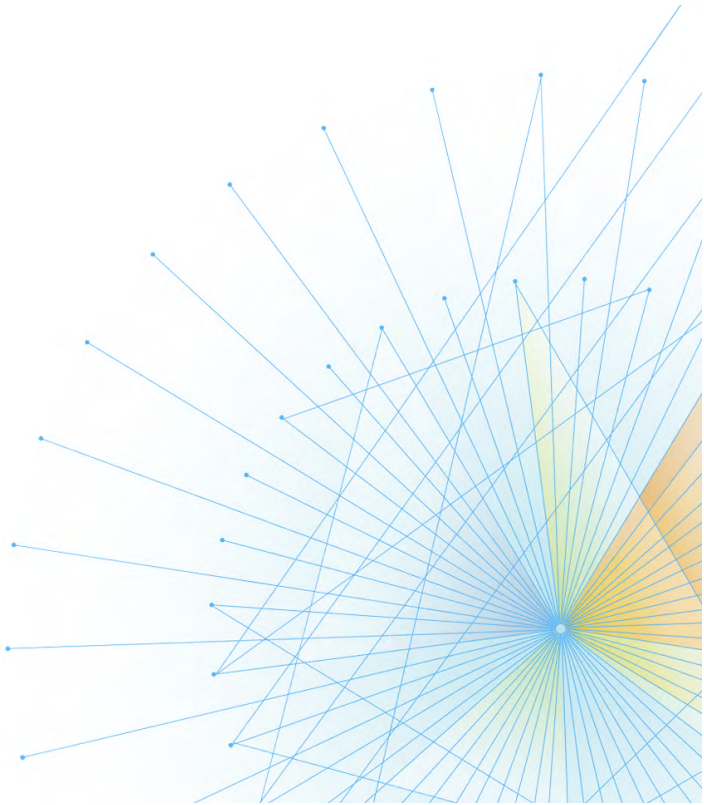
Compuware | The Mainframe Software Partner For The Next 50 Years

ThruPut Manager
User Control Services (UCS) User Guide

Release 18.02



Please direct questions about ThruPut Manager
or comments on this document to:

ThruPut Manager **Customer Support**

https://go.compuware.com/

Doc. APR2019
February 25, 2019

Introduction

Summary of Changes

V1802-7118 (*April 2019*)	• No changes
V1802-7117 (*January 2019*)	• No changes
V1802-7116 (*October 2018*)	• No changes
V1802-7115 (*July 2018*)	• No changes
V1802-7114 (*April 2018*)	• No changes
V1802-7113 (*January 2018*)	• No changes
V1802-7112 (*October 2017*)	• No changes
V1802-7110 (*July 2017*)	• Rebranding of MVS Solutions to Compuware. This includes update of cover style, copyright, and changing version release to 18.02.
V7R1-7109 *(April 2017)*	• No changes
V7R1-7108 *(Feb 2017)*	• No changes
V7R1-7107 *(May 2016)*	• No changes
V7R1-7106 *(November 2015)*	• Added Appendix B: JES2 Job Execution Control Comparison. This appendix is the comparison of z/OS 2.2 and ThruPut Manager Dependent Job Control and deadline scheduling capabilities.
V7R1-7104 *(July 2015)*	• No changes
V7R1-7101 *(July 2014)*	• This is a base manual for ThruPut Manager Version 7 Release 1.0.

About This Manual

This manual is applicable to anyone who submits sets of jobs and needs to control them based on time or in relation to one another. It allows a person to set up a suite of jobs for testing, without relying on Operations to start, hold, or release jobs.

Contents

Chapter 1
Introduction to User Control Services

This chapter introduces the User Control Services (UCS). The remainder of this guide consists of chapters describing each of the sub-components of UCS and how to use them.

What is User Control Services?

User Control Services (UCS) is a component of ThruPut Manager that provides functions to grant users independence from the need to make requests to Operations staff.

Dependent Job Control (DJC)

Dependent Job Control allows you to control the order of execution of jobs that depend on the completion of other jobs. You can use JECL to organize jobs that depend on each other into DJC Groups, and control the order of execution of jobs within a DJC Group. A DJC dialog under ISPF allows you to display and modify DJC Groups, DJC Events, and jobs in DJC Groups.

Job Timing Services (JTS)

Job Timing Services provides a way to control when a job is made available for JES2 job selection. A simple JECL statement provides a way for you specify the "hold until" date and time at job submission. (A JTS dialog under ISPF lets you display your JTS jobs and, if necessary, remove the hold. You can also use the dialog to apply a JTS hold to jobs you have already submitted.)

User Hold Services (UHS)

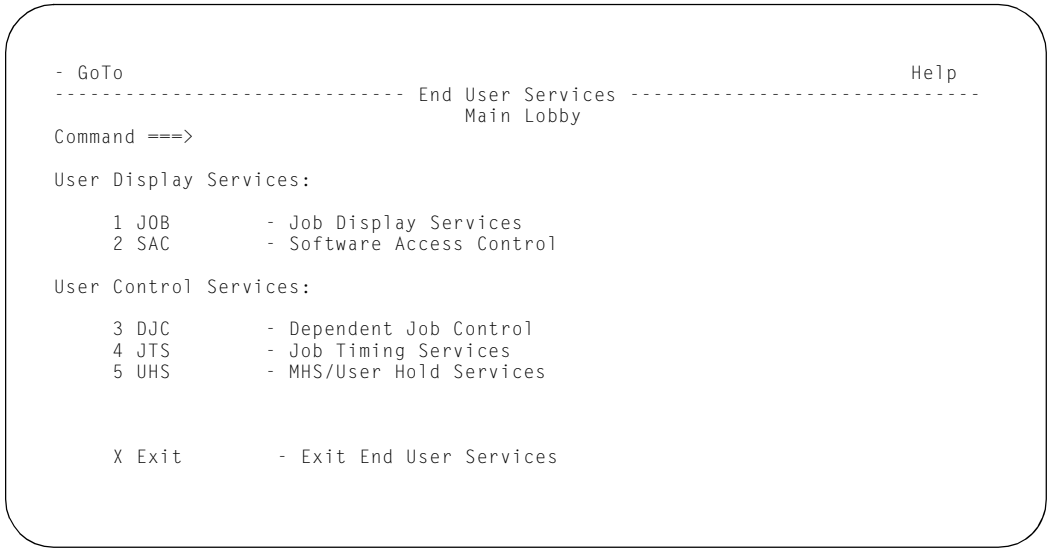
User Hold Services is an extension of ThruPut Manager Multi-hold Services (MHS). To use UHS, you include a JECL statement to request that your job be held at job submis- sion. A major benefit of UHS is the ability to add notes when you apply your hold. (To release jobs that you have held using UHS, you use the UHS dialog under ISPF. You can also use this dialog as an alternate way to apply a UHS hold to jobs that are awaiting execution.)

How do you invoke UCS?

The UCS is a feature of End User Services (EUS), which runs under ISPF. In order to use this facility, the appropriate ISPF datasets must be concatenated during the INSTALL process. You can invoke EUS through a TSO command:

```
TMEUS
```

This command opens the EUS Main Lobby, as illustrated below.



Chapter 2
Dependent Job Control (DJC)

This chapter describes how to use Dependent Job Control to manage jobs that depend on the completion of other jobs.

What is Dependent Job Control (DJC)?

Dependent Job Control is a UCS facility that allows you to manage groups of jobs that must execute in a pre-determined order. DJC definition and control facilities are designed to simplify the process of breaking multi-step jobs into dependent jobs. DJC provides:

- JECL control statements to associate jobs with each other and signal events such as end of job or step.
- UCS dialogs that allow you to alter DJC information for jobs and manage groups of jobs or the events on which they depend.

DJC Groups

Dependent Job Control allows job submitters to group jobs by using the concept of a ***DJC Group***. Each job must explicitly request membership. A job associates itself with a DJC Group through a DJC control statement that is included with the job's JCL. DJC Groups:

- have an arbitrary two level name chosen by the user, e.g. PAYROLL.DAILY.
- have some characteristics, such as the duration, associated with them.
- are used to logically "park" all the information about job relationships.
- are dynamically created and removed, that is, they are not predefined.
- can have an optional password.

Your JAL can interrogate the characteristics and name of the group, allowing your installation to monitor and enforce usage conventions.

Job Relationships

With DJC, relationships are always defined as "what must happen before this job can run". For example, assume you have a simple relationship: JOBX must run after JOBA has completed. With DJC, JOBA (the predecessor) does not need to be aware of JOBX (its successor). The only requirement for JOBA is to request membership in a DJC Group. JOBX then requests membership in the same DJC Group and indicates that it depends on JOBA. This means that the jobs can arrive in any sequence.

How It Works

DJC uses the ThruPut Manager Control File to store and manage the information for DJC Groups.

When a job is submitted to the system, ThruPut Manager determines whether or not DJC is to manage it. The minimum requirement for a job to be DJC managed is a request for membership in a DJC Group. When a job requests membership, DJC takes over the scheduling of that job and its relationships to other jobs.

First, if the installation chooses, the DJC Group name is validated through JAL. Next, the job is assigned to a DJC Group. If a Group does not exist, one is created. If a Group already exists, verification takes place to ensure that the job has not been resubmitted accidentally. Then the job is added to the DJC group.

A job, which is a member of a DJC group, is allowed to send or receive signals to other jobs that belong to the same Group. Signals between Groups are also allowed, but must be requested explicitly. In the context of DJC, a signal is a notification that a particular job has reached a stage that might be of interest to other jobs that are waiting to run. The implication here is twofold:

- DJC holds jobs that are awaiting a signal or signals.
- Signals are used either to release jobs so they can be selected for execution, or to flush jobs because they are no longer needed. The corollary here is that once the DJC hold is removed from a job, it cannot receive any further signals; however, the job is still a member of its Group unless it is flushed.

Any job that is a member of a DJC Group generates an automatic signal, either when it terminates or when it is flushed. This is an *implicit signal*. In addition to the automatic termination signal, once selected for execution a job can generate *explicit signals*:

- The DJC SIGNAL statement defines a DJC Event that other jobs can depend on. For example, a DJC Event can be associated with the end of a specified step, or when a particular message is generated, or just the fact that the DJC SIGNAL statement has been processed. When the DJC Event occurs, an explicit signal is sent. DJC Events are discussed in more detail later in this introduction.
- The DJC SIGNAL operator command can also be used to send an explicit signal associated with a DJC Event.

The GROUP STATEMENT

Every job in a DJC Group must have a GROUP statement as the first DJC JECL statement. A DJC Group is distinguished by a Group name, which is a 1-8 character name, optionally followed by a period (.) and a second 1-8 character name.

The GROUP statement specifies:

- The Group to which a job belongs.
- The duration of the Group.
- Whether this job is to signal the closing of the Group when it terminates.
- Optionally, a password for the Group. A Group can have these states:
  - Open,
  - Closed,
  - Completed,
  - Loading.

A Group is created when a job or DJC Event reference specifies a Group and there is no open Group with the same Group name.

An ***Open Group*** is one that accepts new jobs or new DJC Events. A new Group automatically becomes open when it is created.

A ***Closed Group*** no longer accepts new jobs or DJC Events from other Groups, but still has jobs that have not completed. Any new job or event referring to this Group creates a new instance of the Group that has a state of loading (see below).

A group can be closed in any of several ways:

- The completion of a job requesting a close-group signal in the GROUP statement.
- The end of the period specified in the cycle parameter for the GROUP statement.
- An operator or user command requesting that the Group be closed.

A ***Completed Group*** is a Group that is closed and has ceased all possible job activity. When a Group is closed, DJC checks the status of each job in the Group. If any job is executing or is awaiting execution and is not held by DJC, the Group remains closed but not completed. In essence, the Group is quiesced. The status of completed is reached when all jobs in the Group have either terminated, been flushed, or are being held by DJC. DJC then flushes all jobs in the Group that have not executed and the Group is marked completed.

The GROUP statement allows you to retain one or more generations of history for the completed Group. You can use the DJC DISPLAY command to examine these histories. The history display shows the dependencies and DJC Events associated with the Group, including which events were signaled.

A *Loading Group* is created when a new job or DJC Event uses the name of a Group that already exists and has a state of closed but not yet complete. A Group that is in a loading state accepts jobs and DJC Event relationships but the whole Group is in hold status until the previous version of the Group is complete.

The GROUP statement can contain a password for the Group. The rules governing the use of Group passwords are:

- A password can be assigned only by a DJC GROUP JECL statement.
- The rules for coding a password are the same as those for coding a job name.
- A password can be assigned to a Group when it is created or at any time thereafter. The password is assigned the first time a password is encountered for a specific Group.
- The password is retained until the Group is closed. Once a password is established, it cannot be removed or altered.
- After a password has been established, all jobs specifying that Group must provide the correct password. Failure to provide the password causes the job to fail:

```
DTM6720A DJC GROUP PASSWORD FAILURE
DTM1455I yyyy.ddd hh:mm:ss JOB jobname FAILED, JECL ERROR
```

- Group passwords do not apply to operator commands. If a password is mistyped or forgotten, operator commands must be used to manage the Group.
- You are prompted to provide the password when you attempt to reference the Group using the DJC dialog.

DJC JOBS

Any job that has a GROUP statement is a DJC job. If an open Group exists with the specified Group name, then the job becomes a member of the Group unless there is already a member job with the same name. In that case, the duplicate job is rejected. If no active Group exists when a job is submitted, then an active Group with that Group name is created.

DJC holds jobs in a Group only if they have dependencies that have not yet been accounted for, or if they are held through an operator command or through the use of the HOLD parameter of the GROUP statement. JECL is used to define the dependencies that the jobs have, either in the same Group or in a different Group. The DJC conditional statements are described in a section below.

DJC Events

A DJC Event is identified by a unique nine-character name that starts with a percent sign. An event must belong to a DJC Group. The event name must be unique within that Group.

Events are used to provide additional flexibility in the control of a DJC job. This is done by allowing jobs to specify event names in the condition clauses of the condition statements. The condition clauses can query the state of the event. Similar to the evaluation of conditions associated with job

termination, the condition referencing an event cannot be evaluated until the event has been signaled. The signaling of an event can be accomplished in several ways:

- The use of a DJC SIGNAL JECL statement that is executed at the start of the job or at the termination of a specified step.
- The use of a DJC SIGNAL JECL statement in a job that is triggered by an API associated with a message through a DJC MESSAGE JECL statement.
- A DJC SIGNAL operator or user command, which must specify the DJC Group associated with the signal.

In addition to signaling an event, a completion code can be associated with the event. The possible values of the completion code depend on the method used to signal the event. Once an event has been signaled the event remains "in effect" until the Group is completed. The implication is that subsequent jobs that reference this event have the references resolved as soon as these jobs are entered.

Events can be signaled more than once. The second and all subsequent signals can alter the completion value of an event. This does not affect any references that have already been resolved, but future references are resolved with the new value.

Every Group has one event automatically associated with it: the Close Group Event. The name for this event is %CLOSE. A Group can be closed by signaling this event.

Defining Dependencies and Conditions

JECL is used to define the dependencies that the job has on other jobs or events (either in the same Group or in a different Group).

A job can have the following DJC statements in its JECL:

```
/*DJC RUNIF ...

/*DJC FLUSHIF ...

/*DJC CONDIF ...
```

These statements define one or more conditions that DJC uses to determine the DJC status of a job. When all the conditions of a single RUNIF or FLUSHIF are satisfied, the job is allowed to run (released for execution by DJC) or is flushed (purged from the system and designated as flushed within the DJC Group).

There can be multiple RUNIF or FLUSHIF statements. They are treated as "OR" conditions. That is, if any of them is satisfied the job is released or flushed. In addition, a single RUNIF or FLUSHIF can have multiple conditions through use of the ANDIF JECL statement.

The CONDIF statement is provided to make it easier to convert multi-step jobs that use the COND parameter of an EXEC statement into a series of smaller jobs that use DJC and the CONDIF statement.

The jobs that are specified in the condition clauses of these JECL statements are referred to as predecessor jobs. When a predecessor job completes, DJC evaluates all the conditions that refer to this predecessor job and decides what, if any, actions need to be taken. Furthermore, subsequent jobs that are entered into the Group and have conditions relating to completed predecessor jobs have those conditions evaluated as soon as the job is entered.

Conditions can reference jobs that are in another Group. You must use caution when relating conditions across Groups, because their timing (cycle) might be rather different.

DJC Examples

Next, we use examples to illustrate some of the DJC facilities. These cases are not intended to be realistic because their purpose is to illustrate the actual facilities.

Example 1

A batch job called DBUPDATE runs daily. If it completes successfully, DBBACKUP then runs. From time to time, DBUPDATE terminates abnormally, so DBRELOAD is run, to revert to the status before DBUPDATE started to run.

We want DJC to manage these relationships. In this installation, the standard naming requirements for DJC Groups are userid.user-choice. Let's say these jobs run under userid DBGR01, so the following name is chosen for the Group:

```
DBGR01.DBCUST
```

The DJC statements for the jobs look like this:

```
//DBUPDATE JOB ......
//*+DJC GROUP DBGR01.DBCUST
…
//
//DBBACKUP JOB ......
//*+DJC GROUP DBGR01.DBCUST CLOSE
//*+DJC RUNIF DBUPDATE NORMAL
…
//
//DBRELOAD JOB ......
//*+DJC GROUP DBGR01.DBCUST CLOSE
//*+DJC RUNIF DBUPDATE FAILS
…
//
```

The jobs can be submitted in any order. The first job to arrive causes DJC to create the Group DBGR01.DBCUST, then the DJC information for that job is stored in that Group.

The only job that has no dependencies is DBUPDATE, so as far as DJC is concerned, it is allowed to run as soon as it arrives.

If it terminates normally, DBBACKUP is released. Because the parameter CLOSE was coded with the GROUP statement for this job, a signal to close the Group is generated at the end of this job. This causes the Group to be closed. As a result, DBRELOAD is evaluated and because no conditions can satisfy this job, it is flushed. Since the Group is now empty, it becomes completed.

A completed Group might or might not vanish, depending on the options chosen. As a default, it is kept so the history of the last time the Group was active can be examined.

Example 2

Using the previous example, assume that DBUPDATE can end up in one of four states:

- Normally, with a condition code of 0.
- Normally, with a condition code of 4.
- Abnormally, with a user abend.
- Abnormally, with a system abend.

Let's further assume, for the sake of illustration, that for each one of these conditions, a different job is to be run. The jobs are:

- DBDAILY after normal termination with a condition code 0 (daily run).
- DBWEEKLY after normal termination with a condition code of 4 (weekly run).

- DBPATCH after a user abend.
- DBRELOAD after a system abend. The jobs look like this:

```
//DBUPDATE JOB ......
//*+DJC GROUP DBGR01.DBCUST
...
//
//DBDAILY JOB ......
//*+DJC GROUP DBGR01.DBCUST CLOSE
//*+DJC RUNIF DBUPDATE CODE EQ 0
...
//
//DBWEEKLY JOB ......
//*+DJC GROUP DBGR01.DBCUST CLOSE
//*+DJC RUNIF DBUPDATE CODE EQ 4
...
//
//DBPATCH JOB ......
//*+DJC GROUP DBGR01.DBCUST CLOSE
//*+DJC RUNIF DBUPDATE USER_ABEND
...
//
//DBRELOAD JOB ......
//*+DJC GROUP DBGR01.DBCUST CLOSE
//*+DJC RUNIF DBUPDATE SYSTEM_ABEND
...
//
```

Example 3

In this case, we have the following:

- JOBA and JOBB, which have no dependencies.
- JOBC depends on the successful completion of JOBA and JOBB. Once this job executes the Group is no longer needed.
- If JOBA runs into problems JOBFIX1 is to run, otherwise this job is not needed.
- If JOBB runs into problems JOBFIX2 is to run, otherwise it is not needed.

This is an example of the use of a RUNIF with two conditions to be satisfied. Both JOBA and JOBB must complete before JOBC is to run. Here we combine the RUNIF with the ANDIF statement.

We also show the FLUSHIF statement with the recovery jobs once they are not needed.

The DJC statements look like this:

```
//JOBA JOB ......
//*+DJC GROUP Z5999.GROUP1
...
//
//JOBB JOB ......
//*+DJC GROUP Z5999.GROUP1
...
//
//JOBC JOB ......
//*+DJC GROUP Z5999.GROUP1 CLOSE
//*+DJC RUNIF JOBA NORMAL
//*+DJC ANDIF JOBB NORMAL
...
//
//JOBFIX1 JOB ......
//*+DJC GROUP Z5999.GROUP1
//*+DJC RUNIF JOBA ABENDS
//*+DJC FLUSHIF JOBA NORMAL
...
```

```
//
//JOBFIX2 JOB ......
//*+DJC GROUP Z5999.GROUP1
//*+DJC RUNIF JOBB FAILS
//*+DJC FLUSHIF JOBB NORMAL
...
//
```

Example 4

Here we illustrate the use of event signaling at the step level.

- Let's assume we have a multi-step job that runs for a considerable time.
- For a variety of reasons, it is not desirable to break the job into multiple jobs.
- We have two other jobs that depend on files created by this job.
  - One job, call it JOBSTEP5, depends on a file that the multi-step job, JOBLONG, creates but no longer needs after STEP05.
  - The other job, called JOBSTEP9, depends on another file that JOBLONG creates but does not need after STEP09.

With the step event technique, these two jobs do not have to wait until JOBLONG terminates and can take advantage of the parallel processing of production job streams.

The DJC statements look like this:

```
//JOBLONG JOB ......
//*+DJC GROUP LT021.GROUP3
//*+DJC SIGNAL %PRICE AT=STEP05
//*+DJC SIGNAL %PARTS AT=STEP09
...
//
//JOBSTEP5 JOB ......
//*+DJC GROUP LT021.GROUP3
//*+DJC RUNIF %PRICE
...
//
//JOBSTEP9 JOB ......
//*+DJC GROUP LT021.GROUP3
//*+DJC RUNIF %PARTS
...
//
```

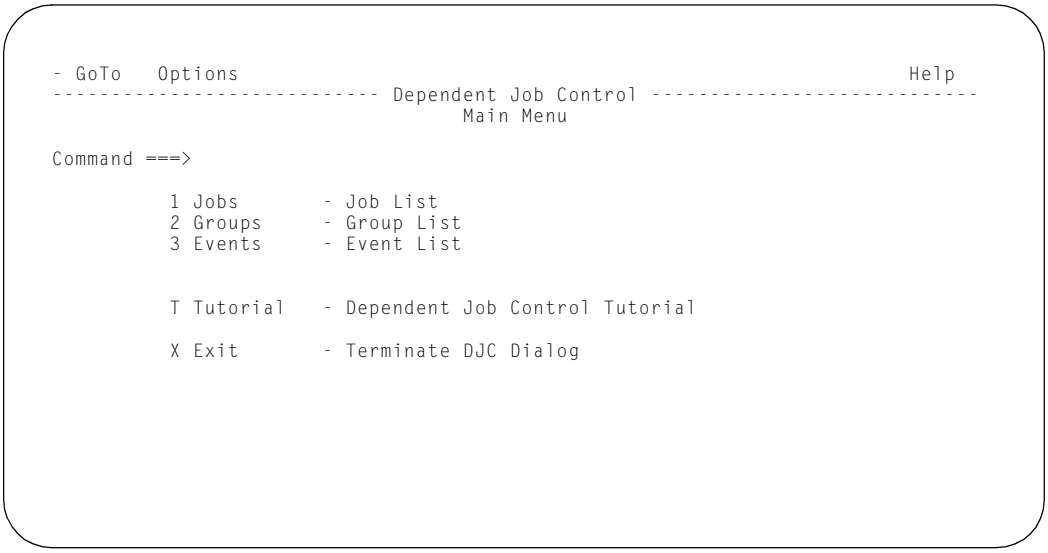
JECL for DJC

In order to take advantage of the services offered by DJC, you need a way to indicate to ThruPut Manager how to manage individual jobs. Requests for DJC services are specified through the use of ThruPut Manager JECL statements.

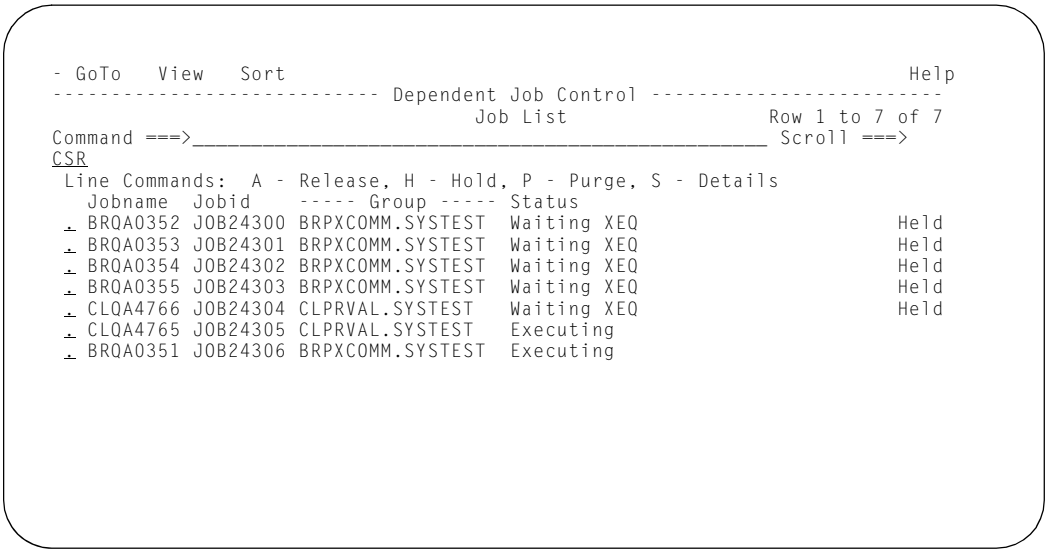
The ThruPut Manager JES2 control statements follow JES2 conventions regarding placement in the job stream and system processing. Their parsing, error messages and other system actions are all handled by ThruPut Manager. For syntax details of these JECL statements, refer to JECL Syntax for UCS.

UCS Dialog for DJC

The DJC dialog of User Control Services gives you the ability to monitor and control jobs managed by DJC, DJC Groups, and DJC Events. Selecting DJC from the Main Lobby opens the DJC Main Menu.

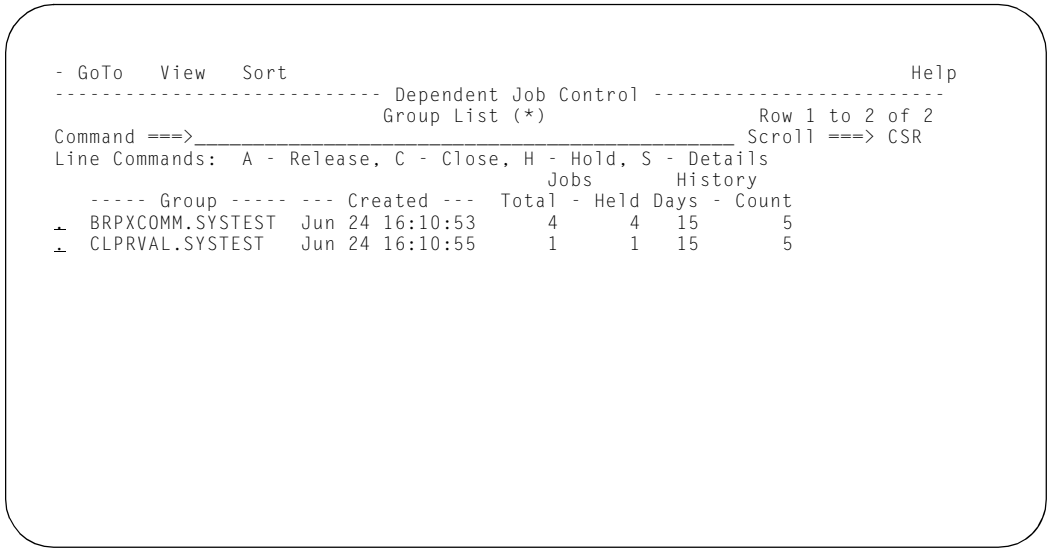


From this menu, you can select the type of DJC information to display or alter.



The Job List above lets you use line commands to display and manage jobs in DJC Groups. You can:

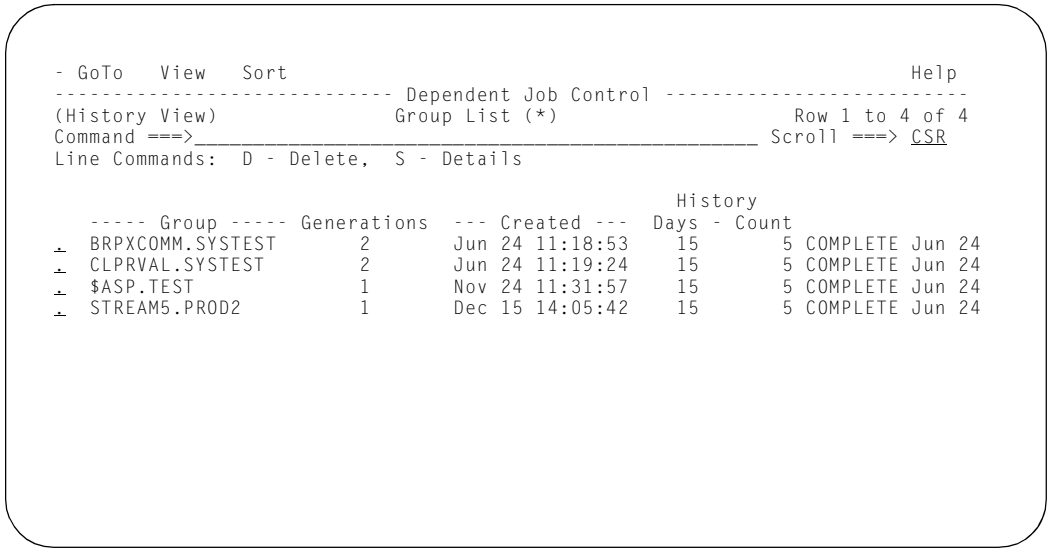
- Apply a DJC user hold.
- Remove a DJC user hold.
- Display additional DJC information about a job.
- Purge a job from DJC control.



The Group List above supports two modes, Active and History. Active mode lets you use line commands to display and manage DJC Groups. You can:

- Apply a DJC Group hold, which applies a user hold to all jobs in a DJC Group.
- Remove a DJC Group hold, which removes the user hold from all jobs within a DJC Group that were previously held by a DJC Group hold. Note that a DJC hold is independent of other ThruPut Manager and JES2 holds, and that removing a DJC hold from your job does not necessarily mean that it can be selected for execution.
- Close an active Group, equivalent to a CLOSE signal from a job.
- Display additional information about DJC Groups.

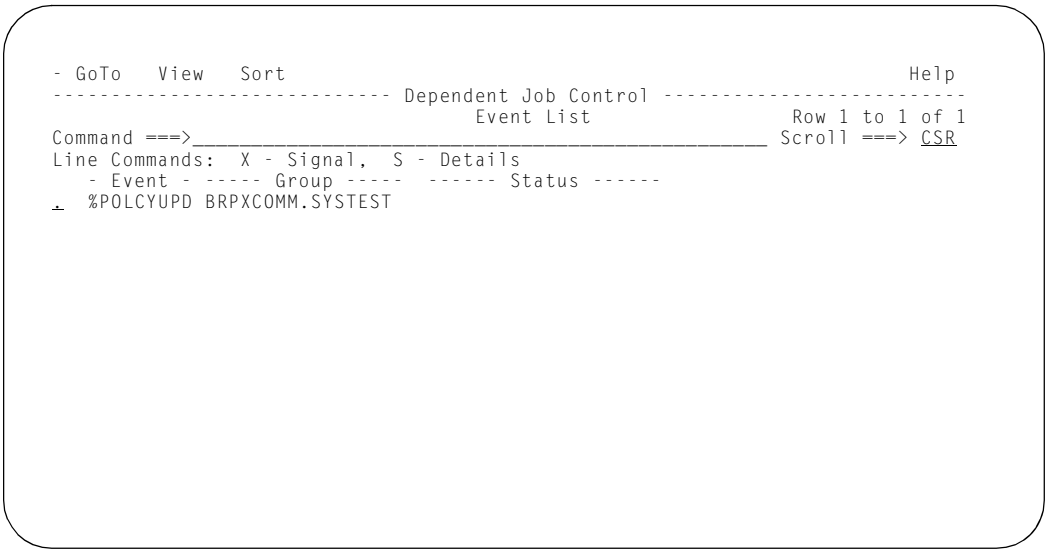
Refer to the online help for details about using the line commands.



You can select History mode above from the View drop-down menu. This lets you use line commands to display and manage DJC Group histories. You can:

- Delete a DJC Group's history information.
- Display additional information about DJC Groups.

Refer to the online help for details about using the line commands. For details about setting the number of generations of history that are kept for a DJC Group and the maximum number of days that history is retained, refer to the description of the /*DJC GROUP.



The Events List (above) lets you:

- Signal the completion of DJC Events.
- Display additional information about DJC Events.

For complete details about using these displays, use the online help.

Chapter 3
Job Timing Services (JTS)

This chapter describes how to use Job Timing Services to manage jobs with date and time dependencies.

What is Job Timing Services (JTS)?

JTS is a facility of UCS that allows you to manage jobs with date and time dependencies by using these facilities:

- A JECL Control Statement.
- A UCS dialog that allows you to change the JTS parameters or release the job.

JTS is not a job scheduler, and has no facilities for submitting jobs.

How It Works

If you have a job that has been submitted but must not run before a particular time, JTS provides the means to hold the job until the time arrives when you want it released, and then removes the hold automatically. You can submit the job in advance without the need to remember to remove it from hold when it is time to run.

To use JTS, you add the JECL statement /*JTS HOLD_UNTIL to your job, specifying the date and/or time until which the job is to be held. Upon submission, ThruPut Manager applies a JTS hold to the job that is not removed until the specified date and/or time.

You can use the UCS dialog for JTS to display information about your job, change the JTS parameters, or release the job from JTS control. Note that a JTS hold is independent of other ThruPut Manager and JES2 holds, and that removing a JTS hold from your job does not necessarily mean that it can be selected for execution.

A JTS Example

You are taking a vacation, but you have a job that you must run next Monday. JTS lets you submit the job before you leave:

```
//WEEKLY JOB .....
/*JTS HOLD_UNTIL MONDAY
...
```

This job will run first thing on Monday, right after midnight. Some maintenance must run first, however. This will be completed by Monday afternoon, so you can further qualify the release time:

```
//WEEKLY JOB .....
/*JTS HOLD_UNTIL MONDAY AFTERNOON
...
```

The AFTERNOON keyword causes JTS to remove the hold at the time your installation has determined the period AFTERNOON begins.

Assume that the job has to run next Friday, and the day you're submitting it is already Friday. Just use the alphabetic date FRIDAY, since JTS always assumes the next occurrence of the day of the week.

What if today is Friday and you're leaving at noon but need the job to run later today? Use the TODAY keyword and specify the time you want the job released:

```
//WEEKLY JOB .....
/*JTS HOLD_UNTIL TODAY AT=1800
...
```

JECL for JTS

JTS provides one JECL statement to allow you to specify when your job will be released. If JTS has not been activated, this statement is ignored. Refer to JECL Syntax for UCS for syntax details.

UCS Dialog for JTS

The JTS dialog below is entered from the EUS Main Lobby window. The jobs that are displayed by this dialog are determined by your settings for the Owner and Prefix filters.

```
- GoTo   View   Options   Display   Sort   Filter                        Help
------------------------------ Job Timing Services -------------------------------
                            Job List (JTS Managed)           Row 1 to 3 of 3
Command ===> ____________________________________________ Scroll ===> CSR
Sort=Jobname  Prefix=* Owner=USERID1  Confirm=On
  Jobname  Jobid    Owner    Queue      Class    Status
. BRQA0351 Job24314 USERID1 Analysis   D Held until 16:30 Tue Jun 24
. CLQA4765 Job24313 USERID1 Analysis   D Held until 16:45 Tue Jun 24
. SPLPURGE Job24278 USERID1 Analysis   D Held until 23:30 Tue Jun 24
****************************** Bottom of data ********************************
```

This dialog allows you to use line commands to modify the JTS settings for a job, or to remove the JTS hold for the job.

Note that if you release a job, it could still be held by other ThruPut Manager services, or by JES2. For details about using this dialog, use PF1 for the online help.

Chapter 4
User Hold Services (UHS)

This chapter describes how to use the User Hold Services to apply and remove a ThruPut Manager user hold for jobs.

What is User Hold Services (UHS)?

User Hold Services is an extension to the Multi-hold Services (MHS) of ThruPut Manager that gives you independence from the need to makes requests to Operations staff to hold and release your jobs. UHS allows you to apply and remove a ThruPut Manager User Hold for your jobs. UHS provides these facilities:

- A UCS dialog that allows you to apply, display, and remove User Holds.
- A JECL statement that you can use to request User Holds for your job.

How It Works

ThruPut Manager can apply and remove holds that are independent of JES2. There are a number of categories of ThruPut Manager holds, including one category called the *User Hold*.

User Holds are beneficial because you do not require an operator to manage them. Using UHS, you can apply, remove, and display User Holds:

- For jobs that are already submitted, you can use the UCS dialog for UHS.
- When submitting jobs, you can include the JECL statement /*MHS_USER HOLD in your JCL stream.

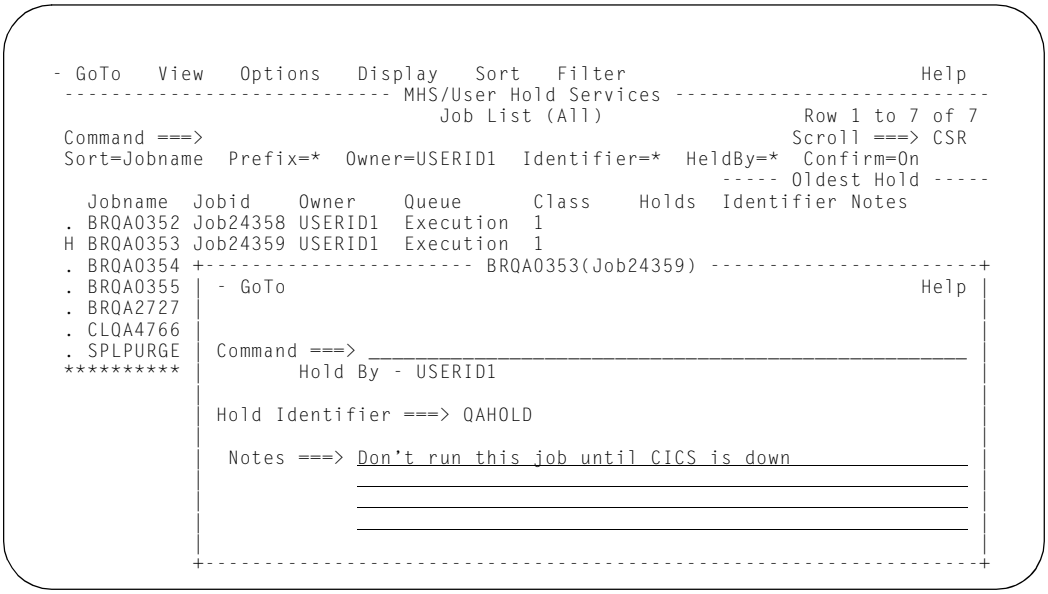
In addition to giving you the ability to apply your own User Hold, UHS has the benefit of letting you include notes that can document the hold. The note is displayed when you use the UHS dialog.

You can apply up to 24 different User Holds to a single job. Each User Hold requires an identifier that is used to distinguish between individual User Holds when more than one is applied.

Note that User Holds are independent of other ThruPut Manager and JES2 holds, and that releasing your job from User Hold does not necessarily mean that it can be selected for execution. Because it is truly a User Hold, there is no operator command to apply one, although if necessary an operator can remove a User Hold for you.

A UHS Example

You have just submitted a test job when you realize that you forgot to prepare the test data. Rather than purge the job, you can use the UHS dialog below to apply your own User Hold.



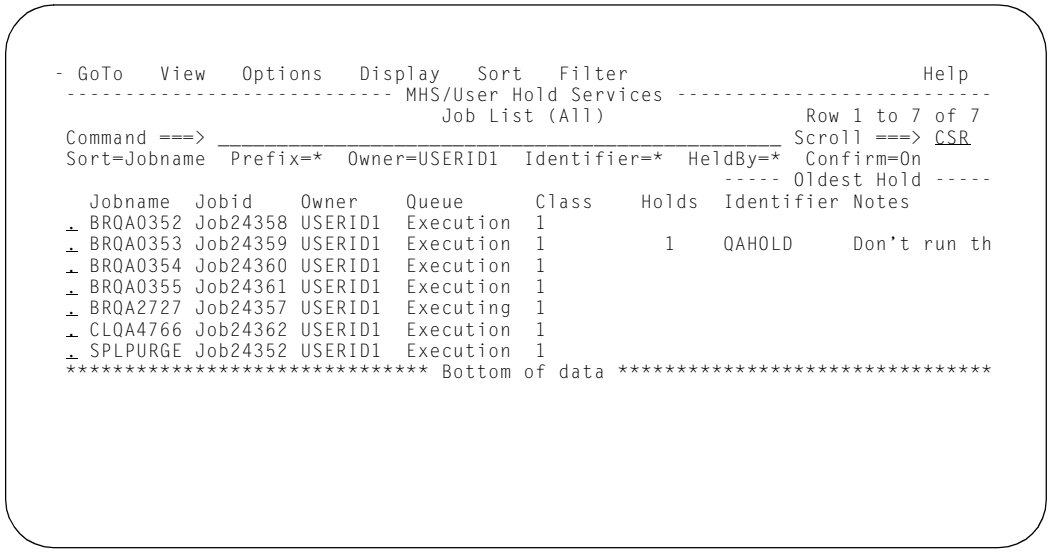
The H line command opens a dialog window that you use to identify your User Hold to distinguish it from other User Holds that you might apply to the same job. Because your data isn't ready, you add a reminder to yourself in the **Notes** field, for example **CP not ready**, as shown above. ThruPut Manager holds your job until you are ready, at which time you can release it yourself, and all without bothering Operations.

JECL for UHS

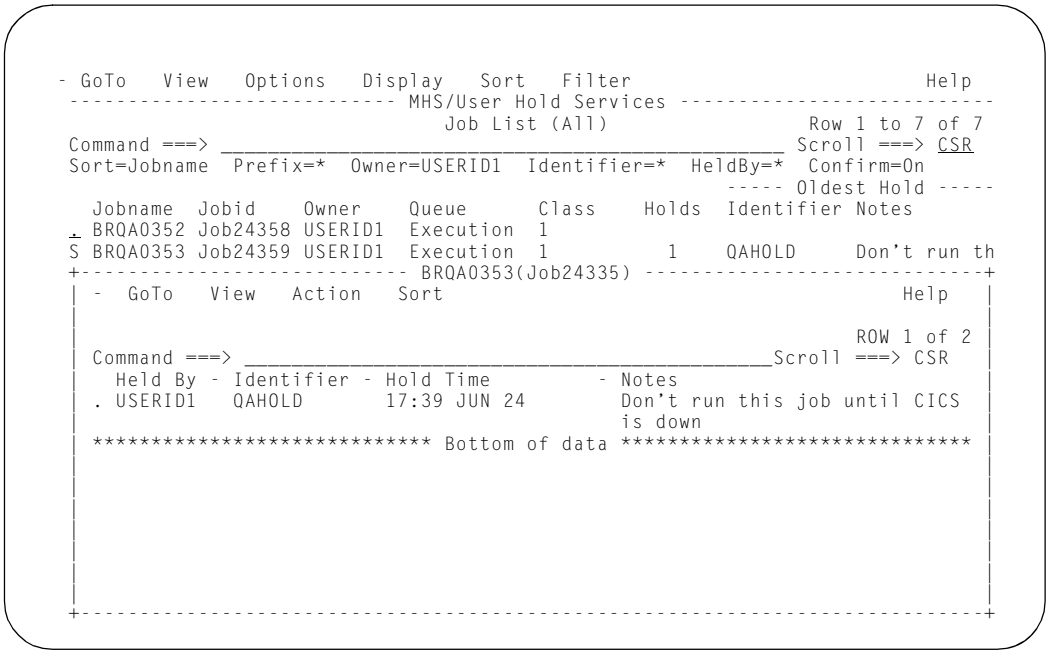
UHS provides a JECL statement to allow you to request a User Hold for your job. For syntax details, refer to JECL Syntax for UCS.

UCS Dialog for UHS

The UHS dialog below is entered from the EUS Main Lobby window. The jobs that are displayed by this dialog are determined by your settings for the Owner and Prefix filters.



This dialog allows you to use line commands to remove a User Hold from your job, or to apply User Holds (see below).



The S line command provides additional information about a job that has a User Hold, displaying the TSO userid of the person who applied the hold, and the date and time the hold was applied.

Multiple UHS holds

You can apply multiple User Holds to a job. The UHS dialog normally displays the oldest hold. If you use the A line command to remove a User Hold for a job with multiple User Holds, UHS releases the hold that is being displayed.

To remove any other hold, use the S line command to display all User Holds for the job. This results in a display that allows you to select the hold you want to remove.

For details about using this dialog, use PF1 for the online help.

Chapter 5
JECL Syntax for UCS

This chapter contains the syntax descriptions for the JECL provided to support UCS.

Considerations

The JECL control statements provided are unique to ThruPut Manager; therefore, they are not recognized in a system without ThruPut Manager, and result in errors that terminate the offending job. This could present you with problems in some cases. For example, if you are evaluating ThruPut Manager in a MAS complex you might install the product in only one LPAR. Unless you exercise tight control of your testing jobs, they could be read on one of the LPARs without ThruPut Manager. If this happens, they will be failed by JES2.

Provisions have been made to handle circumstances like this. The identification software for ThruPut Manager JECL statements allows for:

- Standard JES2 JECL, where statements begin with '/*'. Note that this format for ThruPut Manager JECL statements is processed regardless of placement within the job.
- Unique identification, where statements begin with '//*+'. In a system without ThruPut Manager, the statement is treated as a comment. This identification also lets you include ThruPut Manager JECL statements in procedures for batch jobs.

JECL Syntax Notation Conventions

The syntax of a ThruPut Manager JECL statement follows JES2 conventions. Note the following:

- Uppercase letters and words are coded on the control statement exactly as they appear in the format model, as are the following characters:

  comma ,

  parentheses ( )

- Lowercase letters, words, and symbols appearing in the format model represent variables for which specific information is substituted when the parameter is coded.

  For example:

  ```
  /*DJC ANDIF jobname...
  ```

  is part of the format description for the DJC ANDIF control statement. When you code the statement, substitute the job's name for the symbol jobname.

- The bar | is a special notation and is never coded on a control statement. The bar is used to separate elements in a list of choices from which only one can be selected.

  For example:

  ```
  CLOSE_TIME=nn | CT=nn
  ```

  indicates that you should choose one of the two forms for the CLOSE_TIME keyword on the /*DJC GROUP statement.

- Braces { } are a special notation and are never coded on a control statement. Braces are used to indicate a set of items from which one must be chosen.

  For example:

  ```
  COND=(nn,{GE | GT | EQ | LE | LT | NE})
  ```

  illustrates a keyword choice for the COND keyword of the /*DJC COND statement. The braces indicate that you must specify one and only one of the list of condition operators.

  Note that braces are used only when needed to clarify the syntax. If there is little chance for confusion, choices are separated by bars (|) only, as described above. The intention is to simplify syntax descriptions.

- An underlined choice indicates the value that is used as the default if no choice is specified.

  For example:

  ```
  GLOBAL | LOCAL
  ```

  If no keyword is specified, the default is LOCAL.

- Brackets [ ] are a special notation and are never coded on a control statement. Brackets indicate that the enclosed item is optional and you can omit it entirely.

  For example:

  ```
  [AT=stepname]
  ```

  is part of the format of the /*DJC SIGNAL control statement. When coding the /*DJC SIGNAL statement you can specify the AT keyword or omit it entirely.

- Keywords on ThruPut Manager JECL statements can be separated by either blanks or commas; however, we recommend the use of commas except where spaces provide superior clarity.

  For example:

  ```
  /*DJC GROUP TESTING,CLOSE_TIME=3,CLOSE
  ```

  is preferred, but:

  ```
  /*DJC RUNIF JOB001,CODE GE 4
  ```

  is more easily understood than:

  ```
  /*DJC RUNIF JOB001,CODE,GE,4
  ```

/*DJC ANDIF

Additional Conditions to be Satisfied

This statement allows you to specify additional conditions that must be satisfied for a RUNIF or FLUSHIF statement. It is a way to specify “and” conditions.

/*DJC ANDIF //*+DJC ANDIF	name [,GROUP=groupname] [,conditional-clause]

`name`

Indicates which job-related condition or event must be satisfied before DJC can release this job, and can be either:

– A jobname, or

– A DJC Event name, which consists of 2-9 alphanumeric or national characters, the first of which must be a per cent sign (%).

`GROUP=groupname`

Specifies that the dependency is to be satisfied by another DJC Group. If the Group does not already exist, it is created and the dependency is recorded.

`groupname`

Is a DJC Group name, which consists of 1-8 alphanumeric characters, the first of which must alphabetic or national, optionally followed by a period and a second level name, also consisting of 1-8 alphanumeric characters.

`conditional-clause`

This can be one of the following:

`CODE ComparisonOperator CompletionCode`

The comparison operators allowed are:

`GE, GT, EQ, LE, LT, NE`

The completion code must have the following format:

`nnnn`

Refers to a step completion code, or a completion code signaled by a DJC Event. `nnnn` represents a 1-4 digit number.

`CODE ComparisonOperator AbendCode`

The comparison operators allowed are:

`EQ, NE`

The abend code must have one of the following formats:

`Sxxx`

Refers to a system abend, such as SB37. `xxx` represents a three digit hexadecimal number. All the digits must be coded, e. g. S001.

`Unnnn`

Refers to a user abend code. `nnnn` represents a four digit number between 0-4095. All the digits must be coded, e.g. U0001.

`EVEN`

The job or DJC Event ends normally or abnormally, but does not fail on a runtime condition, and is not flushed.

`FAILS`

The job fails on a runtime condition, such as “unit not available.”<Body indent 2>

`FLUSH`

The referenced job was flushed from the DJC Group because the Group completed or because of a specific action.

`NORMAL`

The job or DJC Event ends without a system or user abend code, does not fail on a runtime condition, and is not flushed. This is the default.

`ONLY`

The job or DJC Event ends with a system or user abend code.

`SYSTEM_ABEND`

The job or DJC Event ends with a system abend code.

`USER_ABEND`

The job or DJC Event ends with a user abend code.

Examples:

```
//JOBC JOB ......
//*+DJC GROUP Z5999.whatever CLOSE
//*+DJC RUNIF JOBA NORMAL
//*+DJC ANDIF JOBB NORMAL
...
```

The job in this example will run on the successful completion of JOBA and JOBB. Once this job executes the Group is no longer needed.

See Also: /*DJC FLUSHIF and /*DJC RUNIF.

Notes: The ANDIF statement is associated with the most recent RUNIF or FLUSHIF statement. If there is no preceding RUNIF or FLUSHIF statement, the job is flushed with a JCL error. The ANDIF statement cannot follow a CONDIF.

/*DJC CONDIF

CONDIF Statement

This statement allows you to specify the condition or conditions that must be satisfied before this job is released by DJC. The CONDIF statement is patterned after the JCL COND parameter. It simplifies the process of breaking up multi-step jobs into independent jobs. It has capabilities similar to RUNIF and FLUSHIF.

/*DJC CONDIF //*+DJC CONDIF	name [,GROUP=groupname] [,COND=(nn,{GE | GT | EQ | LE | LT | NE}) | EVEN | ONLY]

`name`

Indicates which job-related conditions or event must be satisfied before DJC can release this job, and can be either:

- A jobname, or
- A DJC Event name, which consists of 2-9 alphanumeric or national characters, the first of which must be a per cent sign (%).

`GROUP=groupname`

Specifies that the dependency is to be satisfied by another DJC Group. If one does not already exist, it is created and the dependency is recorded.

`groupname`

Is a DJC Group name, which consists of 1-8 alphanumeric characters, the first of which must alphabetic or national, optionally followed by a period and a second level name, also consisting of 1-8 alphanumeric characters.

COND

The specification for running or flushing jobs follows the format of the JCL COND code facility.

(nn,op)

The condition code that is being tested in the (nn,op) clause is the condition code that was returned from the job that terminated or from the event being posted.

If the condition is TRUE the job is flushed, if the condition is FALSE the job is released for execution.

Note that this is not the same behavior as a similar FLUSHIF statement. If the FLUSHIF condition is FALSE then nothing is altered with respect to the state of the job.

EVEN

The job containing the statement is released when the designated job finishes or when the designated event is signaled.

ONLY

The job containing the statement is released if the designated job is terminated abnormally, else the job is flushed.

Examples:

```
/*DJC CONDIF UPDATE01 COND=(0,NE)
```

In this example, the job containing the CONDIF statement runs if COND=(0,NE), i.e. the highest return code from UPDATE01 is equal to 0.

Notes: When converting a COND keyword that refers to a specific step, you must ensure that the referenced step is converted to a single step job. This is necessary because you cannot refer back to a specific step within the referenced job.

/*DJC FLUSHIF

When to Flush Dependent Jobs

The FLUSHIF statement defines dependencies that the job has on other jobs or events in the DJC Group. A dependency can also be on a job/event in another Group. The job can contain many FLUSHIF statements. The first (possibly only) statement for which the condition (or conditions) is true will be used to determine the disposition of the job (in the context of the DJC Group).

A single FLUSHIF can have as many conditions as you wish. Multiple conditions are specified by following the FLUSHIF with a DJC ANDIF statement.

/*DJC FLUSHIF //*+DJC FLUSHIF	name [,GROUP=groupname] [,conditional-clause]

name

Indicates which job-related conditions or event must be satisfied before DJC flushes this job, and can be either:

– A jobname, or

– A DJC Event name, which consists of 2-9 alphanumeric or national characters, the first of which must be a per cent sign (%).

`GROUP=groupname`

Specifies that the dependency is to be satisfied by another DJC Group. If one does not already exist, it is created and the dependency is recorded.

`groupname`

Is a DJC Group name, which consists of 1-8 alphanumeric characters, the first of which must alphabetic or national, optionally followed by a period and a second level name, also consisting of 1-8 alphanumeric characters.

`conditional-clause`

This can be one of the following:

`CODE ComparisonOperator CompletionCode`

The comparison operators allowed are:

`GE, GT, EQ, LE, LT, NE`

The completion code must have the following format:

`nnnn`

Refers to a step completion code, or a completion code signaled by a DJC Event. `nnnn` represents a 1-4 digit decimal number between 0-4095.

`CODE ComparisonOperator AbendCode`

The comparison operators allowed are:

`EQ, NE`

The abend code must have one of the following formats:

`Sxxx`

Refers to a system abend, such as SB37. xxx represents a three digit hexadecimal number. All the digits must be coded, e. g. S001.

`Unnnn`

Refers to a user abend code. `nnnn` represents a four digit number between 0-4095. All the digits must be coded, e.g. U0001.

`EVEN`

The job or DJC Event ends normally or abnormally, but does not fail on a runtime condition, and is not flushed.

`FAILS`

The job fails on a runtime condition, such as “unit not available.”<Body indent 2>

`FLUSH`

The referenced job was flushed from the DJC Group because the Group completed or because of a specific action.

`NORMAL`

The job or DJC Event ends without a system or user abend code, does not fail on a runtime condition, and is not flushed. This is the default.

`ONLY`

The job or DJC Event ends with a system or user abend code.

`SYSTEM_ABEND`

The job or DJC Event ends with a system abend code.

`USER_ABEND`

The job or DJC Event ends with a user abend code.

Examples:

```
/*DJC FLUSHIF JOB0001,CODE EQ 0
```

If job JOB0001 (in the same DJC Group) terminates with a return code of 0, this job is flushed.

```
/*DJC FLUSHIF %NOFIX,GROUP=PROGR01.DAILY
```

If the event %NOFIX is signaled from group PROGR01.DAILY, flush this job.

See Also: /*DJC ANDIF.

Notes:

If a job contains only FLUSHIF statements, it is not held.

If the conditions under which the job is to be flushed occur before the system selects the job for execution, then it is flushed.

/*DJC GROUP

Associate a Job with a DJC Group

This statement associates a job with a DJC Group. If the Group does not exist, it is created.

/*DJC GROUP //*+DJC GROUP	groupname [,CLOSE] [,CLOSE_TIME=nn | CT=nn] [,FORCE] [,HISTORY_DAYS=nnn | HD=nnn] [,HISTORY_COUNT=n | HC=n] [,HOLD] [,OUTPUT_IF_NOT_RUN | OUTPUT | PURGE_IF_NOT_RUN | PURGE] [,PASSWORD=password]

`groupname`

Is a DJC Group name, which consists of 1-8 alphanumeric characters, the first of which must alphabetic or national, optionally followed by a period and a second level name, also consisting of 1-8 alphanumeric characters.

`CLOSE`

Requests that the CLOSE Event be signaled at job termination, whether normally or abnormally.

`CLOSE_TIME=nn | CT=nn`

Specifies the maximum number of hours before the DJC Group closes automatically.

`nn`

The range 1 to 99.

The short form for this keyword is CT.

`FORCE`

Specifies that this job should be accepted into the current active (NOT CLOSED) group *even if a job of the same name has already executed*. The job that it is replacing cannot be executing or waiting for execution.

`HISTORY_DAYS=nnn | HD=nnn`

Specifies the maximum number of days that the Group history is kept. The default is the installation value from the DJC SET operator command or the DJC initialization statement.

`nnn`

The range is 1 to 365.

The short form for this keyword is HD.

`HISTORY_COUNT=n | HC=n`

Specifies the maximum number of generations of Group history to be kept. The default is the installation value from the DJC SET operator command or the DJC initialization statement.

`n`

The range 1 to 9.

The short form for this keyword is HC.

`HOLD`

Specifies that the DJC Group will be held by DJC regardless of dependencies. The Group must subsequently be released with a DJC RELEASE GROUP operator command.

`OUTPUT_IF_NOT_RUN | OUTPUT`

Indicates that jobs flushed by DJC are to be placed on the JES2 output queue. This keyword is mutually exclusive with PURGE_IF_NOT_RUN, and is the default.

The short form for this keyword is OUTPUT.

`PASSWORD=password`

Specifies a Group password.

`password`

1-8 characters, following the same rules that apply to coding a job name.

`PURGE_IF_NOT_RUN | PURGE`

Indicates that jobs flushed by DJC are to be placed on the JES2 purge queue. This keyword is mutually exclusive with OUTPUT_IF_NOT_RUN.

The short form for this keyword is PURGE.

Examples:

```
/*DJC GROUP DEV505.BACKUP,CLOSE_TIME=8
```

This statement indicates that the job is requesting membership in a DJC group called DEV505. BACKUP. The group is to be closed after 8 hours.

Notes:

The value from the last job processed is used if more than one job for a particular Group specifies any of:

```
CLOSE_TIME
HISTORY_DAYS
HISTORY_COUNT
OUTPUT_IF_NOT_RUN
PURGE_IF_NOT_RUN
```

A Group can be closed before CLOSE_TIME expires by a DJC Event or an operator or user command.

HISTORY_DAYS and HISTORY_COUNT are maximums. When either maximum is exceeded, the oldest generation of the Group's history is deleted.

A Group password can be specified at any time by any arriving job, but once a password has been specified, all subsequent jobs specifying the same Group name must also provide the correct password. Failure to do so results in the job failing with a JECL error.

Group passwords cannot be altered or removed.

Group passwords must be specified when using the EUS DJC dialog, but do not apply to operator commands.

/*DJC MESSAGE

DJC API Message Statement

This JECL statement allows jobs to describe a system or application message that is to trigger the signal of an event or events. There must be a SIGNAL statement that matches the API id specified. The message must occur within the address space running the job with the MESSAGE statement.

/*DJC MESSAGE //*+DJC MESSAGE	msgmask,API=id

`msgmask`

Represents a character string pattern. It can contain either the wildcard characters "*" or "?", and specific text within quotes. If your actual text contains quotes, then you express it as paired quotes.

* Represents 0 or more characters of any kind.

? Represents 1 character of any kind.

'text within quotes' represents the text that requires an exact match for the API request to be triggered.

`API=id`

"Connects" the message statement with the actual DJC SIGNAL statement that describes the event.

Examples:

```
/*DJC SIGNAL %DBGOING,API=10
/*DJC MESSAGE *'IST009A'*'STARTED'*,API=10
```

The above statement matches any message with IST009A and STARTED in its text. When a match occurs, the event %DBGOING is signaled because of the API number match.

```
/*DJC MESSAGE *'USERID='???'PROD'*,API=10
```

The above statement matches any message with USERID=cccPROD, where **c** represents any character.

```
/*DJC SIGNAL %CICSOFF,API=10
/*DJC MESSAGE *'IEA123I'*'CICSPROD'*,API=10
/*DJC MESSAGE *'IEA125I'*'CICSPROD'*,API=10
```

In the previous example, a match on either message triggers the signal of the event named %CICSOFF.

See Also: /*DJC SIGNAL.

Notes: You can associate multiple messages with one SIGNAL statement. They are connected with the API id number. You can also associate one message with multiple SIGNAL statements.

/*DJC RUNIF

When to Run Dependent Jobs

The RUNIF statement defines dependencies that the job has on other jobs or events in the DJC Group. A dependency can also be on a job or DJC Event in another Group. If the job contains multiple RUNIF statements, they are treated as OR conditions. in other words, the first (possibly only) statement for which the condition (or conditions) is true determines the disposition of the job in the context of the DJC Group.

A single RUNIF can have as many conditions as you wish. The multiple conditions are specified by following the RUNIF with a DJC ANDIF statement.

/*DJC RUNIF //*+DJC RUNIF	name [,GROUP=groupname] [,conditional-clause]

`name`

Indicates which job-related conditions or event must be satisfied before DJC can release this job, and can be:

- A jobname, or
- A DJC Event name, which consists of 2-9 alphanumeric or national characters, the first of which must be a per cent sign (%).

`GROUP=groupname`

Specifies that the dependency is to be satisfied by another DJC Group. If one does not already exist, it is created and the dependency is recorded.

`groupname`

Is a DJC Group name, which consists of 1-8 alphanumeric characters, the first of which must alphabetic or national, optionally followed by a period and a second level name, also consisting of 1-8 alphanumeric characters.

`conditional-clause`

This can be one of the following:

`CODE ComparisonOperator CompletionCode`

`GE, GT, EQ, LE, LT, NE`

The completion code must have the following format:

`nnnn`

Refers to a step completion code, or a completion code signaled by a DJC Event. `nnnn` represents a 1-4 digit decimal number between 0-4095.

```
CODE ComparisonOperator AbendCode
```

The comparison operators allowed are:

```
EQ, NE
```

The abend code must have one of the following formats:

```
Sxxx
```

Refers to a system abend, such as SB37. xxx represents a three digit hexadecimal number. All the digits must be coded, e. g. S001.

```
Unnnn
```

Refers to a user abend code. nnnn represents a four digit number between 0-4095. All the digits must be coded, e.g. U0001.

`EVEN`

The job or DJC Event ends normally or abnormally, but does not fail on a runtime condition, and is not flushed.

`FAILS`

The job fails on a runtime condition, such as “unit not available.”<Body indent 2>

`FLUSH`

The referenced job was flushed from the DJC Group because the Group completed or because of a specific action.

`NORMAL`

The job or DJC Event ends without a system or user abend code, does not fail on a runtime condition, and is not flushed. This is the default.

`ONLY`

The job or DJC Event ends with a system or user abend code.

`SYSTEM_ABEND`

The job or DJC Event ends with a system abend code.

`USER_ABEND`

The job or DJC Event ends with a user abend code.

Examples:

```
/*DJC RUNIF JOB0001,CODE LE 8
```

If job JOB0001 (in the same DJC Group) terminates with a maximum step return code less than or equal 8, this job is run.

```
/*DJC RUNIF %READY,GROUP=PROGR01.DAILY,CODE EQ 4
```

If the event %READY is signaled from Group PROGR01.DAILY with a condition code of 4, this job is run.

```
/*DJC RUNIF JOBXXXX,CODE EQ SB37
```

If JOBXXXX terminates with a system abend of B37, this job is run.

See Also: /*DJC ANDIF.

Notes:

If a job contains RUNIF statements it is not released for execution until at least one of the RUNIF statements is satisfied (i.e. the specified condition yields a result of TRUE).

A job is not held unless it contains at least one RUNIF (or CONDIF) statement or the job is held when it is introduced by some other means.

/*DJC SIGNAL

Request to Signal Event

This statement is used to signal a DJC Event. The event is always associated with the DJC Group of the job issuing the signal. The signal can be specified to occur at the beginning of the job, at a designated step end, or upon the occurrence of a specified message from the job.

When the event is signaled, it causes any statements that were associated with this event to be evaluated and acted upon.

If there is no AT or API keyword, the event is signaled *at the start of the job* and the CODE parameter always results in a normal condition code of zero.

/*DJC SIGNAL //*+DJC SIGNAL	name | %CLOSE [,AT=stepname | API=id] ,SET=[nnnn | STEPCODE | Sxxx | Unnnn | FLUSH | FAILS]

`name`

Represents a DJC Event name, which consists of 2-9 alphanumeric or national characters, the first of which must be a percent sign (%).

`%CLOSE`

The special DJC Event name %CLOSE is reserved. The first time %CLOSE is signalled, the DJC Group becomes closed, i.e. it no longer accepts new jobs or DJC Events from other Groups. You can set and test values for %CLOSE just like any other DJC Event. When %CLOSE is signalled as a result of a /*DJC GROUP statement or an operator command, it is given a value of 0.

`AT=stepname`

Indicates that the event is to be signaled at the completion of the named step.

`stepname`

The name of the step.

In cases where there are multiple steps with the same name, the form:

`stepname.procstepname`

can be coded with the keyword AT. Otherwise the first step that matches the name triggers the signal.

`API=id`

Indicates that the job intends to signal the event through the API interface.

`id`

This is a two digit identifier that connects the API request to a message described with the DJC MESSAGE JECL statement.

`SET=nnnn`

This keyword associates a value with the signal of the event.

nnnn

Represents a fixed integer from the range 0-4095.

SET=STEPCODE

When the SET keyword is coded with this parameter, if the step terminates normally then the signal is posted with a value equal to the condition code for the step.

If the step terminates abnormally, the signal is posted with a value of Sxxx or Unnnn equal to the value of the user or system abend code.

SET=Sxxx

When the SET keyword is coded with this parameter, the signal is posted with a value of Sxxx, exactly as though a system abend had taken place. Note that this does not necessarily indicate that there actually was a system abend.

Sxxx

xxx represents 3 hexadecimal characters. All the digits must be coded, e.g. S001.

SET=Unnnn

When the SET keyword is coded with this parameter, the signal is posted with a value of Unnnn, exactly as though a user abend had taken place. Note that this does not necessarily indicate that there actually was a user abend.

Unnnn

nnnn represents a four digit number between 0-4095. All the digits must be coded, e.g. U0001.

SET=FAILS

When the SET keyword is coded with this parameter, the signal is posted with a value of FAILS, exactly as though the job had failed. Note that this does not necessarily indicate that the job triggering the signal failed, but the signal is sent as if it had.

SET=FLUSH

When the SET keyword is coded with this parameter, the signal is posted with a value of FLUSH, exactly as though a FLUSH had taken place. Note that this does not necessarily indicate that the job triggering the signal was flushed, but the signal is sent as if it were.

Examples:

```
//JOBLONG JOB ......
//*+DJC GROUP whatever.whatever
//*+DJC SIGNAL %PRICE AT=STEP05
//*+DJC SIGNAL %PARTS AT=STEP09
...
```

This job will signal other jobs in the Group as soon as STEP05 and STEP09 are completed. Other jobs do not have to wait until JOBLONG terminates and can take advantage of the parallel processing of production job streams.

See Also: /*DJC MESSAGE.

Notes:

The same event can be signaled multiple times with different values. Every time the event is signaled pending dependencies are evaluated. That is, references that have already been resolved are not affected.

A normal completion automatically posts the signals NORMAL and EVEN.

An abnormal completion automatically posts the signals ONLY and EVEN.

/*JTS HOLD_UNTIL

JTS User Statement

This JECL statement allows job submitters to apply a JTS hold, and describe when the hold is to be removed.

/*JTS HOLD_UNTIL //*+JTS HOLD_UNTIL	[DATE=numeric-date	alphabetic-date] [[AT | TIME]=numeric-time	alphabetic-time]

DATE=numeric-date

This keyword allows you to specify the date on which the JTS hold is to be removed.

If DATE is not specified, TODAY is assumed.

numeric-date

To describe the actual date, you can use one of the following Julian date formats:

```
yyyy.dddd
yy.ddd
ddd
```

In these formats:

yyyy represents all the digits for a calendar year, for example 2001.

yy represents the last two digits of a calendar year. For example, for 2001, yy is 01.

ddd represents the day (from 1 to 365 or 366 depending on the year) including leading zeros.

alphabetic-date

Instead of the DATE keyword, you can use one of the following alphabetic keywords:

```
MONDAY | MON
TUESDAY | TUE
WEDNESDAY | WED
THURSDAY | THU
FRIDAY | FRI
SATURDAY | SAT
SUNDAY | SUN
TODAY
TODAY[+ddd]
```

The day that you specify represents the next occurrence of the day of the week. For example, if you specify FRIDAY and the job is submitted on a Friday, the job is delayed until next Friday.

TODAY

Indicates that the JTS hold is to be removed on the day it is submitted. TODAY begins one minute after midnight, therefore this form should be accompanied by a TIME keyword (see below).

TODAY+ddd

Specifies a relative day on which the JTS hold is to be removed.

ddd

Represents the number of days to be added to the date.

`AT=numeric-time`

`TIME=numeric-time`

This keyword allows you to specify the time that the JTS hold is to be removed.

If TIME or AT is not specified, a default value specified by your installation is used.

`numeric-time`

The valid numeric values that you can specify are:

`hh:mm | hhmm`

In these formats:

`hh` represents the hour as read from a 24-hour clock.

`mm` represents the minutes.

`alphabetic-time`

Instead of the TIME or AT keyword, you can use one of the following alphabetic keywords:

```
MORNING
AFTERNOON
TONIGHT | NIGHT
NOW+mmmm
```

`NOW+mmmm`

The actual time values of the above keywords (except NOW) are set by your installation.

`NOW+mmmm`

Indicates a relative number of minutes from the time of job submission. This keyword is mutually exclusive with any DATE keyword. `mmmm` represents the number of minutes (1-1440) to be added to the current time.

Examples:

```
/*JTS HOLD_UNTIL DATE=yyyy.ddd TIME=09:00
/*JTS HOLD_UNTIL MONDAY AT=08:30
//*+JTS HOLD_UNTIL MONDAY NIGHT
//*+JTS HOLD_UNTIL NOW+120
```

/*MHS_USER

Request a user hold

This JECL statement allows a job submitter to request that a ThruPut Manager user hold be applied to the job. Explanatory notes can be added to the user hold.

`/*MHS_USER HOLD` `//*+MHS_USER HOLD`	`ID=identifier[,NOTES='notes']`

`ID=identifier`

Specifies a unique identifier that is used to distinguish between multiple user holds.

`identifier`

1 to 8 alphabetic, numeric, or national characters that uniquely identify this hold. If more than one JECL statement specifies the same identifier, the last one encountered takes effect.

`NOTES='notes'`

Specifies additional information that can be used to document the hold.

`'notes'`

1 to 60 characters that will be displayed in addition to the hold identifier. Notes that do not include commas or blanks can omit the enclosing apostrophes.

Examples:

```
/*MHS_USER HOLD ID=POOL3,NOTES='UNTIL JOE READY'
/*MHS_USER HOLD ID=9202,NOTES='NEEDS APPROVAL'
```

Chapter 6
UCS Messages

This chapter documents the messages associated with UCS.

Message Description Format

ThruPut Manager messages use this format:

```
DTMnnnns TEXT
```

Where:

`DTM`

identifies this as a ThruPut Manager message.

`nnnn`

is a four digit message number.

`s`

is an action indicator as follows:

`A` Immediate action required.

`E` Eventual action required.

`I` Information message.

Messages are described in numeric sequence. The following information is provided for each message:

- An explanation of why the message occurred.
- The action that the system takes.
- The suggested operator and programmer actions.

Messages are shown in upper case, while data which can vary in the message is shown in lower case:

`DTM0008I TMSS TASK` *`task-name`* `WILL NOT SIGNAL SUCCESSFUL START`

In some messages, certain text might not appear under some circumstances. This text is shown in brackets:

`DTM0062I AUTHORITY INSUFFICIENT FOR` *`command-name`* `[keyword] COMMAND`

Messages

The following messages support UCS:

DTM6705A NET/GROUP JECL STATEMENTS ARE MUTUALLY EXCLUSIVE

Explanation: A job has included both the NET and GROUP statements.

System Action: The job is failed with a JCL error.

Operator Response: None.

Programmer Response: Remove one of the conflicting statements.

DTM6706A "GROUP'' JECL STATEMENT, DUPLICATE DJC JOBNAME

Explanation: A job was submitted for a DJC Group, but the Group already contains a job with the same jobname.

System Action: The job is failed with a JCL error.

Operator Response: None.

Programmer Response: Use a different jobname.

DTM6707A DJC JECL STATEMENT, NO GROUP STATEMENT

Explanation: A job included a DJC JECL statement, but did not include a /*DJC GROUP statement.

System Action: The job is failed with a JCL error.

Operator Response: None.

Programmer Response: Add a /*DJC GROUP JECL statement or remove all DJC JECL statements.

DTM6708A GROUP ID INVALID

Explanation: A DJC JECL statement has specified a DJC Group name with incorrect syntax.

System Action: The job is failed with a JCL error.

Operator Response: None.

Programmer Response: Correct the syntax for the DJC Group name.

DTM6709A DJC EXCESSIVE NUMBER OF GROUP JECL STATEMENTS

Explanation: A job contains more than one /*DJC GROUP JECL statements.

System Action: The job is failed with a JCL error.

Operator Response: None.

Programmer Response: Remove the extra /*DJC GROUP statement(s).

DTM6710A DJC GROUP MUST BE COMPLETED BEFORE NEW GROUP LOADED

Explanation: After DJC CLOSE was issued for a DJC Group that was loading, a job was submitted for that Group.

System Action: The job is failed with a JCL error.

Operator Response: None.

Programmer Response: Re-submit after the active DJC Group has completed. The loading DJC Group will have become active, allowing a new loading Group to be created.

DTM6713E DJC HOLD–*jobname* HAD 'RELEASE FORCE' NEGATED

Explanation: A DJCHOLDcommand was issued for a job that had previously been released from DJC hold by using the FORCE keyword.

System Action: The job is placed in DJC hold.

Operator Response: None.

Programmer Response: None.

DTM6714E DJC RELEASE–*jobnumber* NOT RELEASED–NOT USER HELD

Explanation: A DJCRELEASEcommand was issued for a job that was not held by a DJCHOLDcommand.

System Action: The command is ignored.

Operator Response: None.

Programmer Response: None.

DTM6715E DJC SET INVALID HISTORY COUNT

Explanation: A DJC SET command was entered, but the HISTORY_COUNT parameter was invalid.

System Action: The command is ignored

Operator Response: Re-enter correctly.

Programmer Response: None.

DTM6717E DJC *command-name* COMMAND IGNORED–SPECIFIED GROUP IS NOT ACTIVE

Explanation: The command was entered specifying the name of an existing DJC Group, but the Group is no longer active (is now a history Group).

System Action: The command is ignored.

Operator Response: None.

Programmer Response: None.

DTM6718I DJC RELEASE–*jobname* WAS NOT BEING HELD–COMMAND IGNORED

Explanation: A DJC RELEASE command was entered for a job that was not being held by DJC.

System Action: The command is ignored.

Operator Response: None.

Programmer Response: None.

DTM6719I DJC HOLD–*jobname* WAS ALREADY HELD–COMMAND IGNORED

Explanation: A DJC HOLD command was entered for a job that was already held by DJC.

System Action: The command is ignored.

Operator Response: None.

Programmer Response: None.

DTM6720A DJC GROUP PASSWORD FAILURE

Explanation: The Group specified by the job is password protected, but the job's DJC GROUP JECL statement did not provide a password, or the password provided was incorrect.

System Action: The job is failed with a JECL error.

Operator Response: None.

Programmer Response: Re-submit specifying the correct password.

DTM6722E DJC *command-name* FAILED, GROUP NOT ACTIVE

Explanation: The command specified a DJC Group that is not active.

System Action: The command is ignored.

Operator Response: Make sure the DJC Group name is entered correctly.

Programmer Response: None.

DTM6723E DJC *command-name* SYNTAX ERROR

Explanation: The command specified invalid syntax.

System Action: The command is ignored.

Operator Response: Re-enter with correct syntax.

Programmer Response: None.

DTM6725E DJC *command-name* FAILED, GROUP NOT FOUND

Explanation: The command specified a DJC Group that does not exist.

System Action: The command is ignored.

Operator Response: Make sure the DJC Group name is entered correctly.

Programmer Response: None.

DTM6726E DJC *command-name* COMMAND FAILED, GROUP STATEMENT MISSING

Explanation: The command was entered for a job that did not contain a DJC GROUP statement.

System Action: The command is ignored.

Operator Response: Specify a job belonging to a DJC Group.

Programmer Response: None.

DTM6727E DJC *command-name* COMMAND FAILED, EVENT NAME INVALID

Explanation: The command specified a DJC Event name that is invalid.

System Action: The command is ignored.

Operator Response: Re-enter with a valid DJC Event name.

Programmer Response: None.

DTM6728E DJC *command-name* COMMAND FAILED, EVENT NAME MISSING

Explanation: The command did not specify a DJC Event name, but one is required.

System Action: The command is ignored.

Operator Response: Re-enter with a valid DJC Event name.

Programmer Response: None.

DTM6729E DJC *command-name* COMMAND, GROUP NAME MISSING

Explanation: The command did not specify a DJC Group name, but one is required.

System Action: The command is ignored.

Operator Response: Re-enter with a valid DJC Group name.

Programmer Response: None.

DTM6730E DJC *command-name* COMMAND, GROUP NAME INVALID

Explanation: The command specified an invalid DJC Group name.

System Action: The command is ignored.

Operator Response: Re-enter with a valid DJC Group name.

Programmer Response: None.

DTM6731E DJC CLOSE COMMAND, INVALID KEYWORD

Explanation: The DJC CLOSE command was entered, but one of the keywords was invalid.

System Action: The command is ignored.

Operator Response: Re-enter with correct syntax.

Programmer Response: None.

DTM6734E DJC *command-name* INVALID KEEP VALUE

Explanation: The command specified an invalid value for the KEEP keyword.

System Action: The command is ignored.

Operator Response: Re-enter with a valid value for KEEP.

Programmer Response: None.

DTM6735E nn HISTORY GENERATIONS WERE PURGED

Explanation: A DJC DELETE command was entered, causing the indicated number of history generations to be purged.

System Action: Purges the oldest nn history generations.

Operator Response: None.

Programmer Response: None.

DTM6738E DJC command-name INVALID SET VALUE

Explanation: The command specified an invalid value for the SET keyword.

System Action: The command is ignored.

Operator Response: Re-enter with a valid value for SET.

Programmer Response: None.

DTM6739E DJC *command-name* INVALID HISTORY_DAYS

Explanation: The command specified an invalid value for the HISTORY_DAYS keyword.

System Action: The command is ignored.

Operator Response: Re-enter with a valid value for HISTORY_DAYS.

Programmer Response: None.

DTM7300I HOLD _ UNTIL JECL STATEMENT, REQUESTED A HISTORICAL DATE–yyyy.ddd<Mess _Num>

Explanation: The date indicated in JTS is past.

System Action: This message can either be followed by a warning or the is job failed, depending on the installation options.

Operator Response: None.

Programmer Response: If it resulted in a job failure, then correct the statement so that the date represents a future date.

DTM7301I HOLD_UNTIL JECL STATEMENT, REQUESTED A TIME EARLIER TODAY–*hh:mm*

Explanation: The time indicated in JTS is past.

System Action: None. This message is a warning.

Operator Response: None.

Programmer Response: None.

DTM7302I HOLD _ UNTIL JECL STATEMENT, REQUESTED A DATE MORE THAN nnn DAYS AHEAD–yyyy.dd

Explanation: The DATE indicated in JTS exceeds the maximum allowed.

System Action: This can be a warning, or the job can be failed, depending on installation options.

Operator Response: None.

Programmer Response: Correct the statement so that the date is within the value nnn.

DTM7330E JTS RESET COMMAND FAILED, DUPLICATE JOB NAME

Explanation: There are, at least, two jobs with the same name.

System Action: No further action is taken by the command processor.

Operator Response: Identify the job by job number.

Programmer Response: None.

DTM7331I JTS RESET COMMAND FAILED, JOB NOT ELIGIBLE

Explanation: The job is not awaiting execution.

System Action: No further action is taken by the command processor.

Operator Response: Verify that the job name or number was correct.

Programmer Response: None.

DTM7332E JTS RESET, INVALID DATE SYNTAX

Explanation: Incorrect DATE syntax, or the date specified is beyond the maximum number of days specified by MAXDAYS on the JTS OPTIONS initialization statement.

System Action: No further action is taken by the command processor.

Operator Response: Reenter with a valid DATE parameter.

Programmer Response: None.

DTM7333E JTS RESET, INVALID TIME SYNTAX

Explanation: Incorrect TIME syntax.

System Action: No further action is taken by the command processor.

Operator Response: Reenter with correct TIME parameter.

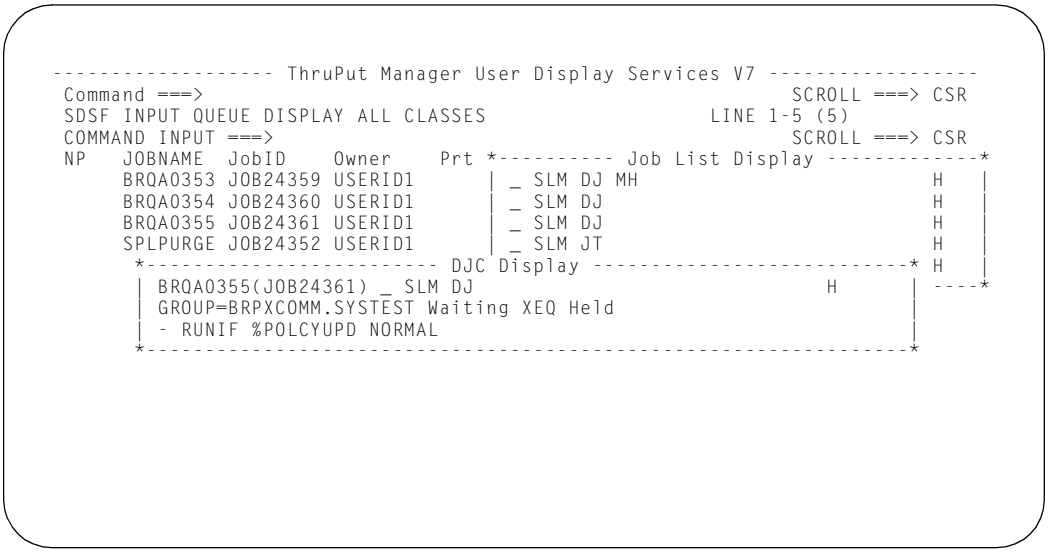
Programmer Response: None.

Appendix A.
UCS User Display Facility Support

This appendix documents the additional User Display Facilities screens that are provided to support UCS.

User Display Facility for DJC Jobs

The User Display Facility (UDF) can display information about jobs that are managed by DJC. The DJC Display Window shows the DJC Group a job belongs to and the dependencies that affect the job. Jobs that are being managed by DJC are shown in the UDF Job List Window with the DJ acronym in the job's Information Summary Line.



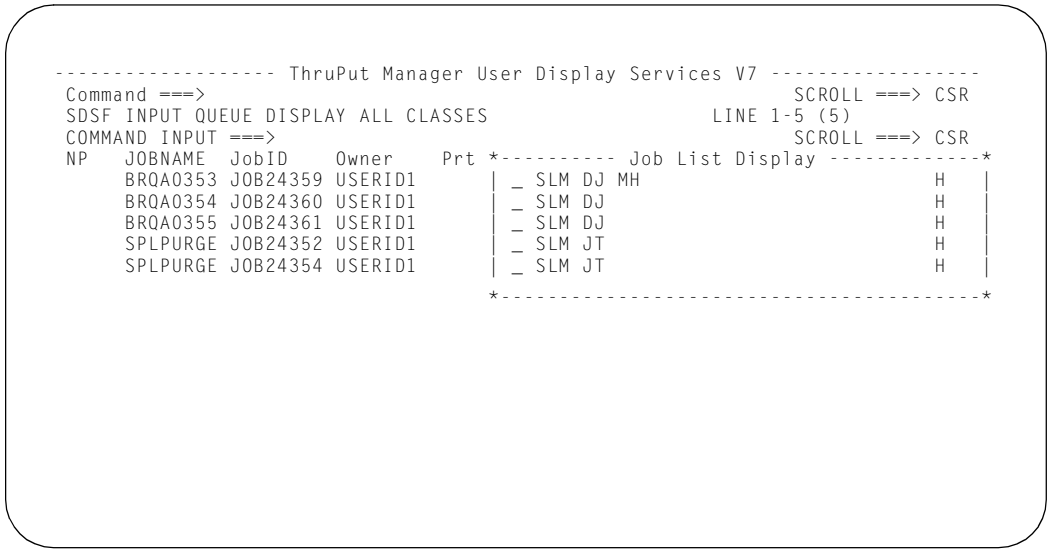
In the sample display above , job BMPFIX belongs to the DJC Group BMP.GROUP and has two dependencies:

- Job BMP202 must end with a system completion code of D37. In our example, this line is highlighted, indicating that the condition has not occurred and is therefore causing the job to be held.
- The signal %STEP5 must have been sent. In this case, the line is not highlighted indicating that this condition is not holding the job and therefore the signal has been sent. This is confirmed by the (T) status for the signal.

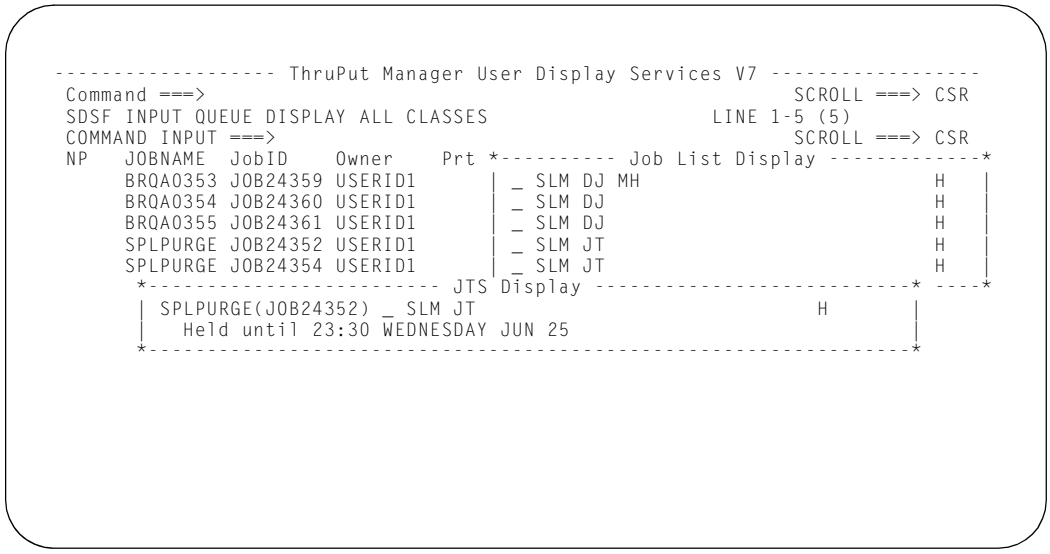
For more information about using these display facilities, use the PF1 key for online help.

User Display Facility for JTS

The ThruPut Manager User Display Facility (UDF) includes support for JTS. Jobs that are being managed by JTS are shown in the UDF Job List Window with JT in the job's Information Summary Line, as shown below.

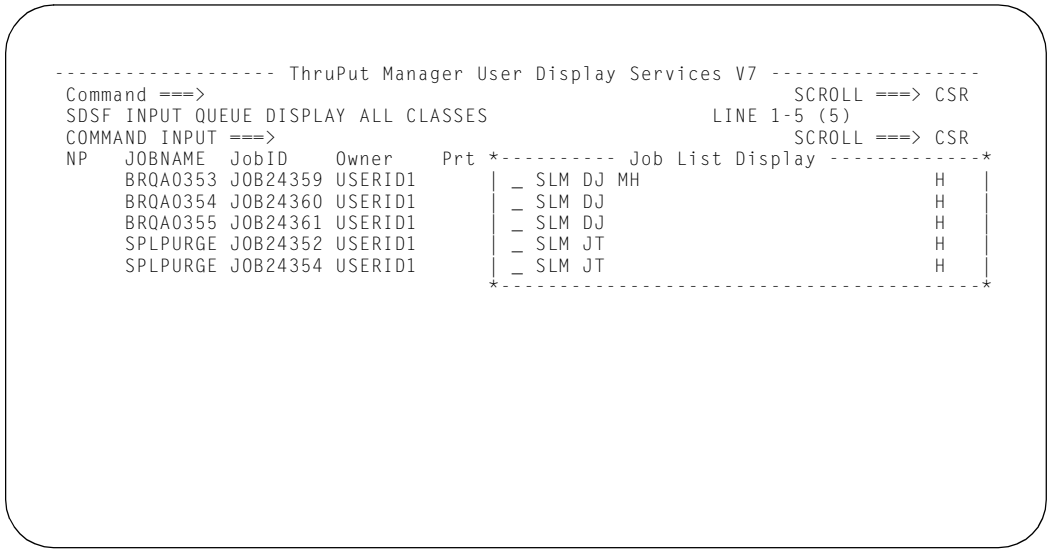


If you place the cursor on JT and press Enter, UDF presents the JTS Display Window, as shown below.

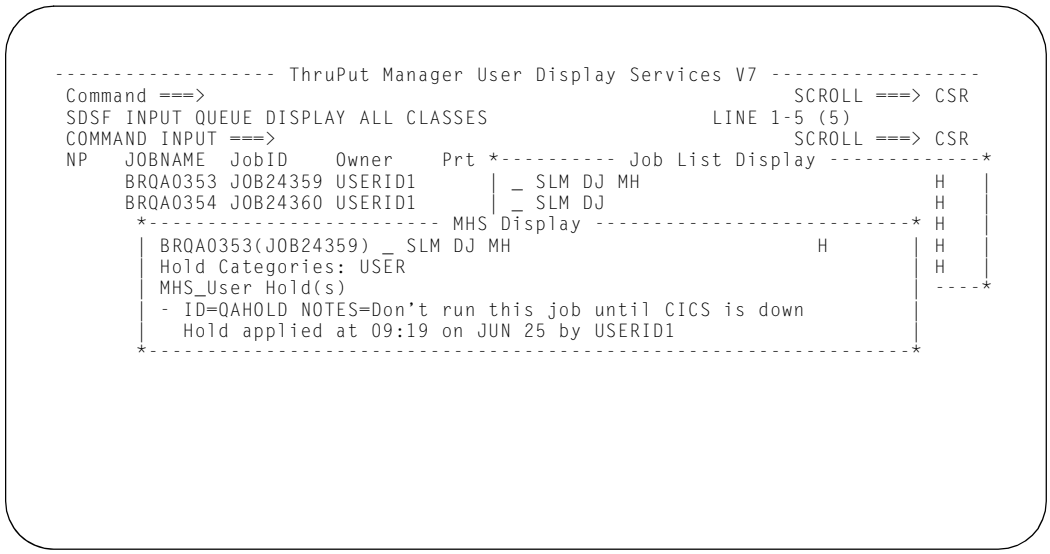


User Display Facility for UHS

The ThruPut Manager User Display Facility (UDF) includes support for UHS. Jobs that have a user hold applied are shown in the UDF Job List Window with MH in the job's Information Summary Line, as shown below.



Note that jobs that have MHS holds are indicated by MH. If you place the cursor on MH and press Enter, UDF presents the MHS Display Window, as shown below.



Appendix B.
JES2 Job Execution Control Comparison

This appendix is a comparison between JES2 Job Execution Control (JEC) and ThruPut Manager Dependent Job Control (DJC) and deadline scheduling capabilities.

Dependent Job Control and Deadline Scheduling

Since 1995 ThruPut Manager has controlled batch job dependencies using Dependent Job Control (DJC) and has provided deadline scheduling capabilities through Job Timing Services (JTS) and Job Chaining Services (JCS).

In 2015, with z/OS 2.2, IBM introduced JES2 Dependent Job Control, also referred to as Job Execution Control (JEC) and JES2 deadline scheduling.

Functionally, the dependent job controls provided by IBM z/OS and MVS Solutions ThruPut Manager are quite similar. In both cases, this is a facility that can be used to group a number of jobs and define dependencies between jobs that belong to the group.

This appendix highlights the substantive differences between the two approaches, under the following topics:

- Dynamic Group Definition
- Simple Dependencies
- Conditional Dependencies
- Treatment of Unsatisfied Dependencies
- Group Display Commands
- Scheduling Parameters: WITH, HOLDUNTIL, and STARTBY
- Group Affinity
- Concurrent Jobs

Refer to the respective User Guides for syntax and further details.

Dynamic Group Definition

ThruPut Manager

With ThruPut Manager, DJC groups are defined dynamically by JECL statements inserted in the JCL of the jobs that are in the group. The group is defined by the first reference to it by a participating job.

IBM

In contrast, the IBM JEC group or network is predefined by a new construct called a JobGroup. The JobGroup is submitted as a block of JCL that defines the jobs that belong to the group (distinguished by a user chosen JobGroup name) and the dependencies amongst the jobs. The block of JCL containing the definition consists of JES2 Execution Control Statements.

The JCL for the jobs in the group contains a SCHEDULE statement which ties them to the JobGroup by name. The network definition is static and must be ‘registered’ before any of the jobs are submitted.

Value Added with ThruPut Manager

The DJC group is dynamically defined where as the IBM JEC Group is not.

Simple Dependencies

ThruPut Manager

ThruPut Manager provides RUNIF and ANDIF statements. A BEFORE condition is expressed by inserting a RUNIF in the target job. ThruPut Manager uses the ANDIF statement when 'and-ing' is desired, and multiple RUNIF statements when 'or-ing' is desired.

The GROUP parameter is specified on RUNIF, ANDIF, FLUSHIF, and CONDIF, to refer to jobs from other groups.

IBM

IBM provides BEFORE and AFTER statements to define dependencies where a job must run either before or after another job or jobs in the same group. These statements are effectively 'and-ed'.

Dependencies are restricted to jobs within the group.

Value Added with ThruPut Manager

- Dependencies can be 'and-ed' and 'or-ed'.
- Cross-group dependencies are supported. The dependent jobs can be in a different group, (using the GROUP parameter on RUNIF, ANDIF, FLUSHIF, CONDIF statements).

Conditional Dependencies

Both DJC and JEC support conditional dependencies based on Job return codes. The codes can be system abend codes, user abend codes, or numerical return codes.

Value Added with ThruPut Manager

ThruPut Manager goes a bit further allowing the following capabilities:

- Conditional dependencies can refer to jobs in another DJC group.
- An EVENT can be created and SIGNALed via JECL or operator commands. The SIGNAL can also 'post' a return code to the EVENT. Conditional DJC statements can refer to the EVENT the same as they do to Jobs. Events can be referred to in other groups, so, as an example, a step in one DJC group could signal an EVENT causing a job (perhaps in another DJC group) to be started.
- An event may also be SIGNALed whenever a message with specified text is detected in the address space of the job containing the associated DJC MESSAGE statement.

Treatment of Unsatisfied Dependencies

ThruPut Manager

A job with dependencies is HELD in the execution queue until the dependencies are satisfied. Since ThruPut Manager analyzes jobs on submission, any errors are found early.

IBM

A job with dependencies is HELD in the SETUP queue until the dependencies are satisfied and then it is submitted. If it is also managed by ThruPut Manager, then analysis is delayed until the hold is resolved.

Value Added with ThruPut Manager

Problems with jobs with dependencies are exposed earlier and this allows them to be dealt with earlier.

Group Display Commands

ThruPut Manager

DJC DISPLAY GROUP command is used to display the group. The JOBS and DETAILS parameters display the conditions that were specified and whether they were True or False (assuming the condition has been set). For example:

```
/DJC D GROUP TST002 ALLJOBS DETAILS
DTM6420I DJC DISPLAY
Group TST002 Created Apr 13, 0 Jobs HD(15) HC(5)
 JOB05975 TTW0001 Ended RC(1) Apr 13 15:51:59
- has a RUNIF from TTW0004
- has a FLUSHIF from TTW0005
- has a RUNIF from TTW0006
- has a FLUSHIF from TTW0007
- has a FLUSHIF from TTW0009
- has a FLUSHIF from TTW0009
 JOB05976 TTW0004 Ended RC(4) Apr 13 15:52:03
- RUNIF TTW0001 NORMAL (T)
 JOB05977 TTW0005 Ended RC(5) Apr 13 15:51:56
- FLUSHIF TTW0001 NORMAL (T)
 JOB05978 TTW0006 Ended RC(6) Apr 13 15:52:03
- RUNIF TTW0001 CODE EQ 1 (T)
 JOB05979 TTW0007 Ended RC(7) Apr 13 15:51:58
- FLUSHIF TTW0001 CODE EQ 1 (T)
 JOB05980 TTW0008 Flushed (PREDECESSOR)
 JOB05981 TTW0009 Ended RC(9) Apr 13 15:51:58
- FLUSHIF TTW0001 CODE EQ 2 (F)
              - FLUSHIF TTW0001 CODE NE 2 (T)
```

IBM

The $DG command displays the group. The parameters JOB and JOBS are added to show the basic relationships between the jobs in the group. For example:

```
$DG5966,LONG,JOB,JOBS
$HASP890 JOB(TST002)
$HASP890 JOB(TST002)    JOB GROUP DEPENDENCY LIST VIA  JOBS
$HASP890                   PARENT        DEP JOB         DEP STAT      COMP ACT
$HASP890                  --------          --------          --------          -
--------
$HASP890                    TTW0001     TTW0009      COMPLETE    FLUSH
$HASP890                    TTW0001     TTW0008      COMPLETE    FLUSH
$HASP890                    TTW0001     TTW0007      COMPLETE    FLUSH
$HASP890                    TTW0001     TTW0006      COMPLETE    SATISFY
$HASP890                    TTW0001     TTW0005      COMPLETE    FLUSH
$HASP890                    TTW0001     TTW0004      COMPLETE    SATISFY
$HASP890                    JOB GROUP  JOB LIST
$HASP890                    JOB NAME    JOBID           JOB STAT        COMP STAT
$HASP890                   --------          --------          --------          -
--------
$HASP890                    TTW0009     JOB05973     Q=HARDCPY  FLUSHED
$HASP890                    TTW0008     JOB05972     Q=HARDCPY  FLUSHED
$HASP890                    TTW0007     JOB05971     Q=HARDCPY  FLUSHED
$HASP890                    TTW0006     JOB05970     Q=HARDCPY  COMPLETE
$HASP890                    TTW0005     JOB05969     Q=HARDCPY  FLUSHED
$HASP890                    TTW0004     JOB05968     Q=HARDCPY  COMPLETE
$HASP890                    TTW0001     JOB05967     Q=HARDCPY  COMPLETE
```

Value Added with ThruPut Manager

- Layout allows dependencies to be determined more easily.
- More details are included: true or false results for the conditions, return codes

Scheduling Parameters – WITH, HOLDUNTL, STARTBY

ThruPut Manager

ThruPut Manager provides WITH and (JTS) HOLD_UNTIL. Though it does not support the STARTBY feature, SLM provides methods to ensure that the job flow is as desired through other means.

IBM

The IBM SCHEDULE statement has the WITH, HOLDUNTL and STARTBY keywords.

Group Affinity

ThruPut Manager

With ThruPut Manager you set the affinity of each job in the group in the job's JCL or using JAL statements.

IBM

z/OS has a parameter to set the affinity at the JEC Group Level.

Concurrent Jobs

Concurrent jobs are supported by z/OS. ThruPut Manager does not have an analogous feature.

ThruPut Manager treats jobs with CONCURRENT as ThruPut Manager exempt; they are not analyzed and are not managed by SLM.