a pending mandatory predecessor entry is created and the occurrence is added in the waiting status. The pending mandatory predecessor entry is created also when LTP and DP batch start running and the predecessor is not found. **P** The dependency is mandatory at plan level. The predecessor is expected to exist at the time the occurrence that includes the successor is dynamically added into the current plan (via the MCP panel). If it does not, the addition of the occurrence fails. Also LTP and DP batch will fail if the predecessor is not found when they run.

Table 120. Keywords for ADDEP (continued)

Keyword	Description
	Note: The `PREMAND` keyword is available only starting from HCL Workload Automation for Z V9.1, or later.
`PREOPNO`	The operation number of a predecessor operation to this operation.
`PREWSID`	The four-character workstation name of a predecessor operation to this operation.
`PREJOBN`	The job name of a predecessor operation to this operation.
`TRANSPT`	When HCL Workload Automation for Z creates the plan, it allows these minutes between the completion of the predecessor and the start of the successor operation that is being defined. You must specify an integer. The default is the time specified for the workstation.
`DESCR`	A free-format description of the dependency. It can be up to 50 characters. HCL Workload Automation for Z holds descriptions only for external dependencies. This field cannot be used to hold a description for an internal dependency.

`PRINT`	Specifies the Long Term Plan print option:
	`A`
	Always
	`C`
	Conditional

ADEXT – Extended information (ADOPEXT segment)

Use the `ADOPEXT` statement to define extended information for an operation.

Note: The `ADEXT` statement is available only starting from HCL Workload Automation for Z V8.2, or later.

Table 121. Keywords for ADEXT

Keyword	Description
`EXTNAME`	A free-format name for the operation. It can include blanks and special characters for a maximum of 54 characters.
`EXTSE`	The name of the scheduling environment for this operation. Special characters are allowed.

ADLAT – User-defined late operation

Use the **ADLAT** statement to specify your own settings by which an operation is to be considered late, and the alerts or actions to be taken.

Note: The **ADLAT** statement is available only starting from HCL Workload Automation for Z V9.5.

Table 122. Keywords for ADLAT

Keyword	Description
ACTIONACT	Specifies the action to be taken if the operation has not yet started when the specified day and time is reached. If you specify an action, **ACTIONDD** and **ACTIONDT** are required. Allowed values are: **A** Only an alert message is issued. **C** The operation is set to Complete, if its status allows it. Otherwise it is NOPed. **E** The operation is set to Error with OLAT, if its status allows it. Otherwise, this setting is postponed at the time when the status allows it. **N** The operation and all its internal successors are NOPed, if their status allows NOPing. Otherwise, it is ignored.
ACTIONBASE	Always **F**. It means that the input arrival time of the operation is used as the base date to apply the offset.
ACTIONDD	Specifies the day offset, related to the application start day, used to check if the operation has not yet started. If the operation has not started, the action that you set is taken. It can be a value from 0 to 99 (where 0 means the start day, 1 means one working day after the start day, and so on). This check is applied only to operations with status Ready.
ACTIONDIR	Always **A** (After). It means that the day offset and time are always added, and never subtracted, to the base date.
ACTIONDT	Required if you have set **ACTIONDD**. Specifies the time, related to the application start day, used to check if the operation has not yet started. If the operation has not started, the action that you set is taken.

Table 122. Keywords for ADLAT (continued)

Keyword	Description
	This value is specified in the format *HHMM*. This check is applied only to operations with status Ready.
ALERTBASE	Always **F**. It means that the input arrival time of the operation is used as the base date to apply the offset.
ALERTDD	Specifies the day offset, related to the application start day, used to check if the operation has not yet started. If the operation has not started, an alert message is issued. It can be a value from 0 to 99 (where 0 means the start day, 1 means one working day after the start day, and so on).
ALERTDIR	Always **A** (After). It means that the day offset and time are always added, and never subtracted, to the base date.
ALERTDT	Required if you have set **ALERTDD**. Specifies the time, related to the application start day, used to check if the operation has not yet started. If the operation has not started, an alert message is issued. This value is specified in the format *HH.MM*.

ADOP - Operation

Use the **ADOP** statement to define an operation.

Table 123. Keywords for ADOP

Keyword	Description
ADOPDLACT	Specifies the action to be taken if the operation is still not completed when the deadline day and time are reached. Blank means no action. If you specify an action, an alert message is always issued. Allowed values are: **A** Only an alert message is issued. **C** The operation is set to Complete, if its status allows it. Otherwise it is NOPed. **E** The operation is set to Error with ODEA, if its status allows it. Otherwise, this setting is postponed at the time when the status allows it.

Table 123. Keywords for ADOP (continued)

Keyword	Description
	N The operation and all its internal successors are NOPed, if their status allows NOPing. Otherwise, it is ignored.
AUTOPRED	Defines an automatic predecessor for operations following this one that do not have an explicit dependency set: **FIRST** Make dependencies to the default first operation. **LAST** Make dependencies to the default last operation. **PREV** To make the operation automatically dependent on the one preceding it in the batch loader statements. ***nnn*** Identifies the operation number to which the dependency can be made. **OFF** Ends the most recent setting of **AUTOPRED** and returns to any previous value, if set within the application. A second argument of **ALL** (for example, **AUTOPRED(FIRST,ALL)**) causes automatic dependencies to be made for all following operations, including ones with explicit dependencies. **Note:** **AUTOPRED** is not allowed with **OPTIONS DBMODE(COPY)** or **OPTIONS DBMODE(UPDATE)**.
AUTOSUCC	Defines an automatic successor for operations following this one that do not have an explicit dependency set: **FIRST** Make dependencies to the default first operation. **LAST** Make dependencies to the default last operation. ***nnn*** Identifies the operation number to which the dependency can be made.

Table 123. Keywords for ADOP (continued)

Keyword	Description
	OFF Ends the most recent setting of **AUTOPRED** and returns to any previous value, if set within the application. A second argument of **ALL** (for example, **AUTOPRED(FIRST,ALL)**) causes automatic dependencies to be made for all following operations, including ones with explicit dependencies. **Note:** 1. The **ADOP** statement for the operation named by **AUTOSUCC** must be included in the Batch Loader statements *after* any operations that you want it to be made an automatic successor to. 2. **AUTOPRED** is not allowed with **OPTIONS DBMODE(COPY)** or **OPTIONS DBMODE(UPDATE)**.
PREOPNO	The operation number of an internal predecessor operation to this operation.
PREJOBN	The job name of an internal predecessor operation to this operation.
PREWSID	The workstation name of an internal predecessor operation to this operation.
OPNO	The operation number for this operation, in the range 1 to 255. Each operation within an application must have a unique number.
WSID	Specified the workstation on which the operation will run.
DURATION	The estimated duration of this operation in minutes or seconds according to the **DURUNIT** value in **OPTIONS**. It must be an integer greater than 0. The maximum value is 99 hours 59 minutes 00 seconds. If you specify 99 hours 59 minutes 01 seconds, you do not receive an alert message if the actual duration is greater than the planned duration.
CLEANUP	The cleanup type for this operation: **A** Automatic. When the operation is ready to be submitted and the controller selects it for submissions, the controller automatically finds the cleanup actions to be taken and also inserts them as the first step in the JCL of the restarted job. Whenever the operation is started from the panels, the cleanup actions are shown to the user for confirmation, only if the AUTOMATIC CHECK OPC panel option is set to YES.

Table 123. Keywords for ADOP (continued)

Keyword	Description
	I Immediate. data set cleanup is immediately performed if the operation ends in error. The operation is treated as if it had the automatic option when it is rerun. **M** Manual. data set cleanup actions are deferred for the operation. They are performed when initiated manually from the panel. **N** None. No data set cleanup actions are performed. Formerly **ADOPCATM** with EQQYLTOP.
EXPJCL	Specifies if the scheduler uses the JCL extracted from the JES JCL sysout: **Y** Uses the fully expanded JCL. **N** Uses the JCL contained in the libraries of the scheduler. Formerly **ADOPEXPJCL** with EQQYLTOP.
CRITICAL	Specifies if the operation is to be considered critical: **W** The operation is considered eligible for WLM assistance, if late. **P** The operation is considered the target job of a critical path and eligible, with all the operations belonging to that critical path, for WLM assistance. **N** The operation is not to be considered critical. This is the default. For W and P, the scheduler automatically sends a request to WLM to promote the job or started task to the specified WLM service class, whenever the conditions of the specified assistance policy are met. Formerly **ADOPJOBCRT** with EQQYLTOP.
MH	Specifies how to set the value for the MH (Manually Hold) option of the operation when the operation is added to the AD database:

Table 123. Keywords for ADOP (continued)

Keyword	Description
	N Default. The MH option for the operation is set to N. **Y** The MH option for the operation is set to Y.
NOP	Specifies how to set the value for the NOP option of the operation when the operation is added to the AD database: **N** Default. The NOP option for the operation is set to N. **Y** The NOP option for the operation is set to Y.
POLICY	Specifies which policy is to be applied for WLM assistance, if the job is defined as critical: **blank** Default. WLM uses the policy specified in OPCOPTS. If no WLM policy is specified and the operation belongs to a critical path, the policy of the critical path target job is applied. **L** Long duration. The job is assisted if it runs beyond its estimated duration time. **D** Deadline. The job is assisted if it has not finished when its deadline time is reached. **S** Latest start time. The job is assisted if it is submitted after the latest start time. **C** Conditional. An algorithm calculates whether to apply the Deadline or the Latest start time policy. Formerly **ADOPJOBPOL** with EQQYLTOP.
USRSYS	Specifies if user sysout support is needed. If you specify Y, the data store logs user sysout.

Table 123. Keywords for ADOP (continued)

Keyword	Description
	Formerly **ADOPUSRSYS** with EQQYLTOP.
WLMCLASS	The name of the WLM service class to which late critical jobs are promoted. It can be an existing service class or a new service class created for this purpose. If you do not set this keyword and the operation belongs to a critical path, the WLM service class of the critical path target job is used. Formerly **ADOPWLMCLASS** with EQQYLTOP.
WTO	If you specify Y, a WTO message is issued if the operation passes its deadline and is in started status. Formerly **ADOPWTO** with EQQYLTOP.
AEC	For operations on automatic reporting workstations, HCL Workload Automation for Z does some processing when a job completes to determine whether the operation should be given error status or completed status. If you specify **AEC(N)**, HCL Workload Automation for Z does not check for errors and assigns to the operation the Completed status, regardless of any error reported by job tracking.
AJR	Jobs can be placed in HOLD status on the job queue by the event writer (an event writer option). Such jobs can either be released when all HCL Workload Automation for Z scheduling conditions are met, or be released immediately. **AJR(Y)** means that HCL Workload Automation for Z should control the scheduling. **AJR(N)** means that HCL Workload Automation for Z releases the job immediately. The automatic job release option is applicable only when the **HOLDJOB** keyword of the **EWTROPTS** is set to USER or YES.
AJSUB	Automatic job submission.
CLATE	Specify Y to cancel this operation if it is time-dependent and late. **Note:** HCL Workload Automation for Z never cancels a job that has already started running.
DESCR	A free-format description of the operation. It can be up to 24 characters.
DLDAY	Specifies the number of days, relative to the start of the application, when this operation must be completed. This must be an integer. 0 means that the deadline is on the same day as the occurrence input arrival.
DLTIME	Required if you have specified **DLDAY**. **DLTIME** specifies the time, on the day specified by the **DLDAY** keyword, by which this operation should be completed. This must be in the format ***hhmm***.

Table 123. Keywords for ADOP (continued)

Keyword	Description
FORM	If this operation is a printing operation, the printer form number that will appear on the daily plan and on ready lists. For printer workstations with automatic reporting, the printer class and form number let HCL Workload Automation for Z identify the different print operations belonging to a specific job. This can be up to 8 characters. **Note**: Operations not on Print workstations do not use this field, but it can still be set and read.
HIGHRC	If this operation is a z/OS job, the highest return code that should not be considered an error. If the job ends with this return code or less, the operation will be treated by HCL Workload Automation for Z as completed. This must be an integer less than 4096. If you leave out the parameter, HCL Workload Automation for Z takes the value you specified in the **JTOPTS** statement.
JOBCLASS	A single character that appears on workstation ready lists for information only. This must be the z/OS job class from the JCL.
JOBN	The job name for this operation, if applicable.
LIMFDBK	The default is the value you set in the job-tracking initialization statement **JTOPTS**. The feedback limit must be an integer in the range 100-999.
MONITOR	If you specify Y, the operation is monitored by an external monitor, for example by Tivoli Business Systems Manager.
PRTCLASS	If this operation is a printing operation, the printer SYSOUT class that appears on the daily plan and on ready lists. For printer workstations with automatic reporting, the printer class and form number let HCL Workload Automation for Z identify the different print operations belonging to a specific job. This is a single character, and must be specified for print operations.
PSNUM	The number of workstation parallel servers required by this operation. This must be an integer.
R1NUM	The amount of workstation type 1 resources required by this operation. This must be an integer.
R2NUM	The amount of workstation type 2 resources required by this operation. This must be an integer.
REROUTABLE	This keyword specifies the reroute option for the operation. The default is that the operation is reroutable if the **WSFAILURE** initialization statement **RESTART** keyword is set to **REROUTE**.

Table 123. Keywords for ADOP (continued)

Keyword	Description
	Y The operation is always reroutable. **N** The operation is never reroutable.
RESTARTABLE	This keyword specifies the restart option for the operation. The default is that the operation is restartable if the **WSFAILURE** initialization statement **RESTART** keyword is set to **RESTART**. **Y** The operation is always restartable. **N** The operation is never restartable.
SMOOTHING	The default is the value you set in the job-tracking initialization statement **JTOPTS**. The smoothing factor must be an integer in the range 0-999.
STARTDAY	Specifies the input arrival day of this operation, as a number of days offset from the occurrence input arrival day (0 means the same day). This must be an integer. Specifying a separate input arrival day and time for an operation can be useful if the operation is time-dependent and you want to ensure that it will not start before the specified time.
STARTTIME	Required if you have specified **STARTDAY**. It specifies the input arrival time of this operation, on the day specified with the **STARTDAY** keyword. This must be in the format ***hhmm***. If you specify **STARTTIME** but not **STARTDAY**, **STARTDAY** defaults to 0 (zero), which is the occurrence input arrival day.
TIME	If you specify Y, the job is made time-dependent.
USESAI	Determines whether the system automation information for the operation is used in the current plan. This keyword must be set to **Y** if the workstation has the system **AUTOMATION** option set to **Y**.
USEXTNAME	Determines whether or not the operation extended name is used in the current plan.
USEXTSE	Determines if the Scheduling Environment name of the operation is used in the current plan: **Y** Scheduling Environment name specified and stored in CP by the DP or dynamic addition process.

Table 123. Keywords for ADOP (continued)

Keyword	Description
	N Scheduling Environment name not specified or specified in AD and not stored in CP by the DP or dynamic addition process.

ADRE – Remote job information

Use the **ADRE** statement to define remote job information for an operation.

Note: The **ADRE** statement is available only starting from HCL Workload Automation for Z V8.6, or later.

Table 124. Keywords for ADRE

Keyword	Description
RECOMPL	Specifies if the shadow job status must be automatically set to complete, if the remote job does not exist: **Y** Sets the operation status to complete. **N** Sets the operation status to error.
REJOBNAME	Specifies the remote job name. It can be up to 40 characters and must be specified between single quotation marks. This parameter is required if the remote job runs on an HCL Workload Automation for Z remote engine.
REJSNAME	Specifies the name of the remote application (for HCL Workload Automation for Z) or of the remote job stream (for HCL Workload Automation for Z). It can be up to 16 characters and must be specified between single quotation marks.
REJSWS	Specifies the name of the remote job stream workstation. It can be up to 16 characters and must be specified between single quotation marks. This parameter is required if the remote job runs on an HCL Workload Automation for Z remote engine.
REOPNO	Specifies the remote operation number. It must be a number in the range 1-255. This parameter is required if the remote job runs on an HCL Workload Automation for Z remote engine.

ADRULE - Rule

The **ADRULE** statement defines a run cycle rule.

Note: Individual keywords on `ADRULE` statements cannot be specified individually when using `OPTIONS DBMODE(UPDATE)`, you must specify the entire rule. Set `ONLY` or `EVERY` as the first keyword.

Table 125. Keywords for ADRULE

Keyword	Description

`EVERY`
 	
`ONLY`
	Specifies the number (for `ONLY`) of the day or days to be selected. For `EVERY`, this specifies the interval of the series. The number is in the range 1 to 999.
	
	Use `EVERY` to specify a series of days. For example, `EVERY(2) DAY(DAY) FEBRUARY` specifies days 1, 3, 5, 7 etc. in February. The origin of the series is 1 unless you also specify `ORIGINSHIFT`.
	
	Use `ONLY` to specify the days precisely. For example, `ONLY(2) DAY(DAY) FEBRUARY` specifies only February 2.

`LAST`	Specifies the number (for `ONLY`) of the day or days to be selected. For `LAST(3)`, read "third last," so `ONLY LAST(3) DAY(DAY) JANUARY` specifies JANUARY 29. For `EVERY`, this specifies the interval of the series, starting from the end, so `EVERY LAST(3) DAY(DAY) JANUARY` specifies January 31, 28, 25 etc. The origin of the series is the last day unless you also specify `ORIGINSHIFT`. The number is in the range 1 to 999.
`DAY`	Specifies the day or days. You can abbreviate the names of the days to `MON` `TUE` `WED` `THU` `FRI` `SAT` `SUN` `WORK` and `FREE`.
`WEEK`	Specifies the week number or numbers. The number might range from 1 to 53. Week 1 is defined as the first week with at least 4 days of the new year. If you omit the number, the rule selects every week.
`MONTH`	Specifies the month or months. If you omit the name of the month, the rule selects every month. **Note:** You can abbreviate the names of the months to the first three characters. • `JANUARY` • `FEBRUARY` • `MARCH` • `APRIL` • `MAY` • `JUNE` • `JULY` • `AUGUST` • `SEPTEMBER` • `OCTOBER`

Table 125. Keywords for ADRULE (continued)

Keyword	Description
	• `NOVEMBER` • `DECEMBER`
`YEAR`	Used to specify that the cycle is a year, as in `EVERY(2) DAY(DAY) YEAR`, which gives January 1, January 3, January 5 etc. for each year. `ONLY LAST DAY(FRIDAY) YEAR` gives the last Friday in each year.
`PERIOD`	The name of a user-defined period, which must be in the period database. If you specify a period name such as `JULY`, which is the same name as a predefined cycle, HCL Workload Automation for Z looks for a user-defined period `JULY`, and gives an error if one does not exist.
`ORIGINSHIFT`	Specifies the origin shift in days. The number is in the range 1 to 999. Use this only with the `EVERY` keyword, when the origin is not the first (or, with `LAST`, the last) day of the cycle or period. If you specify `EVERY(4) DAY(DAY) MONTH ORIGINSHIFT(1)`, for example, the rule selects a series starting at the *second* day of each month, with an interval of 4 days: January 2, 6, 10 etc., then February 2, 6, 10 etc., for each month in the year.

ADRUN – Run cycle

Use the `ADRUN` statement to define a run cycle.

Table 126. Keywords for ADRUN

Keyword	Description
`SEQ`	When using `OPTIONS DBMODE(UPDATE)`, the SEQ keyword can be used to identify a specific run cycle. The value must be the numeric sequence number of the run cycle. **Note:** This must be the sequence as shown by Workload Automation Programming Language when the application is output. Do not rely on the sequence as displayed through the dialogs, because you may have sorted the display.
`NAME`	For rule-based run cycles, this is the name of the rule (up to 8 characters) and unique for this application. Specify `NAME`, and type `R` or `E`, if you are creating a rule-based run cycle.
`PERIOD`	For offset-based run cycles, this is the name of a cyclic or noncyclic period defined in the calendar database, or a run cycle group. Specify `PERIOD`, and type `N` or `X`, if you are creating an offset-based run cycle.
`JCLVTAB`	Specifies the JCL variable table to be used for the occurrences generated by this run cycle. For offset-based run cycles, this JCL variable table overrides the one specified

Table 126. Keywords for ADRUN (continued)

Keyword	Description
	for the period. For rule-based run cycles, specify the JCL variable table here, because HCL Workload Automation for Z ignores any JCL variable table associated with the period. The first character must be alphabetic, up to 16 characters. Formerly **ADRJTAB** with EQQYLTOP.
DESCR	A free-format description of the run cycle, up to 50 characters
DLDAY	Specifies the number of days from the input arrival day that the application should be completed in: 0 means that the deadline is on the same day as the input arrival day. This must be an integer.
DLTIME	Specifies the time on the deadline day that the application should be completed by, in the format ***hhmm***.
EIADAYS	Depending on the **TYPE** keyword, this keyword specifies one or more days relative to the start of the period when the application should be scheduled (**TYPE(N)**) or when it should not be scheduled (**TYPE(X)**). The numbers count from the *end* of the period; 1 is the *last* day.
IADAYS	Depending on the **TYPE** keyword, this keyword specifies one or more days relative to the start of the period when the application should be scheduled (**TYPE(N)**) or when it should not be scheduled (**TYPE(X)**). The numbers count from the start of the period; 1 is the first day.
IATIME	Specifies the time, in the format ***hhmm***, that the application is to arrive at the first workstation.
RPTEND	Specifies the repeat end time for the EVERY options, in the format ***hhmm***. It must be a time between the IA time of the run cycle and the calendar work day end time of the application.
RPTEVRY	Specifies the repeating frequency for the EVERY options, in the format ***hhmm***. It specifies that the application has an occurrence in the long-term plan every ***hhmm***, starting from the IA time to the repeat end time (**RPTENDT** keyword). If this keyword is not set, only the occurrence related to the IA time is added to the long-term plan.
RULE	Defines which free-day rule is in effect: **E** Count only work days when using the rule or offset. That is, free days are excluded. This option ensures that the scheduled day will always be a work day. This is the default for offset-based run cycles.

Table 126. Keywords for ADRUN (continued)

Keyword	Description

	`1`
	Count work days and free days when using the rule or offset. If this gives a free day, schedule the application on the closest work day *before* the free day.
	`2`
	Count work days and free days when using the rule or offset. If this gives a free day, schedule the application on the closest work day *after* the free day.
	`3`
	Count work days and free days when using the rule or offset. If this gives a free day, schedule the application *on* the free day. This is the default for rule-based run cycles.
	`4`
	Count work days and free days when using the rule or offset. If this gives a free day, *do not* schedule the application at all.

`SHIFT`	The number of days to shift the rule dates. This field is optional. It provides the means to define a run cycle relative to another, where the run cycle without the shifting offset is used to schedule an application in relation to which, using the same rule with a negative or positive shift of days, another application is scheduled. By default, the value is considered positive, and will make the new date be *after* the original run cycle group. Prefix the number with the minus sign (-) to shift the date before the run cycle group. **Note:** 1. The `SHSIGN` keyword is supported, but not necessary for Workload Automation Programming Language, because + and – can be used within `SHIFT`. 2. The `SHIFT` keyword is available only starting from HCL Workload Automation for Z V9.1, or later.
`SHTYPE`	Defines the type of days that are to be counted for the shift. `W` implies work days, while `D` implies any day in the calendar. This keyword is required if you used `SHIFT`.

Table 126. Keywords for ADRUN (continued)

Keyword	Description
	Note: The **SHTYPE** keyword is available only starting from HCL Workload Automation for Z V9.1, or later.
TYPE	Specify **R** (Regular) or **E** (Exclusion) *without* **IADAYS** or **EIADAYS** when you create a rule-based run cycle. You must specify the **NAME** keyword, and an **ADRULE** statement must follow this **ADRUN** statement. **R** means that the **ADRULE** statement specifies days when the application should be scheduled. **E** means that the **ADRULE** statement specifies days when the application should not be scheduled. Specify **N** (Normal) or **X** (Negative) together with **IADAYS** or **EIADAYS** when you create an offset-based run cycle. You must specify the **PERIOD** keyword. **N** means that the **IADAYS** and **EIADAYS** parameters define days when the application should be scheduled. **X** means that the **IADAYS** and **EIADAYS** parameters define days when the application should not be scheduled.
VALFROM	The start date of validity of this run cycle, in the format ***yymmdd***. See the note described for **VALTO**.
VALTO	The end date of validity of this run cycle, in the format ***yymmdd***. **Note:** HCL Workload Automation for Z interprets the ***yy*** part in the **VALTO** and **VALFROM** keywords as follows: **YY Year** **72 - 99** 1972 - 1999 **00 - 71** 2000 – 2071 **Note:** **VALTO** specifies the last day when the run cycle is valid. This is not the same as displayed in the ISPF panels, which uses “Out of effect” date that shows the first date when the run cycle is no longer valid. The “Out of effect” date is the day after the **VALTO** date, with the exception of the use of **HIGHDATE** (711231) as **VALTO**, which will show the same in the ISPF panels.

ADSAI – System Automation information (ADOPSAI segment)

Use the **ADOPSAI** statement to define System Automation Information for an operation.

Note: The `ADSAI` statement is available only starting from HCL Workload Automation for Z V8.3, or later.

Table 127. Keywords for ADSAI

Keyword	o LouDescription

`CT1`
`CT2`
`CT3`
`CT4`
	Free-format name for the command text of the operation. It can include blanks and special characters for a maximum of 255 characters.
	`CT1` to `CT4` specify the four lines of Command Text in accordance with how the field is broken up within the ISPF interface. `CT1` to `CT3` can be up to 64 characters in length, `CT4` can be up to 63 characters in length.
	Formerly `COMTEXT` with EQQYLTOP.

`AUTOFUNC`	Alphanumeric name for the automated function field of the operation. It can be up to 8 characters.
`SECELEM`	Free-format name for the security element of the operation. It can be up to 8 characters and include blanks and special characters.
`CI`	The completion information for the operation. It can be up to 64 characters. This parameter is positional. You can specify the following information, in the following order, separated by commas: 1. Maximum wait time, in the format *hh:mm:ss* 2. Maximum return code accepted as successful running 3. Name of an optional user-supplied completion checking routine To delete this information, set `CI` to blank. Formerly `COMPINFO` with EQQYLTOP.

ADSR – Special Resource reference

Use the `ADSR` statement to define a special resource requirement for an operation.

Table 128. Keywords for ADSR

Keyword	Description
`RESOURCE`	The name of the resource required by this operation. You can specify up to 44 characters.
`USAGE`	Defines whether the resource should be allocated as shared (`S`) or exclusive (`X`).
`KEEPONERR`	Defines whether the resource should be kept if the operation ends in error. If you do not specify this keyword, the default action is taken from the resource definition or `RESOPTS` statement.

Table 128. Keywords for ADSR (continued)

Keyword	Description
QUANTITY	The number of this resource that the operation needs, in the range 1 to 999999. If you do not specify this keyword, the operation takes all the resource exclusively, if **USAGE** is **X**, or prevents the exclusive use of any of this resource by any other operation, if **USAGE** is **S**.
ONCOMPLETE	Defines the value to which the global availability of the resource is reset at operation completion. If you do not specify this keyword, the default action is taken from the resource definition or **RESOPTS** statement.

ADSTART – Application common details

Use the **ADSTART** statement to define the common part of an Application Description.

Table 129. ADSTART keywords

Keyword	Description
ADID	Specifies the application name.
OWNER	Specifies the owner of the application, up to 16 characters.
STATUS	Specifies the status of the application: **A** Active **P** Pending **Note:** This keyword is enabled by **SPE(PEND)** for HCL Workload Automation for Z V8.2 when **SYNTAX(LEGACY)** is in use. Formerly **ADSTAT** with EQQYLTOP.
VALFROM	Specifies the start date of the validity period of the AD. Only the start date can be specified. The end of the validity period is set so that the time up to 31 December 2071 is covered, taking existing application descriptions into account. HCL Workload Automation for Zinterprets *yy* as follows: **YY Year** **72 - 99** 1972 - 1999 **00 - 71** 2000 - 2071 For **DBMODE(ADD)**, the **VALFROM** keyword sets the valid from date of the application. For all other **DBMODE** values, **VALFROM** identifies the application to **UPDATE** or **COPY**. The nearest match

Table 129. ADSTART keywords (continued)

Keyword	Description
	with the same value or earlier will be selected. To set a new valid from date for an existing application, use **NEW_VALFROM**. Formerly **ADVALFROM** with EQQYLTOP.
GROUPDEF	Specifies the name of the group definition used by this application to generate run cycle information. This keyword is valid only for an ADTYPE of A and should not be specified with CALENDAR. Formerly **GROUPID** with EQQYLTOP.
TYPE	Specifies the type of object: **A** Application definition **G** Group definition Formerly **ADTYPE** with EQQYLTOP.
CALENDAR	Specifies the name of the calendar to be used when run days are calculated for this application or group definition. Do not specify this keyword for applications that are members of a group.
DESCR	Specifies the description of the Application, up to 24 characters.
DLIMFDBK	The deadline limit for feedback. This keyword determines if the estimated deadline in the application description run cycle or operation is updated when an occurrence of the application reaches the complete status. The **DLIMFDBK** keyword value you set in this keyword is used only if no value is set in the application description. Feedback values are in the range 100 through 999, or 0 if the deadline must be always updated, regardless of the estimated and actual values. The feedback limits for *ADL* are calculated as follows: Lower limit = *ODL* * 100/*DLF* Upper limit = *ODL* * *DLF*/100 Where: ***ADL*** The actual deadline, considered as the elapsed minutes between the IA and the completion time of the occurrence or operation.

Table 129. ADSTART keywords (continued)

Keyword	Description
	ODL The old deadline estimated for the run cycle or operation (considered as offset in minutes from the IA) currently stored in the application description database. ***DLF*** The deadline limit for feedback. When the deadline feedback limit is set to 100, no new estimated deadline is stored in the application description database. If the actual deadline lies within the feedback limits, a smoothing factor is applied before the application description is updated. If the deadline feedback limit is set to 0, the application description database is always updated, unless: • The same limit is also specified in the application. • The smoothing factor does not allow the change. If the completion time occurs before the IA time, the deadline is not updated and a missed feedback record is generated. When the occurrence is generated, an identifier of the run cycle that generates the occurrence is stored in the occurrence record. This identifier is used to determine which run cycle must be updated. If the application description or the occurrence input arrival was modified, the run cycle might no longer be matchable. In this case, the deadline is not updated and a missed feedback record is generated.
`DSMOOTHING`	The smoothing factor. It determines how much the actual deadline influences the new deadline estimated for a run cycle or operation in the application description database. The smoothing factor is applied only if the actual deadline lies within the deadline feedback limits. The DSMOOTHING keyword value is used only if you did not set a smoothing factor in the application description. The smoothing factor is in the range 0 through 999. The value 0 means that the deadline is not updated, the value 100 means that the actual deadline replaces the existing estimated deadline. The new deadline is calculated as follows: *NDL* = *ODL* + ((*ADL* - *ODL*) * *DSF*/100) Where:

Table 129. ADSTART keywords (continued)

Keyword	Description
	NDL The new deadline estimated for the run cycle or operation (considered as offset in minutes from the IA) to be stored in the application description database. ***ODL*** The old deadline estimated for the run cycle or operation (considered as offset in minutes from the IA) currently stored in the application description database. ***ADL*** The actual deadline, considered as the elapsed minutes between the IA and the completion time of the occurrence or operation. ***DSF*** The smoothing factor.
AUTHGRP	Specifies the name of the application authority group to be used for additional authority checking, up to 8 characters. Formerly **GROUP** with EQQYLTOP.
ODESCR	Description of the application owner, up to 24 characters.
PRIORITY	The scheduling priority of the application. Must be a single digit in the range 1-9. This keyword is valid only for an **ADTYPE** of A.

When using **ACTION(SUBMIT)**, the following additional keywords are available:

Table 130. Additional ADSTART keywords when using ACTION(SUBMIT)

Keyword	Description
IA	Input Arrival in the format *YYMMDDHHMM*.
DEADLINE	Deadline in the format *YYMMDDHHMM*.
JCLVTAB	Name of a JCL Variable Table to attach to this occurrence.
SUFFIX	Allows an HCL Workload Automation for Z variable to be appended to the Application Name. For more details, see Suffixing on page 248. **Note:** If you use**SUFFIX** unique occurrences could be created in the plan; but if another application with the same name exists, this is never run and the JCL for

Table 130. Additional ADSTART keywords when using ACTION(SUBMIT) (continued)

Keyword	Description
	these occurrences remains in the JS file indefinitely. Consider using the HCL Workload Automation for Z sample EQQPIFJX to maintain the JCL repository

ADUSF – User Field (ADUSRF segment)

Use the `ADUSF` statement to define a user field for an operation.

Note: The `ADUSF` statement is available only starting from HCL Workload Automation for Z V8.5.1 SPE, or later.

Table 131. Keywords for ADUSF

Keyword	Description
`UFNAME`	User Field name, up to 16 characters long.
`UFVALUE`	User Field value, up to 54 characters long.

ADVDD – Variable values for operation

Use the `ADVDD` statement to define variable values applicable to the operation instances that are generated by the specified run cycles.

Table 132. Keywords for ADVDD

Keyword	Description

`CRIT`	Specifies a variable job critical indicator to be associated with a variable run cycle:
	`N`
	Not eligible for WLM assistance.
	`P`
	Critical Path target.
	`Y`
	Eligible for WLM assistance.

`DLDAY`	Relative deadline day (00-99).
`DLTIME`	Deadline time (*HHMM*).
`DURATION`	Estimated duration for this run (in seconds).
`MH`	Specifies a variable `MH` (Manually Hold) option to be associated with a variable run cycle:

Table 132. Keywords for ADVDD (continued)

Keyword	Description
	Y The **MH** option is set to **Y** in the AD database that is associated with the related variable duration and deadline run cycle. **N** The **MH** option is set to **YN** in the AD database that is associated with the related variable duration and deadline run cycle.
NOP	Specifies a variable **NOP** option to be associated with a variable run cycle: **Y** The NOP option is set to Y in the AD database that is associated with the related variable duration and deadline run cycle. **N** The NOP option is set to N in the AD database that is associated with the related variable duration and deadline run cycle.
RCGROUP	Run cycle or run cycle group name to which you associate the variable duration and deadline.
VACTIONACT	Specifies the action to be taken if the operation has not yet started when the specified day and time is reached. If you specify an action, **VACTIONDD** and **VACTIONDT** are required. Allowed values are: **A** Only an alert message is issued. **C** The operation is set to Complete, if its status allows it. Otherwise it is NOPed. **E** The operation is set to Error with OLAT, if its status allows it. Otherwise, this setting is postponed at the time when the status allows it. **N** The operation and all its internal successors are NOPed, if their status allows NOPing. Otherwise it is ignored.
VACTIONBASE	Always **F**. It means that the input arrival time of the operation is used as the base date to apply the offset.

Table 132. Keywords for ADVDD (continued)

Keyword	Description
VACTIONDD	Specifies the day offset, related to the application start day, used to check if the operation has not yet started. If the operation has not started, the action that you set is taken. It can be a value from 0 to 99 (where 0 means the start day, 1 means one working day after the start day, and so on). This overrides the default value.
VACTIONDIR	Always **A** (After). It means that the day offset and time are always added, and never subtracted, to the base date.
VACTIONDT	Required if you have set **VACTIONDD**. Specifies the time, related to the application start day, used to check if the operation has not yet started. If the operation has not started, the action that you set is taken. This check is made only to operations with status Ready. This value is specified in the format *HH.MM*. It overrides the default value.
VALERTBASE	Always **F**. It means that the input arrival time of the operation is used as the base date to apply the offset.
VALERTDD	Specifies the day offset, related to the application start day, used to check if the operation has not yet started. If the operation has not started, an alert message is issued. This check is applied only to operations with status Ready. It can be a value from 0 to 99 (where 0 means the start day, 1 means one working day after the start day, and so on). This overrides the default value.
VALERTDIR	Always **A** (After). It means that the day offset and time are always added, and never subtracted, to the base date.
VALERTDT	Required if you have set **VALERTDD**. Specifies the time, related to the application start day, used to check if the operation has not yet started. If the operation has not started, an alert message is issued. This value is specified in the format *HH.MM*. It overrides the default value.
VDLACT	Specifies the action to be taken if the operation is still not completed when the deadline day and time is reached. Blank means no action. If you specify an action, an alert message is always issued. Allowed values are: **A** Only an alert message is issued.

Table 132. Keywords for ADVDD (continued)

Keyword	Description
	C The operation is set to Complete, if its status allows it. Otherwise it is NOPed. **E** The operation is set to Error with ODEA, if its status allows it. Otherwise it is ignored. **N** The operation and all its internal successors are NOPed, if their status allows NOPing. Otherwise it is ignored.

ADXIV – External dependency interval

Use the **ADXIV** statement to define the absolute or relative interval specified with the **A** or **R** value in the **ADDEP PRECSEL** parameter.

You can use only one **ADXIV** for each **ADDEP** statement. The statement must be nested within the **ADDEP** to which it refers.

Note: The **ADXIV** statement is available only starting from HCL Workload Automation for Z V9.1, or later.

Table 133. Keywords for ADXIV

Keyword	Description
ADID	The application name of the external predecessor to which the interval applies.
FROMDAYS	The start of the absolute interval in days. The allowed range is 0-7.
FROMHH	The start of the absolute interval in the *HH* format. The allowed range is 00-24. To be specified together with **FROMMM**. For example, if the absolute interval starts at 10:30 of the day before the input arrival time of the successor, it is defined by **FROMHH(10) FROMMM(30) FROMDAYS(1) FROMWHEN(B)**
FROMHHH	The start of the relative interval in hours. The format is *HHH* and the allowed range is 0-167. To be specified together with **FROMMM**.
FROMMM	The minutes fraction of the start of the relative or absolute interval.
FROMWHEN	Specifies if the start of the relative or absolute interval is before (**B**) or after (**A**) the input arrival time of the successor. For relative intervals only, you can choose to make the interval start at an indefinite time in the plan (in this case the mechanism used is similar to

Table 133. Keywords for ADXIV (continued)

Keyword	Description
	that of the closest preceding predecessor). To do this, do not specify this parameter, nor any of the other **FROM..** ones.
OPNO	The operation number of the external predecessor to which the interval applies.
TODAYS	The end of the absolute interval in days. The allowed range is 0-7.
TOHH	The end of the absolute interval in the *HH* format. The allowed range is 00-24. To be specified together with **TOMM**. For example, if the absolute interval ends at 12:30 two days after the input arrival time of the successor, it is defined by **TOHH(12) TOMM(30) TODAYS(2) TOWHEN(A)**
TOHHH	The end of the relative interval in hours. The format is *HHH* and the allowed range is 0-167. To be specified together with **TOMM**.
TOMM	The minutes fraction of the end of the relative or absolute interval.
TOWHEN	Specifies if the end of the relative or absolute interval is before (**B**) or after (**A**) the input arrival time of the successor.
TYPE	The interval type: **A** Absolute interval. Must be defined by the following parameters: **FROMWHEN, FROMHH, FROMMM, FROMDAYS,** **TOWHEN, TOHH, TOMM, TODAYS.** **R** Relative interval. Must be defined by the following parameters: **[FROMWHEN, FROMHHH, FROMMM,]** **TOWHEN, TOHHH, TOMM.**
WSID	The name of the workstation running the external predecessor to which the interval applies.

AWSCL – All Workstations Closed record

The **AWSCL** record is a single segment record with the following structure.

There is one **AWSCL** record for every day that all workstations are to be closed.

AWCSTART – All workstations closed

Use the **AWCSTART** statement to define an entire All Workstations Closed record. A statement begins with **AWCSTART** and can be followed by any of the following additional arguments.

Table 134. Keywords for AWCSTART

Keyword	Description
DATE	Specifies a date, in the *YYMMDD* format, when all workstations are closed.
DESCR	Specifies text of up to 30 characters that describes why all workstations are closed.
FROM	Specifies the start of the period when the workstations are closed, in the *HHMM* format.
TO	Specifies the end of the period when the workstations are closed, in the *HHMM* format.

CL – Calendar record

The CL record is a multi segment record, with the following structure.

```
CLCOM -+- Common segment (1 per CL)
       |
       += CLSD = A specific date (many per CL)
       |
       += CLWD = A day of the week (up to 7 per CL)
```

Note: Some segments have Batch Loader statements with slightly different names:

- CLCOM has a Batch Loader statement of **CLSTART**
- CLSD has a Batch Loader statement of **CLDATE**
- CLWD has a Batch Loader statement of **CLDAY**

CLSTART – Calendar common details (CLCOM segment)

Use the **CLSTART** statement to define the common part of a Calendar.

Table 135. Keywords for CLSTART

Keyword	Description

DROP	Specifies a date in the format *YYMMDD*, or + or – a number of days, which ensures that any calendar date preceding th date specified is dropped from the Calendar.
	
	For example, to drop any dates over a year old issue the following command:
	`OPTIONS DBMODE(UPDATE)`
	`CLSTART CALENDAR(BATCHCAL) DROP(-365)`

Table 135. Keywords for CLSTART (continued)

Keyword	Description
	Note: Using `DROP` when you have defined cyclic periods of type `W` (count only working days), results in interval dates shifting if you drop the dates that are after the period origin date.
`CALENDAR`	Specifies the name of the Calendar.
`DESCR`	Specifies the description of the Calendar, up to 30 characters.
`SHIFT`	Specifies a value, in the *HHMM* format, to define the end of a work day that comes immediately before a free day in the calendar. The default value is 0000.

CLDATE – Specific date (CLSD segment)

Use the `CLDATE` statement to define a specific date.

Table 136. Keywords for CLDATE

Keyword	Description
`DATE`	Specifies a date in the *`YYMMDD`* format, optionally followed by days of the week on which this date must fall to be set as a specific date.
`DESCR`	Specifies a description of the specific date, up to 30 characters.

`STATUS`	Specifies the status of the specific date (which override the status assigned to a day of the week):
	`F`
	A free day
	`W`
	A working day

In the following example, you create a free day for 18 May 2014:

```
CLDATE DATE(140518) STATUS(F)
```

In the following example, you create a free day if 24 December is Monday:

```
VARDATE CL_BRIDGE ONLY(24) MONTH(DEC)
CLDATE DATE(!CL_BRIDGE,MONDAY)
```

In the following example, you create a free day if 25 December is a weekday:

```
VARDATE CL_XMAS ONLY(25) MONTH(DEC)
CLDATE DATE(!CL_XMAS,MONDAY,TUESDAY,WEDNESDAY,THURSDAY,FRIDAY)
```

CLDAY – Day of the week (CLWD segment)

Use the **CLDAY** statement to define a day of the week.

Table 137. Keywords for CLDAY

Keyword	Description
DAY	Specifies a day of the week: • **MONDAY** • **TUESDAY** • **WEDNESDAY** • **THURSDAY** • **FRIDAY** • **SATURDAY** • **SUNDAY**
DESCR	Specifies a description of the day of the week, up to 30 characters.
STATUS	Specifies the status of the day of the week (which can be overridden by the status assigned to a specific date): **F** A free day **W** A working day

ETT – Event Trigger Record

The ETT record is a single segment structure.

ETTSTART – Trigger definition

Use the **ETTSTART** statement to define an entire Event Trigger record.

A statement begins with **ETTSTART** and can be followed by the following additional arguments.

Table 138. Keywords for ETTSTART

Keyword	Description
ETTNAME	Specifies the name of a job or a special resource, up to 44 characters. You can also use the wildcards asterisk (*) and percent sign (%) to specify a generic name.
ETTTYPE	Specifies the type of event that you want to cause a dynamic update of the current plan. Required.

Table 138. Keywords for ETTSTART (continued)

Keyword	Description
	J A job reader event is the triggering event. **R** A special resource availability event is the triggering event.
ADID	Specifies the name of an application defined in the application description (AD) database that you want adding to the current plan when the event occurs.
AS	Availability status switch indicator. Valid only with **ETTTYPE(R)**. Indicates if **ETT** adds an occurrence only if there is a true availability status switch for a special resource from status Available=N to Available=Y, or if **ETT** adds an occurrence each time the availability status is set to Available=Y (regardless of the previous status of the special resource). For **ETTTYPE(J)** this field must have the value **N** or blank. **Y** **ETT** adds an occurrence only when there is a true availability status switch from status Available=N to Available=Y. **N** **ETT** adds an occurrence each time the availability status is set to Available=Y.
DR	Specifies the dependency resolution, which determines if external dependencies should be added when occurrences are added to the current plan. **Y** External dependencies are to be added. **N** External dependencies are not to be added. **P** Add only predecessor dependencies. **S** Add only successor dependencies.
JR	Specifies job-name replace, which is valid only with **ETTTYPE(J)**. This determines if the job name of the first operation in the associated application should be replaced with the triggering job.

Table 138. Keywords for ETTSTART (continued)

Keyword	Description
	Y The name of the first operation is replaced by the job name of the triggering job. **N** The application is added unchanged.
MATCHTYP	This argument is needed when spaces, asterisks (*) or percent signs (%) form part of the name of the object, to indicate how to treat the special characters. **EXA** Use the name exactly as typed, treating asterisks (*) and percent signs (%) as characters to include in the name. **PFX** Treat as a prefix match, to find an item beginning with what was typed but treating asterisks (*) and percent signs (%) as characters included in the name. **SFX** Treat as a suffix match, to find an item ending with what was typed but treating asterisks (*) and percent signs (%) as characters included in the name.

JB – Ad-hoc in the current plan

The JB Batch Loader statements do not map to a record within HCL Workload Automation for Z, instead this is pseudo batch loader generated as output from the **LISTJOB** command.

The **LISTJOB** command extracts application and job details from the database for a single job, but can be made to output this detail in the form of a **JBSTART** command. So where the **ADDJOB** command will extract job details and then submit the job directly to the plan with the extracted attributes, by using **LISTJOB** to create the **JBSTART** command this allows you to manipulate the extracted attributes before submitting the job to the current plan.

For example, issue the following **LISTJOB** command:

```
LISTJOB JOB005 ADID(ALTIF) OUTPUT(OUTDATA) STYLE(LOADER) DISPLAY(N)
                 DETAIL(SUCC)
```

to create the following **JB** command:

```
JBSTART ADID(ALTIF) ADVALFROM(141202) DESCR('ALTIF test') OWNER(TWS)
JBOP WSID(CPU1) OPNO(005) JOBN(JOB005) DURATION(1)
JBPRE-INT WSLCCMD1 PREADID(ALTIF) PREWSID(CMD1) PREOPNO(001)
JBSUC-INT JOB010 SUCADID(ALTIF) SUCWSID(CPU1) SUCOPNO(010)
```

The JB batch loader statements are similar to the equivalent AD statements, with the difference that the **AD** prefix is replaced with **JB**. For a complete description of the **AD** syntax, see AD – Application definition record on page 250.

JBSTART – Application details

The **JBSTART** statement describes the application level information, based on the **ADSTART** statement and using all the same keywords, except for **ADID**. **ADID** is ignored and replaced with a name formed by **PFX**, the job name, and **SFX**.

In addition to the JB versions of the **ADSTART** keywords, the **JBSTART** command also supports the following keywords from the **ADDJOB** command:

COMPSUCC

Specifies how to manage completed successors.

CPDEPR

Specifies whether to resolve dependencies.

CRITERIA

Specifies the dependency resolution criteria.

DEPTIME

Specifies the input arrival time to consider for resolving dependencies.

EXTLINK

Specifies which elements to use to identify an external predecessor.

HOLD

Specifies whether to hold the job that was added.

IA

Specifies the Input Arrival with which to submit the occurrence.

INTLINK

Specifies which elements to use to identify an internal predecessor.

JCL

Provides external JCL to the submitted job.

PFX

Specifies the prefix to use for the application name.

SFX

Specifies the suffix to use for the application name.

UPDATE

Determines whether to actually perform the updates, or just trial the process to see what actions would have been taken.

For detailed information about these keywords, see ADDJOB – Add job to the current plan on page 219.

JBCIV and JBXIV – External dependency selection criteria

These are generated by **LISTJOB** for information purposes only. Dependency selection criteria is handled by the **CRITERIA**, **DEPTIME**, **INTLINK**, and **EXTLINK** keywords of the **JBSTART** statement. They are ignored when the **JBSTART** command is run.

JBCNC and JBCNS – Conditional dependencies

These are generated by **LISTJOB** for information purposes only. **JBSTART** is used to dynamically submit jobs on demand, therefore the decision as whether the job is to be run must be made before running the **JBSTART** statement. They are ignored when the **JBSTART** command is run.

JBDEP, JBPRE and JBSUC – Dependencies

JBDEP is not a supported statement for **JBSTART**, because **JBSTART** allows dependencies in both directions. **JBDEP** is replaced by **JBPRE** to define predecessors and by **JBSUC** to define successors.

Both **JBPRE** and **JBSUC** have a suffix of either **-INT** or **-EXT** to define whether this was originally an internal or external dependency, as the selection criteria can be managed differently for each by using the **INTLINK** and **EXTLINK** keywords of the **JBSTART** statements respectively.

For example, **JBPRE-INT** describes an internal predecessor, and **JBSUC-EXT** describes an external successor.

Both **JBPRE** and **JBSUC** have a positional first argument of the job name of the dependency. If the operation being referred to has no job name, then **-BLANK** is presented by **LISTJOB**. If the operation being referred to does not exist **-MISSING** is presented by **LISTJOB**.

For **JBPRE**, the remaining keywords are identical to **ADDEP**. For **JBSUC** the keywords are similar to **ADDEP**, with the difference that the **PRE** prefix is replaced by **SUC**.

JBRUN and JBRULE - Run cycle and rule

LISTJOB does not generate a **JBRUN** or **JBRULE** statement, because it only lists application level information, then the complete specification of the job with the **JBOP** statement and statements for any child segments of the operation being listed.

JCLV – JCL Variable Table record

The **JCLV** record is a multi segment record, with the following structure.

```
JCLVCOM -+- Common segment (1 per JCLV)
         |
         += JCLVVAR =+= Variable (many per JCLV)
                     |
                     += JCLVDEP = Dependency
                                  (many per Variable)
```

Note: Some segments have Batch Loader statements with slightly different names. `JCLVCOM` has a Batch Loader statement of `JCLVSTART`.

JCLVSTART – Variable table common details (JCLVCOM segment)

Use the `JCLVSTART` statement to define the common part of a JCL Variable table.

Table 139. Keywords for JCLVSTART

Keyword	Description
`JCLVTAB`	Specifies the name of the variable table, up to 16 characters.
`OWNER`	Specifies the owner ID of the variable table, up to 16 characters.
`DESCR`	Specified the description of the variable table, up to 24 characters.

JCLVVAR – Variable details

Use the `JCLVVAR` statement to define a variable.

Table 140. Keywords for JCLVVAR

Keyword	Description
`VARNAME`	Specifies the name of the variable.

`COMP`	Specifies the way in which the `LENGTH` field is used to validate the length of the replaced value. This field is optional, unless you specify a length value.
	
	For example, **EQ 12** means that the replaced value must be exactly 12 characters long.
	
	`EQ or =`
	Equal to
	
	`GT or >`
	Greater than
	
	`LT` **or <**
	Less than
	
	`GE` **or >=**
	Greater than or equal to
	
	`LE` **or <=**
	Less than or equal to
	
	`NE` **or ¬=**
	Not equal to

Table 140. Keywords for JCLVVAR (continued)

Keyword	Description
	NG or **¬>** Not greater than **NL** or **¬<** Not less than
DEFAULT	Specifies the value the variable must have if another values is not explicitly assigned, up to 44 characters.
DEPENDENCY	Specifies the name of a variable on which the value of this variable is dependent. This requires **JCLVDEP** statements to be defined.
DESCR	Specifies a description of, for example, the purpose and use of the variable, up to 20 characters.
DIALOG1	Specifies line 1 (of 4) of information for the preparer. Each line can be up to 51 characters long.
DIALOG2	Specifies line 2 (of 4) of information for the preparer. Each line can be up to 51 characters long.
DIALOG3	Specifies line 3 (of 4) of information for the preparer. Each line can be up to 51 characters long.
DIALOG4	Specifies line 4 (of 4) of information for the preparer. Each line can be up to 51 characters long.
EXIT	Specifies a substitution exit to be used to resolve the variable.
LENGTH	Use this field together with **COMP** to validate the length of the value substituted by the job preparer. This is optional, unless you specify a length comparison. For example, **LT** **11** means that the replaced value must be less than 11 characters.
NUMERIC	Specifies if the data must be numeric: **Y** Data must be numeric. **N** Data can have any values. Specify this filed only if **TYPE** is set to **LIST** or **RANGE**.
PICT	Specifies the character pattern used together with the verification **TYPE** **PICT**. Optional, unless **TYPE** is **PICT**.

Table 140. Keywords for JCLVVAR (continued)

Keyword	Description
	For example, **CC99** means that the first two characters can have any value and characters 3 and 4 must be in the range 0-9. Use the characters **C** (any value), **A** (alphabetic), **N** or **9** (0-9), **X** (0-9, A-F, a-f).
POS	Specifies the starting position, in a range from 1 to 80, for a value assigned to an in-stream positional variable, or 0 if the variable is not positional. The default is 0.
REQD	Specifies whether the variable can be assigned a blank value or not: **N** Not required. The variable can be assigned a blank value (this is the default). **Y** Required. The variable cannot be assigned a blank value.
SETUP	Specifies how variable substitution is managed: **N** No interaction. The variable is replaced at submit time. **P** Promptable. An interaction with the preparer takes place at job setup. **Y** No interaction. The variable is replaced at setup time if a preparation step if present, otherwise at submit time.
SUBPOS	Position in string to start substring selection for dependent variables.
SUBLEN	Length of substring selection for dependent variables.
TYPE	Specifies the type of verification to be carried out on the data specified by the preparer. This follows the same format rules as the ISPF **VER** statement. Optional. Valid types are: • **ALPHA** • **BIT** • **DSNAME** • **ENUM** • **HEX** • **LIST** • **NAME** • **NUM**

Table 140. Keywords for JCLVVAR (continued)

Keyword	Description
	• `PICT` • `RANGE`
`UCASE`	Specify `Y` to get characters a-z translated to upper case. Specify `N` to keep mixed case (the default is `N`).
`VALID1`	Line 1 of `LIST/RANGE` values for use in validating the variable (up to 51 characters). For type `LIST`, specify one or more values separated by commas. For type `RANGE`, specify pairs of values separated by commas. For example, `1,3,5,7` allows values in the ranges 1 to 3 and 5 to 7. Specify this keyword only if the `TYPE` is `LIST` or `RANGE`.
`VALID2`	Line 2 of `LIST/RANGE` values for use in validating the variable (up to 51 characters). For type `LIST`, specify one or more values separated by commas. For type `RANGE`, specify pairs of values separated by commas. For example, `1,3,5,7` allows values in the ranges 1 to 3 and 5 to 7. Specify this keyword only if the `TYPE` is `LIST` or `RANGE`.

JCLVDEP – Dependent value pair

Use the `JCLVDEP` statement to define a variable dependency.

Table 141. Keywords for JCLVDEP

Keyword	Description
`INDEPENDENT`	Specifies a value, 1-44 characters. If the independent variable has this value, the dependent variable is assigned the paired value as specified in `DEPENDENT`. This argument is case sensitive, unless the independent variable has uppercase translation.
`DEPENDENT`	Specifies a value that is assigned if the independent variable matches the value specified in `INDEPENDENT`. It can be 1-44 characters or left blank.

JS – Current Plan JCL record

The JS record is a multi segment record, with the following structure.

```
JSCOM -+- Common segment (1 per JS)
       |
       += JST = JCL text line (many per JS)
```

Note:

1. Some segments have batch loader statements with slightly different names. `JSCOM` has a batch loader statement of `JSSTART`.
2. The Workload Automation Programming Language interpretation of the JS record is slightly different from the PIF specification in which interval dates are store in the field `JST` within the Common Segment. Within Workload Automation Programming Language, the Common Segment includes `JST` as an unresolved field, if requested in an `OUTPUT` statement, but will also decode the `JST` into separate `JST` segments for each line of JCL. This makes for more flexible processing in batch loader.

JSSTART behaviour

Current plan JCL records (JS) are related to Current plan operation records (CPOP) in using the same information as the key: Application Name, Input Arrival, and Operation number.

Even if an Operation exists, this does not mean that an equivalent JS record exists. A JS record is created by HCL Workload Automation for Z when job setup runs for an operation, or if no setup occurs at Job Submission. A JS record can also be created by a user editing JCL through the Current Plan ahead of submission and saving the results. Equally after submission someone could delete a JS record.

Because of this behaviour, `JSSTART` searches for the Operation, rather than just the JCL, to obtain all of the operation details, then it determines if a JS record exists so if `DBMODE(ADD)` is used the appropriate actions can be taken. If a matching operation does not exist, the `JSSTART` command fails.

Because the information in the common segment is inherited from the CPOP record and the JCL is specified as a whole, you cannot use `OPTIONS DBMODE(UPDATE)` and `OPTIONS DBMODE(COPY)` together with `JSSTART`.

Note: If you do not specify `ADID`, `IA`, and `OPNO`, the `JSSTART` statement uses the other keywords to find matching jobs and update all the operations matching your criteria.

JSSTART – Current Plan JCL entry (JSCOM segment)

Use the `JSSTART` statement to define the common part of a JCL entry.

Table 142. Keywords for JSSTART

Keyword	Description
`ADID`	The Application name of the operation to contain the JCL.
`IA`	The Input Arrival Time of the operation to contain the JCL.

Table 142. Keywords for JSSTART (continued)

Keyword	Description
OPNO	The Operation number of the operation to contain the JCL.
JOBNAME	The Job name of the operation to contain the JCL.
WSNAME	The Workstation name of the operation to contain the JCL.
STATUS	The status of the operation to contain the JCL. **Note:** This is *not* the status of the JCL (JSST), because this argument might contain some different values from standard Operation Statuses.
DLM	JCL delimiter. Any character string used to terminate in-stream JCL that follows the **JSSTART** statement. The presence of a **DLM** keyword indicates the immediate following line will be a line of JCL, and any subsequent lines until the character string specified in **DLM** is encountered on a line by itself.
MEMBER	Names a member in the EQQJSPDS DD statement that contains the JCL. The JCL is copied from there into EQQTEMP for addition into the JS record. You can specify a DD statement different from EQQJSPDS, for example MYJCL, by setting **OPTIONS JSFILE(MYJCL)**. You can specify a DD statement different from EQQTEMP by setting **OPTIONS TEMPFILE**. **Note:** The **MEMBER** keyword cannot be used with the EQQWAPL load module.

JST – Line of JCL (JST field of JSCOM)

If **DLM** or **MEMBER** was not used, a **JST** statement can be used to define each line of JCL. The JCL must be coded on the line following the **JST** keyword.

OI – Operator Instruction record

The OI record is a multi segment record, with the following structure.

```
OICOM -+- Common segment (1 per OI)
       |
       += OIT = Line of text (many per OI)
```

Note:

1. Some segments have Batch Loader statements with slightly different names. **OICOM** has a Batch Loader statement of **OISTART**.
2. The Workload Automation Programming Language interpretation of the OI record is slightly different from the PIF specification in which interval dates are store in the field **OIT** within the Common Segment. Within

Workload Automation Programming Language, the Common Segment includes **OIT** as an unresolved field, if requested in an **OUTPUT** statement, but will also decode the **OIT** into separate **OIT** segments for each line of text. This makes for more flexible processing in Batch Loader.

3. **DBMODE(UPDATE)** cannot be used to update individual lines of text.

OISTART – Period common details (PRCOM segment)

Use the **OISTART** statement to define the common part of a Calendar.

Table 143. Keywords for OISTART

Keyword	Description
ADID	The identifier of the application. If you use DBCS characters, they must be entered as a quoted string started by a shift-out and ended by a shift-in. You must specify **ADID**.
JOBN	The job name of the operation that this OI is for.
DLM	Text delimiter. Any character string used to terminate instream text that follows the **OISTART** statement. The presence of a **DLM** keyword indicates the immediate following line will be a line of operator instruction text, and any subsequent lines until the character string specified in **DLM** is encountered on a line by itself. If **DLM** is used no **OIT** statements are needed.
MEMBER	You can set the **MEMBER** keyword only if the **OI** text is located in a partitioned data set defined by the EQQOIPDS DD statement. It must be free format in columns 1 to 72. If **MEMBER** is used, no **OIT** statements are needed. **Note:** The **MEMBER** keyword cannot be used with the EQQWAPL load module.
OPNO	The operation number of the operation that this OI is for. You must specify **OPNO**.
VALFROMD	The start date of validity of this OI. You must specify this in the format **yymmdd**. See the notes about **VAL*** keywords. The default is the current date if other **VAL*** keywords are set.
VALFROMT	The start time of validity of this OI. You must specify this in the format **hhmm**. See the notes about **VAL*** keywords. The default is the current time if other **VAL*** keywords are set.
VALTOD	The end date of validity of this OI. You must specify this in the format

Table 143. Keywords for OISTART (continued)

Keyword	Description
	yymmdd. See the notes about **VAL*** keywords. The default is 711231 if other **VAL*** keywords are set.
VALTOT	The end time of validity of this OI. You must specify this in the format **hhmm**. See the notes about **VAL*** keywords. The default is 2359 if other **VAL*** keywords are set.

Note:

1. If you do not specify any of these **VAL*** keywords, HCL Workload Automation for Z assumes that the operator instruction is permanent.
2. **NEW_VALTOD** and **NEW_VALTOT** are not valid for temporary Operator Instructions. They cannot have their validity range changed by **NEW** keywords. To alter the validity of temporary Operator Instructions they must be deleted and recreated.

The following example shows how to use **DLM** to specify operator instructions:

```
OISTART ADID('TESTGROUP02     ') OPNO(001) DLM(-TEXT-END-)
If the job fails you need to wake Doug up
Don't worry what the time is, he lives for this kind of thing
Just to be on the safe side, wake him up if the job works as well
-TEXT-END-
```

OIT – Line of Text (OIT field of OICOM)

If **DLM** or **MEMBER** was not used, a **OIT** statement can be used to define each line of text. The text must be coded on the same line following the **OIT** keyword.

The following example shows how to use **OIT** to specify operator instructions:

```
OISTART ADID('TESTGROUP02     ') OPNO(001)
OIT 'If the job fails you need to wake Doug up'
OIT 'Don't worry what the time is, he lives for this kind of thing'
OIT 'Just to be on the safe side, wake him up if the job works as well'
```

PR – Period record

The PR record is a multi segment record, with the following structure.

```
PRCOM -+- Common segment (1 per PR)
       |
       += PRDATE = A date interval (many per PR)
```

Note:

1. Some segments have Batch Loader statements with slightly different names. `PRCOM` has a Batch Loader statement of `PRSTART`.
2. The Workload Automation Programming Language interpretation of the PR record is slightly different from the PIF specification in which interval dates are store in the field `PRTAB` within the Common Segment. Within Workload Automation Programming Language, the Common Segment includes `PRTAB` as an unresolved field, if requested in an `OUTPUT` statement, but will also decode the `PRTAB` into separate `PRDATE` segments for start and end date pairs. This makes for more flexible processing in Batch Loader.

PRSTART – Period common details (PRCOM segment)

Use the `PRSTART` statement to defines the common part of a Period.

Table 144. Keywords for PRSTART

Keyword	Description

`DROP`	Specifies a date in the format *YYMMDD*, or + or – a number of days, that ensures that any Period interval preceding that date will be dropped from the Period.
	
	Note: For intervals with start and end dates, the end date will be compared with the DROP value, otherwise the start date will be used.
	
	For example, to drop any dates over a year old:
	`OPTIONS DBMODE(UPDATE)`
	`PRSTART PERIOD(YEAREND) DROP(-365)`

`PERIOD`	Specifies the period name.

`PRTYPE`	Specifies the type of period:
	`A`
	All-days cyclic period. A period that includes both work days and free days (requires `INTERVAL` to be set).
	`W`
	Work-days-only cyclic period. A cyclic period that includes only work days (requires `INTERVAL` to be set).
	`N`
	Non-cyclic period (requires `PRDATE` statements, `ADID` or `DATELIST` to be specified).

`DESCR`	Specified the description of the period, up to 30 characters.

Table 144. Keywords for PRSTART (continued)

Keyword	Description
INTERVAL	Specify a value from 1 to 999 for cyclic periods.
TABLE	Specifies the name of a JCL variable table to be associated with the occurrence for Period based run cycles.

PRDATE – Interval (PRTAB field of PRCOM)

Use the **PRDATE** statement to define an interval for a non-cyclic period.

Table 145. Keywords for PRDATE

Keyword	Description
START	Specifies the start date of an interval in the ***YYMMDD*** format.
END	Specifies the end date of an interval in the ***YYMMDD*** format. If an interval end date is omitted, the end date will be the day preceding the start of the next period.

Automatic Interval generation

Use the **PRSTART** statement to generate a non-cyclic period that has interval dates matching the run dates of a named Application or Group.

The following additional keywords can be specified on the **PRSTART** statement.

Table 146. Keywords for PRSTART

Keyword	Description
ADID	Specifies the application name to base the interval dates on.
ADSTAT	Specifies the status of the application to use: **A** Active (default) **P** Pending
ADTYPE	Specifies the type of application to use: **A** Application (default) **P** Group

Table 146. Keywords for PRSTART (continued)

Keyword	Description
IAT	For applications with run cycles with multiple Input Arrival times, this identifies which set of run cycles to generate dates for. If omitted all run cycles will be used (format ***HHMM***).
FROMDATE	The date from which to generate dates (for details, see LIST GENDAYS – Generate dates from a rule on page 153).
TODATE	The date to generate dates until (for details, see LIST GENDAYS – Generate dates from a rule on page 153).
VALID	The date on which the application to use must be valid, in the format ***YYMMDD*** (the default is today).
DATELIST	If used without **ADID**, this specifies a **SAVELIST** output from a previous command that generated a GNDAY **SAVELIST** to use as input to generate interval dates for the period. If used in conjunction with **ADID**, **GETDATES** is used internally to generate run dates for that application, which are stored in the **SAVELIST** specified in **DATELIST**, and then used to generate interval dates for the period.

Note:

1. The ability to base periods on applications is available only starting from HCL Workload Automation for Z V8.6 SPE, or later.
2. Any period generated using an application will have only dates within the **FROMDATE**/**TODATE** range. The period will need to be regenerated when that range expires, or the Application or and prerequisites, such as referenced calendars or periods, are changed.
3. Periods based on Applications must be non-cyclic.
4. Do not use an application that references the period you are updating with **PRSTART**. The command works, but the dates will be based on the values of the period prior to the command executing, which might result in different intervals being placed in the period.
5. You can use **OPTIONS PREMPTY** to control what happens when no dates are generated.

RG – Run cycle group record

The RG record is a multi segment record, with the following structure.

```
RGCOM -+- Common segment (1 per appl)
       |
       += RGRUN =+= Run Cycle(s)
                 |
                 +- ADRULE - Rule (1 per run cycle)
```

Note:

1. Some segments have Batch Loader statements with slightly different names. `RGCOM` has a Batch Loader statement of `RGSTART`.
2. The rule text within the `RGRUN` segment is an `ADRULE` statement, not `RGRULE`.

RGSTART – Run cycle group common details (RGCOM segment)

Use the `RGSTART` control statement to signal the start of a run cycle group definition.

After an `RGSTART` control statement there are one or more `RGRUN` control statements, one for every run cycle of the run cycle group.

The `ADRULE` control statement must immediately follow the `RGRUN` statement. See `ADRULE` for details.

Note: The `RGSTART` statement is available only starting from HCL Workload Automation for Z V9.1, or later.

Table 147. Keywords for RGSTART

Keyword	Description
`CALENDAR`	The name of a calendar used by the entire group (of up to 16 characters). This field is optional. The run cycle group calendar is superseded by the calendar name specified for each run cycle, if any. If none is specified, the DEFAULT calendar is used.
`DESCR`	A description of up to 50 characters. This field is optional.
`DLDAY`	The number of days (from 1 to 99) from the input arrival day that the application should be completed in: 0 means that the deadline is on the same day as the input arrival day. This must be an integer. This value becomes the default deadline day for the entire group. It is overruled at run cycle level by a value in the `DLDAY` keyword of `RGRUN`.
`DLTIME`	The deadline time that the application should be completed by, in the format *hhmm*. This value becomes the default deadline time for the entire group. It is overruled at run cycle level by a value in the `DLTIME` keyword of `RGRUN`.
`IATIME`	The default input arrival time that will be generated by this run cycle group in the *hhmm* format. This field is optional, but if you do not specify here a value for the whole group, you must specify input arrival times for each run cycle of the group in the `RGRUN` control statement.

Table 147. Keywords for RGSTART (continued)

Keyword	Description
JVTAB	The name of the JCL variable table associated with the run cycle group (up to 16 characters). This field is optional. The run cycle group variable table is superseded by the variable table specified for each run cycle, if any.
OWNER	The run cycle group owner's name (from 1 to 16 characters). This field is optional.
RGID	The name of the run cycle group. The name must be from 1 to 8 alphanumeric characters long and must start with a letter or national character. This field is required.

RGRUN – Run cycle group individual run cycle

Use the **RGRUN** control statement to add a run cycle specification within a run cycle group.

RGRUN statements follow the **RGSTART** statement that defines a run cycle group, and are followed each by the **ADRULE** statement that defines the run cycle rule.

Note: The **RGRUN** statement is available only starting from HCL Workload Automation for Z V9.1, or later.

Table 148. Keywords for RGRUN

Keyword	Description
CALENDAR	The name of the calendar used by this run cycle. The name can be of up to 16 characters. If it is not specified, the run cycle uses the calendar specified for the run cycle group.
JVTAB	The name of the JCL variable table to be used for the occurrences generated. The name can be of up to 16 characters. If it is not specified, the run cycle uses the variable table specified for the run cycle group.
DESCR	A free-format description of the run cycle, up to 50 characters and enclosed in single quotation marks.
DLDAY	The number of days (from 1 to 99) from the input arrival day that the application should be completed in: 0 means that the deadline is on the same day as the input arrival day. This must be an integer. A value specified here overrules for this run cycle any value defined with **RGDLDAY** for the entire group.
DLTIME	The deadline time that the application should be completed by, in the format *hhmm*. A value specified here overrules for this run cycle any value defined with **RGDLTIME** for the entire group.

Table 148. Keywords for RGRUN (continued)

Keyword	Description
IATIME	The time, in the format *hhmm*, that the application is to arrive at the first workstation. If it is not specified here, the run cycle uses the input arrival time specified for the run cycle group.
NAME	The run cycle name. It can be of up to 8 characters.
RPTEND	The repeat end time for the EVERY options, in the format *hhmm*. It must be a time between the IA time of the run cycle and the calendar work day end time of the application.
RPTEVRY	The repeating frequency for the EVERY options, in the format *hhmm*. It specifies that the application has an occurrence in the long-term plan every *hhmm*, starting from the IA time to the repeat end time (**RPTENDT** keyword). If this keyword is not set, only the occurrence related to the IA time is added to the long-term plan.
RULE	Defines which free-day rule is in effect: **E** Count only work days when using the rule or offset. That is, free days are excluded. This option ensures that the scheduled day will always be a work day. This is the default for offset-based run cycles. **1** Count work days and free days when using the rule or offset. If this gives a free day, schedule the application on the closest work day *before* the free day. **2** Count work days and free days when using the rule or offset. If this gives a free day, schedule the application on the closest work day *after* the free day. **3** Count work days and free days when using the rule or offset. If this gives a free day, schedule the application *on* the free day. This is the default for rule-based run cycles. **4** Count work days and free days when using the rule or offset. If this gives a free day, *do not* schedule the application at all.
SUBSETID	The run cycle subset identifier. If the run cycle is part of a subset in the run cycle group (this is useful to match more run cycles against negative rules or to use the logical

Table 148. Keywords for RGRUN (continued)

Keyword	Description
	AND condition), enter the name of the subset. It must be from 1 to 8 alphanumeric characters long and must start with a letter or national character.
TYPE	Specify the type of rule-based run cycle: **R (regular)** The **ADRULE** statement specifies days when the application should be scheduled. **E (exclusion)** The **ADRULE** statement specifies days when the application should not be scheduled. **A (regular for all A rules subsets)** The **ADRULE** statement specifies days when the application should be scheduled *only* if they match all **A** types of the set of run cycles belonging to **SUBSETID**. **D (exclusion for all D rules subsets)** The **ADRULE** statement specifies days when the application should not be scheduled ONLY if they match all **D** types of the set of run cycles belonging to **SUBSETID**.
VALFROM	The start date of the validity of this run cycle, in the format *yymmdd*.
VALTO	The end date of the validity of this run cycle, in the format *yymmdd*.

SR – Special Resource record

The SR record is a multi segment record, with the following structure.

```
SRCOM -+- Common segment (1 per SR)
       |
       += SRDWS = Default workstation (many per SR)
       |
       += SRIVL =+= Interval (many per SR)
                 |
                 += SRIWS = Interval workstation
                            (many per interval)
```

Note: Some segments have Batch Loader statements with slightly different names. **SRCOM** has a Batch Loader statement of **SRSTART**.

SRSTART – Special Resource common details (SRCOM segment)

Use the **SRSTART** statement to define the common part of a Special Resource.

Table 149. Keywords for SRSTART

Keyword	Description

DROP	Specifies a date in the format *YYMMDD*, or + or – a number of days, which ensures that any Interval Date preceding that date is dropped from the Interval.
	
	For example, to drop any dates over a year old, issue the following command:
	
	`OPTIONS DBMODE(UPDATE)`
	
	`SRSTART RESNAME(BATCHSR) DROP(-365)`

RESNAME	The name of the resource, up to 44 characters. The name of the special resource is translated to uppercase. You can include national characters in the name, but you are recommended not to include % and *, because HCL Workload Automation for Z uses these for filtering and searching in the panels. It is also good practice not to use the comparison operators: greater-than symbol (>), less-than symbol (<), caret (^), equals sign (=), or blank spaces. These might by used in search arguments passed by programming interface programs.
GROUP	The resource group, up to 8 characters. The group ID is for selecting subsets of resources in the panel (a list filter).
HIPER	Whether the resource represents a Data Lookaside Facility (DLF) object: **N** The resource does not represent a DLF object (default). **Y** The resource represents a DLF object. For more details about hiperbatch and the Data Lookaside Facility, see *Managing the Workload*.
USEDFOR	Whether the resource is used for: **P** Planning, when the current plan is extended **C** Control, when an operation starts **B** Both planning and control **N** Neither planning nor control

Table 149. Keywords for SRSTART (continued)

Keyword	Description
ONERROR	What happens if an operation that allocates this resource ends in error (and does not have an overriding keep-on-error specification in the operation definition): **F** Free the full allocation of this resource, both those allocated exclusive and those allocated shared **FS** Free the full shared allocation of this resource **FX** Free the full exclusive allocation of this resource **K** Keep the full allocation of this resource **Blank** Use the value specified in the **ONERROR** keyword of the **RESOPTS** statement. For details about this statement, see *Customization and Tuning*.
DESCR	A description of the resource, up to 46 characters.
ONCOMPLETE	The value to which the global availability is reset after the operation that uses the resource completes: **Y** Sets the global availability to Yes. **N** Sets the global availability to No. **R** Sets the global availability to blank. **Blank** Uses the system default, according to the following order: 1. The On Complete value set at operation definition level, if not blank. 2. The On Complete value set at special resource definition level, if not blank. 3. The **ONCOMPLETE** or **DYNONCOMPLETE** keyword value, respectively set for the not dynamically added resources or the dynamically added resources, in all the other cases.

Table 149. Keywords for SRSTART (continued)

Keyword	Description
`MAXTYPE`	The value to which the global availability of the resource is reset, when its maximum usage limit is reached: **Y** Sets the global availability to Yes. **N** Sets the global availability to No. **R** Sets the global availability to blank.
`MAXLIMIT`	The number of allocations of this resource after which the resource global availability is changed to the value specified by Max Usage Type.
`QUANTITY`	A value from 0 to 999 999.
`AVAIL`	Whether the resource is available, Y or N.

`MATCHTYP`	This argument is required when spaces, asterisks (*) or percent signs (%) form part of the name of the object, to indicate how to treat the special characters.
	`EXA`
	Use the name exactly as typed, treating asterisks (*) and percent signs (%) as characters to include in the name.
	`PFX`
	Treat as a prefix match, to find an item beginning with what was typed but treating asterisks (*) and percent signs (%) as characters included in the name.
	`SFX`
	Treat as a suffix match, to find an item ending with what was typed but treating asterisks (*) and percent signs (%) as characters included in the name.

SRDWS – Default workstation

Use the `SRDWS` statement to define a default workstation.

Table 150. Keywords for SRDWS

Keyword	Description
`WSID`	Name of the workstation.

SRIVL - Interval

Use the **SRIVL** statement to define an interval within the Special Resource.

Note: To delete an interval, you must specify at least **DAY** or **DATE**, and the **FROM** time.

Table 151. Keywords for SRIVL

Keyword	Description
DAY **DATE**	For **DAY**, the day of the week: • STANDARD (default for days not defined) • MONDAY • TUESDAY • WEDNESDAY • THURSDAY • FRIDAY • SATURDAY • SUNDAY For **DATE**, the date in the format *YYMMDD*.
FROM	The start time of the interval in the format *HHMM*.
TO	The end time of the interval in the format *HHMM*.
QUANTITY	The maximum quantity for the interval, from 0 to 999 999.
AVAIL	The default availability for the interval: Y or N.

SRIWS – Connected workstations

Use the **SRIWS** statement to define a connected workstation.

Table 152. Keywords for SRIWS

Keyword	Description
WSID	Name of the workstation.

WS – Workstation record

The **WS** record is a multi segment record with the following structure.

```
WSCOM -+- Common segment (1 per WS)
       |
       +- WSAM - Access Method (up to 1 per WS)
       |
       += WSSD =+= Specific Date (many per WS)
       |        |
```

```
|        += WSIVL = Workstation Interval
|                   (many per WSSD)
|
+= WSWD =+= Weekday (up to 8 per WS)
|        |
|        += WSIVL = Workstation Interval
|                   (many per WSWD)
|
+= WSDEST = Virtual Workstation Destination
            (many per WS)
```

Note:

1. Some segments have Batch Loader statements with slightly different names. **WSCOM** has a batch loader statement of **WSSTART**.
2. The **WSDEST** statements do not define the full details of the Virtual Workstation Destination, instead the make a reference to a **WSV** record which contains the complete details. **WSV** records must be defined with **WSVSTART** and subordinate statements. A default **WSV** record is automatically created when a **WSDEST** statement is processed.

WSSTART – Workstation common details (WSCOM segment)

Use the **WSSTART** statement to define the common part of a workstation.

Table 153. Keywords for WSSTART

Keyword	Description

DROP	Specifies a date in the format ***YYMMDD***, or + or – a number of days, which ensures that any Interval Date preceding that date is dropped from the workstation.
	
	Fro example, to drop any dates over a year old, issue the following command:
	`OPTIONS DBMODE(UPDATE)`
	`WSSTART WSNAME(CPU1) DROP(-365)`

WSNAME	Name of the workstation (up to 4 characters).
TYPE	Type of workstation: **G** General (default) **C** Computer **P** Print
REPATTR	The workstation reporting attribute:

Table 153. Keywords for WSSTART (continued)

Keyword	Description
	A Automatic (default) **C** Completion only **N** Non reporting **S** Start and Completion
`JOBSETUP`	Specifies whether the workstation is a JCL setup workstation: **N** No (default) **Y** Yes
`TRANSPORT`	Default Workstation transport time that is used for planning if no transport time is specified on a dependency.
`DURATION`	The default duration for an operation on this workstation.
`PRINTOUT`	The DD Name to send the report for this workstation in the planning jobs.
`DESCR`	The description of this workstation
`USAGE`	Parallel server usage: **P** Planning **C** Control **B** Both **N** None
`SPLITABLE`	Whether operations on this workstation are splittable (`Y` or `N`).
`R1NAME`	Name of workstation resource 1.

Table 153. Keywords for WSSTART (continued)

Keyword	Description
`R1PLAN`	Whether workstation resource 1 is used for planning (`Y` or `N`).
`R1CONT`	Whether workstation resource 1 is used for control (`Y` or `N`).
`R2NAME`	Name of workstation resource 2.
`R2PLAN`	Whether workstation resource 2 is used for planning (`Y` or `N`).
`R2CONT`	Whether workstation resource 2 is used for control (`Y` or `N`).
`DEST`	Destination to use, which must have a matching destination in the HCL Workload Automation for Z ROUTOPTS statement.
`STC`	Whether this is a Started Task workstation (`Y` or `N`).
`WTO`	Whether this is a WTO workstation (`Y` or `N`).
`AUTO`	Whether this is an Automation workstation (`Y` or `N`).
`WAIT`	Whether this is a Wait workstation (`Y` or `N`).
`VIRTUAL`	Whether this is a Virtual workstation (`Y` or `N`).
`ZCENTRIC`	Whether this is a z-centric workstation (`Y` or `N`).

WSAM – Access Method

Use the `WSAM` statement defines the Access Method for Extended Agents hosted by tracker agents.

Table 154. Keywords for WSAM

Keyword	Description
`METHOD`	The name of the method.
`NODE`	Node name or IP address if needed by the method.
`PORT`	Port number if needed by the method.

WSSD – Specific date

Use the `WSSD` statement to define a specific date.

Table 155. Keywords for WSSD

Keyword	Description
`DATE`	Date in the format `YYMMDD`
`DESCR`	Description of the specific date

WSWD – Week day

Use the **WSWD** statement to define a day of the week.

Table 156. Keywords for WSWD

Keyword	Description
DAY	Day of the week: • **MONDAY** • **TUESDAY** • **WEDNESDAY** • **THURSDAY** • **FRIDAY** • **SATURDAY** • **SUNDAY** • **STANDARD**
DESCR	Description of the day.

WSIVL – Interval details

Use the **WSIVL** statement to define details of an interval.

Table 157. Keywords for WSIVL

Keyword	Description
START	Start time of the interval (***HHMM***).
END	End time of the interval (***HHMM***).
PS	Number of parallel servers available for the interval.
R1	Number of workstation resource 1 available for the interval.
R2	Number of workstation resource 2 available for the interval.
ALTWS	Alternate workstation.

WSDEST – Virtual Workstation Destination

Use the **WSDEST** statement defines a reference to a Virtual Workstation Destination.

Note: The `WSDEST` statement is available only starting from HCL Workload Automation for Z V8.5, or later.

Table 158. Keywords for WSDEST

Keyword	Description
`DEST`	Name of the destination. For using the controller as a destination, enter a value of `********`. **Note:** After you have referenced a destination with the `WSDEST` statement, HCL Workload Automation for Z automatically creates a Virtual Workstation destination object by using default values.

WSV – Virtual Workstation destination record

The WSV record is a multi segment record, with the following structure.

```
WSVCOM -+- Common segment (1 per WSV)
        |
        += WSVSD =+= Specific Date (many per WSV)
        |         |
        |         += WSVIVL = Workstation Interval
        |                     (many per WSVSD)
        |
        += WSVWD =+= Weekday (up to 8 per WSV)
                  |
                  += WSVIVL = Workstation Interval
                              (many per WSVWD)
```

Note:

1. Some segments have Batch Loader statements with slightly different names. `WSVCOM` has a segment name of `WSVSTART`.
2. A default `WSV` record is automatically created when a `WSDEST` statement is processed. This means a `WSVSTART` statement can never be created by Batch Loader statements, only replaced or updated. Since a full unload of a virtual workstation will include the Batch Loader for both the `WS` and `WSV` portions, `DBMODE(ADD)` makes an exception with `WSV` records and will `REPLACE` the record automatically created when the `WS` record is created by `INSERT` with any values contained within the `WSVSTART` construct.
3. Virtual workstations are available only starting from HCL Workload Automation for Z V8.5, or later.

WSVSTART – Virtual Workstation (WSVCOM segment)

Use the `WSVSTART` statement to define a Virtual Workstation destination in detail.

Note: The `WSVSTART` statement is available only starting from HCL Workload Automation for Z V8.5, or later.

Table 159. Keywords for WSVSTART

Keyword	Description
`WSDEST`	Virtual Workstation Destination name. For using the controller as a destination, enter a value of `********`.
`USAGE`	Parallel Server usage of the destination: **C** Control **N** None
`R1NAME`	Name of workstation resource 1.
`R1PLAN`	Whether workstation resource 1 is used for planning (`Y` or `N`).
`R1CONT`	Whether workstation resource 1 is used for control (`Y` or `N`).
`R2NAME`	Name of workstation resource 2.
`R2PLAN`	Whether workstation resource 2 is used for planning (`Y` or `N`).
`R2CONT`	Whether workstation resource 2 is used for control.
`WSNAME`	Name of the workstation (up to 4 characters).

WSVSD – Specific date

Use the `WSVSD` statement to define a specific date for a destination.

Note: The `WSVSD` statement is available only starting from HCL Workload Automation for Z V8.5, or later.

Table 160. Keywords for WSVSD

Keyword	Description
`DATE`	Date in the format *`YYMMDD`*.
`DESCR`	Description of the date.

WSVWD – Week day

USe the `WSVWD` statement to define a day of the week for a destination.

Note: The `WSVWD` statement is available only starting from HCL Workload Automation for Z V8.5, or later.

Table 161. Keywords for WSVWD

Keyword	Description
`DAY`	Day of the week: • `MONDAY` • `TUESDAY` • `WEDNESDAY` • `THURSDAY` • `FRIDAY` • `SATURDAY` • `SUNDAY` • `STANDARD`
`DESCR`	Description of the day.

WSVIVL – Interval details

Use the `WSVIVL` statement defines details of an interval for a destination.

Note: The `WSVIVL` statement is available only starting from HCL Workload Automation for Z V8.5, or later.

Table 162. Keywords for WSVIVL

Keyword	Description
`START`	Start time of the interval (*`HHMM`*).
`END`	End time of the interval (*`HHMM`*).
`PS`	Number of parallel servers available for the interval.
`R1`	Number of workstation resource 1 available for the interval.
`R2`	Number of workstation resource 2 available for the interval.

Chapter 11. Variable substitution

Workload Automation Programming Language supports variable substitution within its control statements. In this way, the actual values required for each command run are provided (for example, the occurrence name or Input Arrival date).

To activate the variable substitution, use the command **VARSUB SCAN**. By default, the prefix to identify the variables within the command text is the exclamation mark (!). Optionally, you can use a period (.) to terminate the variable name. Because the exclamation mark can be encoded as different character numbers within some code pages, it is recommended that you use **VARSUB SCAN(!)** to guarantee the use of the exclamation mark that corresponds to your code page.

For example, Workload Automation Programming Language can update the operation text of the running job to indicate something about the processing that day, without having to code the SYSIN as instream data in the JCL to use HCL Workload Automation for Z variables:

```
VARSUB SCAN(!)
MODIFY CPOC ADID(!OADID.) IA(!OYMD1.!OHHMM.)
MODIFY CPOP OPNO(!OOPNO) DESC('NO DATA TODAY')
```

This also means that HCL Workload Automation for Z information becomes available to jobs tracked by HCL Workload Automation for Z that were not necessarily submitted by HCL Workload Automation for Z.

For more detailed information about variables, see Variable naming convention on page 323.

Variable naming convention

Variables are tokens within the command statements that are to be replaced with actual values immediately before the statement is run.

A variable name begins with a prefix character, which is usually an exclamation mark (**!**) to avoid conflicts with HCL Workload Automation for Z variable substitution (but this can be changed), and an optional termination character of a period (.). The variable name must start with an alphabetic character, then it can consist of characters A-Z, 0-9, @, #, _ and the special characters £ and $. The special characters can be modified by using the **OPTIONS VARNAMES** statement. The name can be any length. However, if the variables you create in Workload Automation Programming Language are also to be stored in the HCL Workload Automation for Z JCL Variable Tables, their names must also conform to the JCL Variable naming convention.

Variable resolution process

When Workload Automation Programming Language runs a **VARSUB SCAN** or an **ADD** statement, it checks if the job is running within HCL Workload Automation for Z and set the supplied variables at this point.

The **VARSUB SCAN** statement also adds the GLOBAL and APPLICATION tables to the search sequence, unless you set a specific search sequence. This enables the searching for variables when needed. At any point, except within a Batch Loader object construct, you can issue **VARSET** statements to set variable values within Workload Automation Programming Language. You can do it even before the first **VARSUB SCAN** statement.

When a variable is found in a statement, Workload Automation Programming Language first checks if the variable was already set or referenced with the current running of Workload Automation Programming Language. If it was, that value is

used, without looking up a new value from the table. Even if a variable was changed by a **VARSET** or **VARDATE** command, the latest value is always used.

If the variable was not set or referenced, Workload Automation Programming Language looks through the table search sequence by accessing the tables as required. By default, the first table searched is the APPLICATION table belonging to the occurrence running the Workload Automation Programming Language job; then the GLOBAL table is searched. To add extra tables to the beginning of the table search sequence, set the **VARSUB TABLE** statement; to remove the tables, set **VARSUB CLOSE**. You can specify the whole search sequence with the **VARSUB SEARCH** statement.

By default, if a variable is not found the resolution fails and the statement is not run. You can change this default by setting **VARSUB VARFAIL**.

Note:

- If you renamed the GLOBAL table, specify the name in the **OPTIONS GTABLE** statement. For example:

```
OPTIONS GTABLE (MYGLOBAL)
VARSUB SCAN(!)
```

- If you do not have a GLOBAL table, to prevent accessing it, specify **OPTIONS GTABLE** as follows:

```
OPTIONS GTABLE ()
VARSUB SCAN(!)
```

- To prevent the APPL and GLOBAL tables being automatically added to the search path, issue the following command to set an empty search path before activating the variable scanning:

```
VARSUB SEARCH(-) SCAN(!)
```

- If you change the table search sequence by setting **VARSUB CLOSE**, **SCAN**, **SEARCH**, or **TABLE** the message EQQI022 is issued to report the search sequence at the point when it was modified. In the message, the application and global tables are shown as **-APPL-** and **-GLOBAL-**, respectively. This defers the potential search of the occurrence in the current plan until needed, which can save CPU processing if no table based variables are referenced later. The table names are listed if the tables are actually opened later.

Variable parsing rules

When activated, variable parsing searches for the variable prefix character and then treat every character that follows as a variable, until it finds either a character that cannot be part of a variable name or the termination character.

For example, the following command:

```
000001 VARSUB SCAN (!)
000002 VARSET CODE VALUE(YY)
000003 VARSET SUFFIX VALUE(1234)
000004 MODIFY CPOC ADID(!OADID) IA(!OYMD1!OHMM)
000005 MODIFY CPOP OPNO(!OOPNO.)
000006 MODIFY CPUSRF UFNAME(CODE) UFVALUE(XX!CODE.ZZ)
000007 MODIFY CPUSRF UFNAME(COMPOUND) UFVALUE(!CODE..!SUFFIX)
000008 VARSUB NOSCAN
000009 MODIFY CPUSRF UFNAME(OK) UFVALUE(It worked!!!)
```

Is resolved as follows:

```
000001 VARSUB SCAN (!)
000002 VARSET CODE VALUE(YY)
000003 VARSET SUFFIX VALUE(1234)
000004 MODIFY CPOC ADID(MYAPPL) IA(1101241002)
000005 MODIFY CPOP OPNO(005)
000006 MODIFY CPUSRF UFNAME(CODE) UFVALUE(XXYYZZ)
000007 MODIFY CPUSRF UFNAME(COMPOUND) UFVALUE(YY.1234)
000008 VARSUB NOSCAN
000009 MODIFY CPUSRF UFNAME(!OK) UFVALUE(It worked!!!)
```

Note:

1. The `VARSUB SCAN` statement activates variable substitution.
2. A user variable called CODE is set to YY.
3. A user variable called SUFFIX is set to 1234.
4. !OADID is resolved to the occurrence name, terminated by the closing parenthesis. !OYMD1 is resolved to the occurrence IA date, terminated by the prefix of the next variable. !OHHMM is resolved to the occurrence IA time, terminated by the closing parenthesis.
5. !OOPNO is resolved to the number of the running operation, terminated by the period (.), which will *not* appear in the resolved command because it is the optional termination character. In this case, the period was not needed because the closing parenthesis terminates the variable name anyway.
6. !CODE is resolved to YY, terminated by the period (.). In this case, the termination character is *required* to distinguish the variable name CODE from the subsequent ZZ characters. The termination character does not appear in the resolved statements (just like in JCL).
7. !CODE is resolved to YY and !SUFFIX is resolved to 1234. In this case, the code and the suffix are to be intentionally separated by a period in the resolved statement, therefore a second period needs to be coded after the termination character (just like in JCL).
8. The `VARSUB NOSCAN` statement deactivates variable substitution.
9. Even though the last line contains !OK and other exclamation marks, because the variable substitution is turned off the statement is run as is. Hence, the result is a user field called !OK with the value “It worked!!!”

Dependent variables work slightly differently from how they work with HCL Workload Automation for Z JCL Tailoring. The target variable, on which the dependency is related to, does *not* need to have been previously referenced in prior Workload Automation Programming Language statements. Instead, Workload Automation Programming Language automatically searches for and loads the target variable, and if the target variable has a dependency it repeats the process, until a static variable is found. If dependent variables were coded so that the references are circular, the parsing fails and a message highlighting the variables in the loop is issued.

REXX interpretation considerations

Workload Automation Programming Language variables are replaced with their actual value immediately before a command is run.

Because Workload Automation Programming Language is written in REXX, some commands exploit the REXX interpreter. As a consequence, when you use REXX functions or expressions you need to ensure that a variable value does not contain blanks because this might modify the way the command is run. Make also sure that mathematical operators, if any, do not cause the variable to resolve to more than 250 characters. Every time you set a variable that contain blanks or mathematical characters, such as +, -, /, * or %, you must include the variable name by double quotation marks.

The following commands exploit the REXX interpreter:

- `CONSOLE`
- `DISPLAY`
- `DO WHILE`
- `DO UNTIL`
- `IF-THEN`
- `LOG`
- `WRITE`
- When `VARSET =` is used, REXX interpretation is performed. The keyword based variant of `VARSET` does not use REXX interpretation.

For example:

```
IF (“!TAG” = “-JOBNAME”) THEN
DO WHILE RIGHT(“!LINE”,1) <> “1”
```

Alternatively, you can use the `@V` function, which returns a variable value into a REXX expression and automatically returns values between double quotes into the expression. It also works without variable substitution being active, which allows variables to be used in expressions that might include the variable prefix for other purposes. For example:

```
IF (@V(TAG) = “-JOBNAME”) THEN
DO WHILE RIGHT(@V(LINE),1) <> “1”
```

You can use this function together with variable substitution, to subscript a variable, such as an object variable, without the need for a separate `VARSET VARIABLE` statement.

To issue a set of commands 10 seconds apart, use the following commands:

```
VARSUB SCAN
VARSET CMD1 = "F WSIC,STATUS"
VARSET CMD2 = "F WSJC,STATUS"
VARSET CMD3 = "F WSKC,STATUS" VARSET CMD4 = "F WSLC,STATUS"
DO X = 1 TO 4
CONSOLE @V(CMD!X)
WAIT 10
END
```

For commands that use REXX interpretation, you can use any REXX functions to extend the language. These functions are resolved only if they are not included by quotes, and can be mixed with Workload Automation Programming Language functions. Nesting is allowed. For example:

```
VARSET CHAR =
SUBSTR(ENVATTR(DD,DSNAME,MYDD,1),2,1)
```

The only exceptions are the functions whose names begin with the at sign (@), which cannot have anything nested within them, and are resolved prior to the line being executed. The @ functions must be included within double quotes.

To avoid resolution issues, you can use the REXX equivalent of Workload Automation Programming Language variables. Such variables use the same names but are prefixed with `com.VAR.` and do not need the exclamation mark (!) as prefix. For these variables the REXX rules apply, with the exception that if they were not defined in an earlier command, they resolve to a zero length string. These variables are managed like normal REXX variables and are resolved at running time, not before.

For example, `IF (“!TAG” = “-JOBNAME”) THEN` could be replaced with `(com.VAR.TAG = “-JOBNAME”) THEN`

This avoids problems with blanks, mathematical characters, and the risk of exceeding the limit of 250 characters.

Object variables are not directly accessible in REXX mode, but you can transfer elements to REXX variables with the `VARSET` command, as in the following example:

```
VARSET MYLINE VARIABLE(@MYOBJ-!RX)
```

Object variables

Some commands that read data from the databases or plans have an `OBJECT` keyword to create a set of object variables that enables programmatic access to all the fields within the object that was accessed.

Object variables are all prefixed with the at sign (@) followed by the object name. The number sign (#) is used as a prefix to indicate a count of subordinate elements. The hyphen character is then used to separate the rest of the elements of each object.

There are three kinds of object variables:

`LIST`

Object variables that create a count of each database or plan record converted into an object, and then a numerically suffixed object for each record found.

For example, the keyword `OBJECT(MYOBJ)` creates `@MYOBJ`, which will contain the number of records identified. Then, each record will have object variables beginning with the object name with a numeric suffix, such as `@MYOBJ1-ADID`, `@MYOBJ2-ADID`, and so on.

Though not generated from a `LIST`, the object for `SELECT CPUSRF` is similar to `LIST` because can return up to 100 user fields. For example, the command `@USRF` returns the number of user fields selected. The command `@USRF1-CPUFNAME` returns the name of the first user filed selected.

The following example shows another use of the `LIST` object variable:

```
VARSUB SCAN
LIST AD ADID-EQ(DH*) OBJECT(MYAPPS)
DO I = 1 TO !@MYAPPS
   DISPLAY @V(@MYAPPS!I.-ADID)
END
```

SELECT

Object variables that represent a single record in the database or plan.

For example, the keyword **OBJEXT(MYOBJ)** creates object variables beginning with the object name, such as **@MYOBJ-ADID**. The exception to this is SELECT CPUSRF, which returns an object like a LIST request.

FILE

Object variables that represent a sequential file structure.

The object variable returns the number of rows, and each row is represented by the object name followed by a hyphen (-) and row number. For example, **@MYOBJ-1** shows row 1 of the file.

Understanding the structure of the database and plan objects is crucial to understand the syntax of the object variables, because they represent the structure of the records.

HCL Workload Automation for Z records can have up to three levels of information (segments). The common segment is always level one, but then most records have at least a second level, and some of those might also have a third level.

For example, in an application **ADCOM** is at level 1. Then there can be multiple **ADRUN** (run cycles) and **ADOP** (operations) at level 2. Finally, **ADOP** could have many level 3 sub segments, for example **ADDEP** (dependencies) and **ADSR** (special resources). The record structures can be seen in the relationship diagrams in the Batch Loader section.

You access data at level 1 as follows:

```
@<object>-<field>
```

where:

<object>

Name of the object that was set in the **OBJECT** keyword.

<field>

Name of the field.

For example, to obtain the application name of an AD record **@MYOBJ-ADID**, the number of objects for each second-level segment can be accessed as follows:

```
@<object>-#<segment2>
```

where:

<object>

Name of the object that was set in the **OBJECT** keyword.

<segment2>

Name of the second level segment.

To obtain the number of operations **@MYOBJ-#ADOP**, access data at level 2 as follows:

```
@<object>-<segment2>-<n2>-<field>
```

where:

<object>

Name of the object that was set in the **OBJECT** keyword.

<segment2>

Name of the second level segment.

<n2>

Sequence number of the second level segment.

<field>

Name of the field.

To obtain the operation number of the second operation **@MYOBJ-ADOP-2-ADOPNO** , the number of objects for each third level segment can be accessed as follows:

```
@<object>-<segment2>-<n2>-#<segment3>
```

where:

<object>

Name of the object that was set in the **OBJECT** keyword.

<segment2>

Name of the second level segment.

<n2>

Sequence number of the second level segment.

<segment3>

Name of the third level segment.

To obtain the number of special resources for the second operation **@MYOBJ-ADOP-2-#ADSR**, data at level 3 can be accessed as follows:

```
@<object>-<segment2>-<n2>-<segment3>-<n3>-<field>
```

where:

<object>

Name of the object that was set in the **OBJECT** keyword.

<segment2>

Name of the second level segment.

<n2>

Sequence number of the second level segment.

<segment3>

Name of the third level segment.

<n3>

Sequence number of the third level segment.

<field>

Name of the field.

To obtain the special resource name of the third resource of the second operation **@OBJ-ADOP-2-ADSR-3-ADSRN**, you can display the complete object structure of any record type by using the **SHOW OBJECT** command.

In the following example, the **SHOW OBJECT(CL)** command shows all the available object variables for a calendar with ***n*** showing where sequence numbers fit into the syntax:

```
08/22 10.47.39 EQQI200I SHOW OBJECT(CL)
08/22 10.47.39 EQQI601A Object: @OBJ-CLNAME
08/22 10.47.39 EQQI601A Object: @OBJ-CLDAYS
08/22 10.47.39 EQQI601A Object: @OBJ-CLSHIFT
08/22 10.47.39 EQQI601A Object: @OBJ-CLDESC
08/22 10.47.39 EQQI601A Object: @OBJ-CLVERS
08/22 10.47.39 EQQI601A Object: @OBJ-CLLDATE
08/22 10.47.39 EQQI601A Object: @OBJ-CLLTIME
08/22 10.47.39 EQQI601A Object: @OBJ-CLLUSER
08/22 10.47.39 EQQI601A Object: @OBJ-CLLUTS
08/22 10.47.39 EQQI601A Object: @OBJ-#CLSD
08/22 10.47.39 EQQI601A Object: @OBJ-CLSD-n-CLSDDATE
08/22 10.47.39 EQQI601A Object: @OBJ-CLSD-n-CLSDSTAT
08/22 10.47.39 EQQI601A Object: @OBJ-CLSD-n-CLSDDESC
08/22 10.47.39 EQQI601A Object: @OBJ-#CLWD
08/22 10.47.39 EQQI601A Object: @OBJ-CLWD-n-CLWDDAY
08/22 10.47.39 EQQI601A Object: @OBJ-CLWD-n-CLWDNUM
08/22 10.47.39 EQQI601A Object: @OBJ-CLWD-n-CLWDSTAT
08/22 10.47.39 EQQI601A Object: @OBJ-CLWD-n-CLWDDESC
08/22 10.47.39 EQQI299I Statement completed - RC=0 (00000014)
```

The **VARSET VARIABLE** keyword allows loop variables to be used as subscripts to cycle through values within an object.

The following example shows a simple loop cycling through each job name in a application object:

```
DO X = 1 TO @MYOBJ-#ADOP
   VARSET JOB VARIABLE(@MYOBJ-ADOP-!X-ADOPJN)
   DISPLAY !JOB
END
```

FILE objects are a list of records that can be processed in many ways. They are created by commands such as **READ** or by the **SAVELIST** keywords in **LIST**.

Finding specific rows, this example returns only the rows that contain **Are we there**:

```
VARSUB SCAN
READ IEBGENR.SYSUT2 SOURCE(SYSOUT) OBJECT(MYOBJ) CLIP(1)
DO FOREVER
   VARSCAN MYOBJ TARGET(Are we there) CURSOR(RX,CX)
```

```
                ACTION(LEAVE)
   DISPLAY @V(@MYOBJ-!RX)
END
```

Filtering rows, this example returns only rows that do not start with the character `W` nor `I`:

```
VARSUB SCAN
READ IEBGENR.SYSUT2 SOURCE(SYSOUT) OBJECT(MYOBJ) CLIP(1)
DO RX = 1 TO !@MYOBJ
   VARSCAN MYOBJ FILTER(NE) TARGET(W,I) CURSOR(RX,CX)
                COLS(001,001) ACTION(ITERATE)
   WRITE OUTDATA "@V(@MYOBJ-!RX)"
END
```

Note:

1. If the object variable does not exist, the resolution fails in accordance with the `VARFAIL` setting or the `VARSUB` statement. As well as basic syntax, this applies also to the sequence numbers. For example, `@MYOBJ-ADCOM-2-ADOPNO` is not resolved if at least 2 operations do not exist in the object. The `#<segment>` elements can be used to determine how many of each object exists.
2. Field names and segment names for each object can be obtained from the EQQFLALL member of the Workload Automation Programming Language member.
3. Object variables depend on the HCL Workload Automation for Z version you are using. The `SHOW OBJECT` command lists the variables that are allowed for your current version of HCL Workload Automation for Z.
4. When an object variable is created from a Current Plan Operation command, there will be an entry found by the identification arguments. To process only the rows that were selected by the filter arguments, the object has an `@FILTER` attribute, which contains a list of the item numbers returned by the filter arguments.

   For example:

   ```
   VARSUB SCAN (!)
   VARSET LOOPMAX = WORDS(@V(@CPO-@FILTER))
      DO X = 1 TO !LOOPMAX
      VARSET Y = WORD(@V(@CPO-@FILTER),@V(X))
      DISPLAY "IA="||@V(@CPO!Y.-CPOPIA)
   END
   ```

SAVELIST as object variables

You can access a `SAVELIST` as an `OBJECT` variable, provided that an object variable with the same name does not exist.

This would work similarly to a file object, where the `SAVELIST` object variable, for example `!@RESLIST`, returns the number of rows and the object variable followed by hyphen (-), and a number returns the row number (for example, `@V(@RESLIST-1)` returns row 1).

In addition to normal `FILE` object conventions, you can access the values of single keywords within a record as follows:

```
@<savelist-name>-<record-num>-<keyword-name>
```

For example, to list the resource names from a **SAVELIST** use the following command:

```
VARSUB SCAN
LIST SR SAVELIST(RESLIST)
DO X = 1 TO @V(@RESLIST)
   DISPLAY @V(@RESLIST-!X.-RESNAME)
END
```

Note: To make a **SAVELIST** accessible as an **OBJECT** variable, ensure that you use different names. If an **OBJECT** variable and **SAVELIST** use the same name, the **SAVELIST** can be accessed only by **USELIST** keywords and **SHOW SAVELIST**.

VARDATE – Generate date and time values from rule

Use the **VARDATE** command to generate a variable date from a rule.

```
VARDATE variable|= [BASE(yymmdd|EASTER)]
                   [CALENDAR(<calendar-name>)]
                   [DAY(ALL|MON|TUE|WED|THU|FRI|SAT|SUN)]
                   [FORMAT(<format-definition>)]
                   [MONTH(ALL|JAN|FEB|MAR|APR|MAY|JUN|
                     JUL|AUG|SEP|OCT|NOV|DEC|   )]
                   [OFFSET(+n|-n)] [ONLY(n)|LAST(n)]
                   [RULE(BEFORE|AFTER|ON|NEAREST)] [FORMAT(<dateformat>)]
                   [SAVE(YES|NO)]
                   [SETUP(YES|NO|PROMPT)]
                   [TABLE(table)] [TIME(hhmmss)] [TIME(hhmmss|hhmm)]
                    [UNIT(DAY|WEEK|MONTH|YEAR|WORKDAY|<day-of-week>)]
                   [YEAR(nnnn|+n|-n)] [OFFSET(+n|-n)]
```

The **VARDATE** command can either generate a variable, or can be used to set defaults for other **VARDATE** commands by using the equal sign (=) in place of the variable name.

For example, **VARDATE = YEAR(+1)** does not create a variable, but makes all subsequent **VARDATE** commands calculate dates for the following year.

Note:

1. Even if **VARDATE** command looks similar to a run cycle rule, it is designed to define only a single date within the specified year.
2. Because **VARDATE** is designed to help determine the Free days within the calendar, it does not refer to calendars. For the purposes of calculation, it considers only the differences between weekdays and weekends.

BASE(*yymmdd*|EASTER)

Provides a base date for setting an **OFFSET** against. The base date can either be provided in the *yymmdd* format, either hard coded or from a Workload Automation Programming Language variable, or it can be calculated using a special base name.

Special base names currently include **EASTER**. This returns the date of Easter Sunday. Defined as the Sunday following the first full moon after the Vernal Equinox, it cannot be easily calculated by **VARDATE** rules, so a special **BASE** name is used to access the complex calculation needed.

Note: When you have two consecutive holidays that could be moved by weekends, it is best to use **BASE** in the second, to reference the variable generated by the first. In this way if the first date moves, the second date does not clash with it. For example:

```
VARDATE PH_CHRISTMAS ONLY(25) MONTH(DEC) RULE(AFTER)
VARDATE PH_BOXING_DAY BASE(!PH_CHRISTMAS) OFFSET(1) RULE(AFTER)
```

CALENDAR(<calendar-name>)

Sets the calendar to use if **UNIT(WORKDAY)** is used.

DAY(ALL|MON|TUE|WED|THU|FRI|SAT|SUN)

Defines which day to count within the calculation period. The default value is **ALL**, which counts every day in the period.

For example, **VARDATE PH-CHRISTMAST ONLY(25) MONTH(DEC) RULE(AFTER)** returns 25 December as the primary result, but if that day falls on a weekend it moves it to the following Monday.

If a specific **DAY** is chosen, the **ONLY|LAST** count refers only to that type of day. For example, **VARDATE PH_THANKSGIVING ONLY(4) DAY(THU) MONTH(NOV)** returns the fourth Thursday in November, not the fourth day in November.

FORMAT(<date-format>)

By default, **VARDATE** returns a date in the native HCL Workload Automation for Z format ***YYMMDD***.

You can use the **FORMAT** keyword to change the format to an alternative layout by using a combination of the following case sensitive values:

Table 163. Values for the FORMAT keywords

Value	Description	Example
aa	Part of the day in lower case. • midnight for exactly 00.00.00 • am for 00.00.01 – 11.59.59 • noon for exactly 12.00.00 • pm for 12.00.01 – 23.59.59 **Note:** The text is set in the LANG*xx* file using TERM entries DP00, DPAM, DP12 and DPPM respectively. If you want to provide alternative values, a user language member	pm for 1.05 pm

Table 163. Values for the FORMAT keywords (continued)

Value	Description	Example
	can be concatenated *after* the system member to override the supplied value.	
AA	Part of the day in upper case. • MIDNIGHT for exactly 00.00.00 • AM for 00.00.01 – 11.59.59 • NOON for exactly 12.00.00 • PM for 12.00.01 – 23.59.59 **Note:** The text is set in the LANG*xx* file using TERM entries DP00, DPAM, DP12 and DPPM respectively. If you want to provide alternative values, a user language member can be concatenated *after* the system member to override the supplied value.	PM for 1.05 pm
CC	The century part of the year.	20 for 9 July 2014
D	Day of the month with no leading zeros.	9 for 9 July 2014
DD	Day of the month with leading zeros.	09 for 9 July 2014
DDD	Day of the year (julian day).	190 for 9 July 2014
h	Hours in 12 hour format with no leading zeroes.	1 for 1.05 pm.
H	Hours in 24 hour format with no leading zeroes.	2 for 2.28 am.
hh	Hours in 12 hour format with leading zeroes.	01 for 1.05 pm.
HH	Hours in 24 hour format with leading zeroes.	13 for 1.05 pm.
M	Month of the year with no leading zeroes.	7 for 9 July 2014
MM	Month of the year with leading zeroes.	07 for 9 July 2014
mmm	Three character name of the month in mixed case.	Jul for 9 July 2014
MMM	Three character name of the month in upper case.	JUL for 9 July 2014
mmmm	Full name of the month in mixed case.	July for 9 July 2014
MMMM	Full name of the month in upper case.	JULY for 9 July 2014
N	Minutes with no leading zeroes.	5 for 1.05 pm.

Table 163. Values for the FORMAT keywords (continued)

Value	Description	Example
	Note: Minutes is NN to avoid conflict with MM for month.	
NN	Minutes with leading zeroes **Note:** Minutes is NN to avoid conflict with MM for month.	05 for 1.05 pm.
OO	Ordinal day suffix in upper case.	TH for 9 July 2014
oo	Ordinal day suffix in lower case.	th for 9 July 2013
S	Seconds with no leading zeroes.	8 for 1:05:08 pm
SS	Seconds with leading zeroes.	08 for 1:05:08 pm
W	Day number of the week 1=Monday, 7=Sunday	3 for 9 July 2014
ww	Two character name of the day in mixed case.	We for 9 July 2014
WW	Two character name of the day in upper case.	WE for 9 July 2014
WWW	Three character name of the day in upper case.	WED for 9 July 2014
www	Three character name of the day in mixed case.	Wed for 9 July 2014
wwww	Full name of the day in mixed case.	Wednesday for 9 July 2014
WWWW	Full name of the day in upper case.	WEDNESDAY for 9 July 2014
YY	Two digit year.	14 for 9 July 2014
YYYY	Four digit year.	2014 for 9 July 2014

`MONTH(ALL|JAN|FEB|MAR|APR|MAY|JUN|JUL|AUG|SEP|OCT|NOV|DEC| )`

Defines the scope of the period to start counting the days from. By default, **`ALL`** covers the whole year, therefore **`ONLY`** counts from the start of the year and **`LAST`** counts from the end of the year. Specifying a month targets the scope to an individual month.

For example:

```
VARDATE PH_NEW_YEARS_DAY ONLY(1) /* First day of the year */
VARDATE PH_NEW_YEARS_EVE LAST(1) /* Last day of the year */
VARDATE PH_LABOR_DAY ONLY(1) DAY(MON) MONTH(MAY) /* First Monday in May */
VARDATE PH_CHRISTMAS ONLY(25) MONTH(DEC) /* 25th of December */
```

Specifying **`MONTH`** with no value corresponds to the current month, unless you set **`OPTIONS DATE`**, in which case it uses the month specified.

For example, `VARDATE Q2_W9 ONLY(9) DAY(MON) MONTH(APR) /* 9TH Monday of 2ND Quarter */VARDATE FIRST_MON ONLY(1) DAY(MON) MONTH /* 1ST Monday current month */`

Note: Even if the `MONTH` keyword sets the frame of reference to a particular month, it does not restrict the results to that month. For example, `VARDATE Q2_W9 ONLY(9) DAY(MON) MONTH(APR) /* 9TH Monday of 2ND Quarter */`

`OFFSET(+n|-n)`

After the date has been set by either `BASE` or `ONLY|LAST`, the `OFFSET` keyword can be used to move the date backwards or forwards a number of days. For example:

```
VARDATE MYFRI ONLY(5) DAY(WED) MON(AUG) OFFSET(2) /* FRI after 5th WED */
VARDATE GDFRI BASE(EASTER) OFFSET(-2) /* Good Friday */
```

`ONLY|LAST(n)`

Used to calculate dates entirely by rules, rather than use a `BASE` date as the starting point. `ONLY` refers to a number of days from the start of the calculation period, `LAST` refers to a number of days from the end of the calculation period.

`RULE(BEFORE|AFTER|ON|NEAREST)`

After calculation and `OFFSET`, the date could point to a weekend date. This might not be appropriate if `VARDATE` is being used to calculate public holidays. The `RULE` keyword determines what to do with a weekend date:

`BEFORE`

If the date points to a Saturday or Sunday move the date to the Friday.

`AFTER`

If the date points to a Saturday or Sunday move the date to the Monday.

`ON`

Do not adjust the date (default).

`NEAREST`

If the date points to a Saturday move the date to Friday, if it points to a Sunday move the date to Monday.

`SAVE(YES|NO)`

The `SAVE` keyword determines whether and when the variable is written to an HCL Workload Automation for Z JCL variable table:

`YES`

The variable is written directly in the associated table at this point.

`NO`

The variable is not written in the associated table, but might be written later if a `VARSAVE` command is run referencing the table associated to this variable.

If you specify the `SAVE` keyword with no argument, `SAVE(YES)` is assumed.

For example, `VARSET MYVAR VALUE(Hello World) TABLE(MYTABLE) SAVE`

Note: If you want to set several variables to be written to the same table, it is more efficient to use a subsequent `VARSAVE` command than `SAVE` on every `VARSET`.

For example,

```
VARSET MYVAR1 VALUE(Hello) TABLE(MYTABLE)
VARSET MYVAR2 VALUE(Ciao) TABLE(MYTABLE)
VARSET MYVAR3 VALUE(Bonjour) TABLE(MYTABLE)
VARSAVE MYTABLE
```

Note:

1. Variables do not need to already exist within a table. They are added to the table if needed.
2. Tables to not need to exist to have variables saved to them. If a table does not exist, it is created automatically, using `OPTIONS OWNER` to set the Owner ID and the description is set to the creating Job name, Jes Number and current date.

`SETUP(YES|NO|PROMPT)`

Sets the `SETUP` value for any new variables when saved into a JCL Variable Table:

`YES`

Resolve variable at SETUP phase.

`NO`

Resolve variable at SUBMIT phase (default).

`PROMPT`

Promptable variable.

The default value can be changed by `OPTIONS SETUP`.

`TABLE(table)`

Assigns an HCL Workload Automation for Z JCL Variable Table to a Workload Automation Programming Language variable, so that it can be saved in a JCL Variable Table. If the variable has already been referenced from a JCL Variable Table, the `TABLE` keyword is not needed. If a Workload Automation Programming Language variable does not have an assigned table name, it cannot be saved.

In the following example, `RUNCOUNT` might not exist in `MYTABLE`, but `VARFAIL(NULL)` will cause the first table in the search sequence at the point to be assigned as the source table name:

```
VARSUB SCAN(!) TABLE(MYTABLE) VARFAIL(NULL)
VARSET RUNCOUNT VALUE(!RUNCOUNT) DELTA(1) SAVE(YES)
```

You can however use the **TABLE** keyword to override the name of the table from which the variable originally came, so the value can be saved to a new table.

TIME(*hhmmss*) or **TIME(hhmm)**

Provides the time portion of a variable. The time can be specified in either the format ***hhmmss*** or ***hhmm***. If ***hhmm*** is used, zero seconds is assumed. If the **TIME** keyword is not specified, the current time is assumed.

UNIT (DAY|WEEK|MONTH|YEAR|WORKDAY|<day-of-week>|HOURS|MINUTES|SECONDS)

Sets the unit by which the **OFFSET** is counted:

DAY

Adds or subtracts a number of days (default).

WEEK

Adds or subtracts 7 days per value of **OFFSET**.

MONTH

Uses the same day of the month using the **OFFSET** value to move backwards or forwards that number of months. If the resulting month is too short for the origin date, the last day of the month is used.

YEAR

Uses the same day of the month in the resulting year, adjusting 29 February to 28 if necessary.

WORKDAY

Adds or subtracts a number of work days.

Any day of the week

You can also use any day of the week as a **UNIT**, for example**MONDAY**, such that **OFFSET(-1) UNIT(MONDAY)** would find the Monday preceding the origin date, and **OFFSET(+1) UNIT(MONDAY)** would find the Monday following the origin date. If the origin date and the unit are the same day of the week, the offset is seven days.

HOURS

Adds hours to the time portion of the date.

MINUTES

Adds minutes to the time portion of the date.

SECONDS

Adds seconds to the time portion of the date.

YEAR(*yyyy*|+*n*|-*n*)

Sets the year for which the date is calculated. By default, it uses the current Workload Automation Programming Language Date, but can be set it to an individual year, or a relative year. The **YEAR** keyword is most

effectively used by setting the default relative to the running job; in this way the same job can run annually to calculate sets of dates for each year. For example, `VARDATE = YEAR(+1) /* Calculate dates for next year */`

Note: `BASE` is mutually exclusive from the `ONLY|LAST DAY|MONTH` approach. The `YEAR` keyword is relevant to both approaches, because the year affects Special Base Name resolution as well as `DAY|MONTH` calculations.

Workload Automation Programming Language is provided with some members containing rule definitions for public holidays for some countries. Look for members in the SEQQWAPL library for members beginning with DATE and a two character country code. These members can be used in conjunction with `INCLUDE` to define a job to update elements in the database annually with public holidays.

```
OPTIONS DBMODE(UPDATE)
```

The following example shows an annual job to maintain calendars with public holidays. The member **DATEIT** generates variables, which can be referenced in Batch Loader as required:

```
VARDATE = YEAR(+1) /* Generate dates for next year */
INCLUDE EQQFILE(DATEIT) /* Generates Italian holiday variables */
CLSTART CALENDAR(FESTIVI) DESCR(GIORNI FESTIVI PER L'ITALIA)
        DROP(-365) /* Drop days over a year old */
CLDAY DAY(MONDAY) STATUS(W) DESCR(GIORNATA DI LAVORO)
CLDAY DAY(TUESDAY) STATUS(W) DESCR(GIORNATA DI LAVORO)
CLDAY DAY(WEDNESDAY) STATUS(W) DESCR(GIORNATA DI LAVORO)
CLDAY DAY(THURSDAY) STATUS(W) DESCR(GIORNATA DI LAVORO)
CLDAY DAY(FRIDAY) STATUS(W) DESCR(GIORNATA DI LAVORO)
CLDAY DAY(SATURDAY) STATUS(F) DESCR(GIORNO LIBERO)
CLDAY DAY(SUNDAY) STATUS(F) DESCR(GIORNO LIBERO)
CLDATE DATE(!GF_CAPODANNO) STATUS(F) DESCR(CAPODANNO)
CLDATE DATE(!GF_EPIFANIA) STATUS(F) DESCR(EPIFANIA)
CLDATE DATE(!GF_PASQUA) STATUS(F) DESCR(PASQUA)
CLDATE DATE(!GF_PASQUETTA) STATUS(F) DESCR(PASQUETTA)
CLDATE DATE(!GF_LIBERAZIONE) STATUS(F) DESCR(LIBERAZIONE)
CLDATE DATE(!GF_LAVORO) STATUS(F) DESCR(FESTA DEI LAVORO)
CLDATE DATE(!GF_REPUBBLICA) STATUS(F) DESCR(FESTA DEI REPUBBLICA)
CLDATE DATE(!GF_FERRAGOSTO) STATUS(F) DESCR(FERRAGOSTO)
CLDATE DATE(!GF_TUTTI_I_SANTI) STATUS(F) DESCR(TUTTI I SANTI)
CLDATE DATE(!GF_IMMACOLATA) STATUS(F) DESCR(IMMACOLATA CONCEZIONE)
CLDATE DATE(!GF_NATALE) STATUS(F) DESCR(NATALE)
CLDATE DATE(!GF_SANTA_STEFANO) STATUS(F) DESCR(SANTA STEFANO)
```

Note:

1. In some countries, rules relating to public holidays falling on weekends may vary regionally. In this case, the `RULE` keyword is not coded in the DATE member so that a regionally appropriate statement can be coded before referencing the DATE member. See the comments within the DATE member before using it. For example:

   ```
   VARDATE = RULE(NEAREST)
   INCLUDE EQQFILE(DATEUS)
   ```

2. The **DATE*xx*** members are provided to help ease the generation of calendars; if the rules do not meet your requirements, you can create your own versions.
3. When **VARDATE** is run, message 033 is issued to show the value that is set. This also adds the day of the week at the end of the message to confirm that the date was set as you expected, as shown in the following example:

   ```
   EQQI033A Variable TO set to 1503082100 (Sunday)
   ```

   You can modify the contents of this confirmation aid by specifying an alternative format with **OPTIONS VARDATE**.

VARGEN – Manage a Generational Data Variable (GDV)

Use the **VARGEN** command to create, read, or list the contents of a Generational Data Variable (GDV).

```
VARGEN [<varname>)] GDV(<name>)
                    GEN(<NEW|LIST|n>)
                    TABLE(<name>)
                   [LIMIT(n|10)]
                   [SAVE(YES|NO)]
                   [VALUE(<value>|@V<input-var-name>)]
```

Where:

<varname>

Name of the variable into which to load data from a generation of the GDV. This argument is ignored for **GEN(NEW)** or **GEN(LIST)**.

GDV(<name>)

Required. The Generational Data Variable (GDV) to be accessed or created. You can specify from 1 to 7 characters. The same rules valid for the HCL Workload Automation for Z variables apply.

GEN(NEW|LIST|*n*)

The generation to access. You can set one of the following values:

NEW

Creates a new generation by setting the value by either the keyword **VALUE** or **@V** keyword. The new generation becomes generation 0 and all previous generations are shuffled downwards, for example 0 becomes -1, -1 becomes -2, and so forth. If the creation of the generations exceeds the maximum limit, the oldest generation is dropped.

LIST

Lists the current values of the GDV.

n

Reads a specific generation of the GDV. For example, 0 reads the most recent, -1 reads the generation before 0, and so forth. The positional parameter containing the variable name is required when reading a generation.

`TABLE(<name>)`

Required. The table into which the GDV is to be stored. To avoid accidental clash with a non-GDVT, set the table name explicitly, rather than use the table search sequence.

`LIMIT(n|10)`

The number of generations that is kept within the GDV. The maximum allowed is 50, the default is 10.

`SAVE(YES|NO)`

Whether the GDV is to be stored into a JCL Variable table.

Note:

- To set more that one variable in the same table, it is more effective to use a subsequent **`VARSAVE`** command than using **`SAVE`** on every **`VARGEN`**.
- It s not required that a variable already exists within a table. It is added to the table, if needed.
- Tables to not need to exist to have variables saved to them. If a table does not exist, it is created automatically, using the **`OPTIONS OWNER`** to set the Owner ID. The description is set to the creating Job name, Jes Number, and current date.

`VALUE(<value>)`

Value of a new generation in the GDV.

`@V <input-var-name>)`

Another Workload Automation Programming Language variable to provide the new generation with a value.

Because the content of this keyword is a variable name, you do not need to specify the variable prefix; instead you can use variable names to provide all or part of the name, allowing subscripting to be performed. For example, in the command **`@V(@MYOBJ-ADOP-!X-ADOPJN)`** the variable **`!X`** sets the segment sequence number in an object variable, allowing all operations to be looped through.

Use the **`VARGEN`** command to create a Generational Data Variable. This is a special form of variable that allows a history of up to 50 previous values to be stored and accessed. Like a generation data group (GDG), creating a new entry in a GDV will shuffle all the old values down a generation. Unlike a GDV, the relative positions are renamed at once, meaning that when a new generation is created the previous generation 0 becomes generation -1 and must be referred to as such in all subsequent commands.

The GDV is stored as an HCL Workload Automation for Z dependent variable, dependent on a variable with the same name as the GDV, with a T prefix. For example, for a GDV named **`MYGDV`** the independent variable is named **`TMYGDV`**. This enables

you to access a specific generation of the variable within the HCL Workload Automation for Z JCL. If you access that variable within Workload Automation Programming Language, using the dependency mechanism, it is considered a normal variable for the course of that Workload Automation Programming Language step, and will not be accessible with the **VARGEN** command.

Use the **VARGEN** command also to extract values into normal variables, and to **LIST** the contents of the GDV into the log.

You can save a GDV into a table directly at the time when a new generation is created by using the **VARGEN SAVE** keyword, or you can commit it to a table later in the process by using the **VARSAVE** command.

A GDV does not need to be defined as such, the first time you add a new generation it is created automatically.

Examples

The following command creates a new GDV called **MYGDV** for table **GDVTEST** (but not saved yet). It first adds a generation containing **World**, followed by a new generation containing **Hello**. Hence, generation 0 contains **Hello** and -1 contains **World**. As **LIMIT** is not specified, the default 10 is used:

```
VARGEN GDV(MYGDV) TABLE(GDVTEST) GEN(NEW) VALUE(World)
VARGEN GDV(MYGDV) TABLE(GDVTEST) GEN(NEW) VALUE(Hello)
```

The output looks like the following example:

```
EQQI200I VARGEN GDV(MYGDV) TABLE(GDVTEST) GEN(NEW) VALUE(World)
EQQI067A GDV MYGDV in table GDVTEST LIMIT=10 GENS=1
EQQI068A Generation: 0 = World
EQQI299I Statement completed - RC=0

EQQI200I VARGEN GDV(MYGDV) TABLE(GDVTEST) GEN(NEW) VALUE(Hello)
EQQI067A GDV MYGDV in table GDVTEST LIMIT=10 GENS=2
EQQI068A Generation: 0 = Hello
EQQI299I Statement completed - RC=0
```

The following command lists the contents of a **MYGDV**:

```
VARGEN GDV(MYGDV) TABLE(GDVTEST) GEN(LIST)
```

The output looks like the following example:

```
EQQI200I VARGEN GDV(MYGDV) TABLE(GDVTEST) GEN(LIST)
EQQI067A GDV MYGDV in table GDVTEST LIMIT=10 GENS=2
EQQI068A Generation:   0 = Hello
EQQI068A Generation:  -1 = World
EQQI299I Statement completed - RC=0
```

The following command stores the previous version of **MYGDV** in variable **MYVAR**:

```
VARGEN MYVAR GDV(MYGDV) TABLE(GDVTEST) GEN(-1)
```

The output looks like the following example:

```
EQQI200I VARGEN MYVAR GDV(MYGDV) TABLE(GDVTEST) GEN(-1)
EQQI033A Variable MYVAR set to "World"
EQQI299I Statement completed - RC=0
```

Accessing a Generational Data Variable through a JCL

Because a GDV is stored as an HCL Workload Automation for Z dependent variable, you can access a specific generation within an HCL Workload Automation for Z JCL. For example:

```
//*%OPC SCAN
//*%OPC TABLE NAME=GDVTEST
//*%OPC SETVAR TGDVONE=('-1')
//*%OPC SETVAR TGDVTWO=('-2')
//*GDVONE &GDVONE
//*GDVTWO &GDVTWO
```

Use `SETVAR` to create a temporary variable containing the relative generation number of the GDV. This must precede the line containing the GDV reference, though it does not need to immediately precede it. This variable must begin with character `T` and be followed by the name of the GDV. For example, for `GDVONE` the temporary variable must be named `TGDVONE`.

In a JCL, you can reference only one generation of a GDV per job. This can be superseded by maintaining multiple copies of a GDV, so that every time you add a generation you add it to multiple GDVs, as in the following example. In this way, you can access as many generations as the copies of the GDV.

```
VARGEN GDV(GDVONE) TABLE(GDVTEST) GEN(NEW) VALUE(Hello)
VARGEN GDV(GDVTWO) TABLE(GDVTEST) GEN(NEW) VALUE(Hello)
```

VARSAVE – Save variables in a JCL Variable Table

Use the `VARSAVE` command to save all the variables that are assigned to the tables specified.

```
VARSAVE [table-1,table-2,...,table-n]
```

You can specify as many tables as you want. For each table, the command saves any variables flagged as `SAVE(NO)` and update the JCL Variable Table in HCL Workload Automation for Z.

To update multiple variables in the same table, this is the most effective way because the command updates all the variables in the same table contemporaneously. Instead, `VARSET` and `SAVE(YES)` perform individual table updates for each specified variable.

Note:

1. Variables do not need to exist within a table. They are added to the table, if needed.
2. Tables do not need to exist to have variables saved to them. If a table does not exist, it is created automatically using `OPTIONS OWNER` to set the Owner ID; the description is set to the creating Job Name, Jes Number and current date.

Example

The following example shows how to create the table MYTABLE and add the variable MYVAR2 to the table.

Note: `VARSET MYVAR2` is to be specified in the statement where you set the TABLE keyword and the `VARSAVE` command must follow the `VARSET` command.

```
VARSUB SCAN
VARSET MYVAR1 VALUE(16)
VARSET MYVAR2 @V(MYVAR1) SAVE(NO) TABLE(MYTABLE)
VARSAVE MYTABLE
```

VARSCAN – Search a sequential object variable

Use the **VARSCAN** command to locate the position of specified strings or words within a sequential Workload Automation Programming Language object, or to filter rows within a Workload Automation Programming Language object.

```
VARSCAN <object> TARGET(<target1>[, <targetn>)]
                [ACTION(ITERATIVE|LEAVE|RC)]
                [CASE(Y|N)]
                [COLS(<from>,<to>)]
                [CURSOR(<rowvar>,<colvar>)]
                [DISTINCT(Y|N,[<separators>])]
                [FILTER(EQ|NE|GT|LT|LE|GE|CO|NC)]
                [FOUND(<foundvar>)]
                [INSTANCE(<n>|1)]
                [ROWS(CURRENT|FIRST|LAST|<start>,<end>)]
```

Where:

<object>

Name of the object to search. You can omit the at sign (@), because this is a known object. The object must be a sequential object, such as one created by the **READ** command. Complex objects, like those created by **LIST** or **SELECT** cannot be read by this command.

ACTION(ITERATE|LEAVE|RC)

Determines the action to take if no match is found:

ITERATE

Continues to the next iteration of the loop. Typically performed in a **FILTER** loop.

LEAVE

Leaves the current loop. Typically performed in a **SCAN** loop.

RC

Sets return code 4 if the the **FOUND** keyword is not specified. This is the default.

CASE(Y|N)

Determines whether the target must be an exact match, including the case. If you specify **N**, differences in the case are tolerated. For example, **Help** would match with **help**. If you specify **Y** (default), the case must match exactly.

COLS(1|<left>,0|<right>)

Limits the columns across which the search is performed. For example, **COLS(20,30)** searches from column 20 to 30 included. You can use the value **0** for the right column, which allows searching to the end of the row. If a target starts within the column limits but finishes outside of the limits, it is not considered a match. The default is **COLS(1,0)**.

CURSOR(<rowvar>,<colvar>)

Names a pair of variables within which to store a cursor position. Searching starts from the row named in the row name variable, and the column after the col name variable. This allows subsequent use of the same **CURSOR** to find the next occurrence of the target. If the named variables do not exist or do not contain whole numbers, they are initialized to the default start position, otherwise the current values is used. If no match is found, the row variable is set to the number of rows +1 and the column variable set to 0. The default cursor starting position is row 1, column 0, meaning that searching starts from row 1, column 1.

You can set the cursor position by setting each variable with the **VARSET** command (for example, **VARSET MYROW=2**). If you set the column variable, the search will start from the following column.

If the **CURSOR** position falls outside the limits of **COLS** or **ROWS**, the starting point is adjusted to fit within those limits. If it is already within those limits, the search starts from the **CURSOR** position.

If **CURSOR** is not specified, the search starts from the beginning of the object.

DISTINCT(Y|N,[delimiters])

Determines whether the target is found as a character match, or as a distinct word or label. If you select **Y**, a match occurs only if the preceding and following characters are not alphanumeric. Alternatively, you can specify as delimiters a list of characters, instead of the alphanumeric distinction. If the text is found at position 1, it is considered to have a preceding delimiter. The default is **N**, meaning that a match occurs by finding the specified set of characters, regardless of whether it is a distinct word. For example, if you set **DISTINCT(N)**, the keyword **TARGET(one)** will match with **gone**, **bone** and **one**. If you set **DISTINCT(Y)**, the keyword **TARGET(one)** will match only with **one**.

FILTER(EQ|NE|GT|LT|LE|GE|CO|NC)

Puts the scan command into **FILTER** mode. In **FILTER** mode the command compares only the current cursor row, or the nearest within the row limits, with the comparator specified and each or the targets. The comparison is done with the contents of the row between the column limits. For most comparators, only 1 needs to be matched for the row to be selected, with the exception of NE and NC where all must be true to be successful.

The following comparator values are allowed:

EQ

The column range must match a target string.

NE

The column range must not match all target strings.

GT

The column range must be greater than a target string.

LT

The column range must be less than a target string.

GE

The column range must be greater than or equal to a target string.

LE

The column range must be less than or equal to a target string.

CO

The column range must contain a target string.

NC

The column range must not contain a target string.

FOUND(<foundvar>)

Names a variable to store the number of the **TARGET** value that is found. If the first **TARGET** is found, 1 is stored in the named variable; if the second **TARGET** is found, 2 is stored and so on. If no **TARGET** is found, 0 is stored.

If you specify **FOUND**, the **VARSCAN** command issues a return code of 0 regardless of whether a **TARGET** is found, checking the **FOUND** variable be necessary to determine success or failure. If **FOUND** is not specified, the return code 4 is issued in the event of no match being found and **ACTION(RC)** being specified.

INSTANCE(<n>|1)

Specifies the instance of a potentially repeating match, for example **TARGET(Thunderbird) INSTANCE(4)** returns the fourth occurrence of Thunderbird found in the text. If you specify multiple **TARGET** values, it will count the overall matches regardless of the **TARGET**. For example, **TARGET(Scott,Virgil,Gordon,Alan,John) INSTANCE(4)** returns a pointer to **Virgil** from the string **"John, Alan, Gordon, Virgil, Scott"**. To find the fourth instance of Virgil, you need to specify **VARSCAN** with a single **TARGET**.

ROWS(CURRENT|FIRST|LAST|<n>,[endrow])

Sets the limit of the rows to search. The **CURRENT** keyword represents the current cursor position, **FIRST** is the first row of the object and **LAST**is the last row of the object. Specifying **FIRST** allows the search to start from row 1 regardless of **CURSOR** position. Specifying 1 as the starting row will honor the **CURSOR** position. Specifying **CURRENT**, **FIRST** or **LAST** alone will restrict the search to only that row, while combinations will search a range. For example, **ROWS(FIRST,CURRENT)** searches from the first row to the current cursor row.

TARGET(<target1>,[target*n*])

Specifies the string or strings for which to search. You can specify a list of strings separated by a comma or a single string. For example, **TARGET(alpha,beta)** searches for either alpha or beta in the object. **TARGET(alpha)** searches only for alpha.

To search for a target containing commas specify 2 commas, where one is needed in the target. For example, **TARGET(alpha,,beta,gamma)** searches for "alpha,beta" and "gamma" as two separate target strings.

Use the **VARSCAN** command to search an **OBJECT** variable containing file type of data. You can use the command to simply detect whether a single string occurs anywhere within the object, or perform more complex multi-pass, multi-target searches. You can limit the search to a specific set of ROWS and COLS, and use **CURSOR** positioning for subsequent searches.

If you do not set the **FILTER** keyword, the search operates in *scan mode*, meaning that using **CURSOR** variables enables the command to be called repeatedly and to find each successive occurrence of the **TARGET**. If you set the **FILTER** keyword, the search operates in *filter mode*, meaning that each row is compared with the **FILTER** criteria to be processed or not.

In *scan mode* the **VARSCAN** command would typically be in a **DO FOREVER** loop, with an **ACTION(LEAVE)** keyword. This means that the majority of the loop is processed only for matching rows, and the loop ends when no match is found.

In *filter mode* the **VARSCAN** command would typically be in an iterative **DO FOREVER** loop that processes every row of the object (for example, the command **DO row = 1 TO !@MYOBJ**). The **ACTION** would be **ITERATE**, meaning that the main loop does not process any rows that does not match the **FILTER**.

Examples

The following command searches for the string **Wally** within object **MYOBJ**:

```
VARSCAN MYOBJ TARGET(Wally) CASE(N)
```

The following command searches for the string **Wally** within object **MYOBJ**. If **Wally** is found as an exact match, the return code 0 is issued; otherwise the return code 4 is issued (for example, if **WALLY** is found):

```
VARSCAN MYOBJ TARGET(Wally) CASE(N)
```

The following command searches for the string **Wally** or **Gromit** within object **MYOBJ** and sets variable **RESULT** to show which is found, if any. If **Wally** is found, **RESULT** shows 1; if **Gromit** is round **RESULT** shows 2; if neither is found **RESULT** sows 0. Return code 4 is not issued because you specified **FOUND**:

```
VARSCAN MYOBJ TARGET(Wally,Gromit) CASE(N) FOUND(RESULT)
```

The following command searches for the string **Wally** on every row between columns 10 and 30. The **TARGET** string must be contained entirely within those columns, therefore if **Wally** starts in column 26 the match is found, if it starts in column 27 the match is not found:

```
VARSCAN MYOBJ TARGET(Wally) COLS(10,30)
```

The following command searches for the string **Wally** only from the second to fifth row of the file:

```
VARSCAN MYOBJ TARGET(Wally) CASE(N) ROW(2,5)
```

The following command searches for the string **Wally** and sets the variables **WALROW** and **WALCOL** with the row and column where the string was found, respectively. If no occurrences are found, **RESULT** shows 0, **WALROW** is set to the number of records in **MYOBJ + 1**, and **WALCOL** is set 0. If the same command is repeated, **WALROW** and **WALCOL** are set to the second occurrence of the found string.

```
VARSCAN MYOBJ TARGET(Wally) CASE(N)
              CURSOR(WALROW,WALCOL) FOUND(RESULT)
```

The following command searches for the string **Wally**, then checks if **Gromit** exists in the same row as **Wally**. To do this, the second command must either use the same **CURSOR** variables as the first, or use variables that have been copied from the first command.

```
VARSCAN MYOBJ TARGET(Wally) CASE(N)
              CURSOR(WALROW,WALCOL) FOUND(RESULT)
VARSCAN MYOBJ TARGET(Gromit) CASE(N) ROWS(CURRENT)
              CURSOR(WALROW,WALCOL) FOUND(RESULT)
```

The following command searches for the string **one** within the object, meaning that would also match with stone, bone, bones, tones, and similar:

```
VARSCAN MYOBJ TARGET(one) CASE(N)
```

To search only for the characters **one**, use the following command:

```
VARSCAN MYOBJ TARGET(one) CASE(N) DISTINCT(Y)
```

The following command finds every row containing the string **oh** in any case combination. The **LEAVE** action ensures that the loop stops when all the occurrences are found:

```
DO FOREVER
   VARSCAN MYOBJ TARGET(Oh) CURSOR(RX,CX) DISTINCT(Y)
                 CASE(N) COLS(001,080) ACTION(LEAVE)
   WRITE OUTDATA "@V(@MYOBJ-!RX)"
END
```

The following command processes all the records in the object, but skips to the next iteration of the loop if the string **oh** is not found in the row:

```
DO FOREVER
   VARSCAN MYOBJ TARGET(Oh) CURSOR(RX,CX) DISTINCT(Y)
                 CASE(N) COLS(001,080) ACTION(LEAVE)
   WRITE OUTDATA "@V(@MYOBJ-!RX)"
END
```

The following command returns only rows that begin with **F**, **A**, or **B**:

```
DO RX = 1 TO !@MYOBJ
   VARSCAN MYOBJ TARGET(F,A,B) CURSOR(RX,CX) FILTER(EQ)
                 CASE(N) COLS(001,001) ACTION(ITERATE)
   WRITE OUTDATA "@V(@MYOBJ-!RX)"
END
```

The following command returns only rows that do not begin with **S**, **I**, nor **G**:

```
DO RX = 1 TO !@MYOBJ
   VARSCAN MYOBJ TARGET(S,I,G) CURSOR(RX,CX) FILTER(NE)
                 CASE(N) COLS(001,001) ACTION(ITERATE)
```

```
   WRITE OUTDATA "@V(@MYOBJ-!RX)"
END
```

Troubleshooting scanning

If **VARSCAN** cannot find what you are searching for, use the **OPTIONS TRACE(1)** command as in the following example:

```
EQQI200I VARSCAN MYOBJ TARGET(Is it far,high noon) CURSOR(RX,CX)
EQQI200I FOUND(RESULT) CASE(N) INSTANCE(7)
```

The tracing command will show the following information:

- The row and column limits used (EQQI953A).
- The cursor starting position (EQQI954A).
- Because you used **INSTANCE**, you are shown the instance number, target number, row and column of each instance found before the requested instance is reached (EQQI954A).
- The full text of the row on which the target was found (EQQ951A).

```
EQQI952A >>Row range 1 to 101
EQQI952A >>Column range 1 to unlimited
EQQI953A >>Cursor position Row=1, Column=0
EQQI954A >>Instance=1 Target=2 Row=1 Column=1
EQQI954A >>Instance=2 Target=1 Row=8 Column=1
EQQI954A >>Instance=3 Target=1 Row=8 Column=12
EQQI954A >>Instance=4 Target=1 Row=8 Column=23
EQQI954A >>Instance=5 Target=1 Row=29 Column=1
EQQI954A >>Instance=6 Target=1 Row=29 Column=12
EQQI064A Target "Is it far" found on row 29 column 23
EQQI951A >>Is it far, is it far, is it far?
EQQI033A Variable RESULT set to "1"
EQQI033A Variable RX set to "29"
EQQI033A Variable CX set to "23"
EQQI908I >>LAST_RC=0 LAST_XRC=0 MAX_RC=0 MAX_RESP=0
EQQI909I >>LEVEL=1 NUM=14 PARENT=SYSIN REF=SYSIN.7
EQQI299I Statement completed - RC=0
```

VARSET – Set a Workload Automation Programming Language variable

Use the **VARSET** command to set a Workload Automation Programming Language variable, work with it, and save it in a JCL variable table in HCL Workload Automation for Z.

```
VARSET variable = <expression>
```

or

```
VARSET variable [VALUE(value)]
                [USRF(user-field-name)]
                [@V(input-variable-name)]
                [SYMBOL(system-symbol-name)]
                [OBJECT(<object>)]
                [MISSING(ERROR|FAIL|NULL|ASIS)]
                [ENVATTR(major,minor,item)]
```

```
[ADDWORD(string)]
[APPEND(string)]
[BEFORE|BEFOREX(string)]
[CLEAN(n[,characters])]
[CONCAT(string)]
[LEFT(n)]
[RIGHT(n)]
[SUBSTR(start[,length])]
[WORD(n,[ALL|BASIC|COMMA|NONE|PERIOD|SPACE],[<other>])]
[UPPER(YES|NO)]

[DELTA(n)]

[TABLE(table)]
[SAVE(YES|NO)]
[SETUP(YES|NO|PROMPT)]

[OBJECT(<file-object>)]
[CURSOR(<rowvar>,<colvar>)]
[POSITION([+|-]row|=,[+|-]col|=,[+|-]end)]
```

The **VARSET** command can be either a simple REXX style assignment or a keyword-driven process. You can also use the alternative command name **SETVAR**.

The REXX style is identified by an equal sign (=) after the variable name. The characters following the equal sign (=) must be a standard REXX expression and can use REXX functions. For more details about the REXX expressions and available functions, see the *TSO/E REXX Reference* manual.

The first keyword of each **VARSET** command must be the variable to which the command refers. All other keywords are optional. A **VARSET** command without optional keywords, sets the variable to a null value. If the variable was already set or referenced in the Workload Automation Programming Language statements, that value is kept and modified by any following keyword.

The following keywords are optional:

- **Data sources**

  VALUE(value)

  Sets the value of the variable within Workload Automation Programming Language. The **VALUE** keyword alone does not update any JCL variable table.

  For example, **VARSET MYVAR VALUE(Hello World)** causes any subsequent references to **!MYVAR** to be replaced with **Hello World**.

  The **VALUE** keyword can contain a reference to another or the same variable, so that loads that variable at that specific point in your Workload Automation Programming Language statements.

  For example, **VARSET MYADID VALUE(!OADID)** causes **MYADID** to contain the name of the currently running application.

USRF(user-field-name)

Sets the variable to the value of an operation user field with the same name, if attached to the operation running the Workload Automation Programming Language job. Workload Automation Programming Language first searches for a match using the same case specified in **USRF**, then looks for a match regardless of the case. The **USRF** keyword is not available for the versions of HCL Workload Automation for Z without the user field functionality.

@V(input-variable-name)

The **@V** keyword allows you to specify another Workload Automation Programming Language variable and provide it with a value. Because the content of this keyword is the name of the variable, you do not need to specify the variable prefix (such as the exclamation mark **!**), instead you can use variable names to provide all or part of the name, allowing subscripting to be performed.

For example, in the command **VARSET JOB VARIABLE(@MYOBJ-ADOP-!X-ADOPJN)** the variable **!X** is used to provide the segment sequence number in an object variable, allowing all operations to be looped through.

SYMBOL(system-symbol-name)

Sets the variable to the value of a system symbol. If a symbol with that name is not found, an empty value is returned.

OBJECT(<object>)

Indicates a file object, such as the one created with the **READ** command, to be used as input to set a variable. Omit the at sign (@) because only object names are allowed. The object must already exist and have sequential file type. Complex objects, such as those generated by **LIST** or **SELECT** command, are not allowed.

You can use the **OBJECT** keyword together with the **POSITION** keyword, and optionally with the **CURSOR** keyword.

MISSING(ERROR|FAIL|NULL|ASIS)

Determines what to do if a field specified in the **USRF** keyword is not attached to the current operation. The **MISSING** keyword must be specified after a **USRF** keyword. If you specify it before, the **VARSET** command fails.

Valid values are:

ERROR

Workload Automation Programming Language issues an error (RC=8) if the user field is not found.

FAIL

Workload Automation Programming Language fails (RC=12) if the user field is not found.

NULL

The variable is set to a null value if the user field is not found.

ASIS

The value is left unchanged if the user field is not found. The **VALUE** keyword sets a default value, and the **USRF** keyword requires only that you set the user field on the operation if you want to override the default behavior (this is the default behavior for Workload Automation Programming Language).

Note: **ERROR** issues RC=8, which by default allows the following commands to be run; **FAIL** issues RC=12, which by default stops processing further commands.

For example, in the command **VARSET USERNAME VALUE(RANA) USRF(OVR_USERNAME)**, the variable **USERNAME** is set to RANA, unless the user has added a User Field to the running job named **OVR_USERNAME**. In this case, the value in the User Field is kept.

In the command **VARSET USERNAME USRF(User Name) MISSING(FAIL)**, the variable **USERNAME** is set to the contents of a User Field named **User Name**. If not found, the command fails.

ENVATTR(major,minor,item)

Returns information about the running job environment. You can specify up to 3 levels of argument to identify the information that you require:

major

The part of the environment from which the data is extracted. Valid values are:

- **CMD**, for information about Workload Automation Programming Language commands.
- **DD**, for information about DD statements in the running step.
- **JCL**, for information about steps in the JCL.
- **OPTIONS**, for values of a Workload Automation Programming Language **OPTIONS** keyword.

minor

The type of information you want, which varies depending on **major**.

item

Key information to identify the item from which you require information.

For **CMD**, the item can be one of the following:

- The label of the command.
- The absolute command number, for example 1 is the command in the current step.

- The relative command number, for example -1 is the command before the current command.
- Special command labels:

  `LAST_RC`

  The previous command.

  `LAST_XRC`

  The last command that was actually run.

  `MAX_RC`

  The command that issued the highest return code.

  `MAX_RESP`

  The command that issued the highest response code.

For **`DD`** the item can be one of the following:

- A DD name. If the DD name has multiple data sets coded, a fourth argument can be coded to identify the number of the DD statement. For example, **`ENVATTR(DD,DSNAME,MYDD,3)`** refers to the 3rd data set on the *MYDD* concatenation. If omitted, the first DD statement for the DD name is returned.
- Absolute DD number. For example, 1 is the first DD statement in the current step.
- Relative DD number. For example, -1 is the last DD statement in the current step.

For **`JCL`**, the item can be one of the following:

- **`PROCSTEP`** or **`STEPNAME.PROCSTEP`** values tTo specify the step. If duplicates occur, the latest name is used.
- Absolute step number. For example, 1 is the first step in the JCL.
- Relative step number. For example, -1 is the step before the current step, 0 is the current step.
- Special step labels:

  `LAST_RC`

  The previous step.

  `LAST_XRC`

  The last step that was actually run.

  `MAX_RC`

  The step that issued the highest return code, or most recent ABEND.

For **`OPTIONS`**, there is no **`item`** value except when you set **`minor`** to **`SPE`**; in this case, **`item`** can be the name of the **`SPE`**.

Table 164: Valid combinations for ENVATTR on page 354The valid combinations are listed in the following table:

Table 164. Valid combinations for `ENVATTR`

Major	Minor	Item	Description
`CMD`	`#`	no	Number of commands run so far, including currently running command.
`CMD`	`ARGS`	yes	Argument of the identified command.
`CMD`	`LABEL`	yes	Label of the identified command.
`CMD`	`LEVEL`	yes	Message level of the identified command.
`CMD`	`NAME`	yes	Name of the identified command.
`CMD`	`RC`	yes	Return code or response code of identified command (whichever is highest).
`DD`	`#`	no	Number of named DD statements in the step.
`DD`	`#`	yes	Number of DD statements for the identified DD name.
`DD`	`DDNAME`	yes	DD name of the identified DD statement or concatenation.
`DD`	`DSNAME`	yes	Data set name of the identified DD statement. If the DD statement refers to a member in a library, this includes both the data set name and library name.
`DD`	`LIBRARY`	yes	If the identified DD statement refers to a member in a library, this is the data set portion of the name, without the member name. If this is an ordinary data set reference, this value is the same as `DSNAME`.
`DD`	`MEMBER`	yes	If the identified DD statement refers to a member in a library, this is the member name. If this is an ordinary data set reference, this value is blank.

Major	Minor	Item	Description
`JCL`	`#`	no	Number of steps in the job, including the currently running step.
`JCL`	`STEP`	yes	Fully qualified step name of the identified step. If this is a step in a proc, the step name is composed by the step calling the procedure and the step within the procedure that is running, separated by a period. For example, STEPNAME.PROCSTEP. If this is a step running directly within the job JCL, this value is only the single step name without period.
`JCL`	`STEPNAME`	yes	The name of the step calling the procedure, if this step is within a procedure. Otherwise, this value is blank.
`JCL`	`PROCSTEP`	yes	The name of the step actually running the program.
`JCL`	`RC`	yes	The return code of the step.
`JCL`	`PGM`	yes	The name of the program for the current step.
`JCL`	`PARM`	yes	The PARM value for the current step.
`OPTIONS`	`<keyword>`	no	The value of the specified `OPTIONS` keyword.
`OPTIONS`	`SPE`	yes	Whether the SPE has been activated (Y or N).

- **Text modifiers**

  `ADDWORD(<string>)`

  Adds the string of words, separated by blanks, to the end of the current value, if they do not already exist in the list.

AFTER(<string>)

Returns the text following the specified string.

APPEND(<string>)

Adds the contents of the keyword to the end of the current value, separated by a blank.

BEFORE|BEFOREX(<string>)

Returns the text up to the first occurrence of the string. If the string is not found, the whole text is returned. The `BEFOREX` keyword returns only the text before the string. If the string is not found, nothing is returned. This makes it possible to collect the text within delimiters.

In the following example, **MEM1** contains text even if only the left parenthesis **(** is specified in the input variable, while **MEM1X** contains text if both the left and right parenthesis **(** and **)**are set in the input variable.

```
VARSET MEM1 @V(FILE1) AFTER("(") BEFORE(")")
VARSET MEM1X @V(FILE1) AFTER("(") BEFOREX(")")
```

CLEAN(n[,characters])]

Enables a set of cleanup actions to be performed upon the content at that point. Each action can be performed individually by selecting the action number, or together by adding the action numbers together. To perform the action upon other characters, specify them as a second argument. If you add actions together and also set a second argument, all the actions are applied.

- Action 1 – Strips leading characters from the string (the default is blanks). If multiple characters are specified, they are removed from the beginning of the string one at a time in the order specified.
- Action 2 – Strips trailing characters from the string (the default is blanks). If multiple characters are specified, they are deleted from the end of the string one at a time in the order specified.
- Action 4 – Equalizes blanks (the default is one blank). The second argument for this action is the number of blanks to equalize to.
- Action 8 – Deletes punctuation characters when followed by blanks (default . , ! and ?). The characters are removed instead of changing them to another character.
- Action 16 – Translates characters to blanks unconditionally, regardless of what follows them (default is comma).
- Action 32 – Removes specified characters unconditionally, regardless of what follows them. There is no default for this action, therefore a second argument must be specified.

The cleaning actions are designed to remove extraneous characters and rationalize the layout of fields.

CONCAT(<string>)

Adds the contents of the keyword to the end of the current value, without any blank.

LEFT(*n*)

Sets the ***n*** left most characters of the current variable value as the new value of the variable.

For example, `VARSET MYPFX VALUE(!OADID) LEFT(4)` causes `MYPFX` to contain the first 4 characters of the application name.

You can use `LEFT` and `RIGHT` together in the same statement to extract characters relative to the end of value of unknown length. For example, `VARSET MYPFX VALUE(!OADID) RIGHT(4) LEFT(1)` causes `MYPFX` to contain the 4th from last non-blank character of the application name.

`RIGHT(n)`

Sets the *n* right most non-blank characters of the current variable value as the new value of the variable.

For example, `VARSET MYPFX VALUE(!OADID) RIGHT(4)` causes `MYPFX` to contain the last 4 characters of the application name, regardless of how long the application name actually is.

You can use `RIGHT` and `LEFT` together in the same statement to extract characters relative to the end of value of unknown length. For example, `VARSET MYPFX VALUE(!OADID) RIGHT(4) LEFT(1)` causes `MYPFX` to contain the 4th from last non-blank character of the application name.

`SUBSTR(start[,length])`

Allows you to extract a specific portion of the current value of a variable, by specifying the start position and optionally the length. For example, `VARSET MYBIT VALUE(!OADID) SUBSTR(2,1)`.

If the length is omitted, the value is set to the remaining part of the current value from the start position onwards.

For example, `VARSET MYBIT VALUE(!OADID) SUBSTR(3)` sets `MYBIT` to be the application name, minus the first 2 characters.

`WORD(n,[ALL|BASIC|COMMA|NONE|PERIOD|SPACE],[<other>])]`

Extracts individual words from a variable value. The number identifies which word to extract.

For example, `VARSET MYVAR VALUE(The quick brown fox) WORD(2)` returns `quick`.

Negative numbers extracts words from the end of the phrase. -1 identifies the last word, -2 the 2nd to last, and so on.

For example, `VARSET MYVAR VALUE(The quick brown fox) WORD(-2)` returns `brown`.

By default, words are considered to be space delimited, the optional second and third parameters can be used to specify alternative delimiters.

The second parameter can be one of the following:

`ALL`

Uses a space, comma, and period as delimiters.

`BASIC`

Uses a space and comma as delimiters.

COMMA

Uses a comma as delimiter.

NONE

No predefined delimiters are used (the third parameter is used).

PERIOD

Uses a period as delimiter.

SPACE

Uses a space as delimiter.

The third keyword can be used to specify any additional characters. For example, **WORD(2,COMMA,;:)** looks for the second word delimited by comma, semi-colon and colon.

WORD(3,,/) looks for the third word delimited by a slash (/).

Consecutive delimiters without intervening characters return null words for their positions. For examples:

VARSET MYFIRST VALUE(A,,C,,E,F) WORD(1,COMMA) returns **A**

VARSET MY5TH VALUE(A,,C,,E,F) WORD(5,COMMA) returns **E**

VARSET MYLAST VALUE(A,,C,,E,F) WORD(-1,COMMA) returns **F**

UPPER(YES|NO)

Forces the value into upper case. This is useful if the value will eventually be used in something that requires upper case (for example, JCL), but the source of the value cannot be depended upon to have forced upper case. For example, a Variable table or User field.

UPPER(YES) forces upper case; **UPPER(NO)** leaves the value a sis. The **UPPER** keyword without arguments defaults to **UPPER(YES)**.

For example, **SETVAR USERNAME VALUE(RANA) USRF(OVR_USERNAME) UPPER**

- **Mathematical modifiers**

DELTA(*n*)

Takes the current value and adds *n* to it. The current value must be numeric and *n* must be numeric, otherwise the command fails. The only exception is when the variable is set to a null value, which is considered zero. This allows for new counter variables to be self-initializing.

You can use this keyword to increment variables within JCL variable tables. In the following example, the table **MYTABLE** is opened. Before the **VARSET** command is run, the value of **RUNCOUNT** is substituted into the **VALUE** keyword. Then, the **VARSET** command adds **1** to the current value and saves it into **MYTABLE**. If **RUNCOUNT** does not already exist in **MYTABLE**, the **VARFAIL(NULL)** results in the **VALUE** keyword containing a

null value, which the **DELTA** process considers as zero. The end result being **RUNCOUNT** contains **1** and is added to **MYTABLE**.

```
VARSUB SCAN(!) TABLE(MYTABLE) VARFAIL(NULL)
VARSET RUNCOUNT VALUE(!RUNCOUNT) DELTA(1) SAVE(YES)
```

- **Variable management**

TABLE(table)

Assigns a JCL Variable Table to a Workload Automation Programming Language variable, so that it can be saved in a JCL Variable Table. If the variable has already been referenced from a JCL Variable Table, the **TABLE** keyword is not needed. If a Workload Automation Programming Language variable is *not* assigned to a table name, it cannot be saved.

In the following example, **RUNCOUNT** could not exist in **MYTABLE**, but **VARFAIL(NULL)** causes the first table in the search sequence to be assigned as the source table name. You can, however, use the **TABLE** keyword to override the name of the table from which the variable originally came, and save the value in a new table.

```
VARSUB SCAN(!) TABLE(MYTABLE) VARFAIL(NULL)
VARSET RUNCOUNT VALUE(!RUNCOUNT) DELTA(1) SAVE(YES)
```

SAVE(YES|NO)

Determines if and when the variable is written to a JCL variable table:

YES

The variable is written directly into the associated table at this point.

NO

The variable is not written into the associated table, but could be written later if a **VARSAVE** command is run by referencing the table associated to this variable.

If you specify **SAVE** without argument, **SAVE(YES)** is assumed.

Note:

1. To set several variables in the same table, it is more effective to use a subsequent **VARSAVE** command than using **SAVE** on every **VARSET**.
2. Variables do not need to already exist within a table. They are added to the table, if needed.
3. Tables to not need to exist to have variables saved to them. If a table does not exist, it is created automatically, using **OPTIONS OWNER** to set the Owner ID. The description is set to the creating Job name, Jes Number, and current date.

SETUP(YES|NO|PROMPT)

Sets the **SETUP** value for any new variables when saved into a JCL Variable Table:

YES

Resolve variable at SETUP phase.

NO

Resolve variable at SUBMIT phase (default).

PROMPT

Promptable variable.

You can change the default with **OPTIONS SETUP**.

- **Object processing**

CURSOR(<rowvar>,<colvar>)

Names the variables used in a previous **VARSCAN** command to establish the row and column where the string is located. Use the **POSITION** keyword to offset from the cursor position to collect data from the same or adjacent rows.

Both arguments in this keyword must point to existing variables that contain integers.

POSITION([+|-]row|=,[+|-]col|=,[+|-]end)

Enables data to be collected from a row in an object variable. You must set an integer for row, column, and end positions by setting three arguments as follows:

1. The first argument sets the row from which to collect data. Specify an integer (for example, **1** to collect data from row 1), or a relative number with respect to the **CURSOR** (for example, **-1** or **+2** to collect data from 2 rows before or after the **CURSOR**, respectively). Use the equal sign **=** to collect data from the same row as the **CURSOR**).

   If a negative offset results in a row before the start of the file, an empty row is returned. If a positive row is generated beyond the end of the object, an empty row is returned and a warning message issued (RC=4).

2. The second argument sets the column from which to collect data. Specify an integer (for example, **10** to collect data from column 10), or a relative number with respect to the **CURSOR** (for example, **-10** or **+10** to collect data from 10 characters before or after the **CURSOR**, respecitvely). Use the equal sign **=** to collect data from the same column as the **CURSOR**.
3. The third argument sets the column where to end collecting data. Specify an integer (for example, **72** to end collecting at column 72) or a relative number with respect to the **CURSOR** (for example, **-10** or **+10** to end collecting data from 10 characters before or after the **CURSOR**, respectively). Use the equal sign **=** to end collecting data at the same column as the **CURSOR**. If you do not specify any value for the third argument, data is collected until the end of the row.

After data is extracted from the object with the combination of the **OBJECT**, **CURSOR** and **POSITION** keywords, the value can be further refined by using the **Text Modification** keywords.

Use the **VARSET** command set, modify, and optionally save a variable. It is a progressive command where the order the keywords is important for the result, because each keyword is applied to data in the order you specified.

Use Data source keywords first, and possibly Object processing keywords, before being passed into either Text or Mathematical modifiers to manipulate the value. Finally, use the Variable management keywords to determine how and where the variable is to be stored.

You can set any number of modifier keywords, always remembering that the order is important. For example, the following command returns **23**:

```
VARSET MYVAR VALUE(12345678) LEFT(3) RIGHT(2)
```

The following command returns **78**:

```
VARSET MYVAR VALUE(12345678) RIGHT(2) LEFT(3)
```

Examples

The following command sets the Workload Automation Programming Language variable **MYVAR** to **Hello**, which can be used for the remainder of this Workload Automation Programming Language step.

Alternatively **VARSET MYVAR = "Hello"** does the same thing, but this syntax does not have access to any other **VARSET** features.

```
VARSET MYVAR VALUE(Hello)
```

The following command sets:

- **VAR1** to **ABC123**
- **VAR2** to the value of **VAR1**, then takes the 3 leftmost characters, resulting in **ABC**
- **VAR3** to the value of **VAR1**, then takes the 3 rightmost characters, resulting in **123**

```
VARSET EMAIL VALUE(don.lexer@abc.com)
VARSET USER @V(EMAIL) BEFORE(@)
VARSET DOMAIN @V(EMAIL) AFTER(@)
```

The following command returns **MY.DSN** in DSN and MEMBER in **MEM**. If you did not set the right parenthesis **)** the **MEM** variable would be blank.

```
VARSET FILE = "MY.DSN(MEMBER)"
VARSET DSN  @V(FILE) BEFORE("(")
VARSET MEM @V(FILE) AFTER("(") BEFOREX(")")
```

The following command sets the Workload Automation Programming Language variable **MYVAR** to Hello. The variable is then assigned to JCL variable table **MYTAB** and saved in the database.

```
VARSET MYVAR VALUE(Hello) TABLE(MYTAB) SAVE(YES)
```

The following command sets multiple variables to be saved in the same table. Because a JCL variable table is a single record, when you update one variable the entire record is replaced. If you update multiple variables in the same table, it is more efficient to assign them to a table with the **VARSET** command, but not **SAVE** them on the **VARSET** command, instead having a separate **VARSAVE** command to update the table record with all the variables at once.

```
VARSET NAME1 VALUE(Tom) TABLE(MYTAB)
VARSET NAME2 VALUE(Dick) TABLE(MYTAB)
VARSET NAME3 VALUE(Harriet) TABLE(MYTAB)
VARSAVE MYTAB
```

The following command removes commas from a big number and returns `1000000`:

```
VARSET BIGNUM = "1,000,000"
VARSET CLNNUM @V(BIGNUM) CLEAN(32,,)
```

The following command searches for the data set name of the file allocated to DD `INFILE`:

```
VARSET DSNAME ENVATTR(DD,DSNAME,INFILE)
```

VARSUB – Control variable substitution

Use the `VARSUB` command to control how variable substitution is performed within Workload Automation Programming Language. You can specify `VARSUB` commands at any point in the Workload Automation Programming Language command stream, except within a Batch Loader construct for an object.

```
VARSUB [SCAN[(prefix,suffix)]|NOSCAN] [CLOSE[(table)]] [SEARCH(list)]
       [TABLE(table)] [VARFAIL(ASIS|FAIL|NULL)]
```

`SCAN[(prefix,suffix)]`

Activates Workload Automation Programming Language variable substitution. Without any arguments, the variable prefix is set to an exclamation mark (!) and the suffix to a period (.). For example, `VARSUB SCAN` results in a variable convention like `!MYVAR.`

By setting prefix and suffix, you can modify the way variables are specified. For example, `VARSUB SCAN(<,>)` results in a variable convention like `<MYVAR>.`

Note: If you use the same character for prefix and suffix, you must specify both the prefix and suffix each time a variable immediately follows another variable, such as in `%MYVAR1%%MYVAR2%`

When a `VARSUB SCAN` statement is found, it automatically opens the Application and Global tables.If you want to open only specific tables, use the `SEARCH` keyword before the `SCAN` keyword, as in the following example:

```
VARSUB SEARCH(MYTABLE,NOAPPL,NOGLOBAL) SCAN
```

`NOSCAN`

deactivates Workload Automation Programming Language variable substitution. The table search sequence is left as it is; if variable substitution is activated again in the statements, the same search sequence is used.

If substitution is being deactivated and you don not plan to activate it again in the Workload Automation Programming Language statements, close all open tables by using the `CLOSE` keyword in conjunction with `NOSCAN`, as in the following example:

```
VARSUB NOSCAN CLOSE
```

CLOSE[(table)]

Closes a table and remove it from the search sequence. The variable definitions and any dependencies are dropped from storage, so it is good practice for systems with large tables or many dependent variable values. It does *not* drop the value of any variable that has already been referenced from the closed table, those values remain accessible for the remainder of the Workload Automation Programming Language process.

You can close a single table, for example **VARSUB CLOSE(MYTABLE)**, or close all open tables, including the Application and Global tables, by not specifying a table name.

If you need to close multiple tables, the **CLOSE** keyword can be repeated, for example **VARSUB CLOSE(MYTABLE1) CLOSE(MYTABLE2)**

SEARCH(table,table,table,APPL|NOAPPL,GLOBAL|NOGLOBAL)

Sets the order in which tables are searched for variables. By default, Workload Automation Programming Language first searches tables opened with **TABLE** keywords, starting with the most recently opened. Then it searches the Application table (if one is specified for the occurrence running the Workload Automation Programming Language job) and the Global table.

If not specified in the **SEARCH** keyword, the Application and Global tables are automatically added as the last 2 tables in the sequence. Specifying **NOAPPL** means that the Application table is not searched, and specifying **NOGLOBAL** means that the Global table is not searched.

For example:

- **VARSUB SEARCH(MYTABLE1,MYTABLE2)** sets the search sequence to MYTABLE1, then MYTABLE2, then the Application table, followed by the Global table.
- **VARSUB SEARCH(MTABLE1,APPL,MYTABLE2)** sets the search sequence to MYTABLE1, then the Application table, then MYTABLE2, followed by the Global table.
- **VARSUB SEARCH(MTABLE1,APPL,MYTABLE2,NOGLOBAL)** sets the search sequence to MYTABLE1, then the Application table, then MYTABLE2. The Global table is not searched.

Tables already open but not specified in the **SEARCH** keyword, are automatically closed. Tables that were not open and are specified in the **SEARCH** keyword, are automatically opened.

VARFAIL(ASIS|FAIL|NULL)

Defines what to do if a variable referenced in a Workload Automation Programming Language statement is not found:

ASIS

The variable is left unresolved in the Workload Automation Programming Language statement.

FAIL

The parsing for the statement fails and it is not performed (default).

`NULL`

The variable is considered to have a null value and is assumed to belong to the first table in the table search sequence, at the point where it is referenced.

`TABLE(table)`

Opens a table and add it to the front of table search sequence. Opening a table loads the definition, including the dependencies, of each variable within the table, but it does not set any values of Workload Automation Programming Language variables. This occurs only when a variable is referenced within a Workload Automation Programming Language statement.

With the `TABLE` keyword you can open only one table at a time. To add multiple tables to the search sequence, you can repeat the `TABLE` keyword as in the following example:

```
VARSUB TABLE(MYTABLE1) TABLE(MYTABLE2)
```

Note: Because Workload Automation Programming Language first searches the latest opened table, the above command results in `MYTABLE2` being searched before `MYTABLE1`.

Consider that:

- The sequence of keywords on the `VARSUB` statement does affect the result of the command.
- `VARSUB SCAN` turns on variable substitution for subsequent statements. Variables cannot be used within the same `VARSUB` statement that uses the `SCAN` keyword. Hence, `VARSUB SCAN TABLE(MYTABLE)` is valid, while `VARSUB SCAN TABLE(!OADID)` is not valid, because it uses a the variable `!OADID`. These should be coded on separate statements, as in the following example:

```
VARSUB SCAN(!)
VARSUB TABLE(!OADID)
```

Chapter 12. Performance considerations

With Workload Automation Programming Language you can use different methods to perform the same action. Each method has different implications for ease of use and performance, the balance of which might change depending on the volume of records to which it is being directed.

Workload Automation Programming Language is designed to give you access to HCL Workload Automation for Z data as easy as possible. This means that some default **OPTIONS** are focused on ease of use, not speed of execution; for simple and small processes, this does not imply great difference.

The following topics outline the areas of Workload Automation Programming Language that can have an impact on performance, and provide you with recommendations for running large scale more efficiently.

Knowing the context of the current plan

To run, many Workload Automation Programming Language processes need to know the application ID, IA date and time, and operation number of the occurrence that is controlling the Workload Automation Programming Language job. When required, Workload Automation Programming Language attempts to find itself in the current plan so that it can perform these processes.

In details, Workload Automation Programming Language needs to know the following information:

- Access to the supplied variables that reference the occurrence or operation.
- Access to the user fields related to the operation.
- Date calculations that need to know the calendar in use.
- Actions that need to update the operations in the same occurrence (for example, using the = = operator to reference the current application and input arrival).
- Actions that read variables from tables, which need to know the name of the table referred by the occurrence.

If you access any of these functions, Workload Automation Programming Language attempts to find itself in the current plan by searching for an operation in Started status with a job of the same job name and JES number. In big current plans, this can generate large CPU consumption; to prevent the search, you can provide the information in advance.

For the most efficient use of system resources, it is strongly recommended that you use one of the following options in all your Workload Automation Programming Language jobs:

For jobs scheduled by HCL Workload Automation for Z

Set the EQQCPOP DD statement as in the following example:

```
//*%OPC SCAN
//RUNWAPL  EXEC EQQYXJPX,SUBSYS=INDC
//EQQCPOP  DD *
&OADID &OYMD1&OHHMM &OOPNO
//SYSIN    DD *
SHOW OPTIONS
```

Alternatively, you can use the **OPID OPTIONS** keyword in the ARGS symbolic parameter

```
//*%OPC SCAN
//RUNWAPL  EXEC EQQYXJPX,SUBSYS=INDC,
//         ARGS='OPID(&OADID &OYMD1&OHHMM &OOPNO)'
//SYSIN    DD *
SHOW OPTIONS
```

For jobs that are not scheduled by HCL Workload Automation for Z

To prevent the job trying to locate itself in the current plan, set one of the following commands:

- ```
  //EQQCPOP DD *
  -
  ```
- `OPTIONS OPID(-)`
- ```
  //*%OPC SCAN
  //RUNWAPL  EXEC
  EQQYXJPX,SUBSYS=INDC
  //SYSIN DD *
  OPTION OPID(-)
  SHOW OPTIONS
  ```

Note: Workload Automation Programming Language searches the current plan only when the information is actually needed, and the Application and Global tables are accessed only when searching for variables. By setting **`EQQCPOP DD`** or **`OPTIONS OPID`** appropriately, you can additionally prevent this search process.

Designing your queries

The **`LIST`** command is the HCL Workload Automation for Z equivalent of an SQL query, which is used to find data within the database and plans. It is important to understand how to design an efficient query, which in turn needs an understanding of the data you are querying.

To reduce the amount of data processing, define your search to a narrow set of criteria. For example, use the **`STATUS`** argument to limit the number of results by considering only the operations in a specific status.

When a large set of results is expected, you can define a series of smaller queries instead of a larger one. For example, split the results alphabetically by listing all output beginning with `A*`, then `B*` and so on.

The Current Plan Operation commands are also a form of query. They perform a **`LIST`**, then filter it down to size. Ensure that you set the identification keywords to specific values, so that the amount of data searched is limited. To further reduce the data processing, use also the filter keywords.

Large volume processing

Workload Automation Programming Language is a REXX based tool, with the processing engine designed for ease of use and flexibility. Hence, for very large scale processes PIF programming or other utilities might provide more efficient solutions, as described hereafter.

To update the Application Description (AD), Operator Instructions (OI) or Run Cycle Groups (RG), the EQQYLTOP batch program is the most efficient tool to use. For all other types of database objects, you can use Workload Automation Programming Language.

To extract large amounts of Batch Loader data (AD, OI, RG) or data from the current plan, the Batch Command Interface Tool is the most efficient tool to use.

For complex large volume processing, or repeated processing, use Workload Automation Programming Language to produce and test a prototype program, which you can run with `OPTIONS MSGLEVEL(5) TRACE(3)` to show each required PIF call, the arguments used, and the segments and records extracted. You can then use this information to design a PIF program to perform the requested function, without the Workload Automation Programming Language overheads. For a description about how to write a PIF program in various languages, the SEQQSAMP library provides you with some samples.

Considerations about REXX compiler

Workload Automation Programming Language is delivered in compiled form, which is not necessarily faster than interpreted REXX.

If you do not have a REXX compiler, you can use the Alternate REXX Library (SEAGALT) to run the compiled REXX. The alternative library transforms the compiled code into a form that is passed to the REXX interpreter for processing; this does not provide a performance improvement with respect to interpreted REXX.

If you have a REXX compiler, use the REXX Compiler Library (SEAGLPA) to see a significant performance improvement. This is recommended for high volume usage of Workload Automation Programming Language.

Record processing

Record processing within Workload Automation Programming Language and `WAPLEXEC` commands can be very time consuming. When processing thousands of records, there are some considerations you can make within Workload Automation Programming Language to reduce the amount of records processed and save run time.

LIST-SELECT Common Segment vs Record

For a record type that can have more than one segment, you can retrieve either the whole record, or just the common segment. For large or complex records, the difference between retrieving the common segment over the entire record could be noticeable when retrieving thousands of records. Hence, if all the information you need is in the common segment, ensure that you use the common segment as the resource when running a `SELECT` command.

If the `SELECT` statements are being generated as a result of a `LIST` statement using `OPTIONS SELECT(Y)` and *only* the common segment is needed, a `LIST` statement is sufficient to extract the data you need. Setting `OPTIONS SELECT(N)` (default) results in smaller records being retrieved and prevents extra requests to HCL Workload Automation for Z for each identified object.

Note: If you need to produce Batch Loader output, you can use only a **SELECT** command and the complete record must be retrieved.

OUTPUT and LOADDEF

Use the **OUTPUT** statement to determine which segments and fields are to be extracted.

When a record is retrieved from HCL Workload Automation for Z using the **SELECT** statement, the entire record is retrieved and the header is processed to identify each segment. The segment is then processed *only* if there is a reference to it. If you code only **OUTPUT** statements for the segments you need, the processing is reduced.

Record processing involves extracting every field in the segment, creating a data output record for the segment, running a Segment Processing Exit (if requested,) and generating Batch Loader. Some of this processing is avoided if you code only **OUTPUT** statements for the fields from which you require data.

Use the **LOADDEF** command to load in-built **OUTPUT** statements. The following example shows how to load definitions of every field for every segment. Use this command only when you need to extract entire objects, for example for Batch Loader generation.

```
LOADDEF *
```

To load **OUTPUT** definitions for particular records, use **LOADDEF** in a more selective way. For example, to load the in-built **OUTPUT** statements for the Application Definition records specify:

```
LOADDEF AD*
```

Restricting **OUTPUT** statements to just the Segments and Fields you need reduces the amount of processing that Workload Automation Programming Language performs, both in the amount of **OUTPUT** statements that must be parsed at startup, and in the amount of segments that are processed when records are retrieved.

Appendix A. Resource reference

Alternative resource names

The following resource names can be used as alternatives to increase the clarity of the command syntax.

Table 165. Alternative resource names

Action	Resource	Alternative name
DELETE	AD	ADCOM APPLICATION APPL SCHEDULE SCHED JOBSTREAM
DELETE	CL	CLCOM CAL CALENDAR
DELETE	CPOC	OCCURENCE OCC
DELETE	CPOP	CPOPCOM OPERATION OP
DELETE	CPPRE	PRED PREDECESSOR
DELETE	ETT	TRIGGER
DELETE	JCLV	JCLVCOM TABLE
DELETE	JS	JSCOM JCL
DELETE	OI	OICOM INSTRUCTION TEXT
DELETE	PR	PRCOM PERIOD
DELETE	SR	SRCOM RESOURCE
DELETE	WS	WSCOM WORKSTATION
INSERT	CPOC	OCCURENCE OCC
INSERT	CPOP	CPOPCOM OPERATION OP
INSERT	CPPRE	PRED PREDECESSOR
LIST	ADCOM	AD APPLICATION APPL SCHEDULE SCHED JOBSTREAM
LIST	CLCOM	CL CAL CALENDAR
LIST	CPOC	OCCURENCE OCC
LIST	CPOPCOM	CPOP OPERATION OP
LIST	CPWSCOM	CPWS
LIST	ETT	TRIGGER
LIST	JCLVCOM	JCLV TABLE

Table 165. Alternative resource names (continued)

Action	Resource	Alternative name
LIST	JSCOM	JS JCL
LIST	OICOM	OI INSTRUCTION TEXT
LIST	PRCOM	PR PERIOD
LIST	SRCOM	SR RESOURCE
LIST	WSCOM	WS WORKSTATION
LIST	WSVCOM	WSV VIRTUAL WORKSTATION DESTINATION
MODIFY	CPOC	OCCURENCE OCC
MODIFY	CPOP	OPERATION OP
MODIFY	IVL	INTERVAL
SELECT	AD	APPLICATION APPL SCHEDULE SCHED JOBSTREAM
SELECT	CL	CALENDAR CAL
SELECT	CPOC	OCCURENCE OCC
SELECT	CPOP	OPERATION OP
SELECT	ETT	TRIGGER
SELECT	JCLV	TABLE
SELECT	JS	JCL
SELECT	OI	INSTRUCTION TEXT
SELECT	PR	PERIOD
SELECT	SR	RESOURCE
SELECT	WS	WORKSTATION

OUTPUT field definition reference

The following text is the field content of the **EQQFLALL** member from the Workload Automation Programming Language that serves as a reference for all available fields.

```
/*-------------------------------------------------------------------+
 | MODULE  : EQQFLALL                                                |
 | PURPOSE : Output definition with all PIF data (no TRL data)       |
 |                                                                   |
 | HISTORY-----------------------------------------------------------|
 |                                                                   |
 +-------------------------------------------------------------------*/
```

```
/*-------------------------------------------------------------------+
 | GROUP=DB       - All Database objects                             |
 +-------------------------------------------------------------------*/
/*-------------------------------------------------------------------+
 | RECORD=AD        - Application description                        |
 +-------------------------------------------------------------------*/
/*-------------------------------------------------------------------+
 | SEGMENT=ADCOM    - Common                                         |
 +-------------------------------------------------------------------*/
OUTPUT ADCOM DATA(OUTDATA) LOADER(OUTBL)
  KEYS(ADID ,                  /* Application ID                     */
       ADSTAT ,                /* Application status                 */
                               /* A=Active, P=Pending                */
       ADTYPE ,                /* Application type                   */
                               /* A=Application, G=Group Def         */
       ADFROM)                 /* Valid-From date                    */

  FIELDS(ADMONITOR ,           /* Monitor AD                         */
         ADTO ,                /* Valid-To date                      */
         ADDESC ,              /* Descriptive text                   */
         ADGROUP ,             /* Authority group name               */
         ADOWNER ,             /* Owner ID                           */
         ADODESC ,             /* Owner description                  */
         ADPRIOR ,             /* Priority                           */
         ADCAL ,               /* Calendar                           */
         ADLDATE ,             /* Date last updated                  */
         ADLTIME ,             /* Time last updated                  */
         ADLUSER ,             /* Userid of last updater             */
         ADCOMVERS ,           /* Record version number              */
         ADGROUPID ,           /* Group definition ID                */
/*       ADLUTS ,              /* TOD clock at last update           */
         ADDSM ,               /* Deadline smoothing factor          */
         ADDLIM)               /* Deadline feedback limit            */

/*-------------------------------------------------------------------+
 | SEGMENT=ADKEY    - Key                                            |
 +-------------------------------------------------------------------*/
OUTPUT ADKEY DATA(OUTDATA) LOADER(OUTBL)
  KEYS(ADID ,                  /* Application ID                     */
       ADSTAT ,                /* Application status                 */
                               /* A=Active, P=Pending                */
       ADTO)                   /* Valid-To date                      */

/*-------------------------------------------------------------------+
 | SEGMENT=ADRUN    - Run cycle                                      |
 +-------------------------------------------------------------------*/
OUTPUT ADRUN DATA(=) LOADER(=)
  KEYS(ADCOM.ADID ,            /* Application ID                     */
       ADCOM.ADSTAT ,          /* Application status                 */
                               /* A=Active, P=Pending                */
       ADCOM.ADTYPE ,          /* Application type                   */
                               /* A=Application, G=Group Def         */
       ADCOM.ADFROM ,          /* Valid-From date                    */
       ADRSEQ)                 /* Sequence number                    */

  FIELDS(ADRPER ,              /* Period name                        */
         ADRVALF ,             /* Run cycle valid-from               */
         ADRVALT ,             /* Run cycle valid-to                 */
```

```
         ADRUNDESC ,           /* Run cycle description               */
         ADRUNRULE ,           /* Run rule for work/free days         */
         ADRTYPE ,             /* Period based (N/X) |                */
                               /*   Rule based (R/E)                  */
/*       ADRIAD ,              /* Offsets (start days within period)  */
         ADRIAT ,              /* Input arrival time                  */
         ADRDD ,               /* Deadline day relative to start      */
         ADRDT ,               /* Deadline time                       */
         ADRUNVERS ,           /* Record version number=1             */
         ADRJVTAB ,            /* JCL variable table                  */
         ADRSHTYPE ,           /* Shift type (W,D,or blank)           */
         ADRINPOS ,            /* Number of positive                  */
                               /* run cycle offsets                   */
         ADRINNEG ,            /* number of negative                  */
                               /* run cycle offsets                   */
/*       ADRIADALL ,           /* Array of run cycle offset           */
         ADRIADPOS ,           /* Positive run cycle offsets          */
/*       ADRIADPOSX ,          /* Positive run cycle offsets (Hex)    */
         ADRIADNEG ,           /* Positive run cycle offsets          */
/*       ADRIADNEGX ,          /* Positive run cycle offsets (Hex)    */
         ADRREPEATEVRY ,       /* Repeat every                        */
         ADRREPEATENDT ,       /* Repeat end time                     */
         ADRSHIFT ,            /* Shift value (-999 to 999)           */
         ADRSHSIGN ,           /* Shift sign (F/B)                    */
/*       ADRULEL ,             /* Rule length                         */
         ADRULET)              /* Rule text                           */

/*--------------------------------------------------------------------+
 ! SEGMENT=ADAPD    - Application Dependencies                        !
 +--------------------------------------------------------------------*/
OUTPUT ADAPD DATA(=) LOADER(=)
  KEYS(ADCOM.ADID ,            /* Application ID                      */
       ADCOM.ADSTAT ,          /* Application status                  */
                               /* A=Active, P=Pending                 */
       ADCOM.ADTYPE ,          /* Application type                    */
                               /* A=Application, G=Group Def          */
       ADCOM.ADFROM ,          /* Valid-From date                     */
       ADAPDADID)              /* Predecessor applid                  */

  FIELDS(ADAPDWSID             /* predecessor wsid or blank           */
         ADAPDOPNO             /* predecessor opid or 0               */
         ADAPDDESC             /* description                         */
         ADAPDLTP              /* LTP report print option A!C         */
         ADAPDVERS             /* record version number (r)           */
         ADAPDFLAG             /* flags                               */
         ADAPDCSEL             /* matching criteria:                  */
                               /* C/S/R/A                             */
                               /* (only for ext pred)                 */
         ADAPDXMAND            /* mandatory dep:                      */
                               /* P/C/A                               */
         ADAPDIVTYPE           /* interval type                       */
                               /* - A = absolute                      */
                               /* - R = relative                      */
                               /* ----------------------              */
                               /* FROM:                               */
                               /* ----------------------              */
         ADAPDIVFWHE           /* from when:                          */
                               /* - B = before IA                     */
```

```
                              /* - B = after IA                       */
        ADAPDIVFHHH           /* from hours HHH  (only rel)           */
        ADAPDIVFHH            /* from hours HH   (only abs)           */
        ADAPDIVFMM            /* from minutes MM                      */
        ADAPDIVFD             /* from days  (only abs)                */
                              /* ---------------------                */
                              /* TO:                                  */
                             /* ---------------------                */
        ADAPDIVTWHE           /* to   when:                           */
                              /* - B = before IA                      */
                              /* - A = after IA                       */
        ADAPDIVTHHH           /* to   hours HHH  (only rel)           */
        ADAPDIVTHH            /* to   hours HH   (only abs)           */
        ADAPDIVTMM            /* to   minutes MM                      */
        ADAPDIVTD)            /* to   days  (only abs)                */

/*-----------------------------------------------------------------------+
 | SEGMENT=ADOP     - Operation                                          |
 +-----------------------------------------------------------------------*/
OUTPUT ADOP DATA(OUTDATA) LOADER(=)
  KEYS(ADCOM.ADID ,           /* Application ID                       */
       ADCOM.ADSTAT ,         /* Application status                   */
                              /* A=Active, P=Pending                  */
       ADCOM.ADTYPE ,         /* Application type                     */
       ADCOM.ADFROM ,         /* Valid-From date                      */
       ADOPNO)                /* Operation number                     */

  FIELDS(ADOPWSID ,           /* Workstation                          */
         ADOPJN ,             /* Jobname                              */
         ADOPDESC ,           /* Operation description                */
         ADOPDUR ,            /* Duration in minutes                  */
         ADOPSM ,             /* Smoothing factor (or -1)             */
         ADOPLIM ,            /* Limit for feedback (or -1)           */
         ADOPHRC ,            /* Highest OK RC (or -1)                */
         ADOPSTD ,            /* Relative day input arrival           */
         ADOPSTT ,            /* Input arrival time                   */
         ADOPDD ,             /* Relative day deadline                */
         ADOPDT ,             /* Deadline time                        */
         ADOP#R1 ,            /* Number of R1 resources required      */
         ADOP#R2 ,            /* Number of R2 resources required      */
         ADOP#PS ,            /* Number of servers used               */
         ADOPJCL ,            /* Job class                            */
         ADOPPCL ,            /* Print class                          */
         ADOPFOR ,            /* Form number                          */
         ADOPSUB ,            /* Automatic submit (Y/N)               */
         ADOPAJR ,            /* Automatic CPU release (Y/N)          */
         ADOPCAN ,            /* Cancel if late time (Y/N)            */
         ADOPTIM ,            /* Submit job on time (Y/N)             */
         ADOPAEC ,            /* Automatic error compl (Y/N)          */
         ADOPVERS ,           /* Record version number                */
         ADOPWTO ,            /* Deadline WTO (Y/N)                   */
         ADOPRES ,            /* Restartable (Y/N/blank)              */
         ADOPRER ,            /* Rerouteable (Y/N/blank)              */
         ADOPCM ,             /* Restart and Cleanup                  */
                              /* A=Automatic                          */
                              /* I=Immediate                          */
                              /* M=Manual                             */
                              /* N=None                               */
```

```
/*       ADOPWSINFO ,          /* Workstation info                    */
         ADOPWSISET ,          /* Info available (Y/N)                */
         ADOPWSTYPE ,          /* Type G|C|P                          */
         ADOPWSREP ,           /* Reporting attr (A/S/C/N)            */
         ADOPWSSUBT ,          /* Subtype JCL, STC, WTO               */
                               /*  none J/S/W/blank                   */
         ADOPJCRT ,            /* (WLM) Critical job                  */
         ADOPJPOL ,            /* (WLM) Late job policy               */
         ADOPUSRSYS ,          /* User sysout needed                  */
         ADOPEXPJCL ,          /* Expanded jcl needed                 */
         ADOPDURI ,            /* Duration in 100th of sec            */
         ADOPMON ,             /* Operation monitored                 */
         ADOPUSEEXT ,          /* Use ADEXTNAME field                 */
         ADOPUSESE ,           /* Use ADEXTSE field                   */
         ADOPUSESA ,           /* Use System Automation (Y/N)         */
         ADOPWLMCLASS ,        /* WLM Service Class                   */
         ADOPCONDRJOB,         /* Conditional recovery job            */
         ADOPNOP,              /* NOP indicator                       */
         ADOPMH,               /* MH indicator                        */
         ADOPDLACT)            /* Deadline action                     */

/*-------------------------------------------------------------------+
 | SEGMENT=ADEXT    - Extended name                                  |
 +-------------------------------------------------------------------*/
OUTPUT ADEXT DATA(=) LOADER(=)
  KEYS(ADCOM.ADID ,            /* Application ID                      */
       ADCOM.ADSTAT ,          /* Application status                  */
                               /* A=Active, P=Pending                 */
       ADCOM.ADTYPE ,          /* Application type (for EQQILSON)     */
       ADCOM.ADFROM ,          /* Valid-From date                     */
       ADOP.ADOPNO)            /* Operation number (for EQQILSON)     */

  FIELDS(ADEXTOWNOP ,          /* Owning op number                    */
         ADEXTNAME ,           /* Extended name                       */
         ADEXTVERS ,           /* Record version number               */
         ADEXTSENAME)          /* Scheduling environment name         */

/*-------------------------------------------------------------------+
 | SEGMENT=ADRE     - Remote job information                         |
 +-------------------------------------------------------------------*/
OUTPUT ADRE DATA(OUTDATA) LOADER(=)
  KEYS(ADCOM.ADID ,            /* Application ID                      */
       ADCOM.ADSTAT ,          /* Application status                  */
                               /* A=Active, P=Pending                 */
       ADCOM.ADTYPE ,          /* Application type                    */
       ADCOM.ADFROM ,          /* Valid-From date                     */
       ADOP.ADOPNO)            /* Operation number                    */

  FIELDS(ADRE_OWNOP ,          /* Owning operation number             */
         ADRE_JSNAME ,         /* ADID or Jobstream name              */
         ADRE_VERS ,           /* Record version number=1             */
         ADRE_COMPL ,          /* Complete on failed bind             */
         ADRE_OPNO ,           /* Operation number                    */
         ADRE_JSWS ,           /* Jobstream workstation               */
         ADRE_JOBNAME)         /* Job name                            */

/*-------------------------------------------------------------------+
 | SEGMENT=ADSAI    - Operation system automation information        |
```

```
 +-------------------------------------------------------------------*/
OUTPUT ADSAI DATA(OUTDATA) LOADER(=)
  KEYS(ADCOM.ADID ,            /* Application ID                     */
       ADCOM.ADSTAT ,          /* Application status                 */
                               /* A=Active, P=Pending                */
       ADCOM.ADTYPE ,          /* Application type (for EQQILSON)    */
       ADCOM.ADFROM ,          /* Valid-From date                    */
       ADOP.ADOPNO)            /* Operation number (for EQQILSON)    */

  FIELDS(ADSAIOWNOP ,          /* Owning operation number            */
         ADSAICOMMTEXT ,       /* System Automation operation        */
                               /* command text                       */
         ADSAICOMMTXT1 ,       /* Segment 1 of SA Operation Info     */
         ADSAICOMMTXT2 ,       /* Segment 2 of SA Operation Info     */
         ADSAICOMMTXT3 ,       /* Segment 3 of SA Operation Info     */
         ADSAICOMMTXT4 ,       /* Segment 4 of SA Operation Info     */
         ADSAIAUTOOPER ,       /* System Automation automated        */
                               /* function (for operation)           */
         ADSAISECELEM ,        /* System Automation security         */
                               /* element                            */
         ADSAICOMPINFO)        /* System Automation completion       */

/*-------------------------------------------------------------------+
 | SEGMENT=ADSR     - Special resource                               |
 +-------------------------------------------------------------------*/
OUTPUT ADSR DATA(OUTDATA) LOADER(=)
  KEYS(ADCOM.ADID ,            /* Application ID                     */
       ADCOM.ADSTAT ,          /* Application status                 */
                               /* A=Active, P=Pending                */
       ADCOM.ADTYPE ,          /* Application type (for EQQILSON)    */
       ADCOM.ADFROM ,          /* Valid-From date                    */
       ADOP.ADOPNO ,           /* Operation number (for EQQILSON)    */
       ADSRN)                  /* Special resource name              */

  FIELDS(ADSROWNOP ,           /* Owning operation number            */
         ADSRT ,               /* S = Shared, X = Exclusive          */
         ADSRVERS ,            /* Record version number=1            */
         ADSRONER ,            /* Keep on error (Y/N/blank)          */
         ADSRAMNT ,            /* Quantity required.  The value 0    */
                               /* means the total quantity of        */
                               /* special resource.                  */
         ADSRAVACO)            /* On complete (Y/N/R/blank)          */

/*-------------------------------------------------------------------+
 | SEGMENT=ADDEP    - Dependency                                     |
 +-------------------------------------------------------------------*/
OUTPUT ADDEP DATA(=) LOADER(=)
  KEYS(ADCOM.ADID ,            /* Application ID                     */
       ADCOM.ADSTAT ,          /* Application status                 */
                               /* A=Active, P=Pending                */
       ADCOM.ADTYPE ,          /* Application type (for EQQILSON)    */
       ADCOM.ADFROM ,          /* Valid-From date                    */
       ADOP.ADOPNO ,           /* Operation number (for EQQILSON)    */
       ADDEPADID ,             /* External predecessor/blank         */
       ADDEPOPNO)              /* Operation number                   */

  FIELDS(ADDEPOWNOP ,          /* Owning Op (the successor)          */
         ADDEPWSID ,           /* Workstation name                   */
```

```
         ADDEPTPT ,            /* Transport time in minutes           */
         ADDEPDESC ,           /* Description                         */
         ADDEPLTP ,            /* LTP REPORT PRINT OPTION (A/C)       */
         ADDEPVERS ,           /* Record version number=1             */
         ADDEPJOBN ,           /* Jobname (not always set)            */
         ADDEPFLAG ,           /* Flags                               */
         ADDEPCSEL ,           /* Resolution criteria (C/S/R/A)       */
         ADDEPXMAND ,          /* Is mandatory (N/P/C)                */
         ADDEPTYPE)            /* Dependency type (I/E)               */

/*-------------------------------------------------------------------+
 | SEGMENT=ADXIV    - External Dependency Interval Definition       |
 +-------------------------------------------------------------------*/
OUTPUT ADXIV DATA(=) LOADER(=)
  KEYS(ADCOM.ADID ,            /* Application ID                      */
       ADCOM.ADSTAT ,          /* Application status                  */
                               /* A=Active, P=Pending                 */
       ADCOM.ADTYPE ,          /* Application type (for EQQILSON)     */
       ADCOM.ADFROM ,          /* Valid-From date                     */
       ADOP.ADOPNO ,           /* Operation number (for EQQILSON)     */
       ADXIVADID ,             /* Predecessor application name        */
       ADXIVWSID ,             /* Predecessor workstation ID          */
       ADXIVOPNO)              /* Predecessor operation number        */

  FIELDS(ADXIVOWNOP ,          /* Owning Op (the successor)           */
         ADXIVTYPE ,           /* Interval type R/A (relative/        */
                               /*                    absolute)        */
         ADXIVFWHE ,           /* From When B/A (Before/After)        */
         ADXIVFHHH ,           /* From Hours HHH (Only relative)      */
         ADXIVFHH ,            /* From Hours HH  (Only absolute)      */
         ADXIVFMM ,            /* From Minutes MM                     */
         ADXIVFD ,             /* From Days      (Only absolute)      */
         ADXIVTWHE ,           /* To When B/A (Before/After)          */
         ADXIVTHHH ,           /* To Hours HHH (Only relative)        */
         ADXIVTHH ,            /* To Hours HH  (Only absolute)        */
         ADXIVTMM ,            /* To Minutes MM                       */
         ADXIVTD)              /* To Days      (Only absolute)        */

/*-------------------------------------------------------------------+
 | SEGMENT=ADCNC    - Condition                                      |
 +-------------------------------------------------------------------*/
OUTPUT ADCNC DATA(=) LOADER(=)
  KEYS(ADCOM.ADID ,            /* Application ID                      */
       ADCOM.ADSTAT ,          /* Application status                  */
                               /* A=Active, P=Pending                 */
       ADCOM.ADTYPE ,          /* Application type                    */
       ADCOM.ADFROM ,          /* Valid-From date                     */
       ADOP.ADOPNO ,           /* Operation number                    */
       ADCNCID)                /* Condition ID                        */

  FIELDS(ADCNCOWNID ,          /* Owning AD operation                 */
         ADCNCSIMPNO ,         /* Nuber of condition dependencies     */
         ADCNCCOUNT ,          /* Rule type:                          */
                               /* 0 = All                             */
                               /* N>0 = At least N of                 */
         ADCNCVERS ,           /* Record version                      */
         ADCNCDESC)            /* Condition description               */
```

```
/*-------------------------------------------------------------------+
 | SEGMENT=ADCNS    - Condition dependency                           |
 +-------------------------------------------------------------------*/
OUTPUT ADCNS DATA(=) LOADER(=)
  KEYS(ADCOM.ADID ,             /* Application ID                    */
       ADCOM.ADSTAT ,           /* Application status                */
                                /* A=Active, P=Pending               */
       ADCOM.ADTYPE ,           /* Application type (for EQQILSON)   */
       ADCOM.ADFROM ,           /* Valid-From date                   */
       ADOP.ADOPNO ,            /* Operation number (for EQQILSON)   */
       ADCNSID ,                /* Condition ID                      */
       ADCNSSEQ)                /* Sequence number                   */

  FIELDS(ADCNSOWNID ,           /* Owning AD operation               */
         ADCNSPREAD ,           /* Predecessor Application ID        */
         ADCNSPREOPNO ,         /* Predecessor Operation Number      */
         ADCNSPRETYP ,          /* Check type:                       */
                                /* RC: Return code                   */
                                /* ST: Status                        */
         ADCNSPRELOG ,          /* Logical operator type:            */
                                /* GE: >= (Greater than or equal)    */
                                /* GT: > (Greater than)              */
                                /* LE: <= (Less than or equal)       */
                                /* LT: < (Less than)                 */
                                /* EQ: = (Equal to)                  */
                                /* NE: <> (Not equal to)             */
                                /* RG: RC - RC2 (Range)              */
         ADCNSVALRC ,           /* Return code value                 */
         ADCNSVALST ,           /* Status value:                     */
                                /* S: Started                        */
                                /* C: Completed                      */
                                /* X: Suppressed by condition        */
                                /* E: Error                          */
         ADCNSPROC ,            /* Step name                         */
         ADCNSSTEP ,            /* Procedure invocation step name    */
         ADCNSPREWSID ,         /* Predecessor Workstation ID        */
         ADCNSDEPTYP ,          /* Dependency type:                  */
                                /* I: Internal                       */
                                /* E: External                       */
         ADCNSVALRC2 ,          /* Upper limit of return code range  */
         ADCNSVERS ,            /* Version                           */
         ADCNSCCSEL)            /* Resolution criteria (C/S/R/A)     */

/*-------------------------------------------------------------------+
 | SEGMENT=ADCIV    - Conditional Dependency Interval Definition     |
 +-------------------------------------------------------------------*/
OUTPUT ADCIV DATA(=) LOADER(=)
  KEYS(ADCOM.ADID ,             /* Application ID                    */
       ADCOM.ADSTAT ,           /* Application status                */
                                /* A=Active, P=Pending               */
       ADCOM.ADTYPE ,           /* Application type                  */
       ADCOM.ADFROM ,           /* Valid-From date                   */
       ADOP.ADOPNO ,            /* Operation number                  */
       ADCIVADID ,              /* Predecessor application name      */
       ADCIVCID ,               /* Predecessor condition ID          */
       ADCIVOPNO)               /* Predecessor operation number      */

  FIELDS(ADCIVOWNOP ,           /* Owning Op (the successor)         */
```

```
         ADCIVTYPE ,           /* Interval type R/A (relative/        */
                               /*                    absolute)        */
         ADCIVFWHE ,           /* From When B/A (Before/After)        */
         ADCIVFHHH ,           /* From Hours HHH (Only relative)      */
         ADCIVFHH ,            /* From Hours HH  (Only absolute)      */
         ADCIVFMM ,            /* From Minutes MM                     */
         ADCIVFD ,             /* From Days      (Only absolute)      */
         ADCIVTWHE ,           /* To When B/A (Before/After)          */
         ADCIVTHHH ,           /* To Hours HHH (Only relative)        */
         ADCIVTHH ,            /* To Hours HH  (Only absolute)        */
         ADCIVTMM ,            /* To Minutes MM                       */
         ADCIVTD)              /* To Days      (Only absolute)        */

/*-------------------------------------------------------------------+
 | SEGMENT=ADUSF    - User field                                      |
 +-------------------------------------------------------------------*/
OUTPUT ADUSF DATA(OUTDATA) LOADER(=)
  KEYS(ADCOM.ADID ,            /* Application ID                      */
       ADCOM.ADSTAT ,          /* Application status                  */
                               /* A=Active, P=Pending                 */
       ADCOM.ADTYPE ,          /* Application type (for EQQILSON)     */
       ADCOM.ADFROM ,          /* Valid-From date                     */
       ADOP.ADOPNO ,           /* Operation number (for EQQILSON)     */
       ADUSFNAME)              /* Field name                          */

  FIELDS(ADUSFOWNID ,          /* Owning operation number             */
         ADUSFVALUE ,          /* Field value                         */
         ADUSFVERS)            /* Version=1                           */

/*-------------------------------------------------------------------+
| SEGMENT=ADVDD    - Variable duration and deadline                  |
+-------------------------------------------------------------------*/
OUTPUT ADVDD DATA(OUTDATA) LOADER(=)
  KEYS(ADCOM.ADID ,            /* Application ID                      */
       ADCOM.ADSTAT ,          /* Application status                  */
                               /* A=Active, P=Pending                 */
       ADCOM.ADTYPE ,          /* Application type (for EQQILSON)     */
       ADCOM.ADFROM ,          /* Valid-From date                     */
       ADOP.ADOPNO ,           /* Operation number (for EQQILSON)     */
       ADVDDRG)                /* Run cycle name                      */

  FIELDS(ADVDDOWNID ,          /* Owning operation number             */
         ADVDDDUR,             /* Variable duration                   */
         ADVDDDEADD,           /* Variable deadline day               */
         ADVDDDEADT,           /* Variable deadline time              */
         ADVDDNOP,             /* NOP indicator                       */
         ADVDDMH,              /* MH indicator                        */
         ADVDDCRJ,             /* Critical job indicator              */
        ADVDDDLACT,           /* Deadline late action                */
         ADVDDLATE1BAS,        /* BASEDATE (F)                        */
         ADVDDLATE1DIR,        /* DIRECTION (A)                       */
         ADVDDLATE1DD,         /* day offset                          */
         ADVDDLATE1DT,         /* time                                */
         ADVDDLATE2BAS,        /* BASEDATE (F)                        */
         ADVDDLATE2DIR,        /* DIRECTION (A)                       */
         ADVDDLATE2AC,         /* ACTION (A)                          */
         ADVDDLATE2DD,         /* day offset                          */
         ADVDDLATE2DT)         /* time                                */
```

```
/*-------------------------------------------------------------------+
! SEGMENT=ADLAT    - Operation late times                            !
+-------------------------------------------------------------------*/
OUTPUT ADLAT DATA(OUTDATA) LOADER(=)
  KEYS(ADCOM.ADID ,            /* Application ID                     */
       ADCOM.ADSTAT ,          /* Application status                 */
                               /* A=Active, P=Pending                */
       ADCOM.ADTYPE ,          /* Application type (for EQQILSON)    */
       ADCOM.ADFROM ,          /* Valid-From date                    */
       ADOP.ADOPNO)            /* Operation number (for EQQILSON)    */

  FIELDS(ADLATOWNID ,          /* Owning operation number            */
         ADLATVERS,            /* Version 1                          */
         ADLAT1BASE,           /* LATE1 basedate F                   */
         ADLAT1DIR,            /* LATE1 direction (A) after          */
         ADLAT1DD,             /* LATE1 DD                           */
         ADLAT1DT,             /* LATE1 TIME HHMM                    */
         ADLAT2BASE,           /* LATE2 basedate F                   */
         ADLAT2DIR,            /* LATE2 direction (A) after          */
         ADLAT2AC,             /* LATE2 Action (A) Alert only        */
         ADLAT2DD,             /* LATE2 DD                           */
         ADLAT2DT)             /* LATE2 TIME HHMM                    */

/*-------------------------------------------------------------------+
 | RECORD=AWSCL     - All workstations closed                       |
 +-------------------------------------------------------------------*/
/*-------------------------------------------------------------------+
 | SEGMENT=AWSCL    - All workstations closed interval              |
 +-------------------------------------------------------------------*/
OUTPUT AWSCL DATA(OUTDATA) LOADER(OUTBL)
  KEYS(AWCDATE)                /* Date                               */

  FIELDS(AWCFROM,              /* From time                          */
         AWCTO,                /* To time                            */
         AWCDESC,              /* Description closed interval        */
         AWCVERS,              /* Version of record                  */
         AWCLDATE,             /* Date last updated                  */
         AWCLTIME,             /* Time last updated                  */
         AWCLUSER,             /* Userid of last updater             */
         AWCLLUTS)             /* Tod clock at last                  */


/*-------------------------------------------------------------------+
 | RECORD=CL        - Calendar                                      |
 +-------------------------------------------------------------------*/
/*-------------------------------------------------------------------+
 | SEGMENT=CLCOM    - Calendar common segment                       |
 +-------------------------------------------------------------------*/
OUTPUT CLCOM DATA(OUTDATA) LOADER(OUTBL)
  KEYS(CLNAME)                 /* Calender name                      */

  FIELDS(CLDAYS,               /* No. specific and week days         */
         CLSHIFT,              /* End time of shift                  */
         CLDESC,               /* Description                        */
         CLVERS,               /* Version of record                  */
         CLLDATE,              /* Date last updated                  */
         CLLTIME,              /* Time last updated                  */
```

```
         CLLUSER,              /* Userid of last updater              */
         CLLUTS)               /* Tod clock at last update            */

/*-------------------------------------------------------------------+
 | SEGMENT=CLSD     - Specific date                                  |
 +-------------------------------------------------------------------*/
OUTPUT CLSD DATA(=) LOADER(=)
  KEYS(CLCOM.CLNAME,           /* Calender name                       */
       CLSDDATE)               /* Specific date                       */

  FIELDS(CLSDSTAT,             /* Status, work or free                */
         CLSDDESC)             /* Description of the date             */

/*-------------------------------------------------------------------+
 | SEGMENT=CLWD     - Weekday                                        |
 +-------------------------------------------------------------------*/
OUTPUT CLWD DATA(=) LOADER(=)
  KEYS(CLCOM.CLNAME,           /* Calender name                       */
       CLWDDAY)                /* Weekday                             */

  FIELDS(CLWDSTAT,             /* Status, work or free                */
         CLWDDESC)             /* Description of the date             */

/*-------------------------------------------------------------------+
 | SEGMENT=CPLAT    - Late                                           |
 +-------------------------------------------------------------------*/
OUTPUT CPLAT DATA(=)
  KEYS(CPOPCOM.CPOPADI ,         /* Owning application ID             */
       CPOPCOM.CPOPIAD ,         /* Owning application IA date        */
       CPOPCOM.CPOPIAT ,         /* Owning application IA time        */
       CPOPCOM.CPOPNO)           /* Owning operation number           */
  FIELDS(CPLATVERS ,             /* version number                    */
  /*   CPLATALE,                    LATE ALERT                        */
       CPLATALEBASE ,            /* base for compute                  */
       CPLATALEDIR ,             /* always A (after)                  */
       CPLATALEACT ,             /* not used                          */
  /*   CPLATALEDT ,                                                   */
       CPLATALEDATE ,            /* alert date                        */
       CPLATALETIME ,            /* alert time                        */
  /*   CPLATACT,                    LATE ACTION                       */
       CPLATACTBASE ,            /* base for compute                  */
                                    always F (Occurrence IA is used)  */
       CPLATACTDIR ,             /* always A (after)                  */
       CPLATACTACT ,             /* action to do (A,C,N,E, blank)     */
  /*   CPLATACTDT ,                                                   */
       CPLATACTDATE ,            /* action date                       */
       CPLATACTTIME)             /* action time                       */

/*-------------------------------------------------------------------+
| RECORD=CRITSUCS  - Critical Successors                             |
+-------------------------------------------------------------------*/
/*-------------------------------------------------------------------+
| SEGMENT=CRITSUCS - Critical Successors of an operation             |
+-------------------------------------------------------------------*/
OUTPUT CRITSUCS DATA(OUTDATA)
  KEYS(CRITADID,               /* Application ID                      */
       CRITWSN,                /* Operation workstation               */
       CRITOPNO)               /* Operation number                    */
```

```
  FIELDS(CRITCONFACT,          /* Confidence factor                  */
         CRITJOBN,             /* Jobname                            */
         CRITLS,               /* Latest start                       */
         CRITOI,               /* Operation input arrival            */
         CRITPS,               /* Planned start                      */
         CRITAS,               /* Actual start                       */
         CRITOD,               /* Operation deadline                 */
         CRITAE,               /* Actual end                         */
         CRITOPST,             /* Operation status                   */
         CRITLATE,             /* Job is late                        */
         CRITURGPRO,           /* Job promoted to urgent queue       */
         CRITWLMPRO,           /* Job promoted to WLM                */
         CRITLONGR,            /* Job long running                   */
         CRITESTART,           /* Job estimated start                */
         CRITEEND,             /* Job estimated end                  */
         CRITJisonPATH)        /* Y=Input job is on crit path        */
                               /* N=Input job only on crit network   */

/*-------------------------------------------------------------------+
 | RECORD=ETT       - Event triggered tracking criteria              |
 +-------------------------------------------------------------------*/
/*-------------------------------------------------------------------+
 | SEGMENT=ETT      - Event triggered tracking criteria              |
 +-------------------------------------------------------------------*/
OUTPUT ETT DATA(OUTDATA) LOADER(OUTBL)
  KEYS(ETTTYPE,                /* Type of trigger                    */
       ETTNAME)                /* Trigger name                       */

  FIELDS(ETTVERS,              /* Record version                     */
         ETTAPPL,              /* Application to trigger             */
         ETTJREP,              /* Job replace                        */
         ETTLUSER,             /* Last update user                   */
         ETTLDATE,             /* Last update date                   */
         ETTLTIME,             /* Last update time                   */
         ETTDEPR,              /* Dependency resolution              */
         ETTASSW,              /* Availability switched              */
         ETTLUTS)              /* Tod clock at last update           */


/*-------------------------------------------------------------------+
 | RECORD=GENDAYS   - Output from the GENDAYS command                |
 +-------------------------------------------------------------------*/
/*-------------------------------------------------------------------+
 | SEGMENT=GNDAY    - GENDAYS date                                   |
 +-------------------------------------------------------------------*/
OUTPUT GNDAY DATA(OUTDATA)
  KEYS(GNDAYDATE               /* Generated run day                  */
       GNDAYIAT)               /* IA Time (IAT keyword)              */

  FIELDS(GNDAYFLAGS,           /* Flags                              */
         GNDAYFMOB,            /* Moved before - Free day rule       */
         GNDAYFMOA,            /* Moved after - Free day rule        */
         GNDAYFKEP,            /* Kept                               */
         GNDAYFCAN,            /* Cancelled - Free day rule          */
         GNDAYFEIA,            /* Run on free day - Early IA         */
         GNDAYFOUT,            /* Moved outside interval             */
         GNDAYFREM,            /* Work date outside interval         */
```

```
         GNDAYFROM,            /* From date (FROMDATE keyword)        */
         GNDAYTO,              /* To date (TODATE keyword)            */
         GNDAYCAL,             /* Calendar (CALENDAR keyword)         */
         GNDAYFDRULE,          /* Free day rule (FDAYRULE keyword)    */
         GNDAYRULEDEF,         /* Rule definition (RULEDEF keyword)   */
         TAG)                  /* GENDAYS: ADID,NUM,ADRPER,ADRTYPE    */

/*-------------------------------------------------------------------+
 | RECORD=JCLV      - JCL variable table                             |
 +-------------------------------------------------------------------*/
/*-------------------------------------------------------------------+
 | SEGMENT=JCLVCOM  - Common                                         |
 +-------------------------------------------------------------------*/
OUTPUT JCLVCOM DATA(OUTDATA) LOADER(OUTBL)
  KEYS(JCLVCTAB)               /* Jcl variable table id               */

  FIELDS(JCLVCLU,              /* Last updating user                  */
         JCLVCLT,              /* Last update time hh:mm              */
         JCLVCLD,              /* Last update date yymmdd             */
         JCLVC#V,              /* No. of variables in table           */
         JCLVCOWN,             /* Owner id                            */
         JCLVCDSC,             /* Description                         */
         JCLVCLUTS)            /* Tod clock at last update            */

/*-------------------------------------------------------------------+
 | SEGMENT=JCLVVAR  - Variable definition                            |
 +-------------------------------------------------------------------*/
OUTPUT JCLVVAR DATA(=) LOADER(=)
  KEYS(JCLVCOM.JCLVCTAB,      /* Jcl variable table id               */
       JCLVVVAR)               /* Jcl variable name                   */

  FIELDS(JCLVVDFL,             /* Jcl variable def value              */
         JCLVVSTP,             /* Prompt / setup / submit             */
         JCLVVUC,              /* Upper case (Y/N)                    */
         JCLVVLG,              /* Value length                        */
         JCLVVTYP,             /* Verification type                   */
         JCLVVEX,              /* Substitution exit name              */
         JCLVVINP,             /* Input required                      */
         JCLVVPOS,             /* Replace position jcl data           */
         JCLVVNUM,             /* Numeric                             */
         JCLVVCMP,             /* Comparitive operator                */
         JCLVVPAT,             /* Validation pattern                  */
         JCLVVVLD1,            /* Valid values - line 1               */
         JCLVVVLD2,            /* Valid values - line 2               */
         JCLVVTXT1,            /* Dialog text - line 1                */
         JCLVVTXT2,            /* Dialog text - line 2                */
         JCLVVTXT3,            /* Dialog text - line 3                */
         JCLVVTXT4,            /* Dialog text - line 4                */
         JCLVVDES,             /* Description                         */
         JCLVVNRP,             /* Number of dependent values          */
         JCLVVIND,             /* Independent variable name           */
         JCLVVVER,             /* Version number = 0                  */
         JCLVVSUS,             /* Substring start position            */
         JCLVVSUL)             /* Substring length                    */

/*-------------------------------------------------------------------+
 | SEGMENT=JCLVDEP  - Variable dependency                            |
 +-------------------------------------------------------------------*/
```

```
OUTPUT JCLVDEP DATA(=) LOADER(=)
  KEYS(JCLVCOM.JCLVCTAB,       /* Jcl variable table id               */
       JCLVVAR.JCLVVVAR,       /* Jcl variable name                   */
       JCLVDIV)                /* Value of setting variable           */

  FIELDS(JCLVDDV)              /* Override value for depend           */


/*-------------------------------------------------------------------+
 | RECORD=OI        - Operator instruction                           |
 +-------------------------------------------------------------------*/
/*-------------------------------------------------------------------+
 | SEGMENT=OICOM    - Operator instruction                           |
 +-------------------------------------------------------------------*/
OUTPUT OICOM DATA(OUTDATA) LOADER(OUTBL)
  KEYS(OIADID,                 /* Application name                    */
       OIOPNO,                 /* Operation number                    */
       OITOD,                  /* Valid to date                       */
       OITOT,                  /* Valid to time                       */
       OIFROMD,                /* Valid from date                     */
       OIFROMT,                /* Valid from time                     */
       OIWSN)                  /* Workstation name                    */

  FIELDS(OIJOBN,               /* Job name                            */
         OILDATE,              /* Last update date                    */
         OILTIME,              /* Last update time                    */
         OILUSER,              /* Last update user                    */
         OIVERS,               /* Record version                      */
         OILINES,              /* Number of text lines                */
         OILUTS)               /* Tod clock at last update            */

/*-------------------------------------------------------------------+
 | SEGMENT=OIT      - Operator instruction text                      |
 +-------------------------------------------------------------------*/
OUTPUT OIT DATA(=) LOADER(=)
  KEYS(OICOM.OIADID,           /* Application name                    */
       OICOM.OIOPNO,           /* Operation number                    */
       OICOM.OITOD,            /* Valid to date                       */
       OICOM.OITOT,            /* Valid to time                       */
       OICOM.OIFROMD,          /* Valid from date                     */
       OICOM.OIFROMT,          /* Valid from time                     */
       OICOM.OIWSN,            /* Workstation name                    */
       OITSEQ)                 /* Line number                         */

  FIELDS(OICOM.OIJOBN,         /* Job name                            */
         OITMAX,               /* Number of lines                     */
         OITTEXT)              /* Text line                           */


/*-------------------------------------------------------------------+
 | RECORD=PR        - Period                                         |
 +-------------------------------------------------------------------*/
/*-------------------------------------------------------------------+
 | SEGMENT=PRCOM    - Period Common Segment                          |
 +-------------------------------------------------------------------*/
OUTPUT PRCOM DATA(OUTDATA) LOADER(OUTBL)
  KEYS(PRNAME)                 /* Period name                         */
```

```
  FIELDS(PRVERS,               /* Record version=1                   */
         PRTYPE,               /* Cyclic/noncyclic type (A/W/N)      */
         PRDESC,               /* Description of period              */
         PRINTVL,              /* Interval of cyclic origins         */
         PRORIG#,              /* Number of origin dates             */
         PRLDATE,              /* Date last updated                  */
         PRLTIME,              /* Time last updated                  */
         PRLUSER,              /* Userid of last updater             */
         PRJVT,                /* Jcl variable table                 */
/*       PRTAB,                /* Date table                         */
         PRLUTS,               /* Tod clock at last update           */
         PRWXRNG,              /* ANALYSE: Period range in days      */
         PRWXDAY,              /* ANALYSE: Period first day          */
         PRWXLST,              /* ANALYSE: Period last day           */
         PRWXINT,              /* ANALYSE: Intervale type            */
         PRWXALL,              /* ALTERNATE: All dates               */
         PRWXBEG,              /* ALTERNATE: Begin dates             */
         PRWXEND)              /* ALTERNATE: End dates               */

/*-------------------------------------------------------------------+
 | SEGMENT=PRDATE   - Period date                                    |
 |                    (derived from PRTAB in PRCOM)                  |
 +-------------------------------------------------------------------*/
OUTPUT PRDATE DATA(=) LOADER(=)
  KEYS(PRCOM.PRNAME            /* Period name                        */
       PRDSTART)               /* Period start date                  */

  FIELDS(PRDEND)               /* Period end date                    */


/*-------------------------------------------------------------------+
 | RECORD=RG        - Run cycle group                                |
 +-------------------------------------------------------------------*/
/*-------------------------------------------------------------------+
 | SEGMENT=RGCOM    - Common                                         |
 +-------------------------------------------------------------------*/
OUTPUT RGCOM DATA(OUTDATA) LOADER(OUTBL)
  KEYS(RGID)                   /* Run cycle group ID                 */

  FIELDS(RGIAT ,               /* Default input arrival time         */
         RGJVTAB ,             /* Default JCL variable table         */
         RGCAL ,               /* Default calendar                   */
         RGDESC ,              /* Run cycle group description        */
         RGLUSER ,             /* Userid of last updater             */
         RGLDATE ,             /* Date last updated                  */
         RGLTIME ,             /* Time last updated                  */
/*       RGLUTS ,              /* TOD clock at last update           */
         RGCOMVERS ,           /* Record version number              */
         RGOWNER ,             /* Owner ID                           */
         RGDD ,                /* Default deadline day (relative)    */
         RGDT)                 /* Default deadline time              */

/*-------------------------------------------------------------------+
 | SEGMENT=RGRUN    - Run cycle                                      |
 +-------------------------------------------------------------------*/
OUTPUT RGRUN DATA(=) LOADER(=)
  KEYS(RGCOM.RGID ,            /* Run cycle group ID                 */
```

```
       RGRSEQ)                 /* Sequence number                      */

  FIELDS(RGRNAME ,             /* Rule name                            */
         RGRVALF ,             /* Run cycle valid-from                 */
         RGRVALT ,             /* Run cycle valid-to                   */
         RGRDESC ,             /* Run cycle description                */
         RGRRULE ,             /* Rule for work/free days              */
         RGRTYPE ,             /* Type (R/E/A/D)                       */
         RGRIAT ,              /* Input arrival time                   */
         RGRUNVERS ,           /* Record version number=1              */
         RGRJVTAB ,            /* JCL variable table                   */
         RGRIRDLEN ,           /* Rule definition length               */
         RGRREPEATEVERY ,      /* Repeat every                         */
         RGRREPEATENDT ,       /* Repeat end time                      */
         RGRSETID ,            /* Run cycle correlator                 */
         RGRCALENDAR           /* Run cycle calendar                   */
         RGDD ,                /* Deadline day relative to start       */
         RGDT ,                /* Deadline time                        */
/*       RGRIRDALL ,           /* Rule text (with length)              */
         RGRULEL ,             /* Rule length (RGRULEL + RGRULET)      */
         RGRULET)              /* Rule text                            */

/*-------------------------------------------------------------------+
 | RECORD=SR        - Special resource                               |
 +-------------------------------------------------------------------*/
/*-------------------------------------------------------------------+
 | SEGMENT=SRCOM    - Common                                         |
 +-------------------------------------------------------------------*/
OUTPUT SRCOM DATA(OUTDATA) LOADER(OUTBL)
  KEYS(SRCNAME)                /* Special resource name               */

  FIELDS(SRCGROUP,             /* Group id                            */
         SRCHIPER,             /* Dlf resource (Y/N)                  */
         SRCUSEDFOR,           /* Used for (N/P/C/B)                  */
         SRCONERROR,           /* On error option                     */
         SRCIVLNUM,            /* Number of intervals                 */
         SRCDESC,              /* Description                         */
         SRCONCOMPL,           /* On complete (Y/N/R/blank)           */
         SRCMAXTYPE,           /* Max limit type (Y/N/R)              */
         SRCMAXLIMIT,          /* Max limit value                     */
         SRCDEFQUANT,          /* Default quantity                    */
         SRCDEFAVAIL,          /* Default availability                */
         SRCLUSER,             /* Last updating user                  */
         SRCLDATE,             /* Date of last update                 */
         SRCLTIME,             /* Time of last update                 */
         SRCLUTS,              /* Tod clock at last update            */
         SRCVER ,              /* Record version                      */
         SRCWXDBZ  ,           /* WAXD Analyse: In TWSz database      */
         SRCWXREF  ,           /* WAXD Analyse: Referenced            */
         SRCWXMAX  ,           /* WAXD Analyse: Maximum usage         */
         SRCWXUSE  ,           /* WAXD Analyse: Usage type            */
         SRCWXDEF  ,           /* WAXD Analyse: Default workstation   */
         SRCWXCTL)             /* WAXD Analyse: Control usage         */

/*-------------------------------------------------------------------+
 | SEGMENT=SRIVL    - Interval                                       |
 +-------------------------------------------------------------------*/
OUTPUT SRIVL DATA(=) LOADER(=)
```

```
  KEYS(SRCOM.SRCNAME,          /* Special resource name               */
       SRIVLDAY,               /* Day number                          */
       SRIVLDATE,              /* Specific date                       */
       SRIVLFTIME)             /* From time                           */

  FIELDS(SRIVLTTIME,           /* To time                             */
         SRIVLQUANT,           /* Max number of SRs to allocate       */
         SRIVLWSCNUM,          /* Number of connected WSs             */
         SRIVLAVAIL)           /* Workstation name                    */

/*-------------------------------------------------------------------+
 | SEGMENT=SRIWS    - Interval workstation                           |
 +-------------------------------------------------------------------*/
OUTPUT SRIWS DATA(=) LOADER(=)
  KEYS(SRCOM.SRCNAME,          /* Special resource name               */
       SRIVL.SRIVLDAY,         /* Day number                          */
       SRIVL.SRIVLDATE,        /* Specific date                       */
       SRIVL.SRIVLFTIME,       /* From time                           */
       SRIWSNAME)              /* Workstation name                    */

/*-------------------------------------------------------------------+
 | SEGMENT=SRDWS    - Default workstation                            |
 +-------------------------------------------------------------------*/
OUTPUT SRDWS DATA(=) LOADER(=)
  KEYS(SRCOM.SRCNAME,          /* Special resource name               */
       SRDWSNAME)              /* Workstation name                    */


/*-------------------------------------------------------------------+
 | RECORD=WS        - Workstation description                        |
 +-------------------------------------------------------------------*/
/*-------------------------------------------------------------------+
 | SEGMENT=WSCOM    - Common                                         |
 +-------------------------------------------------------------------*/
OUTPUT WSCOM DATA(OUTDATA) LOADER(OUTBL)
  KEYS(WSNAME)                 /* Workstation name                    */

  FIELDS(WSVERS ,              /* Version of record                   */
         WSTYPE ,              /* Workstation type (G/C/P/R)          */
         WSREP ,               /* Reporting attribute (A/S/C/N)       */
         WSPREP ,              /* Jobsetup ability                    */
         WSTRSPT ,             /* Transport time from predecessor WS  */
         WSOPDUR ,             /* Default operation duration          */
         WSDAY# ,              /* Total number of days                */
         WSTOTIVL# ,           /* Number of open intervals            */
         WSROUT ,              /* Printout routing for dp             */
         WSDESC ,              /* Workstation description             */
         WSPSJT ,              /* Control on servers                  */
         WSSPLIT ,             /* Splittable attribute                */
         WSR1NAM ,             /* R1 resource name                    */
         WSR1PLAN ,            /* Resource used at planning           */
         WSR1CONT ,            /* Resource used for control           */
         WSR2NAM ,             /* R2 resource name                    */
         WSR2PLAN ,            /* Resource used at planning           */
         WSR2CONT ,            /* Resource used for control           */
         WSSUDS ,              /* Destination                         */
         WSLDATE ,             /* Date last updated                   */
         WSLTIME ,             /* Time last updated                   */
```

```
         WSLUSER ,             /* Userid of last updater              */
         WSSTC ,               /* Started task ability (Y/N)          */
         WSWTO ,               /* WTO ability (Y/N)                   */
         WSPSPL ,              /* Planning on servers (Y/N)           */
         WSAUTO ,              /* System Automation workstation (Y/N) */
         WSLUTS ,              /* TOD clock at last update            */
         WSOPDURI ,            /* Default op dur in 100th sec         */
         WSWAIT ,              /* Wait workstation (Y/N)              */
         WSVIRT ,              /* Virtual workstation (Y/N)           */
         WSDES# ,              /* Number of destinations              */
         WSUSAGE ,             /* Server usage (B/P/C/N)              */
         WSZCENTR ,            /* Z-Centric Workstation (Y/N)         */
         WSRENG)               /* Remote engine type (Z/D/blank)      */

/*-------------------------------------------------------------------+
 | SEGMENT=WSCPUREC - Workstation CPU record                         |
 +-------------------------------------------------------------------*/
OUTPUT WSCPUREC DATA(OUTDATA) LOADER(OUTBL)
  KEYS(WSNAME)                 /* Workstation name                    */

  FIELDS(CPUOS ,               /* Operating system                    */
         CPUNODE ,             /* Node name                           */
         CPUTCPIP ,            /* Port number                         */
         CPUDOMAIN ,           /* Domain name                         */
         CPUHOST ,             /* Cpu name                            */
         CPUACCESS ,           /* Access method                       */
         CPUTYPE ,             /* Agent type                          */
         CPUAUTLNK ,           /* Auto link (ON/OFF)                  */
         CPUFULLSTAT ,         /* Full status (ON/OFF)                */
         CPURESDEP ,           /* Resolve dependencies (ON/OFF)       */
         CPUSERVER ,           /* Server ID                           */
         CPULIMIT ,            /* Job limit                           */
         CPUTZ ,               /* Time zone                           */
         CPUUSER ,             /* Default user                        */
         SSLLEVEL ,            /* SSL level (OFF/ON/ENABLED/FORCE)    */
         SSLPORT ,             /* SSL port number                     */
         FIREWALL)             /* Behind firewall (NO/YES)            */
/*-------------------------------------------------------------------+
 | SEGMENT=WSIVL    - Open interval                                  |
 +-------------------------------------------------------------------*/
OUTPUT WSIVL DATA(=) LOADER(=)
  KEYS(WSCOM.WSNAME ,          /* Workstation name                    */
       WSIVLTYPE,              /* Type of interval (WSSD/WSWD)        */
       WSIVLDAY ,              /* Interval day or date                */
       WSIVLS)                 /* Start time of interval              */

  FIELDS(WSIVLE ,              /* End time of interval                */
         WSIVLPS# ,            /* Number of parallel servers          */
         WSIVLR1# ,            /* R1 capacity                         */
         WSIVLR2# ,            /* R2 capacity                         */
         WSIVLAWS)             /* Alternate workstation name          */

/*-------------------------------------------------------------------+
 | SEGMENT=WSSD     - Specific date                                  |
 +-------------------------------------------------------------------*/
OUTPUT WSSD DATA(=) LOADER(=)
  KEYS(WSCOM.WSNAME ,          /* Workstation name                    */
       WSSDDATE)               /* Specific date                       */
```

```
  FIELDS(WSSDDESC ,            /* Description of the date             */
         WSSDIVL#)             /* Number of open intervale            */

/*-------------------------------------------------------------------+
 | SEGMENT=WSWD     - Weekday                                        |
 +-------------------------------------------------------------------*/
OUTPUT WSWD DATA(=) LOADER(=)
  KEYS(WSCOM.WSNAME ,          /* Workstation name                    */
       WSWDDAY)                /* Week day                            */

  FIELDS(WSWDDESC ,            /* Description of the date             */
         WSWDDIVL#)            /* Number of open intervale            */

/*-------------------------------------------------------------------+
 | SEGMENT=WSAM     - Workstation access method                      |
 +-------------------------------------------------------------------*/
OUTPUT WSAM DATA(=) LOADER(=)
  KEYS(WSCOM.WSNAME)           /* Workstation name                    */

  FIELDS(WSAMACC ,             /* Access method name                  */
         WSAMADDR ,            /* Node address                        */
         WSAMPORT)             /* Port                                */

/*-------------------------------------------------------------------+
 | SEGMENT=WSDEST   - Workstation destination                        |
 +-------------------------------------------------------------------*/
OUTPUT WSDEST DATA(=) LOADER(=)
  KEYS(WSCOM.WSNAME ,          /* Workstation name                    */
       WSDVDEST)               /* Destination                         */

/*-------------------------------------------------------------------+
 | SEGMENT=WSVCOM   - Common                                         |
 +-------------------------------------------------------------------*/
OUTPUT WSVCOM DATA(OUTDATA) LOADER(OUTBL)
  KEYS(WSVNAME,                /* Workstation name                    */
       WSVDESTN)               /* Workstation destination             */

  FIELDS(WSVVERS ,             /* Version of record                   */
         WSVTYPE ,             /* Workstation type (G/C/P)            */
         WSVREP ,              /* Reporting attribute (A/S/C/N)       */
         WSVDDAY# ,            /* Total number of days                */
         WSVDTOTIVL# ,         /* Number of open intervals            */
         WSVPSJT ,             /* Control on servers                  */
         WSVSPLIT ,            /* Splittable attribute                */
         WSVR1NAM ,            /* R1 resource name                    */
         WSVR1PLAN ,           /* Resource used at planning           */
         WSVR1CONT ,           /* Resource used for control           */
         WSVR2NAM ,            /* R2 resource name                    */
         WSVR2PLAN ,           /* Resource used at planning           */
         WSVR2CONT ,           /* Resource used for control           */
         WSVLDATE ,            /* Date last updated                   */
         WSVLTIME ,            /* Time last updated                   */
         WSVLUSER ,            /* Userid of last updater              */
         WSVSTC ,              /* Started task ability (Y/N)          */
         WSVLUTS)              /* TOD clock at last update            */
```

```
/*-------------------------------------------------------------------+
 | SEGMENT=WSVIVLD  - Open interval                                  |
 +-------------------------------------------------------------------*/
OUTPUT WSVIVLD DATA(=) LOADER(=)
  KEYS(WSVCOM.WSVNAME ,        /* Workstation name                   */
       WSVCOM.WSVDESTN ,       /* Workstation destination            */
       WSVIVLDTYPE ,           /* Type of interval (WSSD/WSWD)       */
       WSVIVLDDAY ,            /* Interval day or date               */
       WSVIVLDS)               /* Start time of interval             */

  FIELDS(WSVIVLDE ,            /* End time of interval               */
         WSVIVLDPS# ,          /* Number of parallel servers         */
         WSVIVLDR1# ,          /* R1 capacity                        */
         WSVIVLDR2#)           /* R2 capacity                        */

/*-------------------------------------------------------------------+
 | SEGMENT=WSVSDD   - Specific date                                  |
 +-------------------------------------------------------------------*/
OUTPUT WSVSDD DATA(=) LOADER(=)
  KEYS(WSVCOM.WSVNAME ,        /* Workstation name                   */
       WSVCOM.WSVDESTN ,       /* Workstation destination            */
       WSVSDDDATE)             /* Specific date                      */

  FIELDS(WSVSDDDESC ,          /* Description of the date            */
         WSVSDDIVL#)           /* Number of open intervale           */

/*-------------------------------------------------------------------+
 | SEGMENT=WSVWDD   - Weekday                                        |
 +-------------------------------------------------------------------*/
OUTPUT WSVWDD DATA(=) LOADER(=)
  KEYS(WSVCOM.WSVNAME ,        /* Workstation name                   */
       WSVCOM.WSVDESTN ,       /* Workstation destination            */
       WSVWDDDAY)              /* Week day                           */

  FIELDS(WSVWDDDESC ,          /* Description of the date            */
         WSVWDDDIVL#)          /* Number of open intervals           */




/*-------------------------------------------------------------------+
 | GROUP=CP      - All Current Plan objects                          |
 +-------------------------------------------------------------------*/
/*-------------------------------------------------------------------+
 | RECORD=CPOC      - Current Plan occurrence                        |
 +-------------------------------------------------------------------*/
/*-------------------------------------------------------------------+
 | SEGMENT=CPOC     - Current Plan occurrence                        |
 +-------------------------------------------------------------------*/
OUTPUT CPOC DATA(OUTDATA)
  KEYS(CPOCADI ,               /* Application ID                     */
/*     CPOCIA ,                /* Input arrival                      */
       CPOCIAD ,               /* Modified if IA is modified         */
       CPOCIAT)                /* else original from plan            */

  FIELDS(CPOCGRP ,             /* Authority group                    */
/*       CPOCIAO ,             /* Input arrival from LTP             */
         CPOCIAOD ,            /* Date                               */
```

```
         CPOCIAOT ,            /* Time                                */
         CPOCDESC ,            /* Descriptive text                    */
         CPOCOID ,             /* Owner ID                            */
         CPOCODES ,            /* Owner description                   */
/*       CPOCDL ,              /* Deadline                            */
         CPOCDLD ,             /* Date                                */
         CPOCDLT ,             /* Time                                */
/*       CPOCAA ,              /* Actual arrival                      */
         CPOCAAD ,             /* if arrived                          */
         CPOCAAT ,             /* else blanks                         */
/*       CPOCAC ,              /* Actual completion                   */
         CPOCACD ,             /* if completed                        */
         CPOCACT ,             /* else blanks                         */
         CPOCERR ,             /* Occurrence error code               */
         CPOCST ,              /* Occurrence status                   */
         CPOCRER ,             /* Rerun requested (Y/N)               */
         CPOCADDED ,           /* Added to current plan (Y/N)         */
         CPOCLATE ,            /* Latest out passed (Y/N)             */
         CPOCADDF ,            /* Adding function (E/D/P/A/ )         */
         CPOCMON ,             /* Monitoring flag                     */
         CPOCPRI ,             /* Priority                            */
         CPOC#OP ,             /* No. ops in occurrence               */
         CPOCOPC ,             /* No. ops completed                   */
         CPOC#ER ,             /* No. ops ended in error              */
         CPOC#UN ,             /* No. ops undecided                   */
         CPOC#ST ,             /* No. ops started                     */
         CPOCRDU ,             /* Remaining dur critical path         */
         CPOCROP ,             /* Remaining ops critical path         */
         CPOCCWS ,             /* Wsname of 1st critical op           */
         CPOCCOP ,             /* Op no. of 1st critical op           */
         CPOCVERS ,            /* Version number=1                    */
         CPOCJVT ,             /* JCL variable table                  */
         CPGROUPID ,           /* Group definition ID                 */
         CPOCCAL ,             /* Calendar name                       */
         CPOCRDUI ,            /* Remain. dur crit. path sec          */
         CPOCOCTO ,            /* Occurrence token                    */
/*       CPOCCLO ,             /* First critical op latest out        */
         CPOCCLOD ,            /* Date                                */
         CPOCCLOT ,            /* Time in 100th of sec.               */
         CPOCETTCRIT ,         /* ETT criteria                        */
         CPOCETTTYP ,          /* ETT type: J or R                    */
         CPOCETTJOB ,          /* ETT job name                        */
         CPOCETTJID ,          /* ETT job ID                          */
         CPOCETTGROOT ,        /* ETT GDG root                        */
         CPOCETTEVNAM ,        /* Complete ETT event name             */
         CPOCETTGGEN)          /* ETT GDG generation                  */

/*-------------------------------------------------------------------+
 ! SEGMENT=CPOCPRE    - Predecessor                                  !
 +-------------------------------------------------------------------*/
OUTPUT CPOCPRE DATA(=)
  KEYS(CPOCCOM.CPOCADI ,       /* Owning application ID               */
       CPOCCOM.CPOCIAD ,       /* Owning application IA date          */
       CPOCCOM.CPOCIAT ,       /* Owning application IA time          */
       CPOCPREADI ,            /* Application ID                      */
       CPOCPRENO ,             /* Operation number                    */
/*     CPOCPREIA ,             /* Input Arrival                       */
       CPOCPREIAD ,            /* Modified if IA is modified          */
```

```
       CPOCPREIAT)             /* else original from plan             */

  FIELDS(CPOCPRECO ,           /* Predecessor completed (Y/N)         */
         CPOCPRENR ,           /* PRED. WS was nonreporting           */
         CPOCPRETT ,           /* Transport time                      */
         CPOCPREND ,           /* Pending pred occurrence             */
         CPOCPREVERS ,         /* Version number=1                    */
         CPOCPREJN ,           /* Predecessor job name                */
         CPOCPREST ,           /* Predecessor status                  */
         CPOCPMATC ,           /* Predecessor resolution criteria     */
                               /* Blank (Manually chosen)             */
                              /* C (Closest preceding)               */
                               /* S (Same day)                        */
                               /* A (Absolute interval)               */
                               /* R (Relative interval)               */
         CPOCPRECRITPATH ,     /* Predecessor of an operation         */
                               /* belonging to a critical path        */
         CPOCPMANDP)           /* Y: Mandatory Pending -Cannot be set */

/*-------------------------------------------------------------------+
 ! SEGMENT=CPOCSUC    - Successor                                    !
 +-------------------------------------------------------------------*/
OUTPUT CPOCSUC DATA(=)
  KEYS(CPOCCOM.CPOCSUCADI ,    /* Owning application ID               */
       CPOCCOM.CPOCSUCIAD ,    /* Owning application IA date          */
       CPOCCOM.CPOCSUCIAT ,    /* Owning application IA time          */
       CPOCSUCADI ,            /* Application ID                      */
       CPOCSUCNO ,             /* Operation number                    */
/*     CPOCSUCIA ,                Input Arrival                       */
       CPOCSUCIAD ,            /* Modified if IA is modified          */
       CPOCSUCIAT)             /* else original from plan             */

  FIELDS(CPOCSUCCR ,           /* On critical path (Y/N)              */
         CPOCSUCVERS ,         /* Version number=1                    */
         CPOCSUCJN ,           /* Successor job name                  */
         CPOCSUCST)            /* Successor status                    */

/*-------------------------------------------------------------------+
 | RECORD=CPOP      - Current Plan operation                        |
 +-------------------------------------------------------------------*/
/*-------------------------------------------------------------------+
 | SEGMENT=CPOPCOM  - Common                                        |
 +-------------------------------------------------------------------*/
OUTPUT CPOPCOM DATA(OUTDATA)
  KEYS(CPOPADI ,               /* Application ID                      */
       CPOPNO ,                /* Operation number                    */
/*     CPOPIA ,                /* Application input arrival           */
       CPOPIAD ,               /* modified if ia is modified          */
       CPOPIAT)                /* else original from plan             */

  FIELDS(CPOPGRP ,             /* Authority group                     */
         CPOPDESC ,            /* Descriptive text                    */
         CPOPJBN ,             /* Op OS jobname / blank               */
         CPOPJES ,             /* Job id                              */
         CPOPWSN ,             /* Workstation name                    */
         CPOPFRM ,             /* Form number / blank                 */
/*       CPOPPS ,              /* Planned start                       */
         CPOPPSD ,             /* Date / blank                        */
```

```
         CPOPPST ,             /* Time / blank                        */
/*       CPOPPE ,              /* Planned end                         */
         CPOPPED ,             /* Date / blank                        */
         CPOPPET ,             /* Time / blank                        */
/*       CPOPOI ,              /* Operation input arrival             */
         CPOPOID ,             /* Date / blank                        */
         CPOPOIT ,             /* Time / blank                        */
/*       CPOPOD ,              /* Operation deadline                  */
         CPOPODD ,             /* Date / blank                        */
         CPOPODT ,             /* Time / blank                        */
/*       CPOPLO ,              /* Latest out                          */
         CPOPLOD ,             /* Date                                */
         CPOPLOT ,             /* Time                                */
/*       CPOPAS ,              /* Actual start                        */
         CPOPASD ,             /* Date / blank                        */
         CPOPAST ,             /* Time / blank                        */
/*       CPOPAA ,              /* Actual arrival                      */
         CPOPAAD ,             /* Date / blank                        */
         CPOPAAT ,             /* Iime / blank                        */
/*       CPOPIS ,              /* Intermed.start,if intrupted         */
         CPOPISD ,             /* Date / blank                        */
         CPOPIST ,             /* Time / blank                        */
/*       CPOPAE ,              /* Actual end                          */
         CPOPAED ,             /* Date / blank                        */
         CPOPAET ,             /* Time / blank                        */
/*       CPOPED ,              /* Estimated duration                  */
         CPOPEDH ,             /* Estimated duration hours HH         */
         CPOPEDM ,             /* Estimated duration mins MM          */
/*       CPOPAD ,              /* Actual duration                     */
         CPOPADH ,             /* Act. duration hrs HHHH / blank      */
         CPOPADM ,             /* Act. duration mins MM / blank       */
         CPOPST ,              /* Current status                      */
         CPOPERR ,             /* Error code                          */
         CPOPXST ,             /* Extended status                     */
         CPOP#PS ,             /* No. parallel servers required       */
         CPOP#R1 ,             /* No. r1 resources required           */
         CPOP#R2 ,             /* No. r2 resources required           */
         CPOPPRI ,             /* Priority                            */
         CPOP#SU ,             /* Number of successors                */
         CPOP#PR ,             /* Number of predecessors              */
         CPOP#PC ,             /* No. completed predecessors          */
         CPOP#SR ,             /* No. of special resources            */
         CPOPPTT ,             /* Transport time if pred, min         */
         CPOPRDD ,             /* SMF reader date                     */
                               /* (00YYDDDF or 01YYDDDF)              */
         CPOPRDT ,             /* SMF reader time                     */
         CPOPJCL ,             /* Jobclass, sysout class / blank      */
         CPOPAEC ,             /* Auto error completion (Y/N)         */
         CPOPASUB ,            /* Auto job submission (Y/N)           */
         CPOPAJR ,             /* Auto hold/release (Y/N)             */
         CPOPTIME ,            /* Time job (Y/N)                      */
         CPOPCLATE ,           /* Cancel if late (Y/N/blank)          */
         CPOPMCPUP ,           /* Time of last mcp update             */
         CPOPCPTH ,            /* On critical path (F/Y/N)            */
         CPOPLATE ,            /* Latest out passed (Y/ )             */
         CPOPURG ,             /* Urgent (Y/ )                        */
         CPOPJST ,             /* Job status (H/Q/ /N)                */
         CPOPPREP ,            /* JCL preparation op. (Y/N)           */
```

```
         CPOPOIST ,            /* Op instr exist (Y/N/+)              */
         CPOPHRC ,             /* Highest OK return code              */
         CPOPVERS ,            /* Version number=1                    */
         CPOPWTO ,             /* Deadline WTO (Y/N)                  */
         CPOPRES ,             /* Restartable (Y/N/blank)             */
         CPOPRER ,             /* Reroutable (Y/N/blank)              */
         CPOPHRCS ,            /* Highest RC set (Y/N/blank)          */
         CPOPMHLD ,            /* Manually held op (Y/N/blank)        */
         CPOPNOP ,             /* NOPed operation (Y/N/blank)         */
         CPOPCATM ,            /* Restart and cleanup A=Autom.,       */
                               /* I=Immed., M=Manual, N=None          */
         CPOPUDA ,             /* User field                          */
/*       CPOPCMDS ,            /* Operation commands                  */
         CPOPCMD ,             /* Operation command                   */
         CPOPCSTA ,            /* Cleanup status                      */
/*       CPOPWSINFO ,          /* Workstation information             */
         CPOPWSISET ,          /* Info available (Y/N)                */
         CPOPWSTYPE ,          /* WS type G/C/P                       */
         CPOPWSREP ,           /* Reporting attr A/S/C/N              */
         CPOPWSSUBT ,          /* Subtype JCL, STC, WTO, NONE         */
                               /* J/S/W blank                         */
         CPOPWSSTAT ,          /* Status A/F/O/U/blank                */
         CPOPWSRRM ,           /* Reroute mode (Y/N)                  */
         CPOPJCRT ,            /* Workload monitor critical job       */
         CPOPJPOL ,            /* Workload monitor late job policy    */
         CPOPEDUI ,            /* Estimated dur. in 100th of sec      */
         CPOPADUI ,            /* Actual dur. in 100th of sec         */
         CPOPPSTI ,            /* Plan. start time 100th of sec       */
         CPOPPETI ,            /* Plan. end time in 100th of sec      */
         CPOPLOTI ,            /* Latest out time 100th of sec        */
         CPOPASTI ,            /* Actual start time 100th of sec      */
         CPOPAATI ,            /* Actual arr. time 100th of sec       */
         CPOPISTI ,            /* Int. start time 100th of sec        */
         CPOPAETI ,            /* Actual end time 100th of sec        */
         CPOPEXPJCL ,          /* Expanded JCL needed                 */
         CPOPUSRSYS ,          /* User sysout needed                  */
         CPOPOCTO ,            /* Occurrence token                    */
         CPOPMON ,             /* Monitoring flag                     */
         CPOPNLVL ,            /* Max nesting level                   */
         CPOPRECIS ,           /* Y if CPREC segment exists           */
         CPOPDELAY ,           /* Started on wait workstation (Y/N)   */
         CPOPCRITPATH ,        /* Belonging to critical path          */
         CPOPWLMCLASS ,        /* WLM service class                   */
         CPOPWAITSE ,          /* Waiting for scheduling              */
                               /* environment (N/S/Y)                 */
         CPOPVIRTDEST ,        /* Submission destination              */
         CPOPEXECDEST ,        /* Execution destination               */
         CPOPDREM ,            /* Removable by DP                     */
         CPOPCONDRJOB ,        /* Conditional recovery job            */
         CPOPUNEXPRC ,         /* Unexpected RC (Y/N)                 */
         CPOPSHADOW ,          /* Shadow job (Y/N)                    */
         CPOP#CPROP ,          /* Number of conditional predecessors  */
         CPOP#CSUOP ,          /* Number of conditional successors    */
         CPOP#CONDTOT ,        /* Number of conditions                */
         CPOP#COND_T ,         /* Number of True conditions           */
         CPOP#COND_F ,         /* Number of False conditions          */
         CPOP#PX ,             /* Number of predecessors in X status  */
         CPOPORIGRC ,          /* Original return code                */
```

```
         CPOPBNDST ,           /* Bind status for shadow jobs         */
                               /* Possible values for CPOPBNDST are:  */
                               /* P - Bind sent                       */
                               /* J - Sending bind                    */
                               /* B - Bind error                      */
                               /* I - Bind OK                         */
         CPOPWPEND ,           /* Waiting for at least 1 P-Pred       */
         CPOPWMPEND ,          /* Waiting for at least 1 Mand Pred    */
         CPOPWMPPEND)          /* Waiting for at least 1 Mand/P-Pred  */

         CPOPLATEL ,           /* If Y job IS late L                  */
         CPOPLATEN ,           /* If Y job IS late N                  */
         CPOPLATEE ,           /* If Y job IS late E                  */
         CPOPRUNC ,            /* Applied run cycle                   */
         CPOPTOD ,             /* Last update TOD                     */
       / *CPOPORIGDL,             original operation                 */
       / *CPOPORIGDLDATE,        deadline date and time               */
         CPOPORIGDLD ,         /* date | blank                        */
         CPOPORIGDLT ,         /* time | blank                        */
         CPOPORIGDLA ,         /* A,N,C,E, blank                      */
         CPOPULATE ,           /* If Y job IS late                    */
         CPOPMOVEDL)           /* deadline is moved to                */
/*-------------------------------------------------------------------+
 | SEGMENT=CPEXT    - Operation extended name                        |
 +-------------------------------------------------------------------*/
OUTPUT CPEXT DATA(=)
  KEYS(CPOPCOM.CPOPADI ,       /* Application ID                      */
       CPOPCOM.CPOPIA ,        /* Application input arrival           */
       CPEXTOWNOP)             /* Owning op number                    */

  FIELDS(CPEXTNAME ,           /* Extended name                       */
         CPEXTVERS)            /* Record version number               */
/*-------------------------------------------------------------------+
 | SEGMENT=CPSAI    - Operation system automation information        |
 +-------------------------------------------------------------------*/
OUTPUT CPSAI DATA(=)
  KEYS(CPOPCOM.CPOPADI ,       /* Owning application ID               */
       CPOPCOM.CPOPIAD ,       /* Owning application IA date          */
       CPOPCOM.CPOPIAT ,       /* Owning application IA time          */
       CPOPCOM.CPOPNO  ,       /* Owning operation number             */
       CPSAIAUTOOPER   ,       /* SA Automated Operator               */
       CPSAISECELEM    ,       /* SA Security Element                 */
       CPSAICOMPINFO   ,       /* SA Completion Info                  */
       CPSAIOWNOP      )       /* Owning Operation Number             */
  FIELDS(CPSAICOMMTEX1 ,         /* Command Text Line 1               */
         CPSAICOMMTEX2 ,         /* Command Text Line 2               */
         CPSAICOMMTEX3 ,         /* Command Text Line 3               */
         CPSAICOMMTEX4 ,         /* Command Text Line 4               */
         CPSAIAUTOOPER ,         /* Aut Func Exit                     */
         CPSAISECELEM  ,         /* Security Code                     */
         CPSAICOMPINFO ,         /* Complete Info Exit                */
         CPSAIOWNOP    )         /* Owning Operation Number           */

/*-------------------------------------------------------------------+
 | SEGMENT=CPREND   - Distributed remote job info                    |
 +-------------------------------------------------------------------*/
OUTPUT CPREND DATA(=)
  KEYS(CPOPCOM.CPOPADI ,       /* Owning application ID               */
```

```
       CPOPCOM.CPOPIA ,        /* Owning application input arrival    */
       CPRDOWOP)               /* Owning operation number             */

  FIELDS(CPRDVERS ,            /* Version                             */
         CPRDCOMP ,            /* Complete on failed bind (Y/N)       */
         CPRDJSN ,             /* Job stream name                     */
         CPRDJSWS ,            /* Job stream workstation              */
/*       CPRDIAD ,             /* Input arrival date (YYMMDD) | blank */
/*       CPRDIAT ,             /* Input arrival time (HHMM) | blank   */
         CPRDIA)               /* Input arrival (YYMMDDHHMM) | blank  */

/*-------------------------------------------------------------------+
 | SEGMENT=CPRENZ   - z/OS remote job info                           |
 +-------------------------------------------------------------------*/
OUTPUT CPRENZ DATA(=)
  KEYS(CPOPCOM.CPOPADI ,       /* Owning application ID               */
       CPOPCOM.CPOPIA ,        /* Owning application input arrival    */
       CPRZOWOP)               /* Owning operation number             */

  FIELDS(CPRZVERS ,            /* Version                             */
         CPRZCOMP ,            /* Complete on failed bind (Y/N)       */
         CPRZOPNO ,            /* Operation number                    */
         CPRZOCCN ,            /* Application ID                      */
         CPRZWS ,              /* Job workstation                     */
         CPRZJOBN ,            /* Job name                            */
/*       CPRZIAD ,             /* Input arrival date (YYMMDD) | blank */
/*       CPRZIAT ,             /* Input arrival time (HHMM) | blank   */
         CPRZIA)               /* Input arrival (YYMMDDHHMM) | blank  */

/*-------------------------------------------------------------------+
 | SEGMENT=CPSR     - Special resource                               |
 +-------------------------------------------------------------------*/
OUTPUT CPSR DATA(=)
  KEYS(CPOPCOM.CPOPADI ,       /* Owning application ID               */
       CPOPCOM.CPOPIA ,        /* Owning application input arrival    */
       CPOPCOM.CPOPNO ,        /* Owning operation number             */
       CPSRN)                  /* Name                                */

  FIELDS(CPSRU ,               /* Usage (S=Shared, X=Exclusive)       */
         CPSRVERS ,            /* Version                             */
         CPSRONER ,            /* On error flag                       */
         CPSRAMNT)             /* Quantity                            */

/*-------------------------------------------------------------------+
 | SEGMENT=CPOPSRU  - Special resource usage                         |
 +-------------------------------------------------------------------*/
OUTPUT CPOPSRU DATA(OUTDATA)
  KEYS(CPOPUADI,               /* Application id                      */
       CPOPUIA,                /* Application input arrival           */
       CPOPUNO)                /* Operation number                    */

  FIELDS(CPOPUIAD,             /* Modified if ia is modified          */
         CPOPUIAT,             /* Else original from plan             */
         CPOPUJBN,             /* Op os jobname                       */
         CPOPUWSN,             /* Ws name                             */
         CPOPULO,              /* Latest out                          */
         CPOPULOD,             /* Date, blank if in-use list          */
         CPOPULOT,             /* Time, blank if in-use list          */
```

```
         CPOPUAS,              /* Actual start                          */
         CPOPUASD,             /* Date, blank if wait queue             */
         CPOPUAST,             /* Time, blank if wait queue             */
         CPOPUEDU,             /* Estimated duration                    */
         CPOPUEDH,             /* Est dur hh                            */
         CPOPUEDM,             /* Est dur mm                            */
         CPOPUST,              /* Current state                         */
         CPOPUVERS,            /* Version                               */
         CPOPUPRI,             /* Priority                              */
         CPOPUSRQ,             /* Sr quantity used/needed               */
         CPOPUWRS,             /* Reason for wait for sr                */
         CPOPUSRU,             /* Sr allocation type                    */
         CPOPUEDUI)            /* Est dur in 100th sec                  */

/*-------------------------------------------------------------------+
 | SEGMENT=CPREC    - Operation recovery                             |
 +-------------------------------------------------------------------*/
OUTPUT CPREC DATA(=)
  KEYS(CPRECAID ,              /* Application ID                        */
       CPRECNO ,               /* Operation number                      */
       CPRECJREID ,            /* ID of recovery job                    */
       CPRECIA)                /* Input arrival                         */

  FIELDS(CPRECIAD ,            /* Modified if IA is modified            */
         CPRECIAT ,            /* else original from plan               */
         CPRECWSN ,            /* workstation name of rec job           */
         CPRECS ,              /* Recovery job start                    */
         CPRECSD ,             /* Date / blank                          */
         CPRECST ,             /* Time SEC*100 / 0                      */
         CPRECE ,              /* Recovery job end                      */
         CPRECED ,             /* Date / blank                          */
         CPRECET ,             /* Time SEC*100 / 0                      */
         CPRECRJST ,           /* Recovery job status                   */
         CPRECTYPE ,           /* Recovery type                         */
                               /* S - Stop                              */
                               /* C - Continue                          */
                               /* R - Rerun                             */
         CPRECDUR ,            /* Recovery job duration                 */
         CPRECPROMPTID ,       /* Recovery prompt ID                    */
         CPRECPRTMSG ,         /* Recovery message                      */
         CPRECPRTSTAT ,        /* Recovery prompt status                */
                               /* ' ' --> No reply                      */
                               /* 'N' --> Reply with N                  */
                               /* 'Y' --> Reply with Y                  */
         CPRECJID ,            /* ID of job to recover                  */
         CPRECERC ,            /* Recovery job error code               */
         CPRECVERS)            /* Version number                        */

/*-------------------------------------------------------------------+
 | SEGMENT=CPPRE    - Predecessor                                    |
 +-------------------------------------------------------------------*/
OUTPUT CPPRE DATA(=)
  KEYS(CPOPCOM.CPOPADI ,       /* Owning application ID                 */
       CPOPCOM.CPOPIAD ,       /* Owning application IA date            */
       CPOPCOM.CPOPIAT ,       /* Owning application IA time            */
       CPOPCOM.CPOPNO ,        /* Owning operation number               */
       CPPREADI ,              /* Application ID                        */
       CPPRENO ,               /* Operation number                      */
```

```
/*     CPPREIA ,               /* Input Arrival                       */
       CPPREIAD ,              /* Modified if IA is modified          */
       CPPREIAT)               /* else original from plan             */

  FIELDS(CPPRECO ,             /* Predecessor completed (Y/N)         */
         CPPRENR ,             /* PRED. WS was nonreporting           */
         CPPRETT ,             /* Transport time                      */
         CPPREND ,             /* Pending pred occurrence             */
         CPPREVERS ,           /* Version number=1                    */
         CPPREJN ,             /* Predecessor job name                */
         CPPREST ,             /* Predecessor status                  */
         CPPMATC ,             /* Predecessor resolution criteria     */
                               /* Blank (Manually chosen)             */
                               /* C (Closest preceding)               */
                               /* S (Same day)                        */
                               /* A (Absolute interval)               */
                               /* R (Relative interval)               */
         CPPRECRITPATH ,       /* Predecessor of an operation         */
                               /* belonging to a critical path        */
         CPPMANDP)             /* Y: Mandatory Pending -Cannot be set */

/*-------------------------------------------------------------------+
 | SEGMENT=CPCPR    - Current Plan conditional predecessor           |
 +-------------------------------------------------------------------*/
OUTPUT CPCPR DATA(=)
  KEYS(CPOPCOM.CPOPADI ,       /* Application ID                      */
       CPOPCOM.CPOPIA ,        /* Application input arrival           */
       CPOPCOM.CPOPNO ,        /* Owning op number                    */
       CPCPREADI ,             /* Predecessor application ID          */
       CPCPREIA ,              /* Predecessor input arrival           */
/*     CPCPREIAD ,             /* Predecessor input arrival date      */
/*     CPCPREIAT ,             /* Predecessor input arrival           */
       CPCPRENO ,              /* Predecessor operation number        */
       CPCPRE_CID)             /* Condition ID                        */

  FIELDS(CPCPRECO ,            /* Predecessor completed (Y/N)         */
         CPCPRENR ,            /* Predecessor WS was non reporting    */
         CPCPRETT ,            /* Transport time                      */
         CPCPREND ,            /* Pending predecessor                 */
         CPCPREVERS ,          /* Version                             */
         CPCPREJN ,            /* Job name                            */
         CPCPREST ,            /* Predecessor status                  */
         CPCPMATC ,            /* Predecessor resolution criteria     */
                               /* Blank (Manually chosen)             */
                               /* C (Closest preceding)               */
                               /* S (Same day)                        */
                               /* A (Absolute interval)               */
                               /* R (Relative interval)               */
         CPCPRECPATH)          /* Critical predecessor                */

/*-------------------------------------------------------------------+
 | SEGMENT=CPSUC    - Successor                                      |
 +-------------------------------------------------------------------*/
OUTPUT CPSUC DATA(=)
  KEYS(CPOPCOM.CPOPADI ,       /* Owning application ID               */
       CPOPCOM.CPOPIAD ,       /* Owning application IA date          */
       CPOPCOM.CPOPIAT ,       /* Owning application IA time          */
       CPOPCOM.CPOPNO ,        /* Owning operation number             */
```

```
       CPSUCADI ,              /* Application ID                      */
       CPSUCNO ,               /* Operation number                    */
/*     CPSUCIA ,                  Input Arrival                       */
       CPSUCIAD ,              /* Modified if IA is modified          */
       CPSUCIAT)               /* else original from plan             */

  FIELDS(CPSUCCR ,             /* On critical path (Y/N)              */
         CPSUCVERS ,           /* Version number=1                    */
         CPSUCJN ,             /* Successor job name                  */
         CPSUCST)              /* Successor status                    */

/*-------------------------------------------------------------------+
 | SEGMENT=CPCSU    - Current Plan conditional succcessor            |
 +-------------------------------------------------------------------*/
OUTPUT CPCSU DATA(=)
  KEYS(CPOPCOM.CPOPADI ,       /* Application ID                      */
       CPOPCOM.CPOPIA ,        /* Application input arrival           */
       CPOPCOM.CPOPNO ,        /* Owning op number                    */
       CPCSUCADI ,             /* Successor application ID            */
       CPCSUCIA ,              /* Successor input arrival             */
/*     CPCSUCIAD ,             /* Successor input arrival date        */
/*     CPCSUCIAT ,             /* Successor input arrival             */
       CPCSUCNO ,              /* Successor operation number          */
       CPCSUC_CID)             /* Condition ID                        */

  FIELDS(CPCSUCCR ,            /* On critical path                    */
         CPCSUCVERS ,          /* Version                             */
         CPCSUCJN ,            /* Job name                            */
         CPCSUCST)             /* Successor status                    */

/*-------------------------------------------------------------------+
 | SEGMENT=CPUSRF   - Operation User field                           |
 +-------------------------------------------------------------------*/
OUTPUT CPUSRF DATA(OUTDATA)
  KEYS(CPUFADID,               /* Application ID                      */
       CPUFOPNO ,              /* ID of recovery job                  */
       CPUFIA ,                /* Input arrival                       */
       CPUFNAME)               /* User field name                     */

  FIELDS(CPUFVALUE)            /* User field value                    */


/*-------------------------------------------------------------------+
 | RECORD=CPCOND    - Current plan condition                         |
 +-------------------------------------------------------------------*/
/*-------------------------------------------------------------------+
 | SEGMENT=CPCONDCO - Current Plan Condition                         |
 +-------------------------------------------------------------------*/
OUTPUT CPCONDCO DATA(OUTBL)
  KEYS(CPCOADI ,               /* Application ID                      */
       CPCOIA ,                /* Input arrival                       */
/*     CPCOIAD ,               /* Input arrival date                  */
/*     CPCOIAT ,               /* Input arrival time                  */
       CPCOOPNO ,              /* Operation number                    */
       CPCOCID)                /* Condition ID                        */

  FIELDS(CPCODESC ,            /* Condition description               */
```

```
         CPCO#SIMP ,           /* Number of condition dependencies    */
         CPCOCOUNT ,           /* Rule type:                          */
                               /* 0 = All                             */
                               /* N>0 = At least N of                 */
         CPCOVALUE ,           /* Final condition status:             */
                               /* U: Undecided T: True F: False       */
         CPCOVERS ,            /* Version                             */
         CPCOXST)              /* Condition extended status           */

/*-------------------------------------------------------------------+
 | SEGMENT=CPSIMP   - Current Plan Condition dependency              |
 +-------------------------------------------------------------------*/
OUTPUT CPSIMP DATA(=)
  KEYS(CPCONDCO.CPCOADI ,      /* Application ID                      */
       CPCONDCO.CPCOIA ,       /* Input arrival                       */
/*     CPCONDCO.CPCOIAD ,      /* Input arrival date                  */
/*     CPCONDCO.CPCOIAT ,      /* Input arrival time                  */
       CPCONDCO.CPCOOPNO ,     /* Operation number                    */
       CPCONDCO.CPCOCID ,      /* Condition ID                        */
       CPSIPREADI ,            /* Predecessor application ID          */
       CPSIPREIA ,             /* Predecessor input arrival           */
/*     CPSIPREIAD ,            /* Predecessor input arrival date      */
/*     CPSIPREIAT ,            /* Predecessor input arrival time      */
       CPSIPREOPNO)            /* Predecessor operation number        */

  FIELDS(CPSITYP ,             /* Check type: RC or ST                */
         CPSILOG ,             /* Operator: GE GT LE LT EQ NE RG      */
         CPSIVALRC ,           /* Return code value                   */
         CPSIVALRC2 ,          /* Upper limit of return code range    */
         CPSIVALST ,           /* Status value                        */
         CPSILVAL ,            /* Condition dependency status: U T F  */
         CPSIVERS ,            /* Version                             */
         CPSIREMOVED ,         /* Condition dependency removed: (Y/N) */
         CPSISTEPMISS ,        /* Missing step end information: (Y/N) */
         CPSISTEP ,            /* Procedure invocation step name      */
         CPSIPSTEP ,           /* Step name                           */
         CPSIJOBNAME ,         /* Job name                            */
         CPSIWSNAME ,          /* Workstation name                    */
         CPSINEWSTAT)          /* New status: T F                     */


/*-------------------------------------------------------------------+
 | RECORD=CPST      - Current Plan status                            |
 +-------------------------------------------------------------------*/
/*-------------------------------------------------------------------+
 | SEGMENT=CPST     - Common                                         |
 +-------------------------------------------------------------------*/
OUTPUT CPST DATA(OUTDATA)
  FIELDS(CPSTVERS,             /* Version number=1                    */
         CPSTCRD,              /* Current plan create date            */
         CPSTCRT,              /* Current plan create time            */
         CPSTENDD,             /* Current plan end date               */
         CPSTENDT,             /* Current plan end time               */
         CPSTBUD,              /* Last backup date                    */
         CPSTBUT,              /* Last backup time                    */
         CPST1ED,              /* 1st event after backup date         */
         CPST1ET,              /* 1st event after backup time         */
         CPST1EDTS,            /* 1st evevt timestamp date            */
```

```
         CPST1ETTS,            /* 1st evevt timestamp time            */
         CPSTTURN,             /* Turnover produces ncp               */
         CPSTCP,               /* Current plan exists (Y/N)           */
         CPSTCPDDN,            /* Current plan ddname                 */
         CPSTJTDDN,            /* Job tracking ddname                 */
         CPSTJSDDN)            /* Jcl repository ddname               */


/*-------------------------------------------------------------------+
 | RECORD=CPWS      - Current Plan workstation                       |
 +-------------------------------------------------------------------*/
/*-------------------------------------------------------------------+
 | SEGMENT=CPWSCOM  - Common                                         |
 +-------------------------------------------------------------------*/
OUTPUT CPWSCOM DATA(OUTDATA)
  KEYS(CPWSN)                  /* Workstation name                    */

  FIELDS(CPWSDESC,             /* Workstation description             */
         CPWSSC#,              /* No of completed ops                 */
         CPWSSCE,              /* Estimated duration - C              */
         CPWSSCR,              /* Actual duration - C                 */
         CPWSSI#,              /* No of interrupted ops               */
         CPWSSIE,              /* Estimated duration - I              */
         CPWSSIR,              /* Actual duration - I                 */
         CPWSSS#,              /* No of started ops                   */
         CPWSSSE,              /* Estimated duration - S              */
         CPWSSR#,              /* No of ready ops                     */
         CPWSSRE,              /* Estimated duration - R              */
         CPWSSW#,              /* No of waiting ops                   */
         CPWSSWE,              /* Estimated duration - W              */
         CPWSR1IU#,            /* No of r1 resources in use           */
         CPWSR2IU#,            /* No of r2 resources in use           */
         CPWSIVL#,             /* No of open intervals                */
         CPWSTYPE,             /* Workstation type (G/C/P/R)          */
         CPWSREP,              /* Reporting attr (A/S/C/N)            */
         CPWSPSC,              /* Control on parallel servers         */
         CPWSR1N,              /* R1 resource name                    */
         CPWSR1C,              /* R1 used for control                 */
         CPWSR2N,              /* R2 resource name                    */
         CPWSR2C,              /* R2 used for control                 */
         CPWSPREP,             /* Job setup ability                   */
         CPWSVERS,             /* Version number=1                    */
         CPWSSTC,              /* Started task ability (Y/N)          */
         CPWSWTO,              /* Wto ability (Y/N)                   */
         CPWSSTAT,             /* Workstation status (A/O/F)          */
         CPWSRERUT,            /* Reroute mode (Y/N)                  */
         CPWSALTWS,            /* Alternat ws name                    */
         CPWSDEST ,            /* Destination                         */
         CPWSZCEN ,            /* Z-Centric workstation (Y/N)         */
         CPWSRETY ,            /* Remote engine type (D/Z/blank)      */
         CPWSSX)               /* Sum of suppressed cond op           */

/*-------------------------------------------------------------------+
 | SEGMENT=CPIVL    - Current Plan workstation open interval         |
 +-------------------------------------------------------------------*/
OUTPUT CPIVL DATA(=)
  KEYS(CPWSCOM.CPWSN,          /* Workstation name                    */
       CPIVLFR)                /* From yymmddhhmm                     */
```

```
  FIELDS(CPIVLFD,              /* From date yymmdd                   */
         CPIVLFT,              /* From time hhmm                     */
         CPIVLTO,              /* To yymmddhhmm                      */
         CPIVLTD,              /* To date                            */
         CPIVLTT,              /* To time                            */
         CPIVL#PS,             /* Max parallel servers               */
         CPIVL#DPPS,           /* Ps set by daily planning           */
         CPIVL#R1,             /* Current r1 capacity                */
         CPIVL#DPR1,           /* R1 set by daily planning           */
         CPIVL#R2,             /* Current r2 capacity                */
         CPIVL#DPR2,           /* R2 set by daily planning           */
         CPIVLVERS,            /* Version number                     */
         CPIVLDPAWS,           /* Dp alternate workstation           */
         CPIVLAWS,             /* Current alternate ws               */
         CPIVLMOD,             /* Y - mcp modified/added             */
         CPIVLDP)              /* Y - originates from wsd            */

/*-------------------------------------------------------------------+
 | SEGMENT=CPWSVCOM - Workstation instance                           |
 +-------------------------------------------------------------------*/
OUTPUT CPWSVCOM DATA(OUTDATA)
  KEYS(CPWSVNAM ,              /* Workstation name                   */
       CPWSVDST)               /* Workstation Destination            */

  FIELDS(                      /*                                    */
/*       CPWSVDESC ,           /* Description (not used)             */
/*       CPWSVSC# ,            /* Number of complete ops (not used)  */
/*       CPWSVSCE ,            /* Estimated duration (not used)      */
/*       CPWSVSCR ,            /* Real duration (not used)           */
/*       CPWSVSI# ,            /* Number of interupted ops (not used) */
/*       CPWSVSIE ,            /* Estimated duration (not used)      */
/*       CPWSVSIR ,            /* Real duration (not used)           */
/*       CPWSVSS# ,            /* Number of started ops (not used)   */
/*       CPWSVSSE ,            /* Estimated duration (not used)      */
         CPWSVSR# ,            /* Number of started ops              */
         CPWSVSRE ,            /* Estimated duration                 */
/*       CPWSVSW  ,            /* Number of waiting ops (not used)   */
/*       CPWSVSWE ,            /* Estimated duration (not used)      */
         CPWSVR1IU# ,          /* No of Resource 1 in use            */
         CPWSVR2IU# ,          /* No of Resource 2 in use            */
         CPWSVIVL# ,           /* No of open intervals               */
         CPWSVTYPE ,           /* Work station type: C only          */
         CPWSVREP ,            /* Reporting attribute A only         */
         CPWSVPSC ,            /* Control on parallel servers        */
         CPWSVR1N ,            /* Resource 1 name                    */
/*       CPWSVR1C ,            /* Res. used at control (not used)    */
         CPWSVR2N ,            /* Resource 2 name                    */
/*       CPWSVR2C ,            /* Res. used at control (not used)    */
         CPWSVVERS ,           /* Version number=1                   */
         CPWSVSTC ,            /* Started task (Y/N)                 */
         CPWSVSTAT)            /* Workstation status (A/O/F)         */

/*-------------------------------------------------------------------+
 | SEGMENT=CPVIVL   - Workstation instance interval                  |
 +-------------------------------------------------------------------*/
OUTPUT CPVIVL   DATA(OUTDATA)
  KEYS(CPWSVCOM.CPWSVNAM  ,    /* Workstation name                   */
```

```
       CPWSVCOM.CPWSVDST  ,    /* Destination                          */
/*     CPVIVLFR  ,             /* Interval start                       */
       CPVIVLFD  ,             /* Date YYMMDD                          */
       CPVIVLFT)               /* Time HHMM                            */

/*FIELDS(CPVIVLTO  ,           /* Interval end                         */
  FIELDS(CPVIVLTD  ,           /* Date YYMMDD                          */
         CPVIVLTT  ,           /* Time HHMM                            */
         CPVIVL#PS  ,          /* Max number of parallel servers       */
         CPVIVL#DPPS  ,        /* PS set by DP batch                   */
         CPVIVL#R1  ,          /* R1 capacity                          */
         CPVIVL#DPR1  ,        /* R1 set by DP batch                   */
         CPVIVL#R2  ,          /* R2 capacity                          */
         CPVIVL#DPR2  ,        /* R2 set by DP batch                   */
         CPVIVLVERS  ,         /* Version number                       */
         CPVIVLMOD  ,          /* Y UMCP modified or added (Y/N)       */
         CPVIVLDP)             /* Y originated by WSD                  */


/*-------------------------------------------------------------------+
 | RECORD=CRITPATH  - Critical path                                  |
 +-------------------------------------------------------------------*/
/*-------------------------------------------------------------------+
 | SEGMENT=CRPTHCOM - Critical path common (undocumented segment)    |
 +-------------------------------------------------------------------*/
OUTPUT CRPTHCOM DATA(OUTDATA)
  KEYS(CRPTADIDHED,            /* Critical job application ID         */
       CRPTOPNOHED)            /* Critical job operation number       */

  FIELDS(CRPTCTR,              /* Number of jobs in critical path     */
         CRPTWSNHED,           /* Workstation of critical job         */
         CRPTWLMCLASSHED,      /* WLM class of critical job           */
         CRPTWLMPOL)           /* WLM policy for critical job         */

/*-------------------------------------------------------------------+
 | SEGMENT=CRPTHJOB - Job in the critical path (undocumented segment) |
 +-------------------------------------------------------------------*/
OUTPUT CRPTHJOB DATA(OUTDATA)
  KEYS(CRPTADID,               /* Application ID                      */
       CRPTOPNO,               /* Operation number                    */
       CRPTOI)                 /* Operation input arrival             */

  FIELDS(CRPTWSN,              /* Workstation                         */
         CRPTDESC,             /* Description                         */
         CRPTJOBN,             /* Jobname                             */
         CRPTXDH,              /* Duration HH                         */
         CRPTXDM,              /* Duration MM                         */
         CRPTXDS,              /* Duration SS                         */
         CRPTLS,               /* Latest start                        */
         CRPTPS,               /* Planned start                       */
         CRPTAS,               /* Actual start                        */
         CRPTOD,               /* Operation Deadline                  */
         CRPTAE,               /* Actual end                          */
         CRPTAETI,             /*                                     */
         CRPTOPST,             /* Operation status                    */
         CRPTXST,              /* Extended status                     */
         CRPTMHLD,             /* Manual hold                         */
         CRPTNOP,              /* NOP                                 */
```

```
         CRPTOPPRI)            /* Priority                          */


/*-------------------------------------------------------------------+
 | RECORD=CSR       - Current Plan special resource                  |
 +-------------------------------------------------------------------*/
/*-------------------------------------------------------------------+
 | SEGMENT=CSRCOM   - Current Plan resource common                   |
 +-------------------------------------------------------------------*/
OUTPUT CSRCOM DATA(OUTDATA)
  KEYS(CSRNAME)                /* Special resource name             */

  FIELDS(CSRGROUP,             /* Group id                          */
         CSRHIPER,             /* Dlf resource (Y/N)                */
         CSRUSEDFOR,           /* Used for (n/p/c/b)                */
         CSRONERROR,           /* On error (f /fx/fs/k /  )         */
         CSROVAV,              /* Overrid availability (Y/N/ )      */
         CSROVQ,               /* Overrid quant, 0 if none          */
         CSRDEVI,              /* Deviation                         */
         CSRIVLNUM,            /* Number of intervals               */
         CSRCIVLN,             /* Current interval number           */
         CSRDESC,              /* Description                       */
         CSRLIFTIEDAT,         /* Lifespan expiration date and time */
         CSRDEFNWSC,           /* No of connected workstations      */
         CSRDEFQUANT,          /* Default quantity                  */
         CSRDEFAVAIL,          /* Default availability              */
         CSRLIFTIEACT,         /* Lifespan action (Y/N/R)           */
         CSRONCOMPL,           /* On complete (Y/N/R/blank)         */
         CSRMAXTYPE,           /* Max usage type (Y/N/R)            */
         CSRLUSER,             /* Last updating user                */
         CSRLDATE,             /* Date of last update               */
         CSRLTIME,             /* Time of last update               */
         CSRLUTS,              /* Tod clock last update             */
         CSRVER,               /* Record version                    */
         CSRACTAVAIL,          /* Actual availability               */
         CSRACTQUANT,          /* Actual quantity                   */
         CSRXUSE,              /* Amount currently used excl        */
         CSRSUSE,              /* Amount currently used shrd        */
         CSRXALL,              /* Any all excl user now (Y/N)       */
         CSRSALL,              /* Any all shrd user now (Y/N)       */
         CSRWAITQ,             /* Any on waitq (Y/N)                */
         CSRCIDATE,            /* Current interval date             */
         CSRCIFTIME,           /* Current interval from time        */
         CSRCITTIME,           /* Current interval to time          */
         CSRCIQUANT,           /* Current interval quantity         */
         CSRCIADJQ,            /* Current interval adjust qty       */
         CSRCIAVAIL)           /* Current interval avail (Y/N)      */

/*-------------------------------------------------------------------+
 | SEGMENT=CSRIVL   - Current Plan special resource interval         |
 +-------------------------------------------------------------------*/
OUTPUT CSRIVL DATA(=)
  KEYS(CSRCOM.CSRNAME,         /* Special resource name             */
       CSRIDATE,               /* Specific date                     */
       CSRIFTIME)              /* From time                         */

  FIELDS(CSRITTIME,            /* To time                           */
         CSRIQUANT,            /* Allocatable amount                */
```

```
         CSRIWSCNUM,           /* Number of connected ws's            */
         CSRIAVAIL)            /* Available (Y/N)                     */

/*-------------------------------------------------------------------+
 | SEGMENT=CSRIWS   - CP resource interval connected workstation     |
 +-------------------------------------------------------------------*/
OUTPUT CSRIWS DATA(=)
  KEYS(CSRCOM.CSRNAME,         /* Special resource name               */
       CSRIVL.CSRIDATE,        /* Specific date                       */
       CSRIVL.CSRIFTIME,       /* From time                           */
       CSRIWSNAME)             /* Workstation name                    */

/*-------------------------------------------------------------------+
 | SEGMENT=CSRDWS   - CP resource default connected workstation      |
 +-------------------------------------------------------------------*/
OUTPUT CSRDWS DATA(=)
  KEYS(CSRCOM.CSRNAME,         /* Special resource name               */
       CSRDWSNAME)             /* Workstation name                    */


/*-------------------------------------------------------------------+
 | RECORD=JCLPREP   - JCL setup variables                            |
 +-------------------------------------------------------------------*/
/*-------------------------------------------------------------------+
 | SEGMENT=JSVCOM   - JCLPREP common                                 |
 +-------------------------------------------------------------------*/
OUTPUT JSVCOM DATA(OUTDATA)
  KEYS(JSVCADID,               /* Application id                      */
       JSVCIAD,                /* Input arrival date YYMMDD           */
       JSVCIAT,                /* Input arrival time HHMM             */
       JSVCOPNO)               /* Operation number                    */

  FIELDS(JSVCIAD,              /* Input arrival date yymmdd           */
         JSVCIAT,              /* Input arrival time hhmm             */
         JSVC#VARS,            /* Number of variables                 */
         JSVCFROM)             /* Jcl from js repository (Y/N)        */

/*-------------------------------------------------------------------+
 | SEGMENT=JSVVAR   - JCLPREP Variable definition                    |
 +-------------------------------------------------------------------*/
OUTPUT JSVVAR DATA(=)
  KEYS(JSVCOM.JSVCADID,        /* Application id                      */
       JSVCOM.JSVCIAD,         /* Input arrival date yymmdd           */
       JSVCOM.JSVCIAT,         /* Input arrival time hhmm             */
       JSVCOM.JSVCOPNO,        /* Operation number                    */
       JSVVNAME)               /* Variable name                       */

  FIELDS(JSVVVALUE,            /* Value set of default value          */
         JSVVTYPE)             /* Usage type (%/&/?)                  */


/*-------------------------------------------------------------------+
 | RECORD=JS        - Job control language                           |
 +-------------------------------------------------------------------*/
/*-------------------------------------------------------------------+
 | SEGMENT=JSCOM    - Job control language segment                   |
 +-------------------------------------------------------------------*/
```

```
OUTPUT JSCOM DATA(OUTDATA) LOADER(OUTBL)
  KEYS(JSADID,                 /* Application id                      */
       JSIA,                   /* Occurrence input arrival           */
       JSOPNO)                 /* Operation number                   */

  FIELDS(JSIAD,                /* Date                               */
         JSIAT,                /* Time                               */
         JSJOBN,               /* Jobname                            */
         JSWSN,                /* Workstation name                   */
         JSST,                 /* Status                             */
         JSUPDT,               /* Last updating function             */
         JSLDATE,              /* Last updated date                  */
         JSLTIME,              /* Last updated time                  */
         JSLUSER,              /* Userid of last updater             */
         JSVERS,               /* Record version number = 1          */
         JSLINES,              /* Number of text rows                */
/*       JST,                  /* JCL text rows                      */
         JSJFROM)              /* Jcl from js repository (Y/N)       */

/*-------------------------------------------------------------------+
 | SEGMENT=JST      - Job control language text                      |
 |                    (derived from the JST field in JSCOM)          |
 +-------------------------------------------------------------------*/
OUTPUT JST DATA(=) LOADER(=)
  KEYS(JSCOM.JSADID,           /* Application id                      */
       JSCOM.JSIA,             /* Input arrival yymmddhhmm           */
       JSCOM.JSOPNO,           /* Operation number                   */
       JSTSEQ)                 /* JCL row number                     */

  FIELDS(JSTMAX,               /* Maximum number of rows             */
         JSTJCL)               /* JCL row text                       */


/*-------------------------------------------------------------------+
 | RECORD=JLCOM     - Job log                                        |
 +-------------------------------------------------------------------*/
/*-------------------------------------------------------------------+
 | SEGMENT=JLCOM    - Common                                         |
 +-------------------------------------------------------------------*/
OUTPUT JLCOM DATA(OUTDATA)
  KEYS(JLADID,                 /* Application id                      */
       JLIA,                   /* Input arrival yymmddhhmm           */
       JLOPNO)                 /* Operation number                   */

  FIELDS(JLIAD,                /* Input arrival date yymmdd          */
         JLIAT,                /* Input arrival time hhmm            */
         JLJOBN,               /* Mvs job name                       */
         JLWSN,                /* Workstation name                   */
         JLJOBID)              /* Jes job number                     */

/*-------------------------------------------------------------------+
 | SEGMENT=JLT      - Job Log Text (derived from segment JLTXT)      |
 +-------------------------------------------------------------------*/
OUTPUT JLT DATA(OUTDATA)
  KEYS(JLCOM.JLADID,           /* Application id                      */
       JLCOM.JLIA,             /* Input arrival yymmddhhmm           */
       JLCOM.JLOPNO,           /* Operation number                   */
       JLTSEQ)                 /* Line number of SYSOUT              */
```

```
  FIELDS(JLTMAX,               /* Total number of SYSOUT lines        */
         JLTTEXT)              /* Single line of SYSOUT text          */



/*-------------------------------------------------------------------+
 | GROUP=LTP     - All Long Term Plan Objects                        |
 +-------------------------------------------------------------------*/
/*-------------------------------------------------------------------+
 | RECORD=LTOC      - Long term plan occurrence                      |
 +-------------------------------------------------------------------*/
/*-------------------------------------------------------------------+
 | SEGMENT=LTOCCOM  - Common                                         |
 +-------------------------------------------------------------------*/
OUTPUT LTOCCOM DATA(OUTDATA)
  KEYS(LTOCIAD,                /* Input arrival date                  */
       LTOCADI,                /* Application id                      */
       LTOCIAT)                /* Input arrival time                  */

  FIELDS(LTOCIAO,              /* Original input arrival              */
         LTOCIAOD,             /* Original input arrival date         */
         LTOCIAOT,             /* Original input arrival time         */
         LTOCDL,               /* Deadline                            */
         LTOCDLD,              /* Deadline date                       */
         LTOCDLT,              /* Deadline time                       */
         LTOCGRP,              /* Authority group                     */
         LTOCOID,              /* Owner id                            */
         LTOCERR,              /* Occurence error code                */
         LTOCRDST,             /* Run day status (W/F)                */
         LTOCVERS,             /* Version number=1                    */
         LTOCPRI,              /* Priority                            */
         LTOC#PRE,             /* No. of external preds               */
         LTOC#SUC,             /* No. of external succs               */
         LTOC#OP,              /* No. of changed operations           */
         LTOCDEL,              /* Deleted online                      */
         LTOCADD,              /* Added to ltp                        */
         LTOCMOD,              /* Modified on ltp                     */
         LTOCMOV,              /* Run date or time modified           */
         LTOCDEPM,             /* External dep modified               */
         LTOCCOMP,             /* Completed by job tracking           */
         LTOCMOVO,             /* Moved because of optnl rule         */
         LTOCJVT,              /* Jcl variable table                  */
         LTGROUPID,            /* Groupid                             */
         LTOCCAL ,             /* Calendar name                       */
         LTOC#CPRE ,           /* Number of conditional predecessors  */
         LTOC#CSUC ,           /* Number of conditional successors    */
         LTOC#MAND)            /* Number of mandatory pending preds   */

/*-------------------------------------------------------------------+
 | SEGMENT=LTOP     - Operation                                      |
 +-------------------------------------------------------------------*/
OUTPUT LTOP DATA(=)
  KEYS(LTOCCOM.LTOCIAD,        /* Input arrival date                  */
       LTOCCOM.LTOCADI,        /* Application id                      */
       LTOCCOM.LTOCIAT,        /* Input arrival time                  */
       LTOPNO)                 /* Operation number                    */
```

```
  FIELDS(LTOPWSN,              /* Workstation name                    */
         LTOPOI,               /* Input arrival yymmddhhmm            */
         LTOPOID,              /* Input arrival date yymmdd           */
         LTOPOIT,              /* Input arrival time hhmm             */
         LTOPOD,               /* Deadline yymmddhhmm                 */
         LTOPODD,              /* Deadline date yymmdd                */
         LTOPODT,              /* Deadline time hhmm                  */
         LTOPDESC,             /* Operation text                      */
         LTOPVERS)             /* Version number=1                    */

/*-------------------------------------------------------------------+
 | SEGMENT=LTPRE    - Predecessor                                    |
 +-------------------------------------------------------------------*/
OUTPUT LTPRE DATA(=)
  KEYS(LTOCCOM.LTOCIAD,        /* Input arrival date                  */
       LTOCCOM.LTOCADI,        /* Application id                      */
       LTOCCOM.LTOCIAT,        /* Input arrival time                  */
       LTPREIAD,               /* Run date yymmdd                     */
       LTPREADI,               /* Application id                      */
       LTPREIAT)               /* Input arrival time hhmm             */

  FIELDS(LTPREDEL,             /* Dependency deleted                  */
         LTPREADD,             /* Manually added                      */
         LTPREDONE,            /* Predecessor completed               */
         LTPREVERS ,           /* Version number=1                    */
         LTPREMPEND ,          /* Y: Is mandatory pending             */
         LTPREMAND)            /* C/P/N is a required value           */

/*-------------------------------------------------------------------+
 | SEGMENT=LTCPRE   - Conditional predecessor                        |
 +-------------------------------------------------------------------*/
OUTPUT LTCPRE DATA(=)
  KEYS(LTOCCOM.LTOCIAD,        /* Input arrival date                  */
       LTOCCOM.LTOCADI,        /* Application id                      */
       LTOCCOM.LTOCIAT,        /* Input arrival time                  */
       LTCPREIAD,              /* Run date yymmdd                     */
       LTCPREADI,              /* Application id                      */
       LTCPREIAT)              /* Input arrival time hhmm             */

  FIELDS(LTCPREDEL,            /* Dependency deleted                  */
         LTCPREPDONE,          /* Predecessor completed               */
         LTCPREVERS)           /* Version number=1                    */

/*-------------------------------------------------------------------+
 | SEGMENT=LTSUC    - Successor                                      |
 +-------------------------------------------------------------------*/
OUTPUT LTSUC DATA(=)
  KEYS(LTOCCOM.LTOCIAD,        /* Input arrival date                  */
       LTOCCOM.LTOCADI,        /* Application id                      */
       LTOCCOM.LTOCIAT,        /* Input arrival time                  */
       LTSUCIAD,               /* Run date yymmdd                     */
       LTSUCADI,               /* Application id                      */
       LTSUCIAT)               /* Input arrival time hhmm             */

  FIELDS(LTSUCDEL,             /* Dependency deleted                  */
         LTSUCADD,             /* Manually added                      */
         LTSUCVERS)            /* Version number=1                    */
```

```
/*-------------------------------------------------------------------+
 | SEGMENT=LTCSUC   - Conditional successor                          |
 +-------------------------------------------------------------------*/
OUTPUT LTCSUC DATA(=)
  KEYS(LTOCCOM.LTOCIAD,        /* Input arrival date                 */
       LTOCCOM.LTOCADI,        /* Application id                     */
       LTOCCOM.LTOCIAT,        /* Input arrival time                 */
       LTCSUCIAD,              /* Run date yymmdd                    */
       LTCSUCADI,              /* Application id                     */
       LTCSUCIAT)              /* Input arrival time hhmm            */

  FIELDS(LTCSUCDEL,            /* Dependency deleted                 */
         LTCSUCVERS)           /* Version number                     */




/*-------------------------------------------------------------------+
 | GROUP=SYS     - All System Objects                                |
 +-------------------------------------------------------------------*/
 +-------------------------------------------------------------------*/
 | RECORD=XENV      - Execution Environment                          |
 +-------------------------------------------------------------------*/
/*-------------------------------------------------------------------+
 | SEGMENT=XENV     - Common                                         |
 +-------------------------------------------------------------------*/
OUTPUT XENV DATA(OUTDATA)
  FIELDS(XENVFMID              /* PIF Base FMID                      */
         XENVCONLVL            /* PIF connector level                */
         XENVFUNCLVL           /* PIF functional level               */
         XENVDEFCAL            /* Default calendar                   */
         XENVCWBASE            /* Base year for PIF dates            */
         XENVHIGHDATE          /* Highest AD valid to date           */
         XENVADDBCS            /* Application id in DBCS (Y/N/ )     */
         XENVOWDBCS)           /* Owner id in DBCS (Y/N/ )           */
```

Setting additional fields

Additional fields are available for each **OUTPUT** segment.

KEY

The fully qualified key of the segment.

TYPE

The type of segment, for example ADOP.

PARENT_KEY

The fully qualified key of the parent segment.

PARENT_TYPE

The type of parent segment, for example ADCOM.

TAG

The data passed into the **LIST** or **SELECT** statement in the **TAG** keyword. This allows output from multiple **LIST** or **SELECT** commands to be correlated back to the originating command by tagging each output record.

One single level of a key is formed from the segment type followed by a hex 00 and then each key field separated by hex 00. Therefore, a single level of a key for an application called MYAPPL with a status of Active that is valid until 31 December 2071 would have a single level key of `ADCOM 00x MYAPPL 00x A 00x 711231`.

A fully qualified key is a sequence of single keys separated by hex 01, to uniquely identify a segment within an object within the database. Therefore, operation 010 within the previously described MYAPPL would be `ADCOM 00x MYAPPL 00x A 00x 711231 01x ADOP 00x 010`.

Reserved fields

Any areas of records marked as *reserved* in the PIF are also available to Workload Automation Programming Language.

The field names for these fields are composed of RSVD and a 3 digit offset, for example RSVD023 for ADCOM.

These fields do not appear in the **EQQFLALL** member.

For more details about the program interface record formats, see *Developer's Guide: Driving HCL Workload Automation for Z*.

Composite fields

Some fields are composed from smaller fields. For example, ADOPWSINFO is composed from ADOPWSISET, ADOPWSTYPE, ADOPWSREP, and ADOPWSSUBT.

Composite fields are listed in the **EQQFLALL** member, but are commented out.

Raw and untranslated fields

Some fields are not translated into their absolute form by Workload Automation Programming Language, and output in Hexadecimal format, such as Last User Timestamps for example **ADLUTS**.

These fields are listed in the **EQQFLALL** member, but are commented out.

Appendix B. OPTIONS keywords

Use the following keywords with the **OPTIONS** statement.

ACTION – See DBMODE

This keyword has been deprecated, use **OPTIONS DBMODE** instead.

ADOICHK – Consistency check

Determines whether AD/OI consistency checks are to be made every time an application is deleted or modified.

Consistency checks involve looking in the application description data base for matches for all the operator instructions in the application. Any operator instruction without a match is deleted. The checks are made immediately after the application description PIF action has completed with a zero return code.

Y

Consistency checks are performed when an application description record is deleted or replaced by using the PIF.

N

Consistency checks are not performed (default).

ADPFX – Prefix for dynamically created applications

Specifies the application name prefix to be used in commands that generate dynamic applications (**ADDJOB**). The default is ADHOC#.

ADSFX – Suffix for dynamically created applications

Specifies the application name suffix to be used in commands that generate dynamic applications (**ADDJOB**). The default is blank.

ADVALFROM – Valid From generation

How to set the Valid From date in generated Batch Loader.

A

As is, list the date as it is defined in TWS database (default)

N

New, list the date with today's date.

yymmdd

A specific date to use as the Valid From date for any generated batch loader.

Note: From Workload Automation Programming Language version 2.4, `OPTIONS VALFROM` is accepted as an alternative syntax.

ADVERS – Application versioning

Application version support

`Y`

When applications are deleted, the `VALTO` and `VALFROM` field of any other existing versions of the same application are adjusted to provide continuous validity periods (default).

`N`

When applications are deleted, the `VALTO` and `VALFROM` field of any other existing versions of the same application are not modified.

ADWS – Workstation for dynamically submitted jobs

The default workstation to use when `ADDJOB` is creating a dynamic occurrence for a job that is not defined in the database.

BLSTYLE – Style of Batch Loader output

The output style to be sent to the Batch Loader output file.

`TWS`

Generates HCL Workload Automation for Z Batch Loader capable of being used to recreate the object.

`XML`

Generates Extended Markup Language.

CALENDAR – Set default calendar name

Used for the commands that need to know a calendar name, when none is supplied in the keywords for that command.

CHARAT – Set the at sign (@) for object variables

Within Workload Automation Programming Language, the at sign (@) is used to designate an object variable.

However, with some code pages the at sign (@) character might be displayed differently. The `CHARAT` keyword enables you to determine what character is actually in use for your code page by issuing a `SHOW OPTIONS` command.

You can also set the character to be used, so that it matches the documented character for the code page, for example `OPTIONS CHARAT(@)`

The characters you can use for these options cannot be standard upper or lower case alphabetic characters, numbers, minus signs (-) or periods (.). They must not be in conflict with any other `CHAR`*xxxx* keywords or `VARNAMES` keyword.

Note: These **OPTIONS** keywords change only these character for the uses specified. When the same characters are used as part of data in your system, or part of field names in **OUTPUT** statements or object variables, the characters are displayed according to your code page.

CHARBANG – Set the exclamation mark (!) for default variable prefix

Within Workload Automation Programming Language, the exclamation mark (!) is used as the default variable prefix.

However, with some code pages the exclamation mark (!) character might be displayed differently. The **CHARBANG** keyword enables you to determine what character is actually in use for your code page by issuing a **SHOW OPTIONS** command.

You can also set the character to be used, so that it matches the documented character for the code page, for example **OPTIONS CHARAT(!)**

The characters you can use for these options cannot be standard upper or lower case alphabetic characters, numbers, minus signs (-) or periods (.). They must not be in conflict with any other **CHAR*xxxx*** keywords or **VARNAMES** keyword.

Note: These **OPTIONS** keywords change only these character for the uses specified. When the same characters are used as part of data in your system, or part of field names in **OUTPUT** statements or object variables, the characters are displayed according to your code page.

CHARHASH – Set the number sign (#) for count object field and ENVATTR

Within Workload Automation Programming Language, the number sign (#) is used to designate the count of the number of a specified segment in object variables and a count within **VARSET ENVATTR**.

However, with some code pages the number sign (#) character might be displayed differently. The **CHARBANG** keyword enables you to determine what character is actually in use for your code page by issuing a **SHOW OPTIONS** command.

You can also set the character to be used, so that it matches the documented character for the code page, for example **OPTIONS CHARAT(#)**

The characters you can use for these options cannot be standard upper or lower case alphabetic characters, numbers, minus signs (-) or periods (.). They must not be in conflict with any other **CHAR*xxxx*** keywords or **VARNAMES** keyword.

Note: These **OPTIONS** keywords change only these character for the uses specified. When the same characters are used as part of data in your system, or part of field names in **OUTPUT** statements or object variables, the characters are displayed according to your code page.

CHARMAIL – Set the at sign (@) for email addresses

Within Workload Automation Programming Language, the at sign (@) is used to build an email address from any unqualified addresses in the **SENDMAIL** command by using the character in combination with the **OPTIONS MAILSERV** setting.

However, with some code pages the at sign (@) character might be displayed differently. The **CHARAT** keyword enables you to determine what character is actually in use for your code page by issuing a **SHOW OPTIONS** command.

You can also set the character to be used, so that it matches the documented character for the code page, for example **OPTIONS CHARAT(@)**

The characters you can use for these options cannot be standard upper or lower case alphabetic characters, numbers, minus signs (-) or periods (.). They must not be in conflict with any other **CHAR*xxxx*** keywords or **VARNAMES** keyword.

Note:

1. These **OPTIONS** keywords change only these character for the uses specified. When the same characters are used as part of data in your system, or part of field names in **OUTPUT** statements or object variables, the characters are displayed according to your code page.
2. The email addresses from **SENDMAIL** are used by the SMTP task on your system, therefore the **CHARMAIL** character must match the user and domain delimiter as understood by your SMTP server setting, for example `dino@rana.com`.

CHECK – Application integrity

Whether to check the existence for workstations with applications, when processing batch loader.

Y

Yes, cause load to fail if workstations do not exist (default).

N

No, create applications regardless.

COMMIT – File output caching

Number of output records to hold in storage before committing to disk (default is 1000).

The value specified by commit is the maximum total across all output files. Workload Automation Programming Language divides the commit total you enter by the number of files opened and commit each file when that total is reached. For example, if you specify COMMIT(1000) and have two output files, each file is committed after 500 records.

The external data queue does not count as an output file and is not affected by COMMIT, neither it influences the calculation.

COMPSUCC – Set the default values for the ADDJOB and JBSTART commands

The **COMPSUCC** keyword of the **ADDJOB** and **JBSTART** commands determines what action to take when a successor is identified as already complete.

The equivalent **OPTIONS** keyword sets the default.

IGNORE

Do not add the dependency and issue an advisory message (RC=0).

WARNING

Do not add the dependency and issue an advisory message (RC=4). This is the default.

ERROR

Do not add the dependency and issue an advisory message (RC=8).

CONINFO – Information level IEExxxI message numbers

Specifies a list of message numbers for IEE*xxx*I messages to be considered as true informational messages and *not* cause a return code of 8.

Use **SHOW OPTIONS** to see the default values for this option.

CONNAME – MCS console name

Sets the MCS console name to be activated to run a **CONSOLE** command.

The default is the name of the job currently running the command.

CONWAIT – Wait timing for response messages

Sets how long the CONSOLE command waits for a response message.

The option has 2 arguments. The first is how many seconds to wait for the first message, the second is how many seconds to wait after the last message received, before considering the command complete.

The default is **OPTIONS CONWAIT(2,1)**

CONWARN – Warning level IEExxxI message numbers

Specifies a list of message numbers for IEE*xxx*I messages to be considered as warning messages to cause a return code of 4.

Use **SHOW OPTIONS** to see the default values for this option.

CONTENTION – Retry limits

What to do in event of contention within HCL Workload Automation for Z.

The parameter has the following arguments:

Delay

The number of seconds to delay before retry (default is 30).

Retries

The number of reattempts to make (default is 10).

For example, `CONTENTION(10,5)` waits 10 seconds before retrying, up to 5 times.

`CONTENTION(0,0)` disables the contention retry feature.

You can specify a third optional positional parameter if you are using an alternate program to perform the delay. The third parameter can be used to specify the delay period directly in the format supported by the alternate delay program.

For example, if the alternate delay program had a format of *HHMMSSTT*, a delay of 1 minute, with 2 retries would be specified `CONTENTION(60,2,00010000)`.

The first two arguments are still required for diagnostic messages.

CPDEPR – Current Plan dependency resolution

Automatic resolution of external dependencies when inserting new current plan occurrences.

`Y`

Add successor and predecessor dependencies

`N`

Do not add any dependencies (default).

`P`

Add predecessor dependencies.

`S`

Add successor dependencies.

CPFAIL – How to handle Current Plan modification failure

PIF commands that update elements in the current plan, such as `MODIFY` and `INSERT`, by default will end with RC=8 if they fail. This does not prevent Workload Automation Programming Language from continuing processing, and might result in incomplete updates being committed by a subsequent `EXECUTE` statement. The `CPFAIL` option prevents this from happening.

Possible values are:

`ABORT`

Message EQQI148F is issued, causing RC=12 which stops any further Workload Automation Programming Language processing (default).

`ERROR`

Leaves the command ending with RC=8.

DATE – Workload Automation Programming Language internal date

When using the equal sign (=) to represent a DATE or TIME within Workload Automation Programming Language, by default the date and time when Workload Automation Programming Language started is used.

Workload Automation Programming Language maintains a static internal date to enable you to generate multiple statements with a current date that is consistent across the statements, regardless of whether the date changes during running.

`OPTIONS DATE` enables you to set a specific current `DATE` to use instead of the equal sign (=) or any other function that requires the current date within Workload Automation Programming Language.

The format of the date can either be *ccyymmdd* or *yymmdd*. If *yymmdd* is used, the century is calculated by using HCL Workload Automation for Z HIGHDATE and CWBASE settings.

The following example shows a result of all 6 occurrences having an IA of 0701241600:

```
OPTIONS DATE(070124) TIME(1600)
INSERT CPOC ADID(MYAPPL1) IA(=)
INSERT CPOC ADID(MYAPPL2) IA(=)
INSERT CPOC ADID(MYAPPL3) IA(=)
INSERT CPOC ADID(MYAPPL4) IA(=)
INSERT CPOC ADID(MYAPPL5) IA(=)
INSERT CPOC ADID(MYAPPL6) IA(=)
```

You can also use `OPTIONS DATE` to set the internal Workload Automation Programming Language date to be relative to the current date.

For example:

`OPTIONS DATE(+1)` sets the date that Workload Automation Programming Language will use as current to tomorrow's date.

`OPTIONS DATE(-1)` sets the date Workload Automation Programming Language will use as current to yesterday's date.

You can reset the internal Workload Automation Programming Language date and time to the current date and time by using `OPTIONS DATE(RESET)`.

DATA – ILSON data destination

File Destination (such as DD Name) to override value specified on the `OUTPUT DATA` keyword.

Using the minus sign (-) suppresses data output.

`OPTIONS DATA(-)`

Note: `OPTIONS DATA` sets a `DATA` output destination for all segments referenced by `OUTPUT` statements, regardless of whether they originally had a `DATA` keyword.

DBMODE – Mode of operation for database updates

Determines the operating mode for Batch Loader.

This keyword replaces **OPTIONS ACTION** to reduce confusion with the **ACTION** keyword on Batch Loader statements. **OPTIONS ACTION** is still supported for backwards compatibility with EQQYLTOP.

Note: When adding a new version of an application already defined but with a different VALFROM value, you must use DBMODE(ADD).

For more details, see Batch loader commands on page 242.

DECODE – Determine which fields to decode

When a segment is retrieved from HCL Workload Automation for Z, it can be automatically decoded into its separate fields. The **DECODE** keyword decides how much of each segment to decode.

ALL

Decodes every field in the segment definition. It takes more processing cycles but simplifies the coding of any **FILTER** statements that might require fields that are not in the **OUTPUT** statement for that segment.

ONLY

Decodes only key fields, and fields specified in the **OUTPUT** statement for that segment, involving much less processing. This is the default.

DELAY – Post update delay specification

Specifies a number of seconds to pause after issuing any update command to the HCL Workload Automation for Z PIF (default 0).

This is specifically to reduce the impact mass updates may have locking out other users by providing gaps in processing. The command and return code are not reported in the output until the delay has completed.

If you are using your own module to provide the WAIT functionality, you can specify the delay as a second argument in the parameter format for your WAIT module, such as **DELAY(5,00000500)**. The first argument is still required for messages.

DELAYCMD – Commands to wait after

PIF commands to consider as update commands to perform a delay after (default **DELETE EXECUTE INSERT REPLACE**).

MODIFY is not considered an update command for the purposes of DELAY, because it does not commit to HCL Workload Automation for Z until the **EXECUTE** command.

DELETE – Automatic delete processing

Sets the default value for the **DELETE** keyword of the **LIST** statement.

Y

Yes, **DELETE** the **LIST**ed records.

N

No, do not **DELETE** the **LIST**ed records (default).

D

Defer, generate **DELETE** statements for each **LIST**ed record and output them to the **DELFILE** output destination for later processing.

Note: When **OPTIONS DELETE(Y)** is used together with **OPTIONS SELECT(Y)**, the automatically generated **SELECT** statements are processed before the **DELETE** statements, allowing Batch Loader to be generated and saved for back out processes.

DELFILE – File to write deferred DELETE to

File Destination where to write deferred **DELETE** statements.

DLM – End of instream data delimiter

Defines the default delimiter for ending instream data within Batch Loader output. The default is **-END-OF-INPUT-TEXT-**.

DROP – Circumvent occurrence split for ALTER DROPSUCC/PRED

The **OPTIONS DROP** keyword provides mechanisms that circumvent occurrence inconsistencies.

Two mechanisms are provided:

Method 1

Makes both sides of the broken dependencies a successor to a specified operation number, providing that the operation number is in a status of Complete.

Method 2

Adds an operation using a named workstation and job name, or reuse a matching pre-existing one that has no successors, and makes this a success to both sides of the broken dependency.

The syntax of the **OPTIONS DROP** keyword is the following:

```
DROP(<predop>,<succws>,<succjob>,<succtext>)
```

where:

<predop>

An operation number, such as 001, to be used as a predecessor to both sides of a dropped dependency, but only if **<predop>** is in a Complete status. It does not add **<predop>**, if it does not already exist.

<succws>

Name of the workstation used to become a successor to both sides of the dropped dependency. It must be a non-reporting general workstation.

<succjob>

Name of the job used to become a successor to both sides of the dropped dependency. The default is **ZRELINK**

<succtext>

The operation description used if a successor operation is inserted. The default is **Relink dropped deps**.

DUPAUTO – Allow automatic SELECT statements to output duplicates

Controls if any duplicates automatically listed by **OPTIONS SELECT** or **OPTIONS EXPAND** are selected for processing.

YES

Any duplicates can be selected (default).

NO

Automatically listed duplicates cannot be selected.

Note: When **OPTIONS EXPAND(YES|FULL)** is set, **DUPAUTO(NO)** is also automatically set.

DURUNIT – Duration unit for Batch Loader

States the unit to specify durations.

You can use either of the following values:

- **MINUTES**
- **SECONDS** (default)

Setting this option determines how the duration against operations and the default duration for workstations is managed.

Setting it to **MINUTES** means that the **DURATION** keyword on the **ADOP** and **WSSTART** Batch Loader statements is output in minutes and is considered to be Minutes when used as input. Setting it to **SECONDS** means that the **DURATION** keyword is output and input in seconds (this is the default).

Setting the **DURUNIT** keyword will also set the PIF option **DURSEC** accordingly, to be in line with the **DURUNIT** setting. There is no need to specify the PIF **DURSEC** option, but if you do the Workload Automation Programming Language **DURUNIT** setting is changed to the appropriate setting to match it.

DYNATTR – Set attributes for dynamic log

The **OPTIONS DYNLOG** keyword causes a dynamic log file to be created to save a copy of the Workload Automation Programming Language log.

The **DYNATTR** keyword defines the TSO ALLOCATE attributes for the creation of the file.

The default is **SPACE(75,15) TRACKS LRECL(80) RECFM(F B)**

See the TSO Command Reference for details about the ALLOCATE command and the attributes you can set. The **FILE**, **DATASET**, **NEW**, and **REUSE** keywords are already set automatically and are not to be included in this keyword.

DYNLOG – Create a dynamic copy of the Workload Automation Programming Language log

During parallel testing, if the **OPTIONS FAIL(N)** or **SPOOF** keywords are used, you cannot identify the problems with the HCL Workload Automation for Z part of the processing. This keyword prompts Workload Automation Programming Language to create a copy of the log in a uniquely named data set, so that the logs can be collated and scanned for errors at points in the testing cycle.

DYNLOG defines the prefix to use for creating the logs. It can be up to 18 characters.

For example, **OPTIONS DYNLOG(MY.WAPLLOG)** creates log files with the convention **MY.WAPLLOG.D*yymmdd*.T*hhmmssx*.*jjjjjjj***

where:

yymmdd

The current date.

hhmmss

The time.

x

Unique suffix, starting with A. If two jobs with the same name run within the same second, the suffix is incremented.

jjjjjjjj

Name of the job.

Note:

1. **DYNLOG** cannot be used with the load module EQQWAPL.
2. All the messages related to the dynamic log are by default set to Advisory severity. Failing to create or write to a dynamic log does not issue a non-zero return code, because the processing performed by the job is not at risk. To change this behavior, use the **SETSEV** command to modify the severity of messages 306 and 308, which are related to failures in the dynamic log process.

EXECUTE – Automatic Current Plan EXECUTE

Automatically performs an **EXECUTE** before termination.

MANUAL

All **EXECUTE** statements must be explicitly coded.

AUTO

If Current Plan **DELETE**, **INSERT**, or **MODIFY** statements have taken place, without an intervening **EXECUTE** or **RESET**, an **EXECUTE** is performed automatically before the **TERM** statement is processed.

EXPAND – LIST related objects

Whether to generate **LIST** statements for any database elements that might be required by a **SELECT**ed database element.

NO

No, this is the default.

YES

It **LIST**s all items that are required by the object to work correctly. For example, an Application that is part of a Group needs the Group Definition; if it is an event triggered application it needs the ETT rule, any workstations, calendars or periods are also needed.

The following items are expanded by **EXPAND(YES)**:

- When an Application is **SELECT**ed, it will **LIST** any ETT rule that references it, any Group Definition it is part of, any Calendar it references, any Period or Variable table referenced in a Run Cycle or Rule, and any Operator Instruction, Workstation or Special Resource referenced by an Operation.
- When an Event Trigger is **SELECT**ed, it will **LIST** the Application it will trigger, and the Special Resource if this is a Resource trigger.
- When a Period is **SELECT**ed, it will **LIST** any Variable Table it may reference.
- When a Workstation is **SELECT**ed, it will **LIST** any Virtual Workstation Destinations it may have.

FULL

LISTs the items covered by **YES** but will also **LIST** extra items not required to make the individual object work, but are referenced by it. The extra items include:

- Applications referenced as external predecessors.
- Members of application groups.

When you use **OPTIONS EXPAND**, any **SELECT** statements generated by a **LIST** statement with **SELECT(Y)** contains also a **SELECT(Y)** keyword. This means that any items identified by the **LIST** statement for the **EXPAND** generate a **SELECT** request. Using automatic **EXPAND** and **SELECT** together can result in Batch Loader being generated for all related objects.

Note:

1. Applications **LIST**ed by the **EXPAND** option use the keyword **VALID(=)** only to **LIST** the versions valid on the running day. The **DATE** option can be used to influence this.
2. When **OPTIONS EXPAND(YES|FULL)** is set, **DUPAUTO(NO)** is also automatically set.

FAIL – Action to take with return codes

In normal use, Workload Automation Programming Language is expected to fail with non-zero return codes in the event of errors being found. However, during parallel testing when migrating from an alternative workload automation product to HCL Workload Automation for Z, it is normal to send commands to HCL Workload Automation for Z first, then to the alternative product that is controlling the workload. This enables HCL Workload Automation for Z to be in the correct state to track events from the alternative product.

The following values are valid for **FAIL**:

YES

Default. Workload Automation Programming Language flushes commands after the **OPTIONS STOPRC** return code is issued, and returns the highest return code back to the running job.

NO

Workload Automation Programming Language continues processing commands even if the **OPTIONS STOPRC** return code has been reached, and in most circumstances ends with return code zero. Severe syntax issues causes the Workload Automation Programming Language step to fail, but these should be eliminating in testing before parallel running. All error messages will be echoed to the SYSLOG to be collected as part of the parallel testing process.

QUIET

Workload Automation Programming Language continues processing commands, even if the **OPTIONS STOPRC** return code has been reached, and in most circumstances ends with return code zero. Severe syntax issues causes the Workload Automation Programming Language step to fail, but these should be eliminating in testing before parallel running. No error messages will be echoed to the SYSLOG.

Note: **OPTIONS FAIL** overrides any return codes from commands, including **SETMAX**, but is overridden if **OPTIONS SPOOF** is coded.

FASTPATH – Current Plan search option

When you want to retrieve only computer and printer operations, **FASTPATH** makes the search for operations faster.

N

All operations matching the search argument criteria that you specified are retrieved (default).

Y

HCL Workload Automation for Z searches the current plan for computer or printer operations matching the job name search argument. It then selects all operations in the occurrences that contain the computer or printer operations (that is, even operations at general workstations), and retrieves the operations based on the remaining search arguments specified.

FIELDSEP – ILSON field separator

The Field Separator character is used to separate fields in the **DATA** output. You can specify either a single character reference, for example a comma, as in **FIELDSEP(,)**, or a hex value as a two byte notation or two bytes suffixed with **X**, for example **FIELDSEP(00)** or **FIELDSEP(00x)**.

For safe parsing of your data, ensure that you use a value that is not contained in any data extracted (default is 00x).

Field Separators can be turned off by setting **FIELDSEP(NONE)**.

Note: If you want to use **EXIT** or the **EQQYXFLD** function, ensure you do not use the same value for **FIELDSEP** (default 00x) and **LABELSEP** (default =). **NONE** must not be used. If this occurs, when an EXIT is called it is not used and EXIT is reset to blank.

FILESPEC – File Specification DD statement

DD name from which to read File Specification.

FIRST – Logical First operation

Sets the default logical first operation within an application for any process that needs a logical start point.

FIRST(<operation>[,LINK][,ALL])

where:

<operation>

The number of the operation to be considered the logical start point of the application.

LINK

Instructs Workload Automation Programming Language to create an automatic **ADDEP** statement to link to the first operation for any operation without a predecessor.

ALL

Together with **LINK**, instructs Workload Automation Programming Language to create automatic **ADDEP** statements to link to the first operation for all operations, including those with predecessors.

Note:

1. **ALL** takes effect only if **LINK** is also specified.
2. **LINK** cannot be used in together with **DBMODE(COPY)** or **DBMODE(UPDATE)**.

FREEMAX – Maximum number of consecutive free days to skip

To calculate dates using workdays as the offset, a calendar might have been defined with no workdays, causing calculations to loop infinitely. Use the **FREEMAX** keyword to set a limit of consecutive free days to process before the command fails.

The default is **14**.

GTABLE – Default Global Table

Used for any commands that need to know a global table name, when none is specified in the keywords for the command.

HIGHRC – Highest accessible return code

Highest acceptable return code.

Anything up to this return code does not affect the highest return code of the job. Available values are 0, 4 or 8 (default is 0).

Note: Any return codes you set before the **HIGHRC** option is processed, affects the job highest return code.

IFCMD – Default step to consider for command return code checking

The **@CMD** function is used to check the return code of a previous command. If the command label is not specified, the default command to check is determined by this **OPTIONS** keyword.

LAST_RC

Refers to the immediately preceding command.

LAST_XRC

Refers to the last command that was run and was not flushed. This is the default.

MAX_RC

Refers to the command that issued the maximum return code.

MAX_RESP

Refers to the command that issued the maximum response code.

IFJCL – Default step to consider for JCL step return code checking

The **@JCL** tag is used to check the return code of a previous JCL step in the current job. If the command label is not specified, the default command to check is determined by this **OPTIONS** keyword.

LAST_RC

Refers to the immediately preceding step.

LAST_XRC

Refers to the last step that was run and was not flushed. This is the default.

MAX_RC

Refers to the step that issued the maximum return code.

IGNORE – Default value for ADDJOB IGNORE keyword

Sets the default specification of internal dependencies to ignore.

The default is **=FIRST =LAST**

For the valid values, see ADDJOB – Add job to the current plan on page 219.

INCLEVEL – Message level for INCLUDE statements

By default, any statements run as a result of **INCLUDE** statements is listed only when either the **MSGLEVEL** is **3** or higher, or the statement fails. Use **INCLEVEL** to specify an alternative message level for listing the contents of the included members.

OPTIONS INCLEVEL(1)

All included statement are listed at the same level as normal SYSIN commands.

OPTION INCLEVEL(2)

All included statements are listed at the same level as any statements generated by automatic **SELECT**, **DELETE**, or **EXPAND** options.

OPTIONS INCLEVEL(3)

All included statements are listed at the same level as statements within the member referenced by the **FILESPEC** symbolic in the JCL.

Levels 4 and 5 are not available for **INCLUDE** statements, because these levels are reserved for internal Workload Automation Programming Language commands.

INPUT – Command input DD statement

You can name a specific DD statement for Workload Automation Programming Language to read for the main command stream, specify an address for a control block containing the main command stream, or specify **INPUT(-OFF-)** to prevent Workload Automation Programming Language from attempting to read an **INPUT** stream from a file, so it reads only from the External Data Queue.

To specify a DD statement, include the name as the keyword value, as in the following example:

```
OPTIONS INPUT(MYCMDS)
```

To specify a control block, set an 8-character hexadecimal address with a prefix of 0x, as in the following example:

```
OPTIONS INPUT(0x0000CAA0)
```

Note: This keyword is effective only if you specify it within **EQQOPTS**, the **ARGS** symbolic parameter, or the argument when calling EQQYXTOP/EQQWAPL directly.

JSFILE – DD name of input JCL for JSSTART

Used to set the input library for the **JSSTART** batch loader command. The default value is EQQJSPDS.

LABELSEP – ILSON label separator

The Label Separator character used to separate field Labels from Values in the **DATA** output.

You can set either a direct character reference, as in **LABELSEP(=)**, or a hex value either as a two byte notation, or two bytes suffixed with **X**, for example **FIELDSEP(01)** or **FIELDSEP(01x)**.

You can turn off label separators with **LABELSEP(NONE)**.

LAST – Last logical operation

Sets the default logical last operation within an application for any process that needs a logical end point.

LAST(<operation>[,LINK][,ALL])

where:

<operation>

The number of the operation to be considered the logical end point of the application.

LINK

Instructs Workload Automation Programming Language to create an automatic **ADDEP** statement to link to the last operation for any operation without a successor.

ALL

Together with **LINK**, instructs Workload Automation Programming Language to create automatic **ADDEP** statements to link to the last operation for all operations, including ones with successors.

Note:

1. `ALL` is effective only if `LINK` is also specified.
2. `LINK` cannot be used together with `DBMODE(COPY)` or `DBMODE(UPDATE)`.

LIMIT – Unconstrained loop limit

`DO WHILE`, `DO UNTIL`, and `DO FOREVER` loops might run infinitely. This option restricts the types of loops to a maximum number of iterations, before the process fails.

The default is 100.

LOADER – Batch Loader output destination

File Destination (such as DD name) to override value specified on the `OUTPUT LOADER` keyword. Setting a minus sign (-) suppresses LOADER output.

```
OPTIONS LOADER(-)
```

Note: `OPTIONS LOADER` sets a `LOADER` output destination for all the segments referenced by the `OUTPUT` statements, regardless of whether they originally had a `LOADER` keyword.

Lock – Batch Loader locking

Whether to enable the extended Batch Loader locking for the `DBMODE UPDATE` and `DBMODE COPY` statements.

YES

Extended locking is enabled (default).

NO

Extended locking is not enabled.

The extended locking for `DBMODE UPDATE` and `DBMODE COPY` performs a lock *before* the record is read. This ensures that, if you are in the process of editing an object, the update or copy process does not read the record until you complete your changes.

If you disable the extended locking, the object is immediately read from the database even if you are modifying it, without taking into account the unsaved changes.

Extended locking is not compatible with the `INIT DATINT(Y)` statement, because this setting creates an additional lock inside the PIF at update time, which will conflict with the extended locking taken at read time. It is recommended that you use `INIT DATINT(N)` when running Workload Automation Programming Language; if you set `INIT DATINT(Y)`, any changes made while editing are not take into account.

LTDEPR – Long Term Plan dependency resolution

Automatic resolution of external dependencies when inserting new LTP occurrences.

Y

Yes.

N

No (default).

MAILDD – DD name of input text for SENDMAIL

Used to set the input library for the **SENDMAIL** command.

The default value is EQQEMAIL.

MAILFROM – Email address of mail sender

Used to set the default **FROM** value for the **SENDMAIL** command.

If no sending domain name is used, the value set in **OPTIONS MAILSERVER** is automatically appended.

MAILSERVER – Domain name of the mail server

Used to set the default **SERVER** value for the **SENDMAIL** command.

If no domain name is used in any email addresses, the value set in **OPTIONS MAILSERVER** is automatically appended.

MAILSMTP – DD name of SMTP output

Used to set the output DD name for the **SENDMAIL** command.

The default value is EQQSMTP.

MEMORY – Memory usage

This **OPTIONS** keyword is no longer effective. Original values are accepted for backwards compatibility.

MMMM – List of months

Specifies the list of month names to be used in the **DATEADD** and **DATEFORM** functions. By default, the names are in English but you can use **OPTIONS MMMM** to set the months in a different language.

This keyword must specify a list of 12 complete month names.

For example, to set the month names in Italian issue the following command:

```
OPTIONS MMMM(GENNAIO FEBBRAIO MARZO APRILE MAGGIO GIUGNO
             LUGLIO AGOSTO SETTEMBRE OTTOBRE NOVEMBRE DICEMBRE)
```

Note: Do not use accented characters, because the **LOWER** function does not support them and this might produce unpredictable results.

MSGLEVEL – Output message level

Determines the level of messages to be issued.

1. Only show commands from user input on data queue or SYSIN, or commands that have not completed successfully (default)
2. Also show commands generated from user input, such as SELECTs from LISTs.
3. Also show commands from FILESPEC source and INCLUDE statements.
4. Also show commands from OPTIONS, SUBSYS and arg source
5. Show system generated commands INIT, TERM and OPTIONS RETMSG along with commands generated as part of composite commands.

The special message levels are as follows:

-1

Lists only Fatal and Critical errors. When invoked with the SILENT immediate option, it suppresses Workload Automation Programming Language startup messages.

0

Lists only commands that did not complete successfully. Any command that issues a return code higher than **HIGHRC** is considered to be unsuccessful for this purpose. Startup and termination messages are issued.

OIFILE – DD name of input text for OISTART

Used to set the input library for the **OISTART** batch loader command. The default value is EQQOIPDS.

OPID – Identify controlling operation

When required, Workload Automation Programming Language finds automatically the occurrence that controls the job where it is running by searching for the job name and JES number in the CP or through the contents of the EQQCPOP file, which is usually populated by supplied variables.

In some scenarios, you could run a Workload Automation Programming Language job on behalf of an operation in the current plan, which could not be the actual job submitted by the controller or tracker. For example, you can submit a Workload Automation Programming Language process from the status-change-exit. To allow these processes to be connected to the operation, use the **OPID** keyword in the ARGS symbolic parameter.

To prevent the search, set the **OPID** keyword to the application ID, occurrence input arrival, and number of the operation with which you want to associate it, in the following format:

```
APPL_ID  YYMMDDHHMM  NNN
```

For example:

```
S EQQYXJPX,ARGS='OPID(MYAPPL 1701241000 010)'
```

For jobs that are not controlled by HCL Workload Automation for Z, set **OPTIONS OPID(-)** to prevent Workload Automation Programming Language from searching for that jobs.

OPMSG – Send messages to console

Whether to issue messages with severity **O** to Operator Console.

Y

Yes.

N

No (default).

OUTMASK – Output mask

Tells Workload Automation Programming Language to treat the percent sign (%) and asterisk (*) as masking characters for updating fields within Batch Loader statements.

N

Do not use masking for updates (default).

Y

Use masking for updates.

Regardless of this setting, you can still use masks in key fields to identify records.

OVERWRITE – Whether to overwrite an object during rename

When using **NEW_** prefixes for fields when using Batch Loader in **UPDATE** or **COPY** mode, they will result in a new name for the object. This new name might point to another pre-existing object. By default, the update fails. The **OVERWRITE** option enables you to modify this behaviour to allow the replacement of the existing object.

N

The **UPDATE** or **COPY** fail if the newly named object already exists.

Y

The pre-existing object is replaced by the copied or renamed object. The warning message EQQI144W is issued.

Note: This works only when the Batch Loader statements refer to an individual object. If wildcards are used in the batch loader key fields, or **USELIST** is used, the command fails if the newly named object already exists. In these

instances additional Workload Automation Programming Language commands must first delete the destination objects.

OWNER – Owner ID for tables created by VAR* commands

When a **VARDATE**, **VARSAVE**, or **VARSET** command tries to save a variable to a table that does not exist, an Owner ID must be provided. This keyword provides the value to use for these eventualities.

The default value is TWA.

PGMPIF – Program to use for HCL Workload Automation for Z communication

Program to call HCL Workload Automation for Z PIF (default is EQQYCOM).

Use this option to manage calls to different versions of HCL Workload Automation for Z from the same LPAR by providing specific versions of EQQYCOM for each version.

PGMSTOR – Program to use to manage storage

Program to allocate storage (default is EQQSTOR).

PGMWAIT – Program to use to wait

Program to use to wait for contention retry (default is EQQRETWM).

POSTPROC – Post process external data queue

After processing all initial command statements on the external data queue and in the **INPUT** source (usually SYSIN), Workload Automation Programming Language terminates. If you dynamically generate more command statements into the External Data Queue during execution, for example Batch Loader, use this option to have Workload Automation Programming Language processed these extra statements.

Y

Process external data queue until it is empty.

N

Ignore any new entries on the external data queue (default).

Note: If during the processing of the additional commands in the external data queue, other commands are generated, they are also processed unless you reset the **POSTPROC** option. This could lead to a looping condition.

PREMPTY – Action to take when creating period with DATELIST and ADID

Determines the action to take when a period automatically created is about to be created with no intervals.

When using **PRSTART** with either **DATELIST** or **ADID**, the list of calculated dates might be empty within the range **FROMDATE** – **TODATE**. If the Period does not already have interval dates, or is being replaced, an error condition is generated, because the Period must have at least one interval.

PREMPTY can have the following values:

FAIL

If empty, the request fails (default).

HIGHDATE

Uses the HCL Workload Automation for Z High date as the only interval.

LOWDATE

Uses the HCL Workload Automation for Z LOW date as the only interval.

yymmdd

Uses the specified date as the only interval.

REPORT – Report output width

Sets the report width for output messages in SYSTSPRT (default is 80).

RETMSG <unavailable option>

Restricted option. Managed internally by Workload Automation Programming Language, not permitted with Workload Automation Programming Language command stream.

RETMSGID <unavailable option>

Restricted option. Handled internally by Workload Automation Programming Language, not permitted with Workload Automation Programming Language command stream.

RUNIF – Set defaults for conditional execution

Use the **RUNIF** command to conditionally prevent jobs running under certain circumstances. The command moves identified workload to non reporting workstations and optionally removes Special Resources and Time Dependencies.

```
OPTIONS RUNIF(<wsname>,<dropsr>,<droptime>)
```

where:

<wsname>

The default name of the workstation where to switch the workload. It must be a general non-reporting workstation. If the workstation is not specified, the **RUNIF** command fails.

<dropsr>

Whether to drop special resources from identified operations. The value can either be **YES** (default) or **NO**.

<droptime>

Whether to drop time dependencies from identified operations. The value can either be **YES** (default) or **NO**.

RUNSTAT – Alter run cycle status

Modifies the unloaded values of the Valid From (**ADRVALF**) and Valid To (**ADRVALT**) dates of each run cycle. This affects the value in any generated Data or Batch Loader output. It does not affect the database, unless you process the generated Batch Loader output.

RUNSTAT is designed to enable you to turn off the planning capability of an application while still allowing it to be submitted, so that the planning capability can be later reinstated. Typically, this option is used for copying production schedules to a test environment, or disaster recovery testing.

For run cycle status within generated batch loader you can set the following values:

ACTIVATE

Puts any run cycles that are Valid To **LOWDATE** back into effect with **VALFROM** being the current Workload Automation Programming Language date and **VALTO** being **HIGHDATE** (for example, 711231).

RETAIN

Do nothing with run cycle **VALFROM** and **VALTO** values (default).

SUSPEND

Sets the **VALFROM** and **VALTO** values to **LOWDATE** (for example, 720101) for any run cycles for which the Valid To date is set to the current Workload Automation Programming Language date, or later.

Note:

1. If you **SUSPEND** a run cycle with a Valid To date set to a future date other than **HIGHDATE**, this information is lost. When the application is later reinstated with **ACTIVATE**, it uses **HIGHDATE** as the new Valid To date.
2. In the ISPF panels, the **VALTO** date is translated to an “Out of Effect” date. This means that the date listed in the “Out Of Effect” column is actually one day after the actual **VALTO** date (with the exception of **HIGHDATE**,

which is listed the same as **VALTO** as opposed to the day after). It is important to understand this as a run cycle showing an “Out Of Effect” date in the panels of “today” will actually have a **VALTO** date of “yesterday” and therefore not be affected by **RUNSTAT(SUSPEND)**.

SENDDATA – Output ILSON data

Whether to send HCL Workload Automation for Z data information to files specified in the **OUTPUT DATA** keyword.

Y

Writes data output (default).

N

Does not write data output.

Note: No output is written if no output destinations are specified.

SENDLOADER – Output Batch Loader

Whether to send Batch Loader information to files specified on the **OUTPUT LOADER** keyword.

Y

Writes data output (default).

N

Does not write data output.

Note: No output is written if no output destinations are specified.

SELECT – Automatic selection

Sets the default value for the **SELECT** keyword of the **LIST** statement.

Y

Automatically generate **SELECT** statements.

N

Does not generate **SELECT** statements (default).

SETMAX – Influence default SETMAX behaviour

Sets the default behaviour of the **SETMAX** command.

MAX_RC

Compares the **POLICY** against the current maximum return code, and modifies the maximum return code if a match is made within the **POLICY**. It also sets the return code of the **SETMAX** command to the same value. This is the default.

MAX_RESP

Compares the **POLICY** against the current maximum response code, and modifies the maximum response code if a match is made within the **POLICY**. It does not set the return code of the **SETMAX** command.

BOTH

Compares against the highest of the current maximum return code and maximum response code. It modifies both values if is a match is found within the **POLICY**. It also sets the return code of the **SETMAX** command to the same value.

SETUP – Default SETUP attribute for Workload Automation Programming Language variables when saved

Sets the default **SETUP** value for any new **VARSET** and **VARDATE** variables when saved into a JCL Variable Table.

YES

Resolves variable at SETUP phase.

NO

Resolves variable at SUBMIT phase.

PROMPT

Promptable variable.

SEVERITY – Message severity levels

Defines the message severities to display.

Within Workload Automation Programming Language, some message severities can be suppressed so that they are not listed in the output. The severities are:

A

Advisory

O

Operator

W

Warning

E

Error

This keyword instructs Workload Automation Programming Language about the severities to display (default is `AOWE`).

Note: If you suppress W or E, the return codes set by these types of messages are also ignored.

SILENT – Silent running

Sets the special message level **-1** from the start of Workload Automation Programming Language that prevents all messages except Fatal and Critical from being issued.

This option allows Workload Automation Programming Language to be called from within REXX programs that form part of ISPF dialogs without disrupting the panel flow unnecessarily.

Note: This is an immediate option and is valid only when specified as an argument to Workload Automation Programming Language.

SHOWDFLT – Show values that are set to defaults

Whether to create batch loader keywords for fields with default values.

`Y`

Output fields with default values (default).

`N`

Does not output fields with default values.

Note: In some cases this can result in Batch Loader statements with no keywords (for example, WSIVL). This is normal behaviour.

SHOWKEYS – Display key information

Whether to display the key information for each record identified by a **`LIST`** statement.

`Y`

Displays message EQQI640A for every record identified by a **`LIST`** statement.

`N`

Does not display message EQQI640A (default).

Note:

1. *All* the listed records match the **LIST** value, regardless of whether a **FILTER** is being used.
2. **OPTIONS SHOWKEYS** causes every common segment to be processed as far as extracting key fields, regardless of whether an **OUTPUT** statement is present for that segment. In large volume jobs, this might affect run time therefore it is not recommended to set **OPTIONS SHOWKEYS** in your site default member.

SPE – Small Product Enhancements

Indicates whether support for certain Small Product Enhancements (SPEs) is to be considered. Each SPE is listed in the format **spename=Y** or **spename=N**, for example **SPE(WLM=Y,SA=N)**.

Specifying only the SPE name corresponds to specifying **=Y**, for example **SPE(WLM,SA=N)**.

For a list of valid SPEs, see Small product enhancements on page 23.

STOPRC – Return code to terminate processing

The return code that causes processing to stop.

By default, Workload Automation Programming Language flushes processing when a return code 12 is found. You can cause subsequent command statements to be prevented from processing by specifying **STOPRC(4)** or **STOPRC(8)**. You can reset this code to 12 by **STOPRC(12)** (this is the default).

Note: This is not impacted by any response codes set with the **LISTSTAT** command.

STRIP – Remove trailing blanks and leading zeroes

Whether to strip blanks and leading zeroes from **DATA** and **LOADER** output.

Y

Strip blanks and leading zeroes.

N

Return data as extracted from HCL Workload Automation for Z (default).

For example:

OPTIONS STRIP(N) produces **ADID(MYAPPL)**

OPTIONS STRIP(Y) produces **ADID(MYAPPL)**

Note: Leading zeros are stripped only from fields that are defined as Signed or Unsigned in the PIF record layouts. This is to stop leading zeros being removed from character based numeric strings such as dates and times. Operation numbers are also always left with leading zeros in place.

SUBSYS – Input file for controller options

Workload Automation Programming Language can read subsystem specific **OPTIONS** statements after it has read the default options from the **OPTIONS** input source. Use the **SUBSYS** keyword to specify the DD statement from which to read the subsystem specific options (the default is <blank>).

This keyword is effective only on **OPTIONS** statements contained within the **OPTIONS** input stream.

From version 3.4 and later, instead of using **OPTIONS SUBSYS** you can code subsystem specific statements at any point in the command stream, including EQQOPTS.

For example:

```
<global settings>
IF @V(ZSUBSYS) = “TWSA” THEN DO
<controller settings>
END
```

SUFFIX – Object name suffixing

How to manage the **SUFFIX** Batch Loader keyword.

DISABLE

The **SUFFIX** keyword has no effect.

FAIL

If adding the **SUFFIX** to the name of the object the maximum length is exceeded, the process fails (default).

IGNORE

If adding the **SUFFIX** to the name of the object the maximum length is exceeded, the **SUFFIX** is ignored.

OVERLAY

If adding the **SUFFIX** to the name of the object the maximum length is exceeded, the **SUFFIX** is overlayed over the rightmost portion of the field.

TRUNCATE

If adding the **SUFFIX** to the name of the object the maximum length is exceeded, the suffix is added to the end of the field and truncated to fit the field width.

SUPMSG – Message suppression

SUPMSG allows you to prevent a message from being written to the message log.

You can prevent more than one message from being written to message log by issuing multiple **OPTIONS** requests with the **SUPMSG** argument specified. The argument to **SUPMSG** is formed by MSG followed by the message identifier. To obtain the message identifier, remove the HCL Workload Automation for Z prefix (EQQ) from the beginning of the message and the severity indicator from the end of the message.

For example, to prevent message EQQW002E from being written to the message log specify the argument value MSGW002.

Note: If **SUPMSG** is used to avoid Workload Automation Programming Language messages prefixed with EQQI, this also prevents the resulting return code for the message being set.

For example, if you specify an output file in an **OUTPUT** statement that is not allocated and then try to write to it, setting **OPTIONS SUPMSG(MSGI103)** prevents Workload Automation Programming Language from issuing RC=4

SYNTAX – Legacy syntax compatibility

Restricts Batch Loader output and input syntax to legacy Batch Loader (EQQYLTOP) compatibility.

EXTENDED

Use extended syntax (default).

LEGACY

Use only EQQYLTOP compatible syntax.

This is to allow output generated by Workload Automation Programming Language to be processed by EQQYLTOP and Batch Loader written to EQQYLTOP rules to be processed the same way.

When set to **LEGACY**, Workload Automation Programming Language applies the following restrictions:

- Any input beyond column 72 is ignored, regardless of record length.
- Batch Loader keywords not supported by EQQYLTOP for the relevant release are not generated.
- Alternative names for Batch Loader keywords are not output, but are accepted as input.

SYSID – Tracker lookup method

SYSID allows you to set the field with which to look up LPAR specific values in lookup tables such as **OPTIONS TRACKERS**.

Valid values are:

SYSNAME

The name of the system as specified in the **SYSNAME** statement in SYS1.PARMLIB (default).

SMFID

The System Management Facility Identifier.

JESNODE

The JES Node name.

TAGMODE – Set automatic tagging

TAGMODE turns on automatic tagging of descriptions for All Workstations Closed, Calendar Specific Dates, and Workstation Intervals.

By setting **TAGMODE** and the related **TAGMASK OPTIONS**, you can add a tag to the description with information containing both the Day of the Week and the Day of the Year. This allows a better understanding and verification of these dates when viewing the related items.

Tagging occurs every time the **DESCR** keyword is set for one of these objects using Batch Loader.

Valid values for **TAGMODE** are:

OFF

No tagging occurs (default).

PREFIX

The tag is added to the front of the description.

SUFFIX

The tag is added to the end of the description.

RIGHT

The tag is added to the end of the description and is right justified to the maximum length of the description.

Note: If the length of the **DESCR** and the tag exceed the length of the field, the tagged description is truncated when using **PREFIX** and **SUFFIX** and text can be overwritten when using **RIGHT**.

TAGMASK – Set tagging mask

Use **TAGMASK** to describe the layout of the tag to be added to a description of a date related option. Its use is activated by **OPTIONS TAGMODE**.

The mask can be any text string containing various characters, but the following character combinations are replaced by values relating to the date:

DDDD

The complete name of the day of the week in upper case.

Dddd

The complete name of the day of the week in mixed case.

DDD

The first 3 characters of the day of the week in upper case.

Ddd

The first 3 characters of the day of the week in mixed case.

DD

The first 2 characters of the day of the week in upper case.

Dd

The first 2 characters of the day of the week in mixed case.

number sign (#)

The day number with in the year (3 characters).

For example, **TAGMASK(' (Ddd #)')** resolves to an output like the following:

```
' (Mon 334)'
```

Note: Only one instance of Day of the Week and Day of the Year is replaced in each mask.

TEMPFILE – DD name of temporary library allocation

Used to set the DD name for allocating individual members of a library for processes that allow member name input from a PDS or PDSE.

The default value is EQQTEMP.

TIME - Workload Automation Programming Language internal time

When using the equal sign (=) to represent a **DATE** or **TIME** within Workload Automation Programming Language, by default the date and time when Workload Automation Programming Language started is used. Workload Automation Programming Language maintains a static internal time to allow you to generate multiple statements with a "current time" that is consistent across all statements, regardless of whether the time changes during run time.

OPTIONS TIME allows you to set an explicit current **TIME** to be used instead of the equal sign (=) or any other function that requires the current time within Workload Automation Programming Language.

The format of the time must be *hhmm*.

For example, the following command results in all 6 occurrences having an **IA** of 0701241600:

```
OPTIONS DATE(070124) TIME(1600)
INSERT CPOC ADID(MYAPPL1) IA(=)
INSERT CPOC ADID(MYAPPL2) IA(=)
INSERT CPOC ADID(MYAPPL3) IA(=)
INSERT CPOC ADID(MYAPPL4) IA(=)
INSERT CPOC ADID(MYAPPL5) IA(=)
INSERT CPOC ADID(MYAPPL6) IA(=)
```

You can also use **OPTIONS TIME** to set the internal Workload Automation Programming Language date to be relative to the current date.

For example, **OPTIONS DATE(+60)** sets the time that Workload Automation Programming Language will use as 60 minutes ahead of its current setting.

OPTIONS DATE(-5) sets the date that Workload Automation Programming Language will use as 5 minutes behind its current setting.

Note: If the addition or subtraction crosses a day boundary, the internal date is also changed.

You can reset the internal Workload Automation Programming Languagedate and time to the current date and time by using **OPTIONS TIME(RESET)**.

TRACE – Perform interface tracing

The **TRACE** keyword causes Workload Automation Programming Language to list detailed HCL Workload Automation for Z Program Interface information to help you debug any difficulties you might be having.

The available levels are:

0

No tracing (default).

1

Lists PIF commands as they are issued and low level diagnostic information.

2

Lists segments as they are found.

3

Lists control structures and full DATA_AREA listing for SELECT statements.

All **TRACE** messages are easily identifiable by >> at the beginning of the message text.

TRACKERS – Tracker lookup

The **TRACKERS** keyword allows you define a lookup table to allow the HCL Workload Automation for Z TSO commands to know the tracker subsystem to which to send events for a combination of Controller and JES Node.

```
          +------<----------<-----+
          V                       |
TRACKERS(>-+-subsys.sysid.tracker,-+-<<)
```

where:

subsys

Name of the controller

sysid

Identifier for the LPAR as designated by **OPTIONS SYSID**.

tracker

Name of the tracker.

If you set **sysid** to an asterisk (*), the default tracker name for the controller is used.

The following example defines the scenario that all trackers for HCL Workload Automation for Z subsystem TWSA are called TRKA, unless they are running on SYS1 or SYSQ in which case they are TRK1 and TRK2, respectively.

```
OPTIONS TRACKERS(TWSA.*.TRKA,TWSA.SYS1.TRK1,TWSA.SYSQ.TRK2)
```

UPDATE – Default value for UPDATE keyword

Sets the default value for any commands that use the **UPDATE** keyword to determine whether to actually perform updates, or just simulate the process.

Y

Perform updates as specified (default).

N

Simulate command without performing updates.

VARNAMES – Special characters to allow in variable names

Workload Automation Programming Language allows variable names to consist of letters, numbers, and the characters: at sign (@), number sign (#) and underscore (_).

Extra characters can be added to this using the **VARNAMES** keyword. The default additional characters are pound sign (£) and dollar sign ($).

VERADGRD – Verify groups exist

Application descriptions that are members of an application group have the name of the group definition in field **ADGROUPID** of segment ADCOM. **VERADGRD** controls the verification of this field when a new application description is created or an existing application is replaced.

The verification is done for active application descriptions.

N

No check is made to verify that the application group exists (default).

F

The group definition is verified to check that it exists, is active, and valid for at least a part of the validity period of the application description being created or updated.

Y

Same as for **F**, except that the application group ID is accepted if the application description already has this application group ID. It could be an update without any change to the application group ID or an insert of a new version when there already are active versions with the same application group ID.

VERSION – HCL Workload Automation for Z version

Specifies the version of HCL Workload Automation for Z with which to communicate. This prevents inappropriate commands from being issued and generates only Batch Loader compatible with the specified version.

In this way, you can generate batch loader commands that can be used with earlier versions of HCL Workload Automation for Z; this is useful, for example, while you are migrating the product and one controller is at a later version than another.

For versions later than 9 and earlier than 16, you can also use the hexadecimal format (for example, VER=A10 for version 10.1).

You can set the **VERSION** keyword to also specify the modification level of the product by using the plus sign (+). For example, to indicate version 10.1 modification level 001 set `OPTIONS VERSION(1010+001)`.

Unless you need to connect or generate output for an earlier version of the product, do not set the **OPTIONS VERSION** keyword to any value. The default is the latest version supported by Workload Automation Programming Language.

VERSRWSN – Verify workstations

The special resource description, SR, has fields representing workstations, the complete workstation names, or generic names; field **SRDWSNAME** of segment **SRDWS** for default connected workstations, field **SRIWSNAME** of segment **SRIWS** for workstations connected to an interval. **VERSRWSN** controls the verification of these fields when a new special resource is created or an existing resource is replaced.

N

No check is made to verify that the workstation description exists (default).

F

The workstation fields are verified against the workstation description file. Each workstation field in the resource description must match at least one of the workstation descriptions.

Y

Same as for **F**, except that the workstation value is accepted if the resource description already has this workstation name. It could be an update without any change to the workstation names.

WWWW – List of days of the week

Specifies the list of day of the week to be used in the **DATEADD** and **DATEFORM** functions. By default, the names are in English but you can use **OPTIONS MMMM** to set the days in a different language.

This keyword must specify a list of 7 complete day names, starting with the equivalent for Monday.

For example, to set the day names in Italian issue the following command:

```
OPTIONS MMMM(LUNEDI MARTEDI MERCOLEDI GIOVEDI VENERDI SABATO DOMENICA)
```

Note: Do not use accented characters, because the **LOWER** function does not support them and this might produce unpredictable results.

XMBLK – Whether to return a message control block

Use this option to specify that when commands are passed to Workload Automation Programming Language through a control block, the log messages are returned in a second control block.

Valid values are:

YES

Returns Workload Automation Programming Language messages in a control block (default).

NO

Does not return Workload Automation Programming Language messages in a control block.

XMSEV – Severity of messages to return in a message control block

Use this option to set the severity of the Workload Automation Programming Language messages that are returned through a control block.

If you set **XMBLK** to **YES**, the Workload Automation Programming Language messages are returned in a control block. Set the level of message severity in the **XMSEV** option by specifying a string of valid suffixes. The default is **WEF**, meaning that Warning, Error, and Fatal messages are returned.

A

Advisory

C

Critical (RC=16)

E

Error (RC=8)

F

Fatal (RC=12)

I

Informational

O

Operational Console

W

Warning

`X`

Excluded (by `SETSEV` command)

Note: Messages are not returned if Workload Automation Programming Language is stopped immediately for a critical error. In this case, messages are stored only in the Workload Automation Programming Language log, regardless of this setting.

Appendix C. Workload Automation Programming Language variables

In addition to the HCL Workload Automation for Z variables and user variables, Workload Automation Programming Language has access to some additional information relating to the job running Workload Automation Programming Language and when scheduled the operation and occurrence within which it is running.

Some variables are available as Workload Automation Programming Language variables and HCL Workload Automation for Z JCL variables, other variables are specific only to Workload Automation Programming Language. In the following sections, the variables unique to Workload Automation Programming Language are highlighted in bold.

Note: Not all the variables provided by HCL Workload Automation for Z are available within Workload Automation Programming Language.

Job level variables

The job level variables provide information about the running job, whether it is scheduled by Workload Automation Programming Language or not.

Note: The variables unique to Workload Automation Programming Language are highlighted in **bold**.

Table 166. Job level variables

Name	Description
OJJESNO	The JES number of the job.
OJJSTEP	The name of the current JOBSTEP.
OJOBNAME	Name of the job running Workload Automation Programming Language.
OJPSTEP	The name of the current PROCSTEP.
OJSWA	Memory type of the initiator running the job: **ABOVE** JOBCLASS defined with WSA=ABOVE. **BELOW** JOBCLASS defined with WSA=BELOW.

Occurrence level variables

The occurrence level variables provide information about the occurrence in which the running job is scheduled. If the job is not being controlled by HCL Workload Automation for Z, these variables are not available.

Note:

1. A job submitted outside of HCL Workload Automation for Z, but tracked by it, is considered to be controlled by HCL Workload Automation for Z.
2. The variables unique to Workload Automation Programming Language are highlighted in **bold**.

Table 167. Occurrence level variables

Name	Description
OADDED	Added to current plan (Y|N).
OADDESC	Application description.
OADDFUNC	Adding function (E|D|P|A|<blank>).
OADGROUP	Application Group name.
OADID	Application ID.
OADODESC	Owner description.
OADOWNER	Occurrence owner.
OAUGROUP	Authority Group.
OCALID	Calendar name,
ODD	Occurrence input arrival day of month, in DD format.
ODD	Occurrence input arrival day of month, in DD format.
ODL	Deadline (*YYMMDDHHMM*).
ODMY1	Occurrence input arrival date in *DDMMYY* format.
ODMY2	Occurrence input arrival date in *DD/MM/YY* format.
OETCRIT	Event triggering policy name from the ETT table. **Note:** This variable can be used only by the ETT added occurrence.
OETEVNM	Complete ETT event name: **Event type J** Contains the same value of OETJOBN. **Event type R** Contains the complete resource name that triggered the event.

Table 167. Occurrence level variables (continued)

Name	Description
	Note: This variable can be used only by the ETT added occurrence.
`OETGGEN`	GDG data set generation number (G*nnnn*V*nn*). For the ETT event type R, generated by a data set triggering for GDG. **Note:** This variable can be used only by the ETT added occurrence.
`OETGROOT`	GDG data set root. For the ETT event type R, generated by a data set triggering for GDG. **Note:** This variable can be used only by the ETT added occurrence.
`OETJNUM`	Job number associated with the OETJOBN. It is set only for the event type J. **Note:** This variable can be used only by the ETT added occurrence.
`OETJOBN`	The complete job name that triggered the ETT event: **Event type J** Contains the job name of the triggering job. **Event type R** Contains the job name or TSO user ID that closed the ETT data set trigger. **Note:** This variable can be used only by the ETT added occurrence.
`OETTYPE`	Event type of the ETT table entry (J=Job, R=Resource). **Note:** This variable can be used only by the ETT added occurrence.
`OGRPDEF`	Application group ID.
`OHH`	Occurrence input arrival hour in *HH* format.
`OHHMM`	Occurrence input arrival hour and minute in *HHMM* format.
`OIA`	Input arrival (*YYMMDDHHMM*).
`OIAA`	Actual Input Arrival (*YYMMDDHHMM*).

Table 167. Occurrence level variables (continued)

Name	Description
OIAO	Original Input Arrival from LTP (*YYMMDDHHMM*).
OJCLVTAB	JCL Variable table attached to occurrence.
OLATE	Latest Out passed (Y|N).
OMM	Occurrence input arrival month in *MM* format.
OMMYY	Occurrence input arrival month and year in *MMYY* format.
OMONITOR	Occurrence monitor flag.
OPRI	Occurrence priority.
OYMD1	Occurrence input arrival date in *YYMMDD* format.
OYMD2	Occurrence input arrival date in *YY/MM/DD* format.
OYY	Occurrence input arrival year in *YY* format.
OYYMM	Occurrence input arrival month within year in *YYMM* format.

Operation level variables

The operation level variables provide information about the operation within the occurrence in which the running job is scheduled.

Note: The variables unique to Workload Automation Programming Language, if any, are highlighted in **bold**.

Table 168. Operation level variables

Name	Description
OCPJOB	Returns **Y** if the job is in the current plan. Returns **N** if the job is running outside of the scope of the current plan.
OCPOPJBN	Job name, as specified in the current plan. In most cases, this is equivalent to the **OJOBNAME**. When you set **OPTIONS OPID**, **OJOBNAME** specifies the name of the running job and **OCPOPJBN** specifies the job name of the connected operation in the current plan.
OCPOPJES	JES number, as specified in the current plan. In most cases, this is equivalent to the **OJJESNO**. When you set **OPTIONS OPID**, **OJJESNO** specifies the number of the running job and **OCPOPJES** specifies the JES number of the connected operation in the current plan.

Table 168. Operation level variables (continued)

Name	Description
OCPUSRF	Returns the number of user fields referenced by the running job in the current plan. For jobs not included in the current plan, this variable returns 0.
OJOBNAME	Operation job name.
OOPNO	Operation number within the occurrence, right-justified and padded with zeros.
OWSID	Workstation ID for current operation.

Current variables

The current variables represent the current date and time.

Note: The variables unique to Workload Automation Programming Language, if any, are highlighted in **bold**.Name

Table 169. Current variables

	Description
CCC	Current century in *YY* format.
CDAY	Current day of the week; 1 corresponds to Monday, 7 corresponds to Sunday.
CDD	Current day of month in *DD* format.
CDDD	Day number in the current year.
CDDMMYY	Current date in *DDMMYY* format.
CHH	Current time in *HH* format.
CHHMM	Current hour and minute in *HHMM* format.
CHHMMSS	Current hour, minute, and second in *HHMMSS* format.
CHHMMSSX	Current hour, minute, second, and hundredths of seconds in *HHMMSSXX* format.
CMIN	Current minute in *MM* format.
CMM	Current month in *MM* format.
CMMYY	Current month in *MM* format.
CSS	Current second in *SS* format.
CYMD	Current date in *YYYYMMDD* format.
CYY	Current year in *YY* format.
CYYDDD	Current Julian date in *YYDDD* format.

Table 169. Current variables (continued)

	Description
`CYYMM`	Current month within year in *YYMM* format.
`CYYMMDD`	Current date in *YYMMDD* format.
`CYYYY`	Current year in *YYYY* format, for example, 1997.
`CYYYYMM`	Current month within year in *YYYYMM* format.
`CXX`	Current hundredths of a second in *XX* format.

Subsystem variables

The subsystem variables provide information about the connected subsystem.

Table 170. Subsystem variables

Name	Description
`ZFMID`	Software FMID for the connected subsystem.
`ZHIDATE`	HCL Workload Automation for Z high date in the format *YYMMDD*.
`ZLODATE`	HCL Workload Automation for Z low date in the format *YYMMDD*.
`ZSUBSYS`	Subsystem name.
`ZVER`	Version in the format *VVRM* (Version, Release, Modification level).
`ZVERSION`	Version *V.R.M* or *V.R* if no Mod (Version, Release, Modification level).

Dynamic variables

The dynamic variables enables quick formatting. They can be used only with the Workload Automation Programming Language and are not shown in **`SHOW VARIABLES`**.

Table 171. Dynamic variables

Name	Description
`BLANK`	Returns a blank space.
`BALNK`*nn*	Returns *nn* blank spaces.

Appendix D. WAPLEXEC programs

The Workload Automation Programming Language for z/OS is a REXX based tool provided to give you easy access to the features of the HCL Workload Automation for Z Program Interface (PIF). Workload Automation Programming Language can be extended by WAPLEXEC programs, which are REXX programs using Workload Automation Programming Language functions to tackle specific tasks or functions.

Workload Automation Programming Language provides you with a set of programs in the SEQQMISC library. The following WAPLEXEC routines are provided with Workload Automation Programming Language, but are no longer supported nor documented:

Table 172. Old and new routines

Old routine	Now replaced by...
EQQWXADD	Workload Automation Programming Language **ADD** command.
EQQWXJBF	Workload Automation Programming Language **LISTJOB** command. Alhough the output format is different, the core function is the same, to find jobs in the database.
EQQWXMOD	Workload Automation Programming Language Current Plan Operation commands. For detailed information, see Current Plan Operation commands on page 185.
EQQWXLNK	Workload Automation Programming Language **ADD** command with **LINK(YES)**.

Running WAPLEXEC programs

WAPLEXEC programs can be run by EQQYXJPX, naming the WAPLEXEC program with the CMD symbolic parameter.

```
//RUNWAPL  EXEC EQQYXJPX,
//         SUBSYS=TWSA,
//           CMD=EQQWXCSR
//OUTBL      DD SYSOUT=*
//INPUTDEF   DD *
MAP(JOB,RC,PRED,FORM,.,WS) ADID(FRED) FROM(DD0001) WS(CPU1)
POLICY(60<26,5<51,3) UPDATE(Y) FIRSTWS(DUMM) LASTWS(DUMM)
DESCR(Excel Application) ODESCR(Scheduling) OWNER(TWS)
//CSVFILE    DD *
JOBA
JOBB,4,JOBA,DD0002,HELLO,WAIT
JOBC,,JOBA JOBQ,,GOODBYE
JOBD,,'JOBB JOBC'
JOBQ
```

For the specific programs, additional DD statements might be required.

Note:

1. WAPLEXEC programs are designed to provide their own `OUTPUT` statements within the code, therefore the EQQFILE DD statement must always point to member `FILENONE` for WAPLEXECs.
2. Input files to WAPLEXEC programs can be catalogued data sets or instream data within the JCL. The command strings can be extremely long, and in the case of Comma Separated Value files much longer than 80 bytes. Care should be taken with WAPLEXEC jobs using instream data to disable “line numbers” (UNNUM or NUMBER OFF in PDF Edit). Because numeric data could form part of the input, the WAPLEXEC code cannot reliably determine whether numbers in columns 73 to 80 were intended as data, or line numbers, so the whole of the input file, regardless of record length, is always treated as input to the process.

EQQWXCSR – Update resources in the Current Plan

Function

The **EQQWXCSR** program allows Special Resources to be updated in the current plan.

You can specify a single resource or use wildcard characters to specify multiple resources. The program searches for all matching resources that are *not* in the desired status and issue **SRSTAT** commands to set them. This means that resources already in the required status do not generate additional events.

If the desired availability and the default availability for a resource are the same, by default the **SRSTAT** will **RESET** the availability to the default. If you specify **RESET(NO)**, the **SRSTAT** sest the override availability to the desired state.

Process control

The program is controlled by keywords in the ARGS symbolic for the EQQYXJPX procedures. Alternatively, you can provide the keywords as SYSIN.

Each keyword has the value specified within parenthesis and separated from the next keyword by a space.

The following keywords are available:

SR

The name of the Special Resource to set (required). You can use wildcard characters.

AVAIL(YES|NO)

The availability you want to set (required).

MATCHTYP

If your resource name includes spaces, the percent sign (%), or an asterisk (*), use MATCHTYP to select the correct resource.

If you do not specify MATCHTYP, all the characters up the first space are considered to be the resource name, and the characters after the space are ignore; any percent sign (%) or asterisk (*) is treated as a wildcard. If you specify MATCHTYP, the spaces, percent signs (%) and asterisks (*) are treated as normal characters and the following matches can be performed:

MATCHTYP(EXA)

An exact match is performed looking for the whole name, exactly as typed.

MATCHTYP(PFX)

Searches for resources that begin with the name specified.

MATCHTYP(SFX)

Searches for resources that end with the name specified.

RESET(YES|NO)

Whether to **RESET** the resource if **AVAIL** matches the default availability (default is **YES**).

SUBSYS

Targets the **SRSTAT** command to a specific tracker. If omitted, Workload Automation Programming Language uses the SUBSYS as determined by **OPTIONS TRACKERS** settings.

UPDATE(YES|NO)

Specify **UPDATE(NO)** to run the command only to see what it would have done, without actually performing any changes. The default is **YES**.

Running the command

The **EQQWXCSR** program uses the EQQYXJPX procedure with no additional DD statements.

The JCL for running the command must specify **EQQWXCSR** as the command.

```
//RUNPIF   EXEC EQQYXJPX,
//         SUBSYS=TWSA,
//         CMD=EQQWXCSR,
//         ARGS='SR(MY.SRTYPE.*) AVAIL(N)'
```

EQQWXCSV – Generate applications from a CSV file

Function

The **EQQWXCSV** program generates a primitive application, or suite of applications, from a comma separated value (CSV) file.

The primitive application has simply the jobs and dependencies defined, with some operations attributes set. They do not include run cycles, special resources, or other features not directly contained in an operation record. You can use **EQQWXCSV** to create the primitive applications, to which you can later add more detailed features by using dialogs or other means.

The CSV file can contain jobs, dependencies, workstations, highest return code, form definition, and duration. The code can easily be extended to include other operation attributes, as required. The minimum content for the CSV file is a column for the names of the jobs to run, and a column for dependencies.

The columns in the CSV file can be mapped to the relevant operation attributes with the **MAP** keyword. This removes any requirement to format the CSV file to meet the requirements of the program.

The program takes the entire content of the CSV file and creates a workflow for all the operations, then breaks this into multiple applications if necessary, based on your operation numbering **POLICY**. If only one application is created, the application name is the same as the name specified in the control statements. If the number of operations causes an application split, the application names are suffixed with numbers. If the specified application name is already 16 characters long, the suffix is overlayed over the last few characters.

Each generated Application has a START and END operation to tie unattached dependencies to.

Dependencies made to jobs defined later in the list will still be made, so it is not necessary to sort the CSV file into estimated execution order. The program highlights dependencies to jobs *not* in the list as external predecessors, which will be defined as being to an application called EXTERNAL_PRED, with the operation number and workstation being the same as used for the END tie up operation. These place holder dependencies will require manual amendment in the resulting primitive application.

Multiple instances of the same job name can be used, any dependencies made to multiple jobs will always be to the most recently specified in the list before the job making the dependency.

If the dependencies would define a loop, all operations in the loop are reported without generating an application.

It is possible to produce Batch Loader output, for review or later implementation on a later system, or the program will call Workload Automation Programming Language directly to update the database with the new applications.

Process control

The INPUTDEF file is used to control the translation of the CSV file into Application Definitions.

The INPUTDEF file contains keywords with the format `KEYWORD(VALUE) KEYWORD(VALUE) KEYWORD(VALUE)`

Each keyword has the value specified within parenthesis and separated from the next keyword by a space.

MAP(field,field,field)

Defines which columns of the CSV file relate to details within an operation.

JOBN

The job name you want to search.

PRED

The predecessors, multiple predecessors must be contained within the same “cell” separated by spaces.

FORM

The form number.

HIGHRC

The Highest Return Code.

WSID

The Workstation.

DURATION

The estimated duration in seconds.

<period>

A period is used as a placeholder for columns that do not contain information pertinent to the creation of the Application.

The following example shows the Job name being in column 2, highest return code in 3, workstation in 4, form number in 5 and duration in 7:

```
MAP(.,JOBN,HIGHRC,WSID,FORM,.,DURATION)
```

POLICY(op<total,op<total,op)

Defines the Operation Interval policy. Each level of the POLICY defines the operation number to use when the total number of operations in the CSVFILE is less than a specified threshold, with a “catch all” interval at the end.

The following example uses an interval of 10 for less than 26 operations, and interval of 5 for less than 51 operations and an interval of 3 for anything higher:

```
POLICY(10<26,5<51,3)
```

`POLICY(5)` simply sets the catch all at 5, meaning that the interval is 5 regardless of the number of operations in the CSVFILE.

SKIP(*n*)

Specifies the number of rows to skip at the beginning of the file before starting processing, to account for header rows in spreadsheets that are used to generate the CSV files. For example, **SKIP(3)** skips 3 rows and start processing with row 4.

UPDATE(Y|N)

Specifies whether the generated Batch Loader is to update the HCL Workload Automation for Z database (Y) or simply output the Batch Loader to OUTBL (N).

ADID(application-name)

Defines the Application name to create. If the number of operations results in more than one Application being generated, the names are suffixed with a numeric count. If the Application name and the numeric suffix would exceed 16 characters, the numeric suffix will overlay the end of the application name.

For example, `ADID(ABCDAILY)` will create ABCDAILY if only one application is needed, or ABCDAILY1, ABCDAILY2 etc if a split occurs.

<Batch-Loader-Token>(value)

Any Batch Loader tokens for ADOP (for details, see *HCL Workload Automation for Z: Managing the Workload*) can be used to set defaults for the mapped columns that have no values. For example, **FORM(DD0001)** creates a default value of DD0001 for any operation that does not have a value in the mapped column for form number.

They can also be used to create lookup tables to translate input values by suffixing the keyword name with a hyphen and a lookup value, with the replacement value being specified within parenthesis. In the following example, you use DD0001 as the form number for any column that contains MY.FIRST.JCLLIB and DD0002 for any column that contains MY.SECOND.JCLLIB:

```
FORM-MY.FIRST.JCLLIB(DD0001)
FORM-MY.SECOND.JCLLIB(DD0002)
```

This technique can be used for any of the mapped columns.

The following keywords perform the same function as their Batch Loader equivalents in the resulting application:

- **DESCR(text)**
- **OWNER(owner_ID)**
- **ODESCR(text)**
- **PRIORITY(n)**
- **ADVALFROM(yymmdd)**
- **STATUS(A|P)**
- **GROUP(authority_group)**
- **CALENDAR(calendar_ID)**
- **ADGROUPID(application_group)**
- **DLIMFDBCK(deadline_limit_for_feedback)**
- **DSMOOTHING(deadline_smoothing)**

The following keywords define the operation number, workstation and job name of the first and last operations generated in the application to tie up dependencies:

- **FIRST(op_num)**
- **FIRSTWS(ws_id)**
- **FIRSTJOB(jobname)**
- **LAST(op_num)**
- **LASTWS(ws_id)**
- **LASTJOB(jobname)**

The default values are as follows:

```
FIRST(1) FIRSTWS(NONR) FIRSTJOB(START)
LAST(255) LASTWS(NONR) LASTJOB(END)
```

Running the command

In addition to normal Workload Automation Programming Language JCL requirements, **EQQWXCSV** needs the following DD statements allocated.

Table 173. DD statements for EQQWXCSV

DD Name	Purpose	Attributes
OUTBL	Capture Batch Loader output when running with **UPDATE(N)**.	Can be an output data set or SYSOUT. Typically FB 80.
INPUTDEF	Contains the control statements that define the behavior of the job.	Input data set or instream SYSIN. Typically FB 80. Line numbers not tolerated. For multiple data set concatenation, use the same LRECL.
CSVFILE	The comma separated value file containing the jobs and dependencies.	Input data set or instream SYSIN. For multiple predecessors allow for a long record length. Line numbers not tolerated.

The JCL for running the command must specify **EQQWXCSV** as the command.

```
//RUNPIF   EXEC EQQYXJPX,
//         SUBSYS=TWSA,
//         CMD=EQQWXCSV
//OUTBL      DD SYSOUT=*
//INPUTDEF   DD *
MAP(JOB,HIGHRC,PRED,FORM,.,WSID)
ADID(FRED) FORM(DD0001) WSID(CPU1)
POLICY(60<26,5<51,3) UPDATE(Y) FIRSTWS(DUMM) LASTWS(DUMM)
DESCR(Excel Application) ODESCR(Scheduling) OWNER(TWS)
//CSVFILE    DD *
JOBA
JOBB,4,JOBA,DD0002,HELLO,WAIT
JOBC,,JOBA JOBQ,,GOODBYE
JOBD,,'JOBB JOBC'
JOBQ
```

EQQWXHTM – Build an HTML version of a calendar

Function

The **EQQWXHTM** program produces an HTML representation of a calendar. The calendar can be a simple calendar for the year with no scheduling information, or it can contain highlighted dates extracted from HCL Workload Automation for Z, such as calendar free days and run dates.

All dates will have the Julian date available as a Tooltip when the mouse is hovered over the date. If the day is normally a free day, the julian day is appended with `-F`. Any date based free days must be set by the **PROCESS** keyword and will contain

their description in the Tooltip, but no `-F`. Any other specific dates can also contain Tooltip text to explain why they have been highlighted.

If multiple events coincide on the same date, for example Free days and named dates, the date is shown in a box and the tooltip will contain text for all events on that date.

Titles, colors, free days, month, and day names can all be manipulated by keywords.

Multiple lists of dates can be loaded into the calendar, each with their own color coding and individual descriptions. Individual dates within the lists can also have specific color coding.

The HTML file produced by this job can then either by transferred to a non-z/OS platform for sending via email, sharing by Windows, Samba, or a web server, or it could be sent directly from a mainframe SMTP server as an email attachment or served by a mainframe based web server.

Process control

You are provided with many keywords to modify the presentation of the calendar, by setting them either within the ARGS symbolic parameter or SYSIN.

The **EQQWXHTM** program operates in one of the following modes:

SINGLE

A single calendar is created with dates set by the **PROCESS** keyword.

MULTI

An IEBUPDATE stream is created to create multiple calendars, one for each application found in the **INAD** file.

All keywords have default values, therefore none of them is required.

CALENDAR(<calendar-name>,[DEFAULT|FORCE])

The **CALENDAR** keyword enables you to specify the calendar to be used to set specific free days.

The second keyword determines how the specified calendar is used when **MODE(MULTI)** is set:

DEFAULT

The calendar is used only if a calendar is not specified in the application.

FORCE

The calendar is used even if a calendar is specified against the application.

If **CALENDAR** is not specified, calendars are not used in the output; only regular **FREEDAYS** are shown.

DAYS(<column-headers>)

The column headers for each day of the week. If you enter less than seven words, the remaining column headers are blank. If you enter more than seven words, the extra words are ignored. The default is **DAYS(MON TUE WED THU FRI SAT SUN)**.

OUTHTNL(<DD-Name>)

The DD name of the output file to be specified. The default is **OUTHTML(OUTHTML)**.

FREEDAYS(<day-list>)

The days of the week that are always free days. The days are specified by number, starting from 1=Monday to 7=Sunday. Any non numeric value is ignored. It is also possible to set a separate color code for Free days by appending html color codes to the list following **==**.

For example, `FREEDAYS(6 7==gray-yellow)` sets Saturday and Sunday as Free days, and sets the background color to gray with the text color yellow.

The default is **FREEDAYS(6 7)**.

Note: In **MODE(MULTI)** day of the week free days are set by this keyword and not extracted from the calendar; only specific date free days are extracted from the calendar.

HEAD(<BG-html-colour>[-<TEXT-html-colour>])

Sets the background color, and optionally the text color, for the header that contains the names of the days. If only one color is specified, it is considered the background color. If two colors are specified, separated by a minus sign (-), the first is the background color and the second is the text color.

The default is **HEAD(blue-white)**.

HILITE(<BG-html-colour>[-<TEXT-html-colour>])

Sets the background color, and optionally the text color, for any highlighted cells whose color is not specified by other means. If only one color is specified, it is considered the background color. If two colors are specified, separated by a minus sign (-), the first is the background color and the second is the text color.

The default is **HILITE(gray-yellow)**.

INAD(<DD-Name>)<only with MODE(MULTI)>

The DD name of the input file that instructs **EQQWXHTM** about the applications for which to generate calendars. The default value is **INAD**.

The format of the file is for each application to process: `application name,calendar or blank,description,`

Any data after the final comma is ignored.

INCL(<DD-Name>)

The DD name of the input file that holds specific free dates from calendars. The default value is **INCL**.

The format of the file is for each specified date in each calendar: `calendar name,date in format YYMMDD,date description,`

Any data after the final comma is ignored. The **INCL** file is read only if **CALENDAR** is specified.

INLTP(<DD-Name>)<only with MODE(MULTI)>

The DD name of the input file that holds run information for applications. The default value is **INTLP**.

The format of the file is for each specified date in each calendar: `IA date in format YYMMDD,AD name,`

Any data after the final comma is ignored.

MODE(SINGLE|MULTI)

Determines whether to create a single HTML calendar or produce one calendar for each application listed in the INAD file.

MONTHS(<month-list>)

The block headers for each month. If you enter less than 12 words, the remaining block headers are blank. If you enter more than 12 words, the extra words are ignored.

The default is **MONTHS(JAN FEB MAR APR MAY JUN JUL AUG SEP OCT NOV DEC)**.

OUTFTP(<DD-Name>) <only with MODE(MULTI)>

The DD name of the FTP output file. This file contains FTP get statements to allow the PDS members to be translated into html files named after the application. If **OUTFTP** is not specified, no FTP statements are generated.

The default value is **OUTFTP(OUTFTP)**.

OUTHTML(<DD-Name>)

The DD name of the HTML output file. This file contains the HTML calendar definitions.

The default value is **OUTHTML(OUTHTML)**.

PROCESS(<file-list>)

Specifies a list of files to be processed to add specific list of dates to the calendar. The files must be fixed block text files with the first word being a date in the format *YYMMDD*, optionally followed by a comma and the text for the tooltip of this date. You can also specify the color properties for the dates, by appending a html color codes to the list following **==**.

For example, the keyword **PROCESS(HOLS,RUNDATES==green-yellow)** first processes a file named HOLS setting the colors to be the values specified by **HILITE**, then it processes a file named RUNDATES setting the foreground color to green and the background color to yellow for the dates in the list. If the same date is referred to in multiple lists, the tooltip contains the text from each list in the order they were processed, the color coding is set to the last reference to that date.

Individual dates within a file can be given their own color coding by appending html color codes to the record in the file following **==**.

The following example shows how to code the 25th and 26th of December with specific colors:

```
//HOLS     DD *
120406,Goede Vrijdag
```

```
120409,2E Passdag
120430,Koninginnedag
120517,Hemelvaart
120728,2E Pinksterdag
121225,1E Kerstdag==green-red
121226,2E Kerstdag==green-red
```

The default is blank, meaning that no files are processed.

ROW1(<BG-html-colour>) ROW2(<BG-html-colour>)

The **ROW1** and **ROW2** keywords set the background for each alternate row of each month, to make the calendar easier to read.

The default is **ROW1(#FFFFCC) ROW2(#99CCCC)**.

RUN(<BG-html-colour>[-<TEXT-html-colour>]) <only with MODE(MULTI)>

Sets the background and optionally the text color for the run dates of applications in **MODE(MULTI)**. If only one color is specified, it is considered the background color. If two colors are specified, separated by a minus sign (-), the first is the background color and the second is the text color.

The default is **RUN(green-white)**.

TEXT(<TEXT-html-colour>)

Sets the default text color for dates within the calendar. This value can be overridden for highlighted dates with the **HILITE** keyword, and for individual processed files and dates using the **==** tagging method.

The default is **TEXT(black)**.

TITLE(<calendar-title>) <only with MODE(SINGLE) >

The title of the calendar. It is displayed at the head of the page, and in the browser title bar. The default is the value set by the **YEAR** keyword.

When in **LTP** mode, the **TITLE** is the name of the application.

TRANSFER(get|put)

Determines the direction to which the FTP statements transfer the members:

get

Pulls the members from the mainframe to a remote server. The FTP job is run on the remote server.

put

Pushes the members from the mainframe to a remote server. The FTP job is run on the mainframe.

YEAR(<year>)

The year for which to produce the report. The value must be specified in the form ***YYYY***.

The default is the current year.

Note:

1. Many of these keywords enable you to specify html colors. You can use both color names (such as gray, blue, red) and RGB color codes (such as #99CCCC). Because the different browsers support a vast range of colors, the contents of the color keywords are not validated. Ensure the you set colors supported by the browser that you will be using to view the calendar. Any colors not supported by the browser are ignored when the calendar is browsed. Many browsers are case sensitive, therefore the SYSIN for **EQQWXHTM** must be in mixed case.
2. In most keywords, you can separate lists with a space or comma.

Running the command

In addition to normal Workload Automation Programming Language JCL requirements, **EQQWXHTM** needs the following DD statements allocated.

Table 174. DD statements for EQQWXHTM

DD Name	Purpose	Attributes
<output>	The information output from the command is written to the file specified in the **FILE** keyword. The default is OUTHTML.	Can be an output data set or SYSOUT. Typically FB 80. **Note:** The maximum record length is the length of the longest individual piece of text in an input file, plus 7 bytes to account for the appending of **. **
<input>	Input files are specified in the **PROCESS** keyword. HTML can process more than one input file in a single run. Each record must begin with a date and can optionally contain explanatory text for that date in the following format: `YYMMDD,explanatory text` Individual dates can be tagged with specific colors, for example `121225,Christmas Day==green-red`	Can be an input data set or SYSIN. Typically FB 80. **Note:** 1. The length of explanatory text has a bearing on the record length for the output file (see above). 2. You cannot use quotes in the explanatory text.

The JCL for running the command must specify **EQQWXHTM** as the command in the CMD symbolic parameter.

The following example shows how to produce a calendar with regular free days, date based free days, and run dates. Though shown as SYSIN in this example, they would more likely be output files from other Workload Automation Programming Language jobs (HTMLXMPL).

```
//RUNWAPL  EXEC EQQYXJPX,
//         CMD=EQQWXHTM,
//         SUBSYS=WSJC
//OUTHTML  DD DISP=SHR,DSN=MYUSER.CAL.HTML
//SYSIN    DD *
TITLE(Calendar for 2012) DAYS(M D W D V Z Z)
MONTHS(JANUARI FEBRUARI MAART APRIL MEI JUNI
       JULI AUGUSTUS SEPTEMBER OKTOBER NOVEMBER DECEMBER)
PROCESS(HOLS,RUNDATES==green-white)
HILITE(gray-yellow)
//HOLS     DD *
120406,Goede Vrijdag
120409,Tweede Passdag
120430,Koninginnedag
120517,Hemelvaart
120728,Tweede Pinksterdag
121225,Eerste Kerstdag==green-red
121226,Tweede Kerstdag==green-red
//RUNDATES DD *
120124,Run 1
120226,Run 2
120331,Run 3
120501,Run 4
120728,Run 5
121006,Run 6
```

Combining EQQWXHTM with other processes

EQQWXHTM provides you with a presentation layer to review date based information in a simple and meaningful way. When combined with other Workload Automation Programming Language elements, you can automate parts of your date data management.

Annual calendar creation and review

The SEQQWAPL library has DATE members containing public holidays for some countries; you can use them to create variables containing dates for public holidays. For countries without DATE members, use VARDATE to create your own rules for public holidays.

The following example shows a sample job for creating public holidays in the calendar for the forthcoming year (CALNL).

```
//RUNWAPL  EXEC EQQYXJPX,
//         SUBSYS=WSJC
//SYSIN    DD *
OPTIONS DBMODE(UPDATE)
VARDATE = YEAR(+1)
INCLUDE EQQFILE(DATENL)
CLSTART CALENDAR(CALNL) DESCR(NEDERLANDSE KALENDER)
CLDAY DAY(MONDAY) STATUS(W) DESCR(MAANDAG)
CLDAY DAY(TUESDAY) STATUS(W) DESCR(DINSDAG)
CLDAY DAY(WEDNESDAY) STATUS(W) DESCR(WOENSDAG)
CLDAY DAY(THURSDAY) STATUS(W) DESCR(DONDERDAG)
CLDAY DAY(FRIDAY) STATUS(W) DESCR(VRIJDAG)
CLDAY DAY(SATURDAY) STATUS(F) DESCR(ZATERDAG)
CLDAY DAY(SUNDAY) STATUS(F) DESCR(ZONDAG)
CLDATE DATE(!FD_GOEDE_VRIJDAG) STATUS(F) DESCR(GOEDE VRIJDAG)
CLDATE DATE(!FD_TWEEDE_PAASDAG) STATUS(F) DESCR(TWEEDE PAASDAG)
CLDATE DATE(!FD_KONINGINNEDAG) STATUS(F) DESCR(KONINGINNEDAG)
CLDATE DATE(!FD_HEMELVAARTSDAG) STATUS(F) DESCR(HEMELVAARTSDAG)
CLDATE DATE(!FD_TWEEDE_PINKSTERDAG) STATUS(F) DESCR(TWEEDE PINKSTERDAG)
CLDATE DATE(!FD_EERSTE_KERSTDAG) STATUS(F) DESCR(EERSTE KERSTDAG)
CLDATE DATE(!FD_TWEEDE_KERSTDAG) STATUS(F) DESCR(TWEEDE KERSTDAG)
```

You can then export the calendar dates into a flat file by running a job similar to the following:

```
//RUNWAPL  EXEC EQQYXJPX,
//         SUBSYS=WSJC
//OUTDATA  DD DISP=SHR,DSN=ADCDMST.CZ.JCL(CALDATES)
//SYSIN    DD *
OPTIONS FIELDSEP(,)
OUTPUT CLSD DATA(OUTDATA) FIELDS(CLSDDATE,CLSDDESC) LABEL(NONE)
SELECT CL CALENDAR(CALNL) SELECT(Y)
```

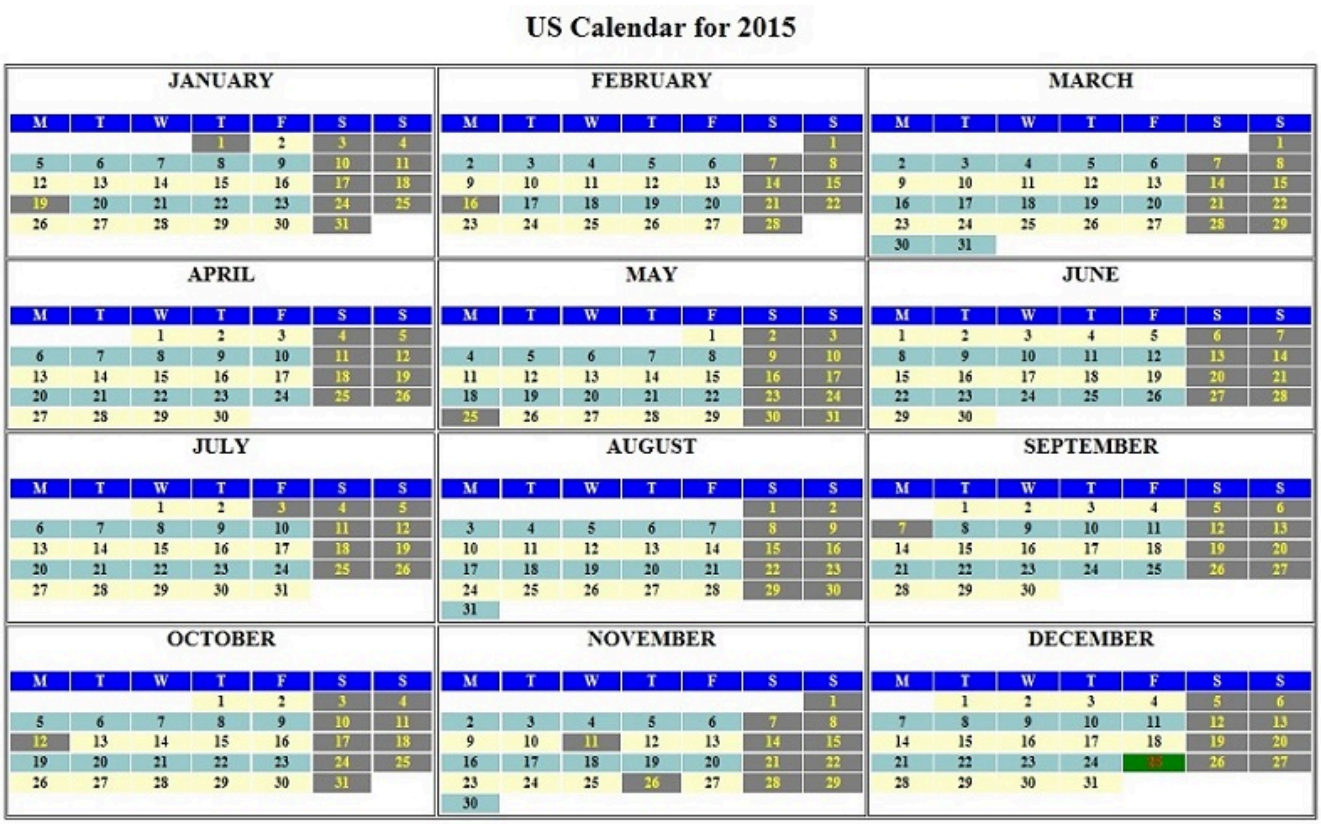
The flat file will contain all the dates contained within the calendar, for as many years as the calendar covers. You can use it with the EQQWXHTM WAPLEXEC to extract specific years and produce the HTML versions. Because the input file contains dates for 2 years, you can run 2 steps which can each concentrate on an individual year by use of the **YEAR** keyword.

The following example shows a sample job for exporting calendar dates into a flat file (CALUNLD).

```
//EXP2012  EXEC EQQYXJPX,
//         ARGS='YEAR(2012)',
//         CMD=EQQWXHTM,
//         SUBSYS=WSJC
//OUTHTML  DD DISP=SHR,DSN=ADCDMST.CZ.JCL(CAL2012)
//CLSDDATE DD DISP=SHR,DSN=ADCDMST.CZ.JCL(CALDATES)
//SYSIN    DD *
DAYS(M D W D V Z Z)
MONTHS(JANUARI FEBRUARI MAART APRIL MEI JUNI
       JULI AUGUSTUS SEPTEMBER OKTOBER NOVEMBER DECEMBER)
PROCESS(CLSDDATE) HILITE(gray-yellow)
/*
//EXP2013  EXEC EQQYXJPX,
//         ARGS='YEAR(2013)',
//         CMD=EQQWXHTM,
//         SUBSYS=WSJC
//OUTHTML  DD DISP=SHR,DSN=ADCDMST.CZ.JCL(CAL2013)
//CLSDDATE DD DISP=SHR,DSN=ADCDMST.CZ.JCL(CALDATES)
//SYSIN    DD *
DAYS(M D W D V Z Z)
MONTHS(JANUARI FEBRUARI MAART APRIL MEI JUNI
       JULI AUGUSTUS SEPTEMBER OKTOBER NOVEMBER DECEMBER)
PROCESS(CLSDDATE) HILITE(gray-yellow)
```

You can then send the output file to review the dates for the forthcoming year, or simply use it as a reference to view the free days.

Figure 2. US calendar for 2015



Run date review

Use the LIST GENDAYS PIF request to generate run dates from rules. You can the use the GETDATES command to find the run dates of any individual application.

The following example shows a sample job for exporting run dates into flat files (RUNDATES):

```
//RUNWAPL  EXEC EQQYXJPX,
//SUBSYS=WSJC
//OPREVAL  DD DISP=SHR,DSN=ADCDMST.CZ.JCL(OPREVAL)
//OPREVAL  DD DISP=SHR,DSN=ADCDMST.CZ.JCL(OPREVAL)
//OMONTH   DD DISP=SHR,DSN=ADCDMST.CZ.JCL(OMONTH)
//OAPPLY   DD DISP=SHR,DSN=ADCDMST.CZ.JCL(OAPPLY)
//SYSIN    DD *
OPTIONS FIELDSEP(,)
OUTPUT GNDAY DATA(OPREVAL) FIELDS(GNDAYDATE,TAG) LABEL(NONE)
GETDATES ADID(FD2PREVAL) FROMDATE(120101) TODATE(121231) OUTPUT(C)
OUTPUT GNDAY DATA(OMONTH)
GETDATES ADID(FD2MONTH)  FROMDATE(120101) TODATE(121231) OUTPUT(C)
OUTPUT GNDAY DATA(OAPPLY)
GETDATES ADID(FD2APPLY)  FROMDATE(120101) TODATE(121231) OUTPUT(C)
```

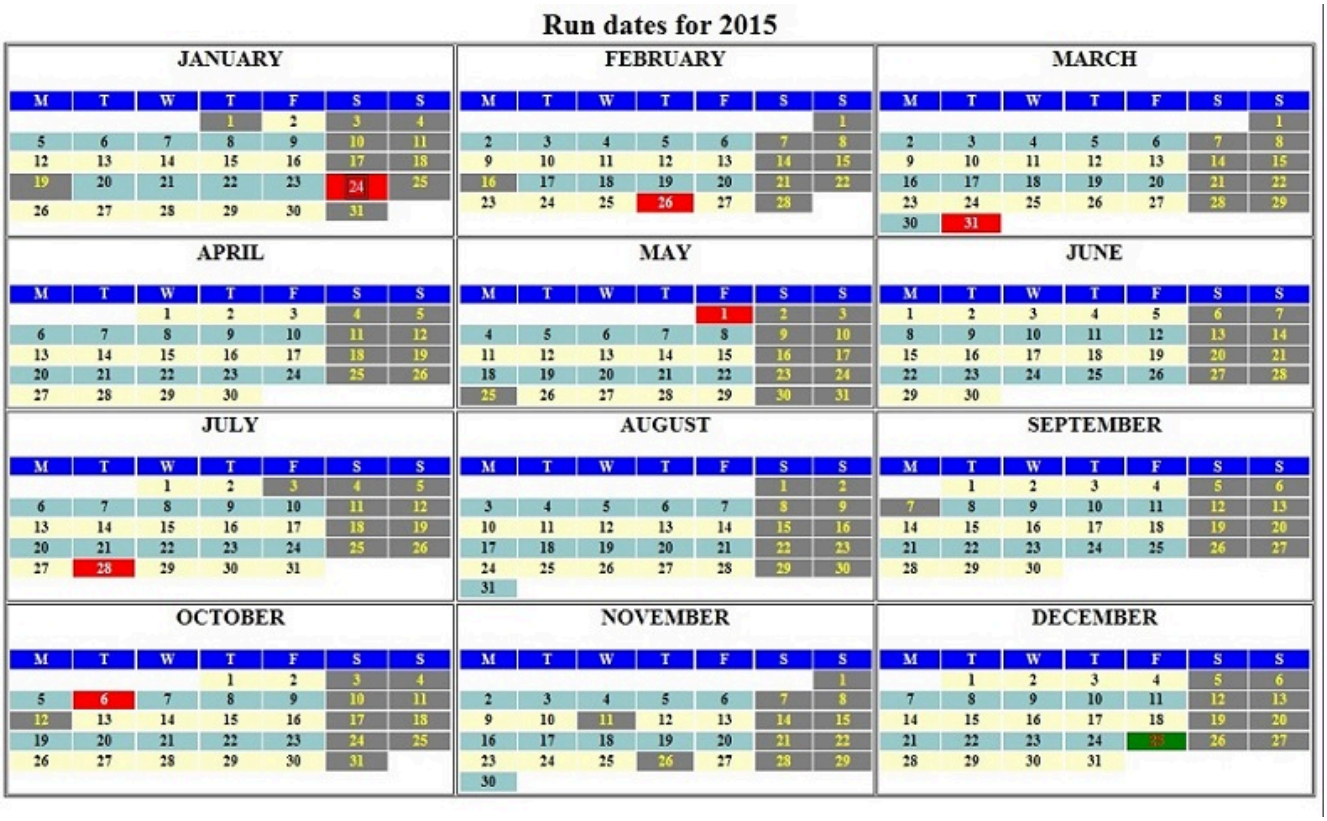
The RUNDATES job finds the run dates for three separate applications across the year. You can the use these files as input in the **EQQWXHTM** command to combine the run dates with calendar information.

The following example shows a sample job for presenting run dates in a calendar (RUNCAL):

```
//RUN2012  EXEC EQQYXJPX,
//         ARGS='YEAR(2012)',
//         CMD=EQQWXHTM,
//         SUBSYS=WSJC
//OUTHTML  DD DISP=SHR,DSN=ADCDMST.CZ.JCL(RUN2012)
//CLSDDATE DD DISP=SHR,DSN=ADCDMST.CZ.JCL(CALDATES)
//OPREVAL  DD DISP=SHR,DSN=ADCDMST.CZ.JCL(OPREVAL)
//OMONTH   DD DISP=SHR,DSN=ADCDMST.CZ.JCL(OMONTH)
//OAPPLY   DD DISP=SHR,DSN=ADCDMST.CZ.JCL(OAPPLY)
//SYSIN    DD *
DAYS(M D W D V Z Z) TITLE(FD2 data voor 2012)
MONTHS(JANUARI FEBRUARI MAART APRIL MEI JUNI
       JULI AUGUSTUS SEPTEMBER OKTOBER NOVEMBER DECEMBER)
PROCESS(CLSDDATE,OPREVAL==green,OMONTH==blue,OAPPLY==red)
HILITE(gray-yellow)
```

Because each application was exported into a separate file, you can assign a different color to each application, making it easier to spot each application that runs on specific dates in a single calendar.

Figure 3. Run dates for year 2015



EQQWXIAX – Input Arrival Cross Reference

Function

The **EQQWXIAX** program builds a list of external dependencies and shows the input arrival times that are most likely to be used to resolve the dependency.

It is not definitive, because in some instances there could be multiple run cycles involved, therefore time could vary depending on day. In these cases, the first run cycle is used to source the input arrival time.

The output looks like the following example:

```
------------ P R E D E C E S S O R -----------    ------------- S U C C E S S O R --------------
 ADID             WSID OP# JOBNAME  DY TIME SRC    ADID             WSID OP# JOBNAME  DY TIME SRC
 ================ ==== === ======== == ==== ===    ================ ==== === ======== == ==== ===
 BANTEST001       CPU1 005 BANTEST6 00 ---- UKN -> BANTEST002       NONR 003 BANTEST6 00 ---- UKN
 DEANO2           ---- 255 -------- -- ---- NTF -> DEANOTEST        NONR 001 FINDMEJU 00 ---- UKN
 WHEREAREYOU      ---- 001 -------- -- ---- NTF -> DEANOTEST        NONR 001 FINDMEJU 00 ---- UKN
 DEANO1           MAN1 001 FIRST    00 ---- UKN -> DEANO2           MAN1 001 FIRST    00 ---- UKN
 HIPPODAILY       NONR 010 HIPAA    00 1800 APL -> HIPPOMTHLY       NONR 001          00 1800 APL
 HIPPODAILY       NONR 010 HIPAA    00 1800 APL -> HIPPOWKLY        NONR 010 HIPAQ    00 1800 APL
 BANK#DAILY#001   NONR 255 END      00 1800 APL -> BANK#DAILY#001   NONR 001 START    00 1800 APL
 GNU#DAILY#001    CPU1 020 GNUD104  00 1800 APL -> BANK#DAILY#001   CPU1 025 TESTJOB5 00 1800 APL
 TEST#EXTDEP1     NONR 001 FIRST    00 ---- UKN -> TEST#EXTDEP2     NONR 001 FIRST    00 ---- UKN
 TEST#EXTDEP2     NONR 001 FIRST    00 ---- UKN -> TEST#EXTDEP3     NONR 001 FIRST    00 ---- UKN
 TEST#GRPMEM1     NONR 001          00 ---- UKN -> TEST#GRPMEM2     NONR 001          00 ---- UKN
 TEST#GRPMEM2     NONR 001          00 ---- UKN -> TEST#GRPMEM3     NONR 001          00 ---- UKN
 TEST#LOOKAHEAD1  CPU1 010 JOBLOOK1 00 0200 APL -> TEST#LOOKAHEAD1  NONR 001 START    00 0200 APL
```

Details about predecessor details are on the left, details about the successors are on the right. Each side has the following columns:

ADID

Application ID

WSID

Workstation name

OP#

Operation number

JOBNAME

Job name

DY

Input Arrival Day

TIME

Input Arrival Time

SRC

Where the Input Arrival time came from.

SRC can have one of the following values:

OPR

The input arrival was specified on the operation.

APL

The input arrival was specified on run cycles in the Application Definition.

GRP

The input arrival was specified on run cycles in the Group Definition.

UKN

No input arrival was specified at any level.

NTF

The predecessor was not found in the database.

Process control

The **EQQWXIAX** program is controlled by keywords in the SCOPE DD statement.

The keywords are filter keywords to select the Applications to be used to form the cross reference. The program is intended only for a small scope of cross referencing and not intended for large full database cross referencing.

Each keyword has the value specified within parenthesis and is separated from the next keyword by a space.

The following keywords are available:

ADID

The name of the Application.

GROUP

Authority Group Name

GROUPDEF

Application Group name

MONITOR(Y|N)

Application has monitored operations

OWNER

Owner ID

STATUS(A|P)

Status:

A

Active (this is the default).

P

Pending

VALID(*yymmdd*)

Date on which the application must be valid.

VALFROM(*yymmdd*)

Valid from date

VALTO(*yymmdd*)

Valid to date

Note: These keywords are the standard keywords for **LIST ADCOM**. You can use wildcard characters and comparators. However, **TYPE** must not be used, because the program needs to separately extract applications and groups.

Running the command

The **EQQWXIAX** program uses the EQQYXJPX procedure with one additional DD statement. The SCOPE DD statement is used to define the scope of the cross reference.

The JCL for running the command must specify **EQQWXIAX** as the command.

```
//RUNPIF   EXEC EQQYXJPX,
//         SUBSYS=TWSA,
//         CMD=EQQWXIAX
//SCOPE    DD *
OWNER(DEANO) VALID(=) STATUS(A)
```

EQQWXJBU – Update applications for a job

Function

The **EQQWXJBU** program finds all the applications where the specified job is defined, and performs the updates requested for the job with the Applications.

The updates can be anything performed by the Batch Loader keywords for the **ADOP** statement.

Because the input could be a simple minor update to a single job, or many fields to many jobs, **EQQWXJBU** works in one of the following modes:

ARG Mode

The ARGS symbolic is used to pass a single job to update and a few arguments to perform updates.

MAP Mode

The ARGS symbolic is used to pass a mapping of columns in a CSV file and the CSV file contains the information to perform the updates for one or more jobs.

Note: HCL Workload Automation for Z is not designed with Job Name as a key field for the database, the **EQQWXJBU** function may help alleviate that, but the process may be slow depending on the size of the database being searched.

Process control

@JOBN(*jobname*)

The job you want to update. You can then use the **JOBN** keyword to change it.

This is an ARG mode keyword.

@WSID(*workstation*)

Specifies the workstation where the job you want to update must be defined. You can then use the **WSID** keyword to change it.

This is an ARG mode keyword.

@ADSTAT(A|P)

Restricts the search to the applications with status **A** or **P**. Though the **@ADSTAT** keyword is specified only in the argument, it also applies to all jobs included for searching within the CSVFILE.

@VALID(*yymmdd*)

Restricts the search to the applications that are valid on the specified date. Though the **@VALID** keyword is specified only in the argument, it also applies to all jobs included for searching within the CSVFILE.

@VALFROM(*yymmdd*)

Restricts the search to the applications by the Valid-From date. Though the **@VALFROM** keyword is specified only in the argument, it also applies to all jobs included for searching within the CSVFILE.

@VALTO(*yymmdd*)

Restricts the search to the applications by the Valid-To date. Though the **@VALTO** keyword is specified only in the argument, it also applies to all jobs included for searching within the CSVFILE.

MAP(*field,field,field*)

Defines which columns of the CSV file relate to details within an operation. The **MAP** can be either the number of the row in the CSV file that contains the field name for each column in the relevant column, or a sequence of comma separated field names.

You can specify the following field names:

@JOBN

The field that specifies the job name to identify the operation to update. There must be a column mapped to **@JOBN**. You can use **JOBN** without the at sign (@) to map to a new job name to update the selected row to.

@WSID

The field that specifies the workstation where the job to be updated must already exist (optional). You can use **WSID** without the at sign (@) to map to a new workstation name to update the selected row to.

<loader argument>

You can use any valid Batch Loader argument for the **ADOP** statement as a mapping field, except **PREOPNO**, **PREJOBN**, and **PREWSID**. For more details about **ADOP**, see *HCL Workload Automation for Z: Managing the Workload*.

<period>

A period (.) is used as a placeholder for columns that do not contain information pertinent to the creation of the Application.

For example, specify **MAP(.,@JOBN,HIGHRC,WSID,FORM,.,DURATION)** to have the Job name in column 2, highest return code in column 3, workstation in column 4, form number in column 5, and duration in column 7.

Specify **MAP(1)** to point to row 1 of the CSV file; each cell in this row will contain the name of the field represented by that column.

This is a MAP mode keyword.

SKIP(*n*)

Specifies the number of rows to skip at the beginning of the file before starting processing, to account for header rows in spreadsheets that are used to generate the CSV files. For example, **SKIP(3)** skips 3 rows and starts processing with row 4.

If you are using a **MAP** row within the CSV file, you must use the **SKIP** keyword to avoid that row.

This is a MAP mode keyword.

PRED(*job job*)

Contains list of job names, separated by a space, to make as a predecessor to the job. If the job is found inside the same application as the **@JOBN** job, only an internal predecessor is made, regardless of whether the job is found in more applications.

If the predecessor needs to be restricted to a job on a particular workstation, use the format job/workstation to specify each predecessor. The Status and Validity of the application are restricted to the same as the job to which the predecessor is being added.

For example, **PRED(ABC123 XYZ789)** makes any instance of these jobs predecessors.

PRED(DEF456/C* GHI321) makes only instances of job DEF456 on a workstation beginning with C a predecessor, but all instances of job GHI321.

Note: When specified in the ARGS symbolic, this is an ARG mode keyword but the same keyword can be mapped as a column to apply to individual CSV file records.

UPDATE(Y|N|SCAN)

Determines whether **EQQWXJBU** applies the updates automatically:

YES

The updates are applied to the database.

NO

The Batch Loader is written to the OUTBL DD statement and not processed (default).

SCAN

The Batch Loader is run against HCL Workload Automation for Z in SCAN mode to validate the syntax, but performs no updates.

This keyword works for both MAP and ARG mode.

<loader argument>

Any valid Batch Loader argument for the **ADOP** statement can be used as a updating field, except **PREOPNO**, **PREJOBN**, and **PREWSID**. For more details about **ADOP**, see *Managing the Workload*.

These are ARG mode keywords.

Running the command

In addition to normal Workload Automation Programming Language JCL requirements, **EQQWXJBU** needs the following DD statements allocated.

Table 175. DD statements for EQQWXJBU

DD Name	Purpose	Attributes
CSVFILE	The comma separated value file containing the jobs and dependencies. The CSVFILE is read only if **MAP** has been specified.	Input data set or instream SYSIN. For multiple predecessors allow for a long record length. Line numbers are not allowed.
OUTBL	The Batch Loader generated by this command to perform the updates is written to OUTBL if **UPDATE(N)** is set.	Can be an output data set or SYSOUT. Typically FB 80

The JCL for running the command must specify **EQQWXJBU** as the command and pass the arguments in the **ARGS** symbolic parameter.

```
//RUNPIF   EXEC EQQYXJPX,
//         SUBSYS=TWSA,
//         CMD=EQQWXJBU,
//         ARGS='JOB(ZAPPED) HEADER(Y) INFILE(MOREJOBS) DATA(*)'
//OUTDATA  DD SYSOUT=*
//MOREJOBS DD *
FRED*
TWSC*
```

EQQWXNOE – Protecting against unconnected applications

Function

When the current plan is extended, an application might start immediately if the external predecessors did not resolve correctly and there are no time dependent or manual operations within the application. The **EQQWXNOE** function provides you with a safety net to help avoid some of these sort of "escapee" applications running when they were not expected.

The premise of the function is that you run the job immediately before the current plan extend job, telling it how long an extension is about to be made to the plan; the **EQQWXNOE** function will identify any occurrences that could run immediately when the current plan is extended. In this case, **EQQWXNOE** fails with RC=8 causing the current plan extend to be delayed until it can be investigated and corrected.

Note: Even with this check in place, some operations might start early, that are not “behind” the external predecessor, time dependency or manual operation, but at least part of the application will be held by at least one of these elements. **EQQWXNOE** will identify only occurrences that have no delaying factors at all.

Process control

The **ARGS** symbolic is used to control the **EQQWXNOE** function.

You can use the following keywords:

EXTEND(*hhmm*|*yymmddhhmm*)

Determines until when the new plan is extended. The value can be either hours and minutes to extend by, or a date and time to extend to.

WS(*wsID*,[*wsID*],[...])

Identifies the workstations to be considered as manual workstations. You can specify one or more workstations, separated by commas.

Running the command

In addition to normal Workload Automation Programming Language JCL requirements, **EQQWXNOE** needs no additional DD statements allocated.

The JCL for running the command must specify **EQQWXNOE** as the command.

In the following example, the plan is being extended 24 hours and MAN1 is a manual workstation:

```
//RUNPIF   EXEC EQQYXJPX,
//         SUBSYS=TWSA,
//         CMD=EQQWXNOE,
//         ARGS='EXTEND(02400) WS(MAN1)'
```

EQQWXPER – Generate week number variables for a period

Function

The **EQQWXPER** program generates a series of dependent variables to enable JCL to determine the week number within an interval of an HCL Workload Automation for Z non-cyclic period.

The process reads the specified HCL Workload Automation for Z period and generates dependent variables for each week within each interval. Whatever day an interval starts on, it is day 1 of the week and each new week starts in a further 7 days.

Each week variable defaults to N, and then only on specified dates it is set to Y. The dates can be made dependent on any of the following HCL Workload Automation for Z variables:

- ODMY1
- ODMY2
- OYMD1
- OYMD2
- OYYDDD
- OLYMD
- OLYYDDD
- CDDMMYY
- CYYDDD
- CYYMMDD

Process control

The **EQQWXPER** program is controlled by keywords either in the ARGS symbolic for the EQQYXJPX procedure, or within SYSIN. If the same keywords are entered in both, the ARGS values overrides the SYSIN value.

Each keyword has the value specified within parenthesis and separated from the next keyword by a space.

You can use the following keywords:

DATEVAR

The name of the HCL Workload Automation for Z supplied date variable that provides the *input* date to set the appropriate week number variable. You can use one of the following variable names:

- ODMY1
- ODMY2
- OYMD1
- OYMD2
- OYYDDD
- OLYMD
- OLYYDDD
- CDDMMYY
- CYYDDD
- CYYMMDD

MAXWEEK

The highest week variable to guarantee the existence of. The default value is 6.

MONTH

Whether to create unique tables for each month (**YES** or **NO**). The default value is **YES**, meaning that the table names are derived from the **TABLE** keyword suffixed with the year and month (for example,`MONTHVARS0912` for December 2009).

OWNER

The **OWNER** to set for the table. This value is needed to create new table instances, because **OWNER** is a required value for an HCL Workload Automation for Z JCL Variable table.

PERIOD

The name of the period for which to calculate week numbers.

TABLE

The name of the table to be created or updated. If **MONTH** is set to **NO**, the **TABLE** will contain the absolute table name; if it is set to **YES** it will be the prefix of the tables created for each month.

UPDATE

Whether to perform updates to the database (**YES** or **NO**). If set to **NO** (this is the default), the process generates the Batch Loader for you to review, without applying it.

VARPFX

The prefix to use for the variables created by this process. The default is **WKNUM**.

For each week being generated, the variable is suffixed by the two character week number, for example `WKNUM01`, `WKNUM02`, and so on.

Running the command

The **EQQWXPER** program uses the EQQYXJPX procedure with no additional DD statements.

The JCL for running the command must specify **EQQWXPER** as the command.

The following example shows how to create tables with the naming convention of MONTHVARS*yymm* containing variables WEEK#01, WEEK#02, WEEK#03, WEEK#04, WEEK#05, and WEEK#06. These week variables are dependent on the occurrence julian date.

```
//RUNPIF   EXEC EQQYXJPX,
//         CMD=EQQWXPER,
//         SUBSYS=TWSA
//OUTBL    DD SYSOUT=*
//SYSIN    DD *
PERIOD(CYCLE) VARPFX(WEEK#) TABLE(MONTHVARS) DATEVAR(OYYDDD)
OWNER(TWS) UPDATE(Y)
```

You can then process it with JCL similar to the following example:

```
//*%OPC SCAN
//*%OPC TABLE NAME=MONTHVARS&OYYMM
//* JULIAN DATE = &OYYDDD
//*%OPC BEGIN ACTION=INCLUDE,COMP=(&WEEK#06..EQ.Y)
//* THIS LINE OF JCL ONLY APPEARS ON WEEK 6
//*%OPC END ACTION=INCLUDE
```

Appendix E. Messages and Return Codes

Messages and return codes provide information for problems occurred.

Workload Automation Programming Language messages have a variety of severities, each with corresponding return codes. Messages with severities A, O, W, and E can be suppressed by using `OPTIONS SEVERITY`. You can also use `OPTIONS MSGLEVEL` to determine the number of messages issued. Use the `SETSEV` command to modify the severity of specific messages to allow individual errors to be considered acceptable for individual running of Workload Automation Programming Language.

Value	Code	Explanation
0	I	Informational messages that cannot be excluded by the SEVERITY option. These messages are self-explanatory and are not documented.
0	A	Advisory messages that can be excluded by the SEVERITY option.
0	O	Advisory messages that may also be routed to the Operator Console by use of the OPMSG option. These can be excluded by the SEVERITY option.
4	W	Warning messages that can be excluded by the SEVERITY option.
8	E	Error messages that can be excluded by the SEVERITY option.
12	F	Fatal messages that cannot be excluded by the SEVERITY option.
16	C	Critical initialization processing error.

With the exception of a few Critical messages, all message text is provided in an external file, therefore message text and severity can be customized to meet the specific requirements of your site. However, you will have to repeat this customization when you upgrade Workload Automation Programming Language.

Message Grouping

Within the Workload Automation Programming Language message log (SYSTSPRT), messages are grouped by the command to which they belong.

A message group always starts with an EQQI200I message and ends with an EQQI299I message. The 200I listing the command and the 299I indicating the return code.

By default. not all commands appear on the Workload Automation Programming Language message log. Only commands directly entered into Workload Automation Programming Language, and commands that have failed will appear. The behaviour can be changed by use of `OPTIONS MSGLEVEL`.

Many commands generate a series of internal commands to perform the required process, for example Batch Loader statements may generate `LIST`, `SELECT`, `INSERT` and `REPLACE` command. Internal commands are listed ahead of the command that generated it, as commands are written to the message log only when they complete, since one of the deciding factors as to whether a message gets written or not is whether it completed successfully or not.

LIST commands generated as a result of **OPTIONS EXPAND(YES)** and **SELECT** commands generated as a result of **OPTIONS SELECT(YES)** are shown after the command that generated them, as the success of the following **LIST** or **SELECT** does not reflect upon the success of the process that generated them.

The format of a message line is **mm/dd hh.mm.ss ppppnnns text**, where:

mm/dd

Date in the format Month/Day

hh.mm.ss

Time in the format Hours:Minutes:Seconds

pppp

Message prefix

nnn

Message number

s

Message severity

text

Message text

Messages automatically wrap to the width of the setting for OPTIONS REPORT (default 80). Continuation lines do not show the date and time, and the message ID is prefixed with . . .

The following message shows sample continued messages:

```
07/08 14.34.20 EQQI200I OUTPUT ADDEP DATA(*) LABEL(NOFIELD) FIELDS(ADCOM.ADID
            ...EQQI200I ADOP.ADOPNO ADDEPADID ADDEPOPNO)
07/08 14.34.20 EQQI299I Statement completed - RC=0
07/08 14.34.20 EQQI200I LIST ADCOM TYPE(G) ADID(*) STATUS(A) VALID(=)
07/08 14.34.21 EQQI299I Statement completed - RC=0
```

Batch Loader requests are listed in a different way. The primary command (for example, **ADSTART**, **ETTSTART**) is listed on EQQI200I message, but any sub segment commands are listed on EQQI203I messages. For example:

```
07/08 15.05.41 EQQI200I ADSTART ADID(FRED) OWNER(FREDDY)
            ...EQQI203I ADOP OPNO(1) WSID(DUMM) JOBN(START) DURATION(1)
            ...EQQI203I DESCR('First operation')
            ...EQQI203I ADOP OPNO(005) WSID(CPU1) JOBN(JOBA) DURATION(1)
            ...EQQI203I FORM(DD0001)
            ...EQQI203I ADDEP PREOPNO(1) PREJOBN(START) PREWSID(DUMM)
            ...EQQI203I ADOP OPNO(010) WSID(CPU1) JOBN(JOBQ) DURATION(1)
            ...EQQI203I FORM(DD0001)
            ...EQQI203I ADDEP PREOPNO(1) PREJOBN(START) PREWSID(DUMM)
            ...EQQI203I ADOP OPNO(015) WSID(DUMM) JOBN(JOBB) DURATION(1)
            ...EQQI203I HIGHRC(4) FORM(DD0002)
            ...EQQI203I ADDEP PREOPNO(005) PREJOBN(JOBA) PREWSID(CPU1)
            ...EQQI203I ADOP OPNO(020) WSID(CPU1) JOBN(JOBC) DURATION(1)
            ...EQQI203I FORM(DD0001)
            ...EQQI203I ADDEP PREOPNO(005) PREJOBN(JOBA) PREWSID(CPU1)
```

```
            ...EQQI203I ADDEP PREOPNO(010) PREJOBN(JOBQ) PREWSID(CPU1)
            ...EQQI203I ADOP OPNO(025) WSID(CPU1) JOBN(JOBD) DURATION(1)
            ...EQQI203I FORM(DD0001)
            ...EQQI203I ADDEP PREOPNO(015) PREJOBN(JOBB) PREWSID(DUMM)
            ...EQQI203I ADDEP PREOPNO(020) PREJOBN(JOBC) PREWSID(CPU1)
            ...EQQI203I ADOP OPNO(255) WSID(DUMM) JOBN(END) DURATION(1)
            ...EQQI203I DESCR('Last operation')
            ...EQQI203I ADDEP PREOPNO(025) PREJOBN(JOBD) PREWSID(CPU1)
07/08 15.05.43 EQQI112I Processing Application ADID(FRED) STATUS(A)
            ...EQQI112I VALTO(711231)
07/08 15.05.44 EQQI116I REPLACE for Application ADID(FRED) ADSTAT(A)
            ...EQQI116I ADVALFROM(080708) ADTYPE(A) completed
07/08 15.05.44 EQQI299I Statement completed - RC=0
```

Messages

EQQI002 - EQQI099, Workload Automation Programming Language control messages

EQQI002A

Loading data definitions for HCL Workload Automation for Z <version> <spe>

Explanation: Workload Automation Programming Language has initialized for a particular version of HCL Workload Automation for Z. The <version> number is the version of the HCL Workload Automation for Z software, <spe> contains a list of Small Product Enhancements considered to be in effect. This message is issued as Workload Automation Programming Language starts and may also be reissued as a result of OPTIONS statements that may require the data mapping to be reloaded.

System action: Reference data is loaded in accordance with the HCL Workload Automation for Z version and applied Small Product Enhancements.

User response: None.

EQQI004E

Parent statement <parent> missing for

Explanation: An attempt has been made to use a segment with a required parent statement missing.

System action: The command is not run.

User response: Specify the missing parent statement and run the command again.

EQQI005A

Segment <name> is not handled by Workload Automation Programming Language

Explanation: A segment has been encountered in the PIF header that is currently not defined to Workload Automation Programming Language.

System action: Workload Automation Programming Language will skip the segment and continue processing from the next segment.

User response: Ensure you are running with the latest release of Workload Automation Programming Language.

Note: While Workload Automation Programming Language is in Early Release status, this message is severity A, as it is likely the current release of Workload Automation Programming Language will not handle every segment. Once Workload Automation Programming Language reaches complete status this message will be raised to W.

EQQI006A

Exit <name> has set return code <rc>

Explanation: A Workload Automation Programming Language Segment Processing Exit has terminated with a nonzero return code.

System action: The return code is passed into the Workload Automation Programming Language processing and may impact the final exit code of WAPL.

User response: Determine if this is expected. If necessary correct the exit and rerun.

EQQI007A

Message <message-ID> changed from <severity> to <severity>

Explanation: The severity of a Workload Automation Programming Language message has been redefined.

System action: Any actions that may issue that message will set a return code in accordance with the new severity.

User response: None.

EQQI008A/W

 not supported for OUTPUT in version <version> of HCL Workload Automation for Z

Explanation: An OUTPUT statement has referred to a segment name that is not valid for the version of HCL Workload Automation for Z that Workload Automation Programming Language is currently set to use.

System action: The OUTPUT statement is ignored and processing continues.

User response: If you are using any of the supplied FILESPEC members (for example, EQQFLALL), this could simply mean you are using a version of HCL Workload Automation for Z earlier than the latest release, or without all of the available Small Product Enhancements loaded. If this is the case this message can be safely ignored. If this is your own OUTPUT statement causing the error then you may have mistyped the segment name, or are referring to a segment not available in the release of HCL Workload Automation for Z you are communicating with. Correct your input and rerun.

Note: This error message is initially delivered as an Advisory message as it is quite possible that many users may not be on the absolute latest release of HCL Workload Automation for Z with all the Small Product Enhancements applied. The FILExxxx FILESPEC members will raise the severity to W after they have loaded.

If you would sooner have this as a warning message to protect against mistyped segment names in your own FILESPEC members then use SETSEV to raise the priority of this message at the top of your FILESPEC member. For example, `SETSEV EQQI008W`

EQQI009A/W

<field> not in for <version> <SPEs>

Explanation: An OUTPUT statement has been processed that refers to a field that does not exist within the specified segment for the version of HCL Workload Automation for Z being used and the Small Product Enhancements that have been activated.

System action: Processing continues, the invalid fields will return no data.

User response: If you are using any of the supplied FILESPEC members (for example, EQQFLALL), this could simply mean you are using a version of HCL Workload Automation for Z earlier than the latest release, or without all of the available Small Product Enhancements loaded. If this is the case this message can be safely ignored. If this is your own OUTPUT statement causing the error then you may have mistyped a field or segment name, or are referring to a field not available in the release of HCL Workload Automation for Z you are communicating with. Correct your input and rerun.

Note: This error message is initially delivered as an Advisory message as it is quite possible that many users may not be on the absolute latest release of HCL Workload Automation for Z with all the Small Product Enhancements applied. The EQQFLALL and FILENONE members will raise the severity to W after they have loaded.

If you would sooner have this as a warning message to protect against mistyped segment names in your own FILESPEC members then use SETSEV to raise the priority of this message at the top of your FILESPEC member. For example, `SETSEV EQQI009W`

EQQI010W

Could not LIST <request-detail>

Explanation:

A LIST request has been processed without providing any results. Possible reasons are:

- There are no records matching the specified criteria.
- You are running the job without having at least READ access to the data being requested.
- You issued a LIST CRITSUCS request for an occurrence that was dynamically added to a current plan with no critical operations. No entries are shown by ISPF option 6.7.

System action: Processing continues, no data is returned, and return code 4 is set.

User response:

If the result is not as expected, perform the appropriate action as follows:

- Review the data in the current plan and revise the LIST request appropriately.
- After ensuring that you have the required authorization to complete the command execution, run the LIST request again.
- If the LIST CRITSUCS did not return any output, run the command again either after a current plan EXTEND or REPLAN.

Note: To prevent that LIST CRITSUCS does not return any data, ensure that at least 1 critical operation is included in the CP. The simplest way to do this is to mark any daily planning jobs EXTEND and REPLAN as CRITICAL=P, because one of these jobs is always included in the current plan and they are critical to the operations of HCL Workload Automation for Z.

EQQI011F

HCL Workload Automation for Z version <IWSver> not supported by Workload Automation Programming Language <WAPLver>

Explanation: An attempt has been made to start Workload Automation Programming Language with a version of HCL Workload Automation for Z that it was not written to support.

System action: Processing terminates.

User response: Contact the customer support to see if an alternate version of Workload Automation Programming Language is available. **Note**: You can specify a supported version of HCL Workload Automation for Z to connect to an unsupported release. Be aware that this may allow you extract information from HCL Workload Automation for Z, but care should be taken if trying to update HCL Workload Automation for Z specifying a version other than the level of the subsystem being communicated with as failures or data loss may occur.

EQQI012A

JOB <jobname>,<JESno,> is external to HCL Workload Automation for Z

Explanation: A command has been run that needs to understand the context of the job executing the command within HCL Workload Automation for Z. This message informs the user of the controlling Job Name and JES number, and determines that the job is not being controlled by HCL Workload Automation for Z.

System action: None.

User response: None.

EQQI013A

JOB <jobname>,<JESno,> in <ADID> <IA> <OPNO>

Explanation: A command has been run that needs to understand the context of the job executing the command within HCL Workload Automation for Z. This message informs the user of the controlling Job Name and JES number, and show the controlling Application ID, Input Arrival, and Operation number.

System action: None.

User response: None.

EQQI017A

Scanning activated for prefix of <prefix>

Explanation: A `VARSUB SCAN` command has been issued.

System action: Any subsequent statements will be scanned for the character listed as `<prefix>` and perform variable substitution.

User response: None.

EQQI018A

Scanning deactivated

Explanation: A `VARSUB NOSCAN` command has been encountered.

System action: Variable substitution will not take place for any subsequent statements.

User response: None.

EQQI019F

Variable <name> not found

Explanation: Variable substitution encountered a variable name that it could not find either as a supplied variable, in a listed variable table or as a user variable.

System action: The command fails and Workload Automation Programming Language stops processing further commands.

User response: Either correct the variable name, define it, or add a new `TABLE` reference that contains it.

EQQI020F

Table <table> not found

Explanation: A reference has been made to a JCL variable table that does not exist.

System action: The command fails and Workload Automation Programming Language stops processing further commands.

User response: Correct the table name or create the table.

EQQI021A

Table <name> loaded

Explanation: A JCL variable table has been opened and the values from it loaded at this point.

System action: Any variables referenced from within that table will use the values as they were at the point it is loaded, unless the values were changed by this Workload Automation Programming Language job.

User response: None.

EQQI022A

Search sequence <list of tables>

Explanation: A command has been issued that either opens or closes a JCL variable table, or otherwise alters the search sequence.

System action: This message lists the order that tables will be searched to find an unloaded variable value.

User response: None.

EQQI023E

Dependent variable loop (1)

Explanation: A dependent variable has been referenced that refers to a chain of dependent variables that lead back to it.

System action: The command will fail.

User response: Correct the dependencies.

EQQI024E

Table <name> was not open

Explanation: An attempt to **CLOSE** a table was made, but the table was not open at this point.

System action: The command will fail.

User response: Remove the **VARSUB CLOSE** for the offending table.

EQQI025A

Table <name> dropped

Explanation: A **VARSUB CLOSE** command has been issued for this table.

System action: The values are unloaded and Workload Automation Programming Language will not search this table for subsequent variable resolution.

User response: None.

EQQI028F

Variable <name> has not been assigned to a table

Explanation: A **VARSET SAVE** command was issued for a variable that is not connected with any JCL variable table.

System action: The command will fail and Workload Automation Programming Language stops processing further commands.

User response: Correct the variable name or assign to a **TABLE**.

EQQI032A

Date <variable-name> adjusted by <amount>

Explanation: A **VARDATE** command has been issued to create a date variable and the rule landed on a weekend date with the **RULE** keyword not set to **ON**.

System action: Workload Automation Programming Language will adjust the date in response to the rule and list the positive or negative number of days that it was adjusted by.

User response: None.

EQQI033A

Date <variable-name> set to <value>

Explanation: A **VARDATE** or **VARSET** command has been issued to a variable.

System action: For **VARDATE**, Workload Automation Programming Language will list the variable name, the date in format yymmdd, or the user specified format if FORMAT is coded. This will be followed by the day of the week the date falls upon to ease validation of the results. For **VARSET** only the variable name and value will be shown.

User response: None.

EQQI036F

<name> not found

Explanation: A command has been issued that referred to a `SAVELIST` or `OBJECT` that has not been created in this job.

System action: Workload Automation Programming Language will stop processing.

User response: Correct the name of the item and rerun.

EQQI049E

POOL/DYNAMIC POOL workstations not permitted with batch loader

Explanation: An attempt has been made to create or update a POOL or DYNAMIC POOL workstation using Workload Automation Programming Language. Though Workload Automation Programming Language will export batch loader for these types of workstation for reference use, they cannot be created or updated via Workload Automation Programming Language as elements of their definition are stored on the dynamic domain manager.

System action: The command terminates with an error. Workload Automation Programming Language continues with the next command.

User response: Define the workstation manually via the Dynamic Workload Console.

EQQI051W

Cannot <action> operation <opno> on <wstype> workstation <wsname>

Explanation: A current plan operation command has been issued that has selected an operation on a workstation type that is incompatible with the action being requested.

System action: Workload Automation Programming Language will continue processing other operations selected by the command, but the command will end with warnings. Workload Automation Programming Language continues with the next command.

User response: If this was not the desired result, refine the identification or filter keywords to avoid the selection of this operation.

EQQI053W

Dependency target missing – ADID(<adid>) OPNO(<opno>)

Explanation: An `ADDJOB` command has identified an operation which has a dependency that points to an operation that does not exist.

System action: Workload Automation Programming Language will continue processing other operations and dependencies discovered by the command, but the command will end with warnings. Workload Automation Programming Language continues with the next command.

User response: Correct the identified application definition.

EQQI054W

Dependency target has no jobname – ADID(<adid>) OPNO(<opno>)

Explanation: An **ADDJOB** command has identified an operation which has a dependency that points to an operation that does not have a job name.

System action: Workload Automation Programming Language will continue processing other operations and dependencies discovered by the command, but the command will end with warnings. Workload Automation Programming Language continues with the next command.

User response: If this job is intended to be used by **ADDJOB** consider revising the target operation to provide a job name, so job name searches can be used for dynamic dependencies.

EQQI056F

INCLUDE loop detected for <member>

Explanation: An **INCLUDE** command has attempted to load a member that has already been loaded higher up the **INCLUDE** chain.

System action: Workload Automation Programming Language terminates.

User response: Correct the member name to remove the loop and rerun.

EQQI057W

Unsupported object level <level> for

Explanation: A segment is being processed that has encountered and unexpected level of nesting for the record structure.

System action: Workload Automation Programming Language terminates.

User response: None.

EQQI058F

<number> consecutive FREEDAYS encountered

Explanation: A date was being calculated using relative FREEDAYS, and the calculation encountered more consecutive FREE days than is permitted. This could be indicative of a badly defined calendar that has no WORK days. The limit of consecutive FREE days allowed before this error occurs is set by **OPTIONS FREEMAX**. The default is 14 days.

System action: Workload Automation Programming Language terminates.

User response: Determine whether the run of consecutive FREE days is genuine, or the calendar is badly defined. Either increase **OPTIONS FREEMAX** or correct the calendar.

EQQI095F

Nest limit of <number> set by OPTIONS LIMIT has been reached

Explanation: A **DO UNTIL/WHILE/FOREVER** construct has executed more times than the limit set by **OPTIONS LIMIT**, the default of which is 100.

System action: Workload Automation Programming Language terminates.

User response: Determine the cause for the nesting level, if this is a desired behaviour then set **OPTIONS LIMIT** to a higher value.

EQQI096F

Loop limit of <number> set by OPTIONS LIMIT has been reached

Explanation: A `DO UNTIL/WHILE/FOREVER` construct has executed more times than the limit set by `OPTIONS LIMIT`, the default of which is 100.

System action: Workload Automation Programming Language terminates.

User response: Determine the cause for the nesting level, if this is a desired behaviour then set `OPTIONS LIMIT` to a higher value.

EQQI097E

Job <jobname>,<JESnum> is not running in the current plan

Explanation: A command is being run that requires the Workload Automation Programming Language job to be run from within HCL Workload Automation for Z so actions can be made in relation to the occurrence in which it is operating.

System action: The command terminates with errors and processing continues from the next command.

User response: Ensure the job is controlled by HCL Workload Automation for Z and rerun. If the job already was submitted by HCL Workload Automation for Z and Workload Automation Programming Language is unable to find itself in the current plan, this could be due to tracking performance issues. To avoid this kind of issue ensure the EQQCPOP DD statement is appropriately coded.

EQQI098W

No actions performed

Explanation: Workload Automation Programming Language has completed without performing any commands.

System action: Workload Automation Programming Language terminates with return code 4.

User response: If this is what you intended, then no action is required. If you expected more commands to have been executed check the following:

- That you have specified commands in the JCL or a referenced member.
- That your input DD statement is correctly spelled as SYSIN, INPUT or whatever you specified in `OPTIONS INPUT`.
- That you do not have any non JCL statements ahead of instream SYSIN as these may cause JES to generate an additional SYSIN statement.
- If calling Workload Automation Programming Language from REXX that you have pushed or queued commands to the external data queue.

EQQI099A

Workload Automation Programming Language complete - highest return code <rc> - <elapsed> (<eqqycom>) sec

Explanation:

Workload Automation Programming Language termination message, including the following key information:

<rc>

The maximum return code, after considerations from `SETMAX` and `LISTSTAT`.

<elapsed>

The total elapsed time of the Workload Automation Programming Language session.

<eqqycom>

The subset total elapsed time taken executing EQQYCOM.

System action: Workload Automation Programming Language terminates.

User response: None.

EQQI100 - EQQI199, Data exception messages

EQQI101F

Unable to read <DD name> <additional info>

Explanation: Workload Automation Programming Language has been unable to read a file, possibly due to the DD name not being allocated, or the input file never having been written to, or the record format not being appropriate to read with EXECIO. Additional info may contain the following:

-NODD

The DD statement is not allocated.

-NOMEM

The member is not found within the DD statement.

-ALLOC <rc>

Allocation of the EQQTEMP file failed with RC=<rc>.

System action: Workload Automation Programming Language terminates.

User response: Determine the problem with the file, correct it, and rerun.

EQQI103W

Output file <DD name> not allocated - skipping

Explanation: Workload Automation Programming Language has encountered an optional output file that has not been allocated.

System action: No output is written to this file, Workload Automation Programming Language continues.

User response: If the file should have been present, correct, and rerun.

EQQI105W

Truncation occurred for file <DD name>

Explanation: A file has been written to with at least one record that is longer than the record length.

System action: The file is written with any oversize records are truncated at the record length.

User response: Correct the record length and rerun.

EQQI106E

Failed writing to <DD name> RC=<RC> (records=<count>)

Explanation: An attempt has been made to write to an output file that has failed for reasons other than truncation. Possibly space or authority. The return code is from the REXX EXECIO service.

System action: The file is not written successfully, it may contain partial data.

User response: Determine the cause of the failure from return codes, additional messages and other additional information, such as file attributes. Correct and rerun.

EQQI108E

<number> <type> versions already exist for <object>

Explanation: An attempt has been made to create a new version of an object that is limited to the number of versions that can exist at once. Controlled types are:

Application

Limited to 4 versions per status.

System action: No update takes place.

User response: Delete unwanted old versions before attempting a rerun.

EQQI109E

Item already exists so cannot be added

Explanation: A Batch Loader request using `OPTIONS DBMODE(ADD)` for an object that is already in the database has been attempted.

System action: The object is not added to the database.

User response: Delete the object from the database or change the `DBMODE` to `REPLACE` and rerun.

EQQI110E

No key information specified

Explanation: A Batch Loader request has been specified that does not include any key fields to identify the object to update.

System action: The database is not updated.

User response: Amend the Batch Loader statements to include key fields or use a `USELIST` and rerun.

EQQI111E

<list-name> contains records for <type1> instead of <type2>

Explanation: A request to use the results of a saved list has been coded, but the list contains objects of a different type to the use that is being applied to.

System action: No update takes place.

User response: Correct the control statements to use an appropriate list and rerun.

EQQI113A

Options <keyword> may not have full effect for this stream

Explanation: The keyword is one that affects the way control statements are parsed. Workload Automation Programming Language will pre-parse an entire input stream of control statements to identify valid statements, consolidate continuation and remove comments before executing any of the statements. This means any subsequent statements in the same stream might not benefit from the changes made by this `OPTIONS` statement and parsing errors, such as unknown commands or keywords, may be reported.

System action: The new parsing behavior takes effect from the next input stream.

User response: Verify whether this is acceptable, and possibly move the OPTIONS statement to an earlier input stream.

Note: This warning is only issued for the input streams for standard input (SYSIN) and the REXX stack.

EQQI114E

 not found to delete

Explanation: An attempt has been made to remove a non-existent segment from an existing object using the `ACTION(DELETE)` keyword in Batch Loader.

System action: The object is not updated.

User response: Correct the Batch Loader statements and rerun.

EQQI117W

<action> for <type> <key information> ended with warnings

Explanation: A Batch Loader request has completed but has produced additional warning messages. This message clarifies the action being attempted and identifies the object that has been used. The action can be `INSERT`, `REPLACE`, or `SUBMIT`..

System action: Batch Loader process completes with warnings.

User response: Review the warning messages, if necessary correct and rerun.

EQQI118E

<action> for <type> <key information> failed

Explanation: A Batch Loader request has failed to complete. This message clarifies the action being attempted and identifies the object that has been used. The action can be `INSERT`, `REPLACE`, or `SUBMIT`.

System action: Batch Loader process failed.

User response: Review the warning messages, correct and rerun.

EQQI120W

EXECUTE not performed

Explanation: Statements to update the Current Plan have been processed but not `EXECUTE` or `RESET` request has been performed.

System action: No updates are made to the Current Plan.

User response: Add an `EXECUTE` or `RESET` command to the control statements or use `OPTIONS EXECUTE(AUTO)`.

EQQI121E

<count> matches found for LISTSTAT

Explanation: More than one record was identified as matching the arguments to the `LISTSTAT` request. Only one record must be identified for `LISTSTAT` to work.

System action: `LISTSTAT` fails.

User response: None.

EQQI124W

OUTPUT statement not found for parent of - ignoring DATA(=)/LOADER(=)

Explanation: An `OUTPUT` statement has been coded with a `DATA` or `LOADER` keyword using `=` as the value. This means that the destination for the parent segment should be used, but when this message appears it means that an `OUTPUT` statement for a parent segment has not been coded.

System action: This message will be issued and the output stream for the keyword will be ignored.

User response: Replace the `=` with a real destination or code an `OUTPUT` statement for the parent before this `OUTPUT` statement.

EQQI125E

COPY cannot replace existing records <key-information>

Explanation: An attempt has been made to copy an object over the top of an existing object using `DBMODE(COPY)`.

System action: No update takes place.

User response: Either delete the existing object or alter the NEW field specifications to create an object that does not already exist.

EQQI126E

Bad predecessor <pred> for <succ>

Explanation: When both workstation and operation number are not specified together for a dependency, Workload Automation Programming Language must look up the values using the information provided. This message means not enough information was specified to uniquely identify a predecessor. Either the input had more than one possible match, or no match.

System action: No update takes place.

User response: Either provide both workstation and operation number, or refine the statements to ensure a unique operation is identified.

EQQI127E

Cannot find a matching operation in the Current Plan

Explanation: A command has been issued that needs to update an operation in the current plan, but the information provided does not match any operation in the plan.

System action: The command terminates without performing any update.

User response: Check the Current Plan to see if the operation you wish to update is that, and if so correct the command to provide the necessary information to locate the required operation.

EQQI128E

Unable to find unique Application <adid> valid on <date>

Explanation: An attempt has been made to `INSERT` an occurrence into the Current Plan using an alias. The ADID specified does not have a Valid Active version for the date it is due to run on.

System action: An occurrence is not submitted.

User response: Correct and rerun.

EQQI129E

Unable to retrieve <type> <key information> from database

Explanation: An attempt to `SELECT` a record from the database has failed.

System action: The record is not retrieved.

User response: Additional messages may indicate why the failure occurred. It may be that the object does not exist in the database. If the `SELECT` was generated as a result of a `LIST` statement it is possible that the object was deleted by another user between the `LIST` and `SELECT`.

EQQI0131F

Unexpected data format for <field>

Explanation: Workload Automation Programming Language has encountered a field when reading a record that does not match the expected format as defined in the Workload Automation Programming Language data mapping.

System action: Subsequent EQQI0132A messages will be issued with diagnostic information. A mini dump of the segment will be displayed. The command fails and Workload Automation Programming Language stops processing further commands.

User response: Check that you are instructing Workload Automation Programming Language to run for the correct version of Workload Automation Programming Language and you have the correct `OPTIONS SPE` settings for that subsystem.

EQQI134E

User field "<name>" not defined for this job

Explanation: A `VARSET USRF` command has been issued with `MISSING(ERROR)` and the named user field was not defiled to the operation.

System action: The command fails.

User response: Correct the user field name or define the user field to the operation.

EQQI135F

User field "<name>" not defined for this job

Explanation: A `VARSET USRF` command has been issued with `MISSING(FAIL)` and the named user field was not defiled to the operation.

System action: The command fails and Workload Automation Programming Language stops processing further commands.

User response: Correct the user field name or define the user field to the operation.

EQQI136F

File <ddname> not allocated

Explanation: A command has been issued referring to a dd name that does not exist in the step.

System action: The command fails and Workload Automation Programming Language stops processing further command.

User response: Correct the DD name in the command or add the DD statement to the step.

EQQI137F

File <ddname> not available for input processing

Explanation: A command has been issued referring to a dd name that is not considered to be an input file, for example, a SYSOUT file.

System action: The command fails and Workload Automation Programming Language stops processing further command.

User response: Correct the DD name in the command or correct the DD statement in the JCL.

EQQI139F

All data sets in <ddname> must be cataloged for search

Explanation: An `INCLUDE` command has been issued referring to a member within a dd name that contains one or more partitioned data sets that are not catalogued. To perform a concatenation search Workload Automation Programming Language requires all of the data sets to be catalogued.

System action: The command fails and Workload Automation Programming Language stops processing further command.

User response: Either ensure all of the data sets are catalogued, or allocate the specific member explicitly in the JCL and `INCLUDE` from that explicit DD statement without specifying a member name.

EQQI140F

Member <name> not found in <ddname>

Explanation: An `INCLUDE` command has been issued referring to a member in a dd name that does not contain any datasets in which the named member exists.

System action: The command fails and Workload Automation Programming Language stops processing further command.

User response: Correct the DD name in the `INCLUDE` statements or correct the DD statement in the step to include a library containing the specified member.

EQQI141F

Unable to allocate <dataset> to <ddname>

Explanation: A command the requires a dynamic allocation to the Workload Automation Programming Language temporary file has been issued. This is done whenever a member is needed from a library that is specified in the JCL without the member explicitly specified.

System action: The command fails and Workload Automation Programming Language stops processing further command.

User response: Identify the reason for failure and rerun. The most likely cause is contention, but if you have also made a lot of dynamic allocations in the step then DYNAMNBR may need specifying in the JCL.

EQQI143E

Cannot rename an item that does not exist

Explanation: Batch loader has been specified with `DBMODE UPDATE` or `COPY` that specifies new key names using `NEW_` fields, but the original key refers to an object that does not exist.

System action: The command will fail.

User response: Correct the batch loader key fields.

EQQI144W

<UPDATE|COPY> will overwrite existing record <key information>

Explanation: Batch loader has been specified with `DBMODE UPDATE` or `COPY` that specifies new key names using `NEW_` fields. The new key points to an existing record but `OPTIONS OVERWRITE(Y)` has been specified.

System action: The existing option will be overwritten.

User response: None.

EQQI145E

No user fields match <mask>

Explanation: An `INCLUDE` statement has been specified that has a mask which does not match any user fields attached to the job.

System action: The command will fail.

User response: Correct the mask, or user fields against the operation.

EQQI146E

Table <name> could not be updated

Explanation: A `VARSAVE` or `SETVAR SAVE` command was issued, but Workload Automation Programming Language was unable to update the table.

System action: The command will fail, no updates were performed to the named table.

User response: Determine the reason for failure. Most likely causes are contention, trying to update a table that doesn't exist, or insufficient access rights.

EQQI148F

Updates to the current plan have failed

Explanation: A `DELETE`, `INSERT` or `MODIFY` command for a resource beginning with CP has failed and `OPTIONS CPFAIL(ABORT)` is in effect. This is to protect against an incomplete sequence of update actions being committed to the current plan.

System action: Return code 12 will be issued and processing will stop unless `OPTIONS HIGHRC` has been amended.

User response: Determine the reason for failure and correct. If you wish to allow processing to continue in such circumstances set `OPTIONS CPFAIL(ERROR)`.

EQQI150W

No matching run cycles found within <application>

Explanation: A `PRSTART` batch loader statement has been executed that includes setting dates from an Application, or a `GETDATES` command extracting dates from an application has been run. There were not run cycles within the application that matched either the Input Arrival or validity criteria.

System action: Workload Automation Programming Language will issue RC=4 and continue.

User response: None.

EQQI152E

An internal command failed – see previous messages

Explanation: A command has been executed that internally calls other Workload Automation Programming Language commands and one of these internal commands has failed.

System action: The reason for the failure will be listed in the message group for the internal command, earlier in the Workload Automation Programming Language log.

User response: Respond to the earlier error messages.

EQQI155W

Application validity <date-range> is outside range <date-range>

Explanation: A `PRSTART` batch loader statement has been executed that includes setting dates from an Application, or a `GETDATES` command extracting dates from an application has been run. The named application is not in effect within the range specified by the `FROMDATE` and `TODATE` keywords.

System action: Workload Automation Programming Language will issue RC=4 and continue.

User response: Determine if this is the correct behaviour. If not alter the FROMDATE/TODATE range or correct the application and rerun.

EQQI160W

No entries found matching criteria

Explanation: A command has been issued to find elements in the database or plans, but nothing was found.

System action: Workload Automation Programming Language will issue RC=4 and continue.

User response: Review the output and correct if necessary.

EQQI164E

<adid> <opno> has no successors for CONNECT(SUCC)

Explanation: The `QUEUE` command has been issued with the `CONNECT(SUCC)` keyword specified.

System action: Workload Automation Programming Language will issue RC=8 and continue.

User response: Review the output and correct if necessary.

EQQI170E

Unable to activate console <console-name>

Explanation: A `CONSOLE` command was unable to activate an extended console.

System action: Workload Automation Programming Language will issue RC=8 and continue.

User response: Review associated messages and correct.

EQQI171E

External command returned negative RC (1)

Explanation: A `CALL` command has executed an external REXX routine that has resulted in a negative return code. This is often caused by the REXX routine not being found in the execution path.

System action: Workload Automation Programming Language will issue RC=8 and continue.

User response: Review the execution path to ensure the REXX routine is accessible.

EQQI172W

Workstation <wsname> not suitable for <command> - Oper <ADID> <IA> <wsname>_<OPNO>

Explanation: A current plan operations command has been issued for an operation that is not valid. The command might have been issued for all the operations in an occurrence, with the intention of applying the command only to the operations that are permitted.

System action: Workload Automation Programming Language will issue RC=4 and continue.

User response: Review the output and correct if necessary.

EQQI173E

Cannot lock <record-type> <key> for <update-mode>

Explanation: A batch loader command has been issued for an object that cannot be locked exclusively, after retrying in accordance with the `OPTIONS CONTENTION` setting.

System action: Workload Automation Programming Language will issue RC=8 and continue.

User response: Identify and resolve the cause of contentions before rerunning.

EQQI174W

An internal command issued warnings - see previous messages

Explanation: A command has been executed that called other commands internally that issued Warning messages effecting the success of the command.

System action: Workload Automation Programming Language will issue RC=4 and continue.

User response: Review the messages from the preceding block of messages.

EQQI175W

Missing interval for <wsname> - Cannot reset

Explanation: A `WSALTER` command has been executed that requested a reset of either parallel servers or resource quantities, but the FROM and TO keywords do not point to an existing interval.

System action: Workload Automation Programming Language will issue RC=4 and continue.

User response: An interval can only be reset if it already exists. Either correct the FROM and TO to point to the existing interval covering the time period you want to reset to planned values, or provide explicit values for PSCAP, R1CAP and R2CAP for the time period concerned.

EQQI176E

Cannot find inserted job to edit JCL

Explanation: An `ADDJOB` or `JBSTART` command has run with the JCL keyword, but the update process is unable to locate the newly inserted job.

System action: Workload Automation Programming Language will issue RC=8 and continue.

User response: Investigate why the inserted job cannot be located and rerun.

EQQI180W

Completed successor AD=<applicationID> IA=<YYMMDDHHMM> OP=<nnn>

Explanation: An `ADDJOB` or `JBSTART` command has run that has identified a defined successor that is already in a complete state. An external predecessor cannot be added to a completed operation. Keyword `COMPSUCC(WARNING)` has been specified.

System action: Workload Automation Programming Language will not apply the dependency, issue RC=4, and continue.

User response: Review the details to determine if further actions are required.

EQQI181E

Completed successor AD=<applicationID> IA=<YYMMDDHHMM> OP=<nnn>

Explanation: An **ADDJOB** or **JBSTART** command has run that has identified a defined successor that is already in a complete state. An external predecessor cannot be added to a completed operation. Keyword **COMPSUCC(ERROR)** has been specified.

System action: Workload Automation Programming Language will not apply the dependency, issue RC=8, and continue.

User response: Review the details to determine if further actions are required.

EQQI195W

File <ddname> not open

Explanation: A **CLOSE** command has been issued for a file that was not open.

System action: Workload Automation Programming Language will issue RC=4 and continue.

User response: Review the message to determine whether the file name needs to be corrected or the **CLOSE** command removed.

EQQI197W

End of file reached for <ddname> at <number> records

Explanation: A **READ** command has been executed that specified a number of records to read, but the file did not have that many records within it.

System action: Workload Automation Programming Language will issue RC=4 and continue.

User response: Investigate the reason for the discrepancy and correct.

EQQI198E

Input file <ddname> not allocated for input

Explanation: A command has been executed that requires an input file, but the named DD is either not allocated, or does not refer to an input file.

System action: Workload Automation Programming Language will issue RC=8 and continue.

User response: Correct the file allocations and rerun.

EQQI199E

Failed reading from file <ddname> RC=<return-code>

Explanation: A command has been executed that reads a file, but has failed.

System action: Workload Automation Programming Language will issue RC=8 and continue.

User response: Consult REXX documentation for the EXECIO command for an explanation of the return code.

EQQI204 - EQQI299, Syntax related messages

EQQI204F

Missing end quote for keyword <keyword>

Explanation: A parsing error has occurred.

System action: Workload Automation Programming Language terminates.

User response: Correct quoting and rerun.

EQQI205F

Missing right parenthesis for keyword (1)

Explanation: A parsing error has occurred.

System action: Workload Automation Programming Language terminates.

User response: Correct parenthesis and rerun.

EQQI206F

Invalid value <value> for <keyword>

Explanation: A parsing error has occurred. An inappropriate value has been found for a keyword or field.

System action: Workload Automation Programming Language terminates.

User response: Consult the documentation, correct and rerun.

Note: The documentation may state that the value is valid for the keyword. Workload Automation Programming Language could still reject this if the value is not valid for the particular version of HCL Workload Automation for Z you are communicating with, or you have not enabled a particular SPE. Check message EQQI002I to see what version of HCL Workload Automation for Z Workload Automation Programming Language believes it is operating against.

EQQI207F

Invalid keyword <keyword>

Explanation: A parsing error has occurred. An inappropriate keyword has been coded.

System action: Workload Automation Programming Language terminates.

User response: Consult the documentation, correct and rerun.

Note: The documentation may state that keyword is valid. Workload Automation Programming Language could still reject this if the keyword is not valid for the particular version of HCL Workload Automation for Z you are

communicating with, or you have not enabled a particular SPE. Check message EQQI002I to see what version of HCL Workload Automation for Z Workload Automation Programming Language believes it is operating against.

EQQI208F

Resource <resource> is not valid for <command>

Explanation: An invalid resource has been specified for a command.

System action: Workload Automation Programming Language terminates.

User response: Workload Automation Programming Language will validate that a valid resource has been used with respect to the type of request, the version of HCL Workload Automation for Z being used and what SPEs have been activated. If the resource name is typed correctly, ensure that the HCL Workload Automation for Z Version and SPEs have been declared correctly, and rerun. If the resource name is not correct, enter the correct resource name and rerun.

EQQI209E

Exit <exit name> not found - OPTIONS EXIT reset

Explanation: An exit has been specified using `OPTIONS EXIT` that is not declared in EQQYXU00.

System action: Workload Automation Programming Language issues an error message and continues without the exit.

User response: Correct the `OPTIONS` statement or update EQQYXU00 to include the new exit before rerunning.

EQQI210E

<keyword> required for <command> command

Explanation: An attempt has been made to use a command that has a required keyword missing. The keyword may be one that is always required by the command, or may only be required in conjunction with other keywords you have specified.

System action: The command is not run.

User response: Provide the missing keyword and rerun.

EQQI211F

Invalid comparator <comparator> for keyword <keyword>

Explanation: An invalid comparator has been specified against the named keyword.

System action: The command will fail, Workload Automation Programming Language will stop processing further commands.

User response: Correct the comparator. See section Using comparators on page 47.

EQQI212F

Unrecognized/unsupported statement <command> in <input stream>

Explanation: An invalid command has been coded in one of the Workload Automation Programming Language input streams. This may either be a command that is not known to Workload Automation Programming Language, or one that is not supported for the version of HCL Workload Automation for Z being used. The input streams can be:

OPTIONS

Site option defaults

SUBSYS

Subsystem option defaults

-ARGS-

The Workload Automation Programming Language argument

FILESPEC

ILSON file specifications

-STACK-

The REXX stack

INPUT

The main input stream (SYSIN)

Anything else will refer to either an **INCLUDE** stream or automatically generated commands from **LIST** or **SELECT** statements.

System action: Workload Automation Programming Language terminates.

User response: Correct and rerun.

EQQI213F

Call to <load module> has abended RC(<rc>)

Explanation: An attempt to call a load module internal to Workload Automation Programming Language has failed.

System action: Workload Automation Programming Language terminates.

User response: Determine the reason for the failure and rerun. Possible causes could be

- Module not found (S806), ensure the library containing the module is defined to Workload Automation Programming Language by STEPLIB or other means.
- Insufficient region.
- Version mismatch. The HCL Workload Automation for Z module EQQYCOM can object to being run against versions of HCL Workload Automation for Z other that the version it is written for, Ensure you are calling the correct version of EQQYCOM for the subsystem you are communicating with.

EQQI214E

<keyword> must be coded before <keyword>

Explanation: A Workload Automation Programming Language command has been coded that requires certain keywords to be coded in a specific order.

System action: Workload Automation Programming Language terminates.

User response: Correct the command and rerun.

EQQI215A

SPE (<spe name>) is not valid for TWS version <version> – ignored

Explanation: An **OPTIONS SPE** statement was encountered that attempted to activate an SPE for a version of HCL Workload Automation for Z earlier than the version from which the SPE was made available.

System action: The SPE is not activated.

User response: Ensure you have declared the correct SPE and or version of HCL Workload Automation for Z.

EQQI216F

<keyword1> is not allowed with <keyword2>

Explanation: Two keywords have been encountered that cannot be specified together.

System action: Workload Automation Programming Language terminates.

User response: Correct and rerun.

EQQI217F

NEW_ not permitted with DBMODE ADD or REPLACE

Explanation: Batch loader has been specified with **DBMODE ADD** or **REPLACE** that specifies new key names using **NEW_** fields. The **ADD** and **REPLACE** modes must specify an entire object as it is to be saved, it cannot rename or copy an existing object.

System action: The command will fail, Workload Automation Programming Language will stop processing further commands.

User response: Remove the **NEW_** keywords or alter the **DBMODE**.

EQQI218E

ADRULE must follow ADRUN TYPE R or E

Explanation: An **ADRULE** statement has been encountered that does not follow and **ADRUN** segment for types R or E.

System action: No update takes place.

User response: Correct and rerun.

EQQI219F

Keyword <keyword1> not permitted with <keyword2> for <command>

Explanation: A command has been specified that uses two keywords that are mutually exclusive.

System action: Workload Automation Programming Language terminates

User response: Correct and rerun.

EQQI220E

<object name> and SUFFIX <suffix> exceeds field length <length>

Explanation: The SUFFIX keyword has been used to generate a new name for an object by combining the original name with a TWS supplied variable. In this case the combination of the original name and the contents of the variable exceed the field length for the object name and `OPTIONS SUFFIX(FAIL)` has been used.

System action: No update takes place.

User response: Amend the input to use an alternate combination of name and variable or use a different value for `OPTIONS SUFFIX`.

EQQI221F

Unexpected THEN or ELSE

Explanation: A `THEN` keyword or `ELSE` statement has been encountered without being adjacent to an `IF` clause and its single action or `DO` block.

System action: Workload Automation Programming Language terminates

User response: Correct and rerun.

EQQI222F

Tracker not defined to Workload Automation Programming Language for <controller> on <SYSID> <LPAR>

Explanation: A TSO command has been issued without a `SUBSYS` but the value pointed to by `OPTIONS SYSID` for this LPAR does not have match in the `OPTIONS TRACKERS` lookup table.

System action: The command is not issued

User response: Add a `SUBSYS` keyword to the command or correct the `OPTIONS TRACKERS` table to include an entry for the LPAR.

EQQI223W

Use of MSTR is not recommended

Explanation: A TSO command has been issued using MSTR as the SUBSYS value. This sends the event to every HCL Workload Automation for Z subsystem on the LPAR, which can lead to error messages on EQQMLOG for controllers that have no match for the event, and may lead to inappropriate triggering of workload.

System action: The command is issued with RC=4.

User response: If possible use the `OPTIONS SYSID` and `OPTIONS TRACKERS` feature to target the event directly to the appropriate subsystem.

EQQI225E

FIELDSEP/LABELSEP incompatible with EXIT - OPTIONS EXIT reset

Explanation: An attempt to call a segment processing exit has been made with inappropriate settings for **FIELDSEP** or **LABELSEP**. Either both of these separators have been set to the same value, or at least one of them has been turned off. Segment processing exits need them both set to different values to be able to process the data.

System action: Processing continues, the exit is turned off and not attempted for subsequent segments.

User response: Use **OPTIONS FIELDSEP** and **OPTIONS LABELSEP** to set different and valid values, or turn off the exit, using **OPTIONS EXIT**.

EQQI226F

Open comment before text block on line <num> of <stream>

Explanation: A comment has been encountered on a line with **DLM** coded that has been opened but not closed.

System action: Processing terminates.

User response: A line containing **DLM** indicates that the following lines contain a text block. Text blocks cannot contain comments. Remove or close the comment on the line containing the **DLM** keyword.

EQQI227F

Potentially inconsistent line numbers detected

Explanation: Message 207 or 212 has been issued indicating an unknown command or keyword. The invalid keyword has been detected as beginning with an 8 digit numeric field. This is most likely caused by line numbers having been turned on at some point in the past, and then turned off again, resulting in a situation where some lines in SYSIN have line numbers, and some do not. When this happens, Workload Automation Programming Language cannot determine whether these were genuine line numbers, or were intended as being part of the SYSIN itself.

System action: Processing terminates.

User response: Remove the lines from columns 73 to 80.

EQQI228F

A date must precede description for tagging

Explanation: Automatic date tagging is in effect (see **OPTIONS TAGMODE**) and a description keyword has been encountered before the corresponding keyword containing the date has been encountered in the batch loader statements.

System action: Processing terminates.

User response: Amend the sequence of the batch loader statements to place the keyword containing the date before the description keyword.

EQQI230F

Incomplete DO/IF <stream> <line>

Explanation: A logical **DO** or **IF** structure is not structured correctly.

System action: Workload Automation Programming Language terminates.

User response: Correct and rerun.

EQQI231F

Unexpected or unmatched END <stream> <line>

Explanation: An `END` statement has been encountered without a matching `DO` statement.

System action: Workload Automation Programming Language terminates.

User response: Correct and rerun.

EQQI232F

Evaluation Error - <explanation>

Explanation: A REXX based expression has failed to evaluate correctly.

System action: Workload Automation Programming Language terminates.

User response: Correct and rerun.

EQQI234F

SUBROUTINE name <name> already used

Explanation: A duplicate subroutine has been defined.

System action: Workload Automation Programming Language terminates.

User response: Correct and rerun.

EQQI235F

<command> not permitted in environment <environment>

Explanation: You are running Workload Automation Programming Language in a non-batch or non-TSO environment such as AF/OPERATOR or NETVIEW. The requirements of the command you are executing is not compatible with this environment.

System action: Workload Automation Programming Language terminates.

User response: Correct and rerun, or execute in an alternate environment such as triggering a started task from the environment you are using.

EQQI236E

User field <name> not found for operation <opnum> <jobname>

Explanation: The `ALTIF` command is being run and has not found a `DO` user field to match the `IF` user field found against a particular operation.

System action: The command ends RC=8, and continues with the next command.

User response: Add corresponding `DO` user field or remove the `IF` user field.

EQQI300 - EQQI399, EQQYCOM control messages

EQQI301F

PIF command <command> abandoned due to INIT failure

Explanation: A PIF process was attempted, but the preceding `INIT` request failed.

System action: Any commands requiring the PIF will not run.

User response:

Identify the reason for the failure of the `INIT` command and rerun. Possible causes could be:

- The controller is not active.
- The server through which PIF is trying to communicate is not active.
- The communication route to the server is failing.

EQQI302F

Unable to GET|FREE <size> bytes of storage

Explanation: An attempt to obtain storage for a PIF request has failed.

System action: Workload Automation Programming Language terminates.

User response: Identify the cause of the storage problem and rerun.

EQQI303O

Contention/Delay, retrying in <seconds> second(s) <attempt>/<max>

Explanation: A contention condition has occurred and the `OPTIONS CONTENTION` feature is active.

System action: Workload Automation Programming Language will reattempt the command in the specified number of seconds, and repeat this process until <attempt> has reached <max>. After which Workload Automation Programming Language will terminate if unsuccessful.

User response: Identify and alleviate the contention.

EQQI304A

Delaying for <seconds> second(s)

Explanation: The OPTIONS DELAY feature has been enabled to cause a time delay to happen after certain categories of command, to reduce risk of contention and allow other processes a share of HCL Workload Automation for Z resources.

System action: The program pauses for the stated number of seconds.

User response: None.

EQQI400 - EQQI499, Validation messages

EQQI401F

Invalid value for addition/subtraction <value>

Explanation: A date based calculation is being performed with an invalid value specified to add or subtract.

System action: Processing terminates.

User response: Correct the value and rerun.

EQQI402F

Invalid value for inFormat <value>

Explanation: A date based calculation is being performed with an invalid input format specified.

System action: Processing terminates.

User response: Correct the format and rerun.

EQQI403F

Invalid value for outFormat <value>

Explanation: A date based calculation is being performed with an invalid output format specified.

System action: Processing terminates.

User response: Correct the format and rerun.

EQQI404F

Invalid value for inDate <value>

Explanation: A date based calculation is being performed with an invalid date used as input.

System action: Processing terminates.

User response: Correct the date and rerun.

EQQI500 - EQQI599, Function based messages

EQQI500E

Job <jobname>,<JESnum> not scheduled by <ADID(ad)|GROUPDEF(<grp>)>

Explanation: The `ADD` command is being used to repeat an Application or Group, but the job running the command is not included in the named application.

System action: Processing terminates.

User response: When running in REPEAT mode the `ADD` command does not need the `ADID` or `GROUPDEF` keywords specifying. If this job is supposed to be repeating the occurrence it is running within, remove the `ADID` or `GROUPDEF` keywords and rerun.

EQQI501A

Repeat <ADID(ad)|GROUPDEF(grp)> from hhmm to hhmm

Explanation: The **ADD** command is being used to repeat an Application or Group. This message indicates the occurrences being repeated and the **FROM** and **UNTIL** limits.

System action: Processing continues.

User response: None.

EQQI502W

Current IA <yymmddhhmm> outside of FROM/UNTIL range

Explanation: The **ADD** command is being used to repeat an Application or Group, but the instance running has an IA outside of the **FROM/UNTIL** limits. This will have been added outside the limits by a process other than the **ADD** command in REPEAT mode.

System action: Processing continues but no further occurrences are scheduled.

User response: None.

EQQI503W

Next IA <yymmddhhmm> outside of FROM/UNTIL range

Explanation: The **ADD** command is being used to repeat an Application or Group. It has calculated that the next instance will be outside of the **FROM/UNTIL** limits.

System action: Processing continues but no further occurrences are scheduled.

User response: None.

EQQI504E

<yymmddhhmm> overlaps with existing <yymmddhhmm>

Explanation: The **ADD** command is being used to repeat an Application or Group. The IA for the next calculated occurrence overlaps with the IA of another occurrence of the same application that has been added by other means.

System action: Processing continues but no further occurrences are scheduled.

User response: Investigate the additional occurrence and act accordingly.

EQQI506E

No Application|Group <adid> found active on <yymmdd>

Explanation: The **ADD** command is being used to submit an Application or Group, but no instance of the requested Application or Group could be found valid on the date specified.

System action: Processing terminates.

User response: Correct the application name or group and rerun.

EQQI507A

The IA of the next occurrence will be <yymmddhhmm>

Explanation: The **ADD** command is being used to add an Application or Group. This message shows the Input Arrival that will be used for submission.

System action: Processing continues.

User response: None.

EQQI508A

No new occurrence will be added

Explanation: The **ADD** command is being used to repeat an Application or Group. It has determined that no further occurrences will be added. There will be an accompanying message 502, 503, 504, or 510 to explain why.

System action: Processing continues but no further occurrences are scheduled.

User response: None.

EQQI509A

This is instance <x> of <y>

Explanation: The **ADD** command is being used to repeat an Application or Group. The **COUNT** keyword is being used, this message indicates how many instances have run so far and what the upper limit is.

System action: Processing continues.

User response: None.

EQQI510W

Count limit of <y> has been reached

Explanation: The **ADD** command is being used to repeat an Application or Group. The **COUNT** keyword is being used and the upper limit has been reached.

System action: Processing continues but no further occurrences are scheduled.

User response: None.

EQQI511E

Command executed before IA of <yymmddhhmm>

Explanation: The **ADD** command is being used to repeat an Application or Group. It has detected that the **ORIGIN** time is earlier than the Input Arrival of the controlling occurrence.

System action: Processing terminates.

User response: Investigate the reason for the early execution. If deliberately released early, the job can be restarted with the keyword **EARLY(CONTINUE)** coded to resume the repeat cycle at the next interval. If the job ran early because the repeating occurrence has no time dependency, correct the application definition in the database before restarting the job.

EQQI512A

IA adjusted from <yymmddhhmm> to <yymmddhhmm>

Explanation: The **ADD** command is being used to submit an Application or Group. Input arrival was specified, but an occurrence already existed, and **FINDIA** was specified as Y. This lists the originally specified IA and what IA was used instead.

System action: Processing continues.

User response: None.

EQQI900 - EQQI999, Trace messages

EQQI901C

Unable to read EQQLANG

Explanation: Workload Automation Programming Language has been unable to read the language file EQQLANG, possibly due to the DD name not being allocated, or the input file never having been written to, or the record format not being appropriate to read with EXECIO.

System action: Workload Automation Programming Language terminates before initialization has completed.

User response: Determine the problem with the file, correct it and rerun.

EQQI902C

Message <msgID> missing

Explanation: Workload Automation Programming Language has attempted to issue a message that is not contained within the EQQLANG file.

System action: Workload Automation Programming Language terminates.

User response: Determine the reason for the failure. It is possible that the cause is that Language File is not the correct file for the version of EQQYXTOP that is executing, or has been customized incorrectly.

EQQI903C

Unrecognized severity <severity>

Explanation: The message file EQQLANG contains a message in which the severity is one not recognized by EQQYXTOP.

System action: Workload Automation Programming Language terminates.

User response: Determine the reason for the failure. It is possible that the cause is that Language File is not the correct file for the version of EQQYXTOP that is executing, or has been customized incorrectly.

Notices

This document provides information about copyright, trademarks, terms and conditions for product documentation.

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.

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