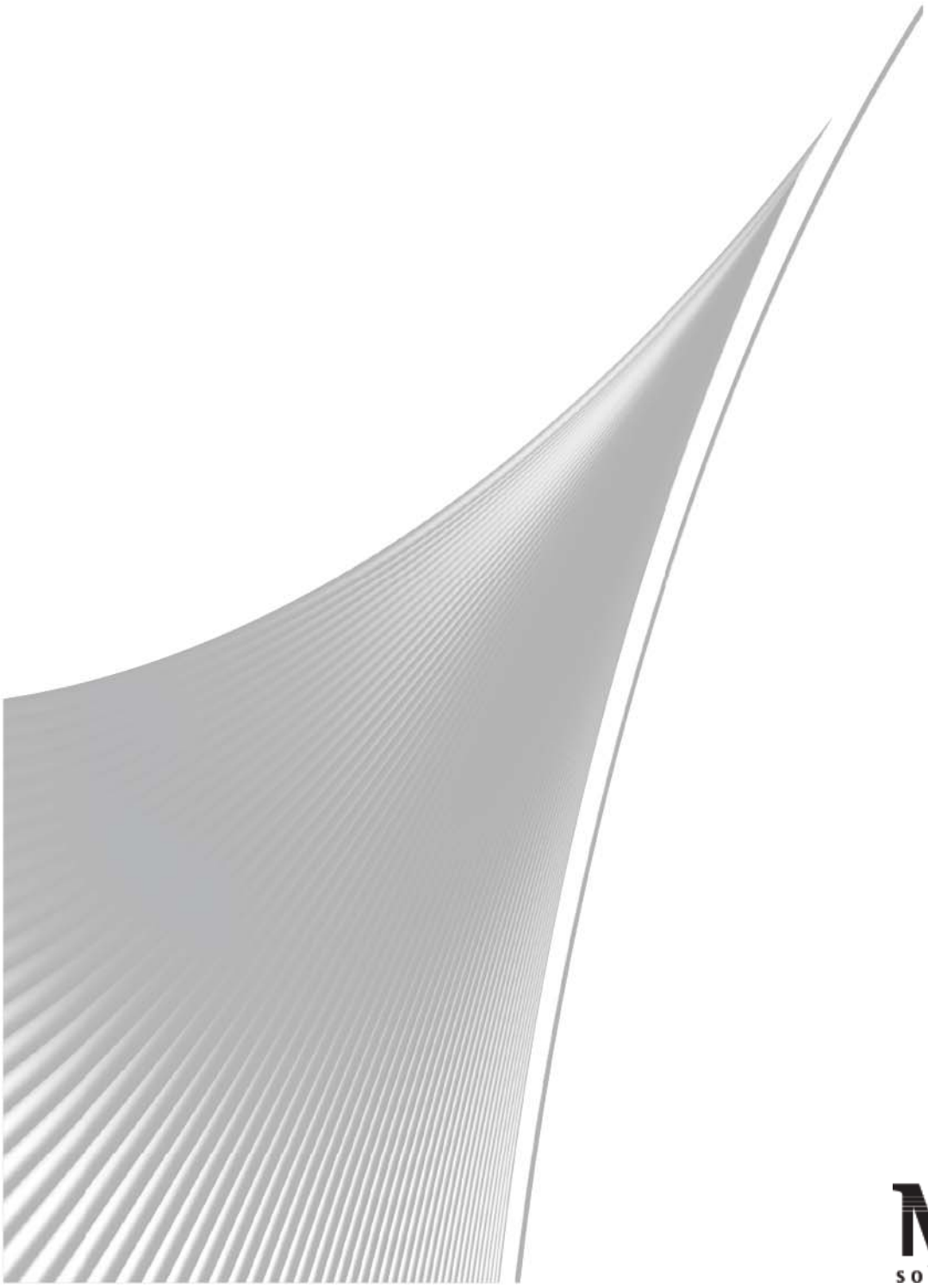
ThruPut Manager

Job Binding Services (JBS)

System Programming Guide



MVS solutions inc.

This manual applies to ThruPut Manager® Version 7 Release 1.0 PTF level 7109 and to all further releases until otherwise indicated. This material contains proprietary information on ThruPut Manager. Use of this manual is subject to the terms of the user's software license agreement for ThruPut Manager.

The complete list of manuals relevant to this release is available from the Customer Center on www.mvssol.com, from where any of the manuals may be downloaded.

ThruPut Manager is a registered trademark of MVS Solutions Inc.

TM/Service Level Manager, TM/Production Control Services, TM/Drive Booking Services, TM/Dataset Contention Services, TM/Job Binding Services, TM/User Control Services, and TM/JES3 Compatibility Services are trademarks of MVS Solutions Inc.

Other trademarks and registered trademarks used in this document are the property of their respective owners and are to be regarded as appearing with the appropriate ™ or ® symbol.

© 1985 to 2017 MVS Solutions Inc. All rights reserved.

The Acrobat® PDF Format

This set of manuals has been formatted to make it suitable for soft copy as well as hard copy formats. If reading the manual online, note that:

- Entries in the Table of Contents are hot-linked and can be used for document navigation.
- Other cross-references appear in blue type and are also hot-linked for easy navigation.

For best results, turn on the Bookmarks feature in your Acrobat® PDF reader.

Preface

About This Manual

This manual provides information for systems programmers involved in planning, installing, and customizing the Drive Booking Services (DBS). There are five system programming manuals as listed below. Information described in other manuals is referred to by guide name and chapter as a quick and easy cross reference.

- Base Product System Programming Guide
- Drive Booking Services (DBS) System Programming Guide
- Dataset Contention Services (DCS) System Programming Guide
- User Control Services (UCS) System Programming Guide
- Exits System Programming Guide

Summary of Changes

V7R1-7109

(April 2017)

- No changes.

V7R1-7108

(February 2017)

- No changes.

V7R1-7107

(May 2016)

- No changes.

V7R1-7106

(November 2015)

- No changes.

V7R1-7104

(July 2015)

- No changes.

V7R1-7101

(July 2014)

- This is a base manual for ThruPut Manager Version 7 Release 1.1.

Table of Contents

Preface

About This Manual . . . 3
Summary of Changes . . . 3

The TM/JBS Component

Chapter 1. The TM/JBS Functions—An Optional Playground

TM/Job Binding Services Option . . . 11
Functions of TM/Job Binding Services . . . 11
Job Action Language (JAL) Extensions for TM/JBS . . . 12
Job Binding Services (JBS) . . . 12
Job Limiting Services (JLS) . . . 13
Job Chaining Services (JCS or Before & After) . . . 14
Mellon Bank Compatibility Services . . . 15
JECL Statement Extensions . . . 15
Started Task Support . . . 15
Application Program Interface Function . . . 16
User Display Facility Extensions . . . 16
User Exits . . . 16
JBS File Requirements . . . 16

Chapter 2. Job Binding Services (JBS) Function

Description . . . 17
Implementation Summary . . . 17
Job Binding Agents . . . 18
What is a Binding Agent? . . . 18
Binding Agent Names . . . 18
Binding Agent Attributes . . . 19
Agent Definition . . . 20
Activating Binding Agents . . . 22
Requesting BINDING . . . 23
JBS Considerations . . . 24
JECL Considerations . . . 24
Binding Agent—Verifying and Reserving . . . 25
Definition/Deletion of Agents . . . 25
API Considerations . . . 25
Job Action Language . . . 26
Activation/Deactivation of PERMANENT Agents . . . 28
UDF Extensions for JBS . . . 30
Displaying Installation-defined Information . . . 31
Facilities Summary . . . 32

Chapter 3. JBS: Incompatible Agents

Incompatible Agents: The Problem . . . 35
Incompatibility Categories: The Solution . . . 35
Usage And Examples . . . 36

Assigning Categories 36
Sample Category Definitions 36
Category 0: The Default 36
Category Relationships 37
Conflicting Definitions 38
Self-incompatibility 38
“Soft” Incompatibility: $$DELETE 38
Multiple Agents In A Single Bind Statement 39
A Case Study 39

Chapter 4. JBS: Software Access Control (SAC)

Software Access Control Overview 43
How SAC Works 43
The SAC Table 45
Table Coding Syntax 45
Table Structure 46
Associating Table Statements with a System 47
Mandatory TYPE Statements 48
Optional TYPE Statements 52
TYPE Statements for SAC Control 53
TSO Access Control 53
Batch Access Control 56
Sample SAC Table 61

Chapter 5. JBS: Environment Services

Prerequisites 65
What is a JBS Environment? 65
JBS Environments vs. Scheduling Environments 65
What Is a Resource Element? 66
JBS Resource Elements vs. SCHENV Resource Elements 66
Components to Support JBS Environments 67
JBS Environment Definition ISPF Dialog 67
JAL for JBS Environments 67
JECL for JBS Environments 68
MHS for JBS Environments 68
Operator Commands for JBS Environments 69
Implementation 69
Defining JBS Environments and Resource Elements 70
Creating Resource Elements 73
Creating JBS Environments 76
Installing a JBS Environment 82
Notes and Considerations 83
Converting from IBM Scheduling Environments 85
Using JAL with JBS Environments 86
Using JECL with JBS Environments 86

Chapter 6. Job Limiting Services (JLS) Function

Description 87
Implementation Summary 88
Job Limiting Agents 88

What Is a Limiting Agent? . . . 88
Activating Limiting Agents . . . 89
The JLS_LIMITDEF Statement in JAL . . . 89
JAL Action Statements . . . 92
Job Limiting Considerations . . . 93
System Level Limiting . . . 96
JLS and JES2 Exit 14 (Job Queue Work Select) . . . 97
UDF Extensions For JLS . . . 99
Displaying Installation-defined Information . . . 99
Facilities Summary . . . 100

Chapter 7. Job Chaining Services (JCS) Function: Before & After

Introduction . . . 103
Requirements . . . 103
Job Chaining Services—Before & After . . . 103
BATCH Name Conventions . . . 104
Implementation Summary . . . 104
JECL Statements . . . 104
Batch Job Sequencing . . . 105
Adding a BATCH Name to TSUs and STCs. . . . 105
Implementation . . . 105
Considerations . . . 106
Example . . . 107
UDF Extensions for JCS. . . . 107
Facilities Summary . . . 108

Chapter 8. Mellon Bank Compatibility Services

Introduction . . . 109
Enabling Mellon Bank Compatibility . . . 109
The Resource Routing Facility . . . 109
Resource Definition . . . 110
Resource Activation/Deactivation . . . 110
Display Facilities. . . . 110
JECL Control Statements . . . 111
Special Cases . . . 111
Additional Considerations . . . 112
The CNTL Facility . . . 112
Implementation . . . 112
The WITH Facility . . . 112
Implementation . . . 113
Facilities Summary . . . 113

Chapter 9. DD Subsystem Interface

Introduction . . . 115
DD SUBSYS Interface . . . 115
DD SUBSYS . . . 116

Chapter 10. Application Program Interface (API)

Description . . . 119
Activating/Deactivating Using the API. . . . 119

Testing Agent Status Using the API 121
Facilities Summary 123

Index

Job Binding Services

Chapter 1. The TM/JBS Functions—An Optional Playground

This chapter describes the functions provided by the Component TM/Job Binding Services.

TM/Job Binding Services Option

ThruPut Manager offers a Component that is installed on top of the ThruPut Manager Base Product. A collection of functions is provided that significantly enhance the control that an installation has over the JES2 job selection process. The Component is named "TM/Job Binding Services." This chapter discusses this package.

We remind you that the name Job Binding Services is used with two different meanings:

1. The ThruPut Manager Component is "TM/Job Binding Services" or "TM/JBS."
2. One of the Applications included with this Component is "Job Binding Services" or "JBS."

We will make the distinction as clear as possible.

Functions of TM/Job Binding Services

The TM/Job Binding Services Component of ThruPut Manager extends the facilities in the Base Product to the job selection process. With the functionality provided, job selection can be made dependent on the availability of logical resources. It can also be limited to a number established by the installation. Transmission of jobs to another NJE node can also be controlled with this facility.

This Component provides the following additional functions:

1. Job Action Language extensions for all TM/JBS components.
2. Job Binding Services.
3. Job Limiting Services.
4. Job Chaining Services.

5. Mellon Bank Compatibility Services.
6. JECL Services Extensions.
7. Started Task Support Services.
8. API Services.
9. Extensions to the User Display Facility.
10. User Exit Extensions.

Job Action Language (JAL) Extensions for TM/JBS

Extensions to JAL are provided to support TM/JBS. They allow you to manage Binding and Limiting Agents.

A summary description of the JAL extensions is provided in ***Base Product: System Programming Guide*** "Chapter 13. Job Action Language (JAL) Function."

For a complete description, see the ***JAL Reference Guide***.

Job Binding Services (JBS)

Job Binding Services, as its name indicates, addresses the problem of relating jobs that provide resources to each other. For example, database managers that provide services to batch jobs, such as IMS or DATACOM, create environments that are difficult to manage. JBS provides a cost-effective solution to the problems associated with that type of environment.

JBS provides a comprehensive set of facilities for the complete management of all aspects of job binding. This includes:

- JBS JES2 Control cards.
- A special "comment" form of JECL statements that permits BIND statements to be included in JCL procedures.
- Started task support using a DD subsystem interface.
- Job Action Language extensions.
- Operator commands.
- Application Program Interface by means of a simple message intercept.
- JBS initialization parameters for initial activation and JBS "cold starts."

With the above facilities, users can:

- Activate and deactivate installation-defined Binding Agents.
- Have jobs automatically scheduled or held, depending on the status of the Binding Agents.
- Automatically "insert" a BIND statement for a job using JAL facilities.
- Schedule jobs that have dependencies by using the BIND facility.
- Direct jobs to a given processor by using the BIND facility.
- Transmit jobs to another NJE node.
- Verify the usage of Job Binding Control statements in the JAL so naming conventions and usage standards can be easily managed.
- Control whether Binding Agents are active simultaneously on the same system.

This function is described in "Chapter 2. Job Binding Services (JBS) Function."

JBS also supports:

- ***JBS Environments***, which are intended to provide extended capabilities to users of IBM scheduling environment (SCHENV) support. This function is described in "Chapter 5. JBS: Environment Services."
- ***Software Access Control (SAC)***, which allows your installation to control software licence compliance. This function is described in "Chapter 4. JBS: Software Access Control (SAC)."

Job Limiting Services (JLS)

Job Limiting Services is designed to limit the parallel execution of certain types of work without resorting to separate classes.

The proliferation of classes makes the task of deploying initiators difficult. For operating staff, a large number of classes makes their job confusing. With JLS, you can greatly reduce the "alphabet soup" required to control different types of work in a standard z/OS JES2 environment.

JLS delivers to your installation a set of powerful facilities to manage Job Limiting. This includes:

- The ability to associate jobs and Limiting Agents using JAL.
- Facilities to create Limiting Agents dynamically in JAL. This greatly simplifies the administrative requirements.

- The ability to associate jobs with a particular Agent but with different "weights" so one job can, if desired, count as several jobs.
- Facilities to request that a particular job must execute in "exclusive mode", that is, no other job associated with the Limiting Agent can be executing at the same time.
- A further refinement of the "exclusive mode" is provided with the DRAIN/NODRAIN facility. When DRAIN is associated with a job requesting "exclusive control", other jobs that use the resource in non-exclusive mode are not selected. This mode expedites the availability of the resource for exclusive usage.
- A further degree of operational control by a comprehensive set of commands. With operator commands, the limits set in JAL can be overridden. This permits rapid response to unusual situations that might be creating shortages of computing resources.

This function is described in "Chapter 6. Job Limiting Services (JLS) Function."

Job Chaining Services (JCS or Before & After)

The Before & After mode of Job Chaining Services provides full compatibility with the Mellon Bank "Before & After" Facilities. Some additional services are also provided. B&A is implemented as a separate Control File Application with the name Job Chaining Services. It includes:

- Support for the /*BEFORE and /*AFTER JECL statements.
- Operating commands to display dependencies.
- An additional JECL statement /*JCS BATCH that allows the user to name a group of jobs. This includes support for the SEQUENCE keyword, which ensures that within a batch, jobs will run in the order in which they were submitted.
- The additional "BATCH=name" keyword in /*BEFORE and /*AFTER statements to restrict the scope of the statements to the named group of jobs.
- JAL extensions to help your installation manage the usage of Batch Names, so conflicts can be avoided.

This function is described in "Chapter 7. Job Chaining Services (JCS) Function: Before & After."

Mellon Bank Compatibility Services

The Mellon Bank Compatibility Services provides the following:

- A facility to activate Mellon Bank Compatibility Services through a keyword in TMPARM.
- A facility to indicate to ThruPut Manager what to do with your existing JECL Mellon Bank statements.
- Support for /*CNTL, /*ROUTE XEQ, and /*WITH Mellon Bank JECL statements.
- A definition statement in JAL to allow you to replicate the function of the /*CNTL statement.

The Mellon Bank Compatibility Services, together with the Before & After facilities described above, provide upwards compatibility to the most commonly implemented Mellon Bank User Mods.

This function is described in "Chapter 8. Mellon Bank Compatibility Services."

JECL Statement Extensions

The TM/Job Binding Services Component extends JECL support to include the functions introduced by TM/JBS. A number of different control statements are provided. JECL statements are available for Job Binding Services, Job Limiting Services, and Job Chaining Services.

If you have JECL statements that provide similar functions and you want to convert them to TM/JBS format, this facility provides a mechanism for capturing and automatically converting your statements to the new format.

For a detailed description of all the JECL statements refer to ***JECL Reference Guide***.

Started Task Support

Started tasks can use the services of TM/JBS JECL facilities, provided that the appropriate JECL keywords are coded on the TMPARM statement. A description of how this support works is included in ***JECL Reference Guide***.

Previous versions of ThruPut Manager provided an equivalent mechanism that used the SUBSYS keyword of the JCL DD statement, allowing you to "code" the equivalent of JECL statements in your started task JCL. This DD subsystem interface has been retained for compatibility, and is described in "Chapter 9. DD Subsystem Interface."

Application Program Interface Function

There are situations where you want to activate or deactivate Binding Agents from a program, or as a result of an application having "done something". A typical example is a CICS region that dynamically allocates and deallocates resources. In this case, no step initiation or step termination takes place. The region might simply deallocate a database dynamically for maintenance purposes. Prior to doing that, the Binding Agent associated with that service should be deactivated.

An API is provided to deal with this case.

This function is described in "Chapter 10. Application Program Interface (API)."

User Display Facility Extensions

TM/JBS extends User Display Facility support to include UDF Windows for each TM/JBS component. Examples of the information displayed in these windows are included in the chapters describing the JBS, JLS, and JCS functions.

User Exits

An additional user exit, exit 19, also can be used to support TM/JBS. This exit allows you to examine JECL statements. A summary of the exit is provided in ***Base Product: System Programming Guide*** "Chapter 11. Installation Exits Summary."

Complete details of installation exits, including design considerations and implementation descriptions, are provided in the manual ***Exits: System Programming Guide***.

JBS File Requirements

TM/JBS requires the TM Control File to be active. The Control File is described in ***Base Product: System Programming Guide "***Chapter 3. File Definition Services (FDS) Function."

Chapter 2. Job Binding Services (JBS) Function

This chapter describes the capabilities and services provided by the JBS Function.

Description

Job Binding Services helps you solve the problem of relating jobs that provide resources to each other. A typical example of an environment that JBS has been designed to cope with is a database manager (e.g. IMS or DB2) that provides services to batch jobs with their associated problems. JBS has a comprehensive set of facilities for complete management of all aspects of Job Binding, including:

- JBS JES2 Control cards.
- A DD Subsystem interface for started tasks.
- Job Action Language extensions.
- Operator commands.
- Application Program Interface employing a simple message intercept.
- JBS initialization parameters for initial activation and JBS "cold starts."
- Addition of a JBS Display Window to the User Display Facility.

Implementation Summary

Before implementing the JBS function you should become familiar with its facilities and purpose. The ***TM/Job Binding Services Concepts and Facilities*** publication provides an overall description of this function. Some examples of usage are included.

To implement the JBS Function you must:

- Determine the names and types of Binding Agents that your installation needs.
- Use the /JBS DEFINE operator command to define the Binding Agents.
- Put in place operational procedures for the Activation/Deactivation of PERMANENT Binding Agents. You should also establish regular backups of your Agent definitions with the JBS DEFINE TODSN command.

- Define the rules and Agents that users can place in their job streams using JECL statements.
- Add any statements needed in JAL to manage JECL and Binding Agents.
- Consider procedures that might benefit from //*+JBS BIND statements.

Job Binding Agents

Job Binding Agents provide the essential mechanism for establishing inter-job relationships. These Agents also allow you to associate jobs with the availability of a particular resource. One example is licensed software in a given processor, such as a compiler.

What is a Binding Agent?

Agents are self-describing logic elements. They contain information about their attributes and serve as anchors to jobs that are "bound" by these Agents.

A Binding Agent does not represent any physical resource. It represents whatever your installation chooses it to represent.

Binding Agent Names

Binding Agents have a name that is chosen by your installation. They also have attributes that indicate certain characteristics. These characteristics, to be explained below, make certain Agents more suitable than others for particular types of work.

The names follow dataset naming conventions. They can be two levels. For example:

```
CICS.PAYROLL
```

is a valid Binding Agent name.

```
COMPSAS
```

is also a valid Binding Agent name.

```
CICS.DEV.G1
```

is not a valid name.

☞ ***Binding Agents are predefined by the installation before they can be used. This allows control over usage and naming standards.***

Binding Agent Attributes

The attributes have been designed to address a variety of situations. You probably have only some of the requirements that these Agents can address. As a result, if some of the characteristics do not appear to have any applicability in your installation, simply ignore that feature.

Some attributes are assigned when an Agent is defined. This allows tight operational control. Some other attributes are assigned at activation time.

Attributes at Definition Time

The first distinction with Binding Agents is two-fold:

1. Is the Agent intended to be job-related?
2. Is it intended to be PERMANENT?

While all the Agents are used by jobs for scheduling purposes, the above attributes refer to *activation:*

If an Agent is to be activated by a job and deactivated at the end of the job then the type of Binding Agent to use is a *Job-related Binding Agent.* A good example of this situation is an IMS region.

If an Agent is to be activated with a command and remain active until explicitly deactivated, then you want a *PERMANENT Agent.* An example could be an Agent that is used to direct jobs to a processor (in a multi-CPU MAS complex) that has a licensed software package that is not available anywhere else. In this case a number of jobs want to be bound to that Agent, but its activation is associated with a particular CPU. Hence, the Agent is activated with a command. It stays active regardless of the status of the processor. Jobs are automatically assigned to that processor and are selected for execution whenever the processor is available.

The second distinction concerns uniqueness:

1. Do you want an Agent to be *unique* in your MAS complex?
2. Do you want an Agent to be able to appear *more than once* in your MAS complex?

If a given service is to be provided only in a single processor, then your Agent should have the attribute of UNIQUE. This prevents an accidental activation of the same Agent in another processor. Note that the attributes UNIQUE and MULTIPLE refer to a MAS complex. Regardless of the attribute, an Agent can be active ***only once on each processor***.

The third distinction allows you to control which Binding Agents can be used to initiate NJE transmission. At definition time you establish whether or not an Agent can be activated with the XMIT attribute (only with an operator command).

Attributes at Activation Time

At activation time a Binding Agent can be given one of the following three attributes:

1. GLOBAL
2. LOCAL
3. XMIT

GLOBAL indicates that the Agent is active in all the systems in a MAS complex. An example of a situation that requires a GLOBAL Agent is when you use the Binding Agent mechanism to signal an event such as "overnight processing can begin." In this case, it applies to all the systems in the MAS complex.

LOCAL indicates that the Binding Agent is to direct jobs to a particular system. The availability of an IMS online region is a good example.

XMIT indicates that jobs associated with this Binding Agent are to be transmitted to the designated NJE node. ***This attribute is valid only if the Agent was defined with that capability, and can only be activated with a command***.

☞ ***Note that transmitted jobs are re-analyzed on the receiving node.***

Agent Definition

Before any Binding Agents can be activated or deactivated, they must be defined to Job Binding Services. Agents can be defined either individually with a command or using Group Definition Services from a file. For a detailed explanation, refer to the ***Operating Guide*** publication for the JBS DEFINE commands.

It should be noted that this is required only when the Control File is *cold started*, or new Agents are to be added. Once the Agents have been defined, they exist until the Control File is reformatted.

If you plan to reformat the Control File but intend to redefine the same Agents, you simply invoke the Group Create Services to have a file created with all the necessary definitions.

Binding Agents can be defined as having one of the following characteristics:

PERMANENT: This indicates that the Agent is not directly associated with an executing job. This type of Agent, once it is activated, must be explicitly deactivated.

MULTIPLE: This type of Agent is Job Related. It normally remains activated while a particular job (or job step) is executing. Once the job that activated the Agent terminates, the Agent is automatically deactivated. MULTIPLE means that in a MAS complex there can be more than one Agent active with the same name; however, *there can be only one per machine*.

UNIQUE: If an Agent has the UNIQUE attribute, JBS schedules only one job that activates/deactivates the Agent in the shared spool complex at any given time. This ensures that jobs using an Agent with this attribute are serialized.

XMIT: This capability applies to PERMANENT Binding Agents. It allows the initiation of transmission to another NJE node for jobs that are bound to that type of Agent. The activation of an Agent with the XMIT keyword can be done ***only with an operator command.*** This provides complete operational control of the facility.

In some circumstances, it might be important to know the source of the last request to activate or deactivate an Agent. To provide this information, you can also combine LOG with any of the above attributes:

LOG: This attribute requests recording in the system log for all ACTIVATE and DEACTIVATE requests by a job for this Agent.

A number of further definition attributes are provided for PERMANENT Agents. Each attribute is designed to deal with a particular situation, so if you do not recognize the purpose of an attribute, it is likely that you do not have that problem in your installation.

The following attributes apply to ***PERMANENT Agents only***, and can be specified alone or together:

WARN: This attribute causes a non-deletable message to be issued when the Agent is *inactive* and a job binding to that Agent enters the system. The message is issued on the system that has selected the job binding to the inactive Agent, but the Agent can be activated from any system in the complex. The system requiring the Agent detects that the Agent has been activated, and deletes the message.

OPER: This attribute specifies that the Agent can be controlled only by operator commands. If a job includes activate or deactivate requests for an Agent with the OPER attribute, it is failed with a JCL error.

Finally, you can use the JBS REDEFINE command to change certain attributes for an existing Agent:

- LOG | NOLOG
- OPER | NOOPER
- WARN | NOWARN

Activating Binding Agents

There are four ways to activate Binding Agents, depending on the circumstances:

1. JECL JES2 Control Statements.
2. DD Subsystem Interface.
3. Application Program Interface (API).
4. Operator Commands.

The following sections provide a brief explanation of each method and the reasons for using it.

JECL JES2 Control Statements

Jobs that activate Binding Agents use the JECL statement mechanism. You can have up to 6 ACTIVATE JECL statements in a job.

The statements allow the activation of Agents either at job initiation or at the initiation of a named step.

For a comprehensive description of JBS JECL statements refer, to ***JECL Reference Guide.***

DD Subsystem Interface

☞ ***Although this interface is still supported, it is no longer necessary. Instead, we recommend using the comment form of ThruPut Manager JECL statements (//*+JBS ACTIVATE ...).***

Started tasks can include JBS ACTIVATE and JBS DEACTIVATE JECL statements. The DD subsystem interface also provides started tasks with the ability to Activate and Deactivate Binding Agents.

Each DD SUBSYS request is the equivalent of a JECL statement for a job. The same rules apply.

For a detailed explanation of the DD subsystem interface, refer to "Chapter 9. DD Subsystem Interface."

Application Program Interface (API)

Two different mechanism can be used with the API:

- A simple message (WTO) intercept with a pre-defined message ID and a pointer to the particular Binding Agent JECL reference. When the message is intercepted the specified Agent is activated or deactivated. This technique is normally used when you have control of the application, so you can issue the message.
- A message intercept where you specify a text pattern to be matched against messages generated by the address space where the job is running. When a message is generated in the particular address space that matches the text pattern, the specified Agent is activated or deactivated. This technique is normally used when you have no control over the application's messages, but you can uniquely identify parts of the message.

For a detailed description of the API, refer to "Chapter 10. Application Program Interface (API)."

Operator Commands

Operator commands are provided to activate and deactivate Agents. If an Agent has been defined using the OPER attribute, it can be activated or deactivated *only* by using operator commands.

For a complete description of the JBS commands for activating and deactivating Agents, refer to the ***Operating Guide***.

Requesting BINDING

Jobs can request Binding in one of the following ways:

- The simplest method is using the JECL BIND statement. Jobs that want to use a Binding Agent for execution purposes include a JECL BIND statement in their JCL. Up to 24 BIND statements can appear in a job.
- In cases where transparency is desired, so that users do not have to take any action with their JCL, JAL can be used to insert BINDING statements for a particular job. Up to 24 BIND statements can be inserted by JAL.

- Again, started tasks can use the DD SUBSYS to request Binding. The same rules used for JECL statements apply.
- A list of Binding requests can be created using the pre-JAL exits (1-5 and 19). To use these Binding requests, your JAL must execute an action statement indicating that requests have been defined in exits:

  ```
  JBS BIND FROM_EXITS
  ```

  Complete details about how to use this facility can be found in the ***Exits*: *System Programming Guide.***

A single JECL BIND statement allows for the inclusion of four Binding Agents. These four Agents represent alternatives, that is, they are OR conditions. Any one of them satisfies the Binding requirement.

Multiple Binding statements represent an AND condition, that is, each statement must be satisfied before the job can be scheduled. Between JECL statements and JAL, a maximum of 48 BIND statements can be associated with a job.

JBS Considerations

This section outlines some considerations that could affect the way you implement JBS in your installation.

JECL Considerations

- The JBS JES2 control statements are processed in the order that they are encountered in the job stream.
- During testing of JBS or if jobs are to be transmitted to NJE nodes that do not have ThruPut Manager, consider using the //*+ form of JBS control statements. They will not cause errors in non-ThruPut Manager environments because they are treated as comments.
- If BIND requests are inserted with JAL for jobs that are to be transmitted, the inserted BIND statements are not seen at the receiving NJE node.
- A job that has JBS ACTIVATE statements for Job-related Binding Agents that are currently active (via another job) *is not eligible for selection until the other job(s) terminate.*
- Care should be taken when multiple BIND statements are included in a job to make sure that conflicting BINDs are not created. Jobs are not selected until all the BINDs can be satisfied. For additional facilities that help you to manage Binding Agent conflicts, see “Chapter 3. JBS: Incompatible Agents.”

Binding Agent—Verifying and Reserving

We recommend that a nomenclature for naming Binding Agents be established from the beginning. This makes verification in JAL a simple task and can avoid usage conflicts by different groups.

All Binding Agent references are verified at Job Analysis time. The Agents must have been defined, otherwise the job is failed with a message.

To ensure that there are no conflicts when activating Agents at the step level (some time after the job started execution), the following takes place:

- As soon as the job is allowed to be selected for execution, all the Job-related Binding Agents that the job refers to with JBS ACTIVATE control statements *are reserved*. This ensures that the job will not run into any problems when it activates the Agents.
- Any Job-related Binding Agent that is *reserved or active* is considered to be *busy*. No other job will be allowed to start if it references a busy Agent in a JBS ACTIVATE statement (unless it has been defined as MULTIPLE). As a result, a Binding Agent can have one of the following states:
  - Defined
  - Reserved but not Active (busy state)
  - Reserved and Active (busy state)

Definition/Deletion of Agents

The JBS DELETE command will only remove Agents that *are not busy*.

The whole concept of Binding Agents is predicated on the presence and status of such Agents. No actions can be taken as a result of the absence of Agents. This is to avoid situations where errors of omission can result in incorrect actions.

If you want to schedule jobs based on, for example, IMS services not available, then define another Agent to represent this situation. For example, IMS.TEST active might indicate that jobs that request binding to that Agent can be scheduled. If there is a class of jobs to be scheduled when IMS is not available, then create another Agent, for example NOIMS.TEST, that is activated when IMS.TEST is deactivated.

API Considerations

The requirement to externalize one aspect of the use of the program interface by connecting it to actual JECL statements is to perform verification and be able to reserve Agents before execution begins. This eliminates situations where,

through the API, a program requests the activation of an Agent that does not exist or is already active and has the attribute of UNIQUE.

Job Action Language

JAL provides facilities to help you ensure that JBS will function as intended in your environment. When Binding management is done in JAL, the situation where a JAL BIND request is made for an invalid Binding Agent (not defined) should not occur. This normally represents either a JAL coding error or an accidental deletion of an Agent. A JBS HOLD statement can be optionally added to hold and requeue jobs when a JBS Environment or JBS Agent is undefined. Refer to the ***JAL Reference Guide***.

JBS includes facilities to eliminate those embarrassing moments. Note that Agent verification techniques verify *all* Agents in your JAL, whether or not the Agent is actually referenced by the JAL logic.

Verification of Agents Using the Language Processor

JBS provides Agent verification at JAL compile time. This requires access to the Control File during JAL compilation. To verify Agents at compile time, include the VERIFY parameter on the EXEC statement for the Language Processor:

```
//JALCOMP  EXEC PGM=DTMJALP4,PARM='VERIFY'
```

Any Agent that is not defined is flagged in the Binding Agent cross-reference report produced by the Language Processor. Here is a sample:

```
THRUPUT MANAGER ACTION LANGUAGE TM/JBS BINDING AGENT CROSS REFERENCE REPORT

BINDING AGENT      STATUS          REFERENCES

AGENTA.UNDEF      *NOT-DEFINED*    00006 00020
AGENTB.UNDEF      *NOT-DEFINED*    00006 00023
AGENTX.DEFINED         DEFINED     00009 00025 00026 00031 00035
AGENTY.DEFINED         DEFINED     00010 00025 00028 00033 00035 00039

TM/JBS AGENTS:   4, DEFINED:   2, NOT-DEFINED:   2

DTM4565I *** WARNING *** TM/JBS BINDING AGENT(S) NOT DEFINED
```

Binding Agent Cross-reference Report

Verification Using the JAL VERIFY Command

If you do not have access to the Control File from the processor where the Language Processor is run, you can use the JAL VERIFY command. It is advisable to use this command just before doing any refresh.

The VERIFY command requests that a particular JAL internal text be verified, to ensure that all the Agents are defined. Just before refreshing your JAL, issue a command similar to this:

```
/JAL VERIFY AGENTS TM.PARMLIB(NEWJAL)
```

This results in a display resembling this example:

```
DTM4743I JAL VERIFY AGENT DISPLAY
   DSNAME=TM.PARMLIB MEMBER=TESTJAL
   --BINDING AGENT-- --STATUS---
   AGENTA.UNDEF      NOT-DEFINED
   AGENTB.UNDEF      NOT-DEFINED
TM/JBS AGENTS:  4, DEFINED:  2, NOT-DEFINED:  2
```

This display shows only those Agents that are not defined. You can display all agents defined in the JAL being verified by using the DISPLAY keyword:

```
/JAL VERIFY AGENTS TM.PARMLIB(NEWJAL) DISPLAY
```

The display then shows all Agents:

```
DTM4743I JAL VERIFY AGENT DISPLAY
   DSNAME=TM.PARMLIB MEMBER=TESTJAL
   --BINDING AGENT-- --STATUS---
   AGENTA.UNDEF      NOT-DEFINED
   AGENTB.UNDEF      NOT-DEFINED
   AGENTX.DEFINED        DEFINED
   AGENTY.DEFINED        DEFINED
TM/JBS AGENTS:  4, DEFINED:  2, NOT-DEFINED:  2
```

Verification of ACTIVATE and DEACTIVATE Statements

The Job Action Language also provides facilities to verify JBS ACTIVATE and DEACTIVATE statements. In this way, naming conventions can be established and enforced. Also, you might want to restrict the jobs that can activate/deactivate Agents.

This verification also applies to the use of the API.

A number of action statements are provided to facilitate the inserting of BIND statements in JAL. You can:

JBS ADD BIND

Add a BIND statement to the job.

JBS BIND FROM_EXITS

Add a BIND statement defined in an exit.

JBS REPLACE BIND

Replace the BIND statement previously added.

JBS DELETE BIND

Delete the BIND statement previously added.

Activation/Deactivation of PERMANENT Agents

Binding Agents defined with the PERMANENT attribute, once activated, will remain in an active state regardless of the status of the system where they were activated. In other words, a system failure or a system shutdown will not cause the Agents to be deactivated. It must be done explicitly. This simplifies the management of resources associate with a processor.

For example, in a multi-system environment, a PERMANENT Agent might describe where a given software product is (e.g a compiler in SYS2). You want that Agent to continue its association with that processor, even when SYS2 is not available. Only when you invoke backup procedures, or a similar arrangement, do you want the Agent to be activated in another processor. So deactivating the Agent in its normal host and activating it in another processor is a explicit action (using commands).

Conditional Activation/Deactivation

Permanent Agents defined with the UNIQUE attribute can be controlled conditionally, that is, based on whether the Agent was active or inactive before the requesting job started. This is accomplished with the COND keyword on the JBS ACTIVATE or JBS DEACTIVATE statement. Its use is best illustrated with an example:

- Assume AGENT.USEFUL is an Agent that represents a particular service.
- This service is offered during low-demand hours. AGENT.USEFUL is a Permanent Agent that is activated and deactivated with a command.

- Occasionally some library maintenance is done that momentarily precludes the use of the service. The job that performs the maintenance cannot run concurrently with jobs using the service.
- At job initiation, the maintenance job deactivates AGENT.USEFUL, so no other job will be selected that uses that service.

At job termination, the maintenance job should reinstate the Agent to its original status, but—*what was the original status?*

This is where the COND keyword is useful. In this case, a JBS ACTIVATE request with the COND keyword should be used:

```
//*+JBS ACTIVATE AGENT.USEFUL,COND
```

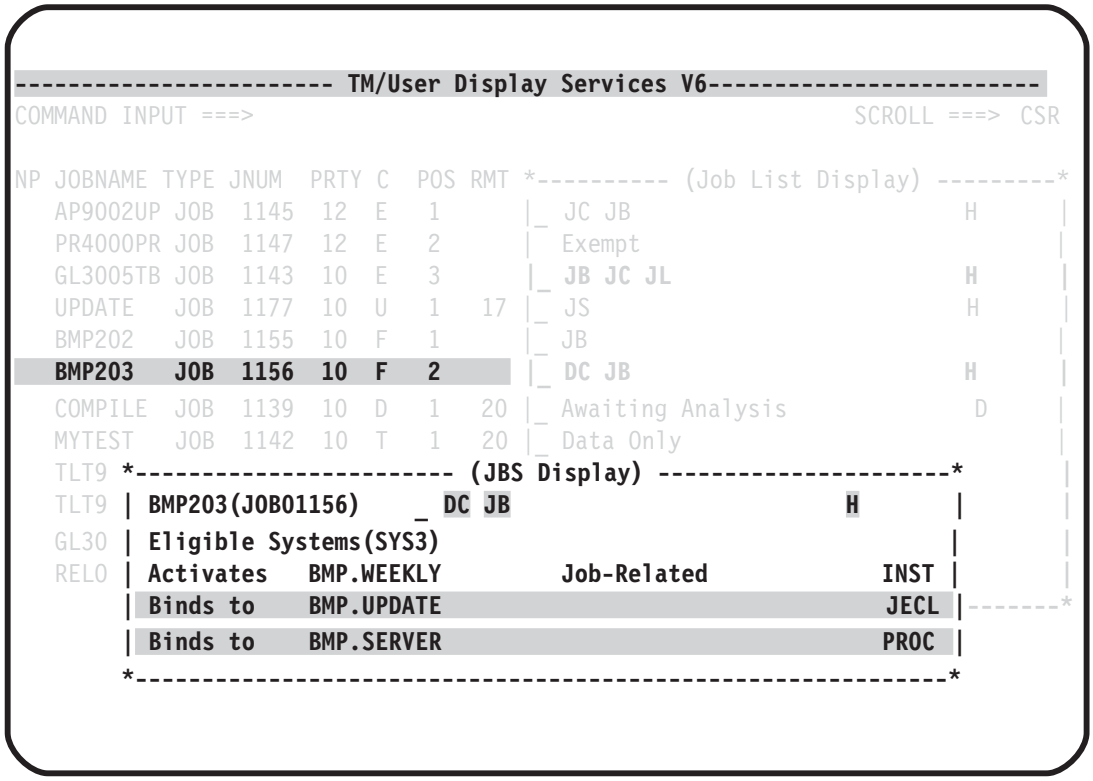
AGENT.USEFUL is then activated *only if it was active* before the job requested the Agent to be deactivated.

Note that you must define AGENT.USEFUL using the UNIQUE attribute for a Permanent Agent:

```
/JBS DEFINE AGENT.USEFUL PERMANENT UNIQUE
```

UDF Extensions for JBS

If the User Display Facility shows the JBS acronym for a job, a user can select that acronym and open the JBS Window. A sample JBS Window is shown here.



The JBS Display Window

The first line of the JBS Window is the Information Summary Line for the job.

The second line shows the job's system affinity.

Subsequent lines list each Binding request affecting the selected job. For each Binding request listed, the display indicates whether an Agent is being activated, deactivated, or bound, whether the Agent is Job-related, and what the source of the request is:

- **PROC** indicates that the request is contained in a JCL PROC.
- **JECL** indicates that the request comes from JECL for the job.
- **INST** indicates that the request came from an installation action such as JAL.

Highlighted lines indicate that the Binding request described on that line causes the selected job to be held.

Displaying Installation-defined Information

UDF allows you to define an ISPF panel that is displayed when a user places the cursor on a JBS Agent name in the JBS Display Window and then hits ENTER. The panel name is associated with the Agent by the PANEL keyword of the JBS DEFINE or JBS REDEFINE operator commands:

```
/JBS DEFINE IMS … PANEL(OURPANEL)
```

OURPANEL must be a valid ISPF selection panel, since it is displayed using the ISPF SELECT service.

UDF initializes several ISPF variables that you can use in your panel, as shown in the table below.

ISPF Variables Initialized for Installation-defined UDF JBS Display

Name	Type	Length	Description
DTMJOBNM	Char	8	JES2 job name.
DTMJOBID	Char	8	JES2 job ID.
DTMRACGR	Char	8	RACF group (RACF only).
DTMRACUS	Char	8	RACF user ID (RACF only).
DTMTSSUS	Char	8	TSS user ID (TSS only).
DTMACFSI	Char	8	ACF2 Source ID (ACF2 only).
DTMACFLI	Char	8	ACF2 logon ID (ACF2 only).
DTMACFUI	Char	24	ACF2 user ID (ACF2 only).
DTMJBNAM	Char	17	JBS Agent name.
DTMJBVRB	Char	10	JBS verb (ACTIVATE, BIND, DEACTIVATE).

Facilities Summary

JBS Operator Commands	
Command	**Description**
JBS ABANDON	Removes a job from Job Binding Services control.
JBS ACTIVATE	Activates a PERM Binding Agent.
JBS DEACTIVATE	Deactivates a PERM Binding Agent.
JBS DEFINE (single)	Defines a single Binding Agent.
JBS DEFINE (group)	Define Agents from a file.
JBS DELETE	Deletes a Binding Agent.
JBS DISPLAY	Displays Agents, jobs, and their relationships.

JBS JECL Statements Refer to *Base Product: System Programming Guide*		
Statement	**Description**	**Chapter**
/*JBS ACTIVATE	Activates a Binding Agent.	7
/*JBS BIND	Request that the job be “bound” to an Agent.	7
/*JBS DEACTIVATE	Deactivates a Binding Agent.	7
/*JBS MESSAGE	Describes a system or application message that triggers the activation or deactivation of a Binding Agent.	7

JBS EXITS		
Exit #	**Exit Name**	**Description**
Exit 19	JECL Statement inspection.	This exit receives control for JECL statements that the installation has defined to ThruPut Manager using tables in the JES2 source module.

Chapter 3. JBS: Incompatible Agents

This chapter explains the facilities provided to help you manage Binding Agent conflicts.

Incompatible Agents: The Problem

Incompatible Agents are Binding Agents that cannot be active simultaneously or must be active on different JES2 systems.

Without a function to manage incompatibility, jobs that bind to incompatible agents would be held by JBS with no chance of release and no external indication that there is anything wrong. Fortunately, JBS provides a solution to this problem: Incompatibility Categories.

Incompatibility Categories: The Solution

An ***Incompatibility Category*** is an attribute given an Agent at definition or redefinition time. If you do not explicitly assign a Category, Agents default to a Category that is compatible with all other Categories.

Through JBS commands, you can define the relationship of each Category to every other Category. The definition indicates what action to take when a job binds to Agents in different Categories.

The possible actions for Agents in incompatible Categories are:

- Fail the job, which generates messages explaining the incompatibility.
- Enter a warning message the job's log and let the job proceed. This is intended to let you warn users about future incompatibilities. If the Agents are really incompatible, the job will not run until the incompatibility is resolved.
- Ignore the incompatibility and allow the job to execute. The Agent definition, through the special name $$DELETE, determines which incompatible Agent is ignored.

Categories thus provide you with a means to prevent jobs from winding up in limbo, and also relieve your users from the need to know which Agents are incompatible.

Usage And Examples

JBS provides 10 Categories numbered from 0 to 9. The Categories from 1-9 are available for you to define as required to meet your needs. For ease of use, you can give a name to each Category (except for Category 0) at definition time. Category 0 is the default Category and cannot be assigned a name.

Assigning Categories

When you define an Agent, you can assign a Category by name or number:

```
/JBS DEFINE COBOL.COMPILER CATEGORY(3)
/JBS DEFINE DB2.PROD CATEGORY(PROD)
```

Categories can also be assigned or changed when you REDEFINE an Agent, but such a change affects only subsequent jobs. Jobs already held or awaiting execution are not affected.

Sample Category Definitions

Here are some sample Category definitions:

```
/JBS DEFINE 1 INCOMPAT(2W,3F)
/JBS DEFINE 2 INCOMPAT(3W) NAME(PROD)
```

In this example:

- Category 1 is defined as incompatible with Category 2. An attempt to bind to Agents in both Categories results in a warning message.
- Category 1 is also defined as incompatible with Category 3. An attempt to bind to Agents in both Categories results in the job being failed.
- Category 2 is defined as incompatible with Category 3, with a warning message resulting.

Note that Category 2 is given a Category Name. Category Names allow you to identify Categories with a name, which can be remembered more easily than just a simple number.

Category 0: The Default

Every Agent belongs to one and only one Category. If you do not assign a Category, the Agent belongs to Category 0. This Category is always compatible with all other Categories and cannot be modified.

Category Relationships

Agent Categories and their relationships can be represented in a simple table. The table below shows the results of our sample definitions.

		INCOMPATIBILITIES									
		0	1	2	3	4	5	6	7	8	9
CATEGORIES	1			W	F						
	2				W						
	3										
	4										
	5										
	6										
	7										
	8										
	9										

Note that there is no row for Category 0 in this table because Category 0 is compatible with all other Categories by definition, therefore:

```
/JBS DEFINE 0 INCOMPAT(5W)
```

is an error.

You can define a Category to be incompatible with Category 0, however:

```
/JBS DEFINE 5 INCOMPAT(0W)
```

is valid. This illustrates that incompatibility definitions are not commutative. In other words, the incompatibility arises from the fact that Category 5 Agents cannot mix with Category 0 Agents, not the other way round. Knowing which Category causes incompatibility can be useful when diagnosing problems.

Conflicting Definitions

When two conflicting definitions are given, the last one encountered prevails. For example:

```
/JBS DEFINE 3 INCOMPAT(4W)
/JBS DEFINE 4 INCOMPAT(3F)
```

results in the job being failed if it attempts to bind to Agents from both Category 3 and Category 4.

Self-incompatibility

A Category can be defined as incompatible with itself:

```
/JBS DEFINE 4 INCOMPAT(4F)
```

This limits the number of Category 4 Agents in a job ***to just one***.

"Soft" Incompatibility: $$DELETE

You can ignore an incompatibility by using $$DELETE instead of an Agent name. For example, assume that for performance reasons, you want some jobs to run on a specific system. Your JAL therefore binds them to an Agent defined on the preferred system:

```
/JBS ADD BIND(PREFER.SYS1,$$DELETE)
```

Before we explain the role of $$DELETE, let us further assume that some jobs require a resource that sometimes is moved to another system. The users deal with this by inserting a JECL statement that binds them to the resource:

```
/*JBS BIND MOVABLE.RESOURCE
```

The definitions for PREFER.SYS1 and MOVABLE.RESOURCE assign them to Categories that allow you to adjust them to the situation:

```
/JBS DEFINE PREFER.SYS1 CATEGORY(1)
/JBS DEFINE MOVABLE.RESOURCE CATEGORY(2)
```

These Categories normally are compatible. When MOVABLE.RESOURCE is not available on SYS1, however, you change the definition for Category 2:

```
/JBS DEFINE 2 INCOMPAT(1F)
```

Since you do not want jobs requesting MOVABLE.RESOURCE to fail simply because you *prefer* that they run on SYS1, you make the incompatibility optional. This is the purpose of the special Agent $$DELETE, shown earlier. If

PREFER.SYS1 is incompatible with another Agent, $$DELETE causes the incompatibility to be ignored by effectively deleting the BIND to PREFER.SYS1.

Multiple Agents In A Single Bind Statement

A JBS Bind statement can specify up to four Agents. Because these Agents are treated as an OR condition, you can specify two or more incompatible Agents together on the same Bind statement. Incompatibility is tested only across individual Bind statements (across groupings on a compound BIND JECL statement).

For the same reason, if a Bind statement contains ***at least one Agent*** that is compatible with other Bind statements for the job, the statement is considered compatible.

A Case Study

Acme Anvils, a well-known ThruPut Manager installation, uses Incompatibility Categories to solve some of its processing problems. Here is how they do it.

First, the situation at Acme Anvils:

- The datacenter has two systems: SYS1 and SYS2
- SYS1 provides access to SAS software.
- SYS2 provides access to production DB2.
- SYS1 provides DB2 testing facilities. A new release of DB2 has arrived.
  - *During the morning*, the old DB2 release is available on SYS1.
  - *During the afternoon*, the new release of DB2 is available on SYS1.
- The situation is controlled with Binding Agents.

Four Agents are created:

SOFT.SAS

This Agent is active on SYS1 only, where SAS is licensed.

PROD.DB2

This Agent is active on SYS2 only

TEST.DB2

This Agent is active on SYS1 only between 8 and 1 o'clock.

TEST.NEWDB2

This Agent is active on SYS1 only between 1 and 6 o'clock.

Acme wants to avoid jobs winding up in "limbo", so they assign these Categories:

```
/JBS DEFINE PERMANENT SOFT.SAS CATEGORY(1)
/JBS DEFINE PERMANENT PROD.DB2 CATEGORY(2)
/JBS DEFINE PERMANENT TEST.DB2 CATEGORY(3)
/JBS DEFINE PERMANENT TEST.NEWDB2 CATEGORY(4)
```

The incompatibility definitions are as follows:

```
/JBS DEFINE 1 INCOMPAT(2F)
/JBS DEFINE 2 INCOMPAT(1F,3F,4F)
/JBS DEFINE 3 INCOMPAT(2F,4F)
/JBS DEFINE 4 INCOMPAT(2F,3F)
```

With the above definitions, JBS can manage the problems.

- SAS and production DB2 (Category 1 and 2) in the same job will not run since they are provided in different machines, therefore:

  ```
  /*JBS BIND SOFT.SAS
  /*JBS BIND PROD.DB2
  ```

  results in an incompatibility error and the job is failed.

- DB2 testing and DB2 production requests in the same job cannot run since they are provided on different systems (Category 2 for production, 3 and 4 for testing), therefore:

  ```
  /*JBS BIND PROD.DB2
  /*JBS BIND TEST.DB2
  ```

  results in an incompatibility error. So does:

  ```
  /*JBS BIND PROD.DB2
  /*JBS BIND TEST.NEWDB2
  ```

- Testing of old and new DB2 in the same job will not run since the service is provided during non-overlapping times (Categories 3 and 4), so:

  ```
  /*JBS BIND TEST.DB2
  /*JBS BIND TEST.NEWDB2
  ```

  also results in an error; however, this is valid:

  ```
  /*JBS BIND TEST.DB2
  /*JBS BIND SOFT.SAS
  ```

So is this:

```
/*JBS BIND TEST.NEWDB2
/*JBS BIND SOFT.SAS
```

Acme Anvils, by simply separating its Binding Agents into Incompatible Categories, can thus ensure that its workload continues to run smoothly, and that jobs requesting incompatible resources are failed rather than entering a permanent and unexplained hold.

Chapter 4. JBS: Software Access Control (SAC)

This chapter introduces the ThruPut Manager Software Access Control extensions to JBS, describes the concepts, and outlines the facilities available. Software Access Control table syntax and usage are explained.

Software Access Control Overview

Software Access Control (SAC) is a ThruPut Manager enhancement to control licence compliancy. A popular use of the job routing ability of ThruPut Manager's Job Binding Services (JBS) function (provided with the TM/JBS component) has been to route batch jobs to the system image(s) where a software product to be executed is licensed. This mechanism, although effective, had two shortcomings:

- It does not handle TSO.
- No "software licencing" view could be provided, either to the administrator or to the end-user.

ThruPut Manager's Software Access Control (SAC) enhances the existing mechanisms to include TSO and to formalize the rules. An upgrade is planned that will provide ISPF dialogs that you can use to request a comprehensive view of how the licenced software is deployed at any particular time within the JES2 node. The mechanism is based on the Binding Agent architecture of JBS:

- For batch, jobs are automatically routed to the correct LPARs.
- For TSO, since commands cannot be routed dynamically like batch jobs can, SAC prevents TSO invocation of software on system images that are not licensed.

How SAC Works

SAC uses an *installation-generated table* (the SAC table) to identify the products that are to be restricted to specific system images. The SAC table links requests to JBS Binding Agents and lets you specify whether the request is failed or allowed to proceed with a warning. You can control requests according to the way they are made. The facilities for TSO are:

- ISPF_PANEL, to control the invocation of specific ISPF panels.
- ISPF_PGM, for programs that are invoked only from ISPF.

- ISPF_COMMAND, for TSO commands issued from ISPF.
- READY_COMMAND, for programs and commands invoked only from a TSO READY prompt (not under ISPF).
- COMMAND, for programs and commands that can be invoked from a TSO READY prompt or under ISPF.

The facilities for batch include matches for:

- Account.
- Accounting field.
- DD name.
- Dataset (DS) name.
- Step library name or LINKLIST.
- PROCSTEP name.
- STEP name.

For unusual circumstances, you can use TM DAL to determine the requirements and set a global variable. This takes advantage of the fact that SAC has the ability to test a GLOBAL variable similar to that which exists in JAL. You can then use this variable to route the job accordingly. (See the JAL_GLOBAL keyword.)

The SAC table is normally loaded during TMSS initialization, and links a product to a TM/JBS Binding Agent. TMSS enhancement provides an initialization statement and operator commands to manage the SAC table. It is worth noting at this point that the scope of the SAC table is limited to the scope of the JES2 subsystem that loads it.

For batch, you simply make a reference in JAL to request that the SAC table be used to control the routing of the job. The normal JBS mechanisms do the rest.

- In the world of TSO, SAC gets control prior to invocation of a product or execution of a command. If a link to one or more Binding Agents is defined in the SAC table for the product or command, all Agents are queried. If at least one of the Agents is active on the ambient system, invocation of the product is allowed.
- If none of the Agents is active on the ambient system, but at least one of the Agents is defined on any system, installation defined message(s) are issued. Invocation of the product is denied or allowed to proceed with a warning, depending on your settings.

- If none of the Agents linked to the product are defined, or if the query cannot be answered (ThruPut Manager is unavailable, for example), an error message is issued and the request is denied.

The SAC Table

The Software Access Control (SAC) Table is an external table. Facilities to support external tables are described in:

- ***Base Product System Programming Guide*** "Chapter 15. Table (TBL) Function."
- ***Command Reference Guide*** (see TBL commands).

Table Coding Syntax

The SAC table can be coded in "free-flowing" format, meaning that they can start in any column and allow any number of intervening blanks between keywords. The conventions used are:

- Continuations are denoted by a "+" as the last character on a line.
- Comments must be enclosed within "/*" and "*/".
- Blank lines are ignored.
- 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    ( )
```

- Lower-case letters, words, and symbols appearing in the format model represent variables for which specific information is substituted when the parameter is coded. For example:

```
panelid
```

  This means that you should substitute a valid ISPF panel identification for the symbol "panelid".

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

```
FAIL(msgid1[,msgid2,...])
```

  is the format for the FAIL keyword. This indicates that when entering the FAIL keyword, the second and subsequent message identifiers are optional.

- Where masking is allowed:
  - The '?' (question mark) masking character represents any single character, but not the absence of a character.
  - The '*' (asterisk) masking character can appear at the beginning of a string, the end, or both. It cannot be embedded in a string. It represents any number of characters, including the absence of a character.

Table Structure

The SAC table is divided into sections by TYPE statements that indicate the type of definitions that follow.

- You must include, in this order:
  - TYPE(MSGID)
  - TYPE(PRODUCT)

These statements define the product that is being controlled and the messages issued to describe any control that is exercised.

- You must also include, in any order, one or more of:
  - TYPE(ISPF_PANEL)
  - TYPE(ISPF_PGM)
  - TYPE(ISPF_COMMAND)
  - TYPE(READY_COMMAND)
  - TYPE(COMMAND)
  - TYPE(BATCH_PGM)

  These statements describe where Software Access Control is applied, and what actions are taken.
- You can also include statements to modify the behavior of SAC itself:
  - TYPE(JES2_NAMES)

    This statement allows you to map JES2 system names to names of your choice.
  - TYPE(EXEMPT_USERS)

    This statement allows you to specify TSO userids that are exempt from SAC checking.
- A TYPE statement must precede any definitions for its type.

- Definitions are terminated by the next TYPE statement or by the end of the SAC table.
- You cannot specify a particular TYPE more than once.

Associating Table Statements with a System

The FOR/ENDFOR statements allow you to share common SAC table statements among multiple systems. Use this facility only when you have unique statements for a particular system. The format is as follows:

```
FOR SMFID(smfid-list) | JES2_NAME(jes2name-list) | SYSNAME(systemname-list)
      statement
      statement
      …
ENDFOR
```

smfid-list

Is a list of one or more SMFIDs, separated by commas, identifying the systems that should honor the SAC table statements that follow.

jes2name-list

Is a list of one or more 1-4 character JES2 names, separated by commas, identifying the systems that should honor the SAC table statements that follow.

systemname-list

Is a list of one or more 1-8 character system names, separated by commas, identifying the systems that should honor the SAC table statements that follow.

To terminate the FOR condition, use the ENDFOR statement.

The rules for using the FOR group facility are as follows:

- A FOR group and its associated definitions must be contained entirely within a TYPE statement.
- Statements that are not within a FOR group are always applicable regardless of the system that processes them and are cumulative with statements within the FOR group.
- Nested FOR groups are not allowed. If another FOR statement is encountered before an ENDFOR terminates the previous group, it is an error.

- The FOR and ENDFOR statements can begin in any column of the record. Only columns 1 through 72 are scanned.

If the identification for the system processing a FOR group does not match the FOR identification, the statements are ignored—almost. The statements receive simple checking, so they must conform to SAC table syntax. Unless there is a syntax error, however, these statements are totally ignored.

Mandatory TYPE Statements

TYPE(MSGID)

This section defines message(s) that might be issued as a result of SAC rules. For products invoked in TSO, messages are issued to the user with PUTLINE, and you can also log messages to SYSLOG. For products invoked in batch, you can direct messages to any or all of JOBLOG, SYSLOG, and SYSMSGS. The syntax is:

```
msgid(message-text)
```

msgid

Is 1-8 alphanumeric characters that identify the message text. This is used to refer back to the message text in subsequent sections of the SAC table. The identification must be unique.

message-text

Is up to 256 characters of message text, enclosed in apostrophes. To code an apostrophe, use two consecutive apostrophes. If a message exceeds this length after the variables are expanded, the message is truncated. If a message results in a text line with all blanks, the message is not issued.

You can use the following inserts, which expand when the message is issued:

&AGENT

The associated JBS Binding Agent name(s).

&CURRENT

The JES2 system name under which the message is issued. Note that you can use TYPE(JES2) to map the JES2 names to names that might be more recognizable to users.

&ENTITY

The current program/panel/command with its TYPE, as specified by the definition invoking the message. For example, this could expand to:

```
ISPF_COMMAND(IMS)
```

&JOBNAME

The job name, or for TSO, the userid.

&JOBNUMBER

The JES2 job number in the format JOBnnnnn or Jnnnnnnn, or the TSU session id in the format TSUnnnnnor Tnnnnnnn.

&NAME

The current program/panel/command without its TYPE, as specified by the definition invoking the message. For example, this could expand to:

```
IMS
```

&PARM

The current parameter field for ISPF SELECT service.

&PRODUCT

The current product name, as specified by the PRODUCT keyword in the definition invoking the message.

&SYSTEMS

A list of JES2 system name(s) on which invocation is allowed. If invocation is not permitted on any system, this variable expands to NONE. Note that you can use TYPE(SYSTEM_NAMES) to map the JES2 names to names that might be more recognizable to users.

Here are some sample message definitions:

```
MSG1('YOU CANNOT USE &PRODUCT ON &CURRENT')
MSG2('&PRODUCT IS AVAILABLE ON &SYSTEMS')
MSG3('&AGENT IS NOT ACTIVE ON &CURRENT AT THIS TIME')
```

TYPE(PRODUCT)

In this section you define a particular product. You give the product a name, for example COBOL, and define the Binding Agents that control the routing of jobs, or in the case of TSO, that allow or disallow invocation. The syntax is:

```
product-name AGENT(name1[,name2,...])
             [SAF_AUTHORITY(ALTER | CONTROL | READ | UPDATE)
             [SAF_CLASS(class)
             [SAF_RESOURCE(resource)
             [FAIL_MSG(msgid1[,msgid2,...])]
             [WARN_MSG(msgid1[,msgid2,...])]
             [LOG_MSG(msgid1[,msgid2,...])]
             [DISPLAY_ONLY]
             [NOTES('Up to 50 characters of text')]
```

product-name

Identifies the product. The identification must be unique. Specifies up to 24 alphabetic, numeric, national, or underscore (_) characters.

AGENT(name1],name2,...])

The name of the Binding Agents to be associated with this product. Up to 4 agents can be specified. The agents must have been defined to JBS. Following JBS conventions, this represents an 'OR' list of Agents.

SAF_AUTHORITY(ALTER | CONTROL | READ | UPDATE)

Indicates that you want to check the access authority.

ALTER

Checks whether the SAF user or group has total control over the resource.

CONTROL

Checks whether the SAF user or group has CONTROL authority for the resource.

READ

Checks whether the SAF user or group can open the resource only to read. This is the default.

UPDATE

Checks whether the SAF user or group can open the resource to read or write.

SAF_CLASS(class)

Specifies that SAF authorization checking is to be performed for a resource of the specified class.

class

Is a 1-8 byte SAF resource class.

SAF_RESOURCE(resource)

Specifies that SAF authorization checking is to be performed for the specified resource.

'resource'

Is a 1-255 byte SAF resource entity.

FAIL_MSG(msgid1[,msgid2,...])

This is the default message or messages to be issued when FAIL is requested in a subsequent section. This keyword is applicable to TSO. The message(s) are displayed (using PUTLINE) before the request is failed. You can specify up to 24 message IDS, as defined in the TYPE(MSGID) section.

WARN_MSG(msgid1[,msgid2,...])

This is the default message or messages to be issued when WARN is requested in a subsequent section. This keyword is applicable to TSO. The message(s) are displayed (using PUTLINE) before the request proceeds. You can specify up to 24 message IDs, as defined in the TYPE(MSGID) section.

LOG_MSG(msgid1[,msgid2,...])

This is the default message or messages to be issued when LOG is requested in a subsequent section. This keyword applies to both batch and TSO. Use it to log such things as access patterns and product usage to SYSLOG. You can specify up to 24 message IDs, as defined in the TYPE(MSGID) section.

DISPLAY_ONLY

Use this keyword when access control is not exercised with SAC, but you want the information about this product to be available to the administrator for display using ISPF dialogs (available in a future release). If this keyword is included, no other SAC action is taken.

NOTES('Up to 50 characters of text')

Use this keyword for annotations, which can be displayed in ISPF dialogs (available in a future release).

Optional TYPE Statements

TYPE(section-type)

The section types described below are optional, but you must include at least one of this form of TYPE statement. These types associate specific ways of invoking licensed products and commands with a previously defined product.

The following section types control TSO:

ISPF_PANEL

Specifies the name of the panel that ISPF is about to select.

ISPF_PGM

Specifies the name of a program invoked within ISPF.

ISPF_COMMAND

Specifies the name of a command invoked from within ISPF.

☞ ***NOTE: The above types depend on the contents of the ISPF SELECT statement used to invoke the product.***

READY_COMMAND

Specifies a program or command invoked from a TSO READY prompt (not from within ISPF).

COMMAND

A combination of ISPF_COMMAND and READY_COMMAND.

For each type, you can specify one or more associations of product or command with JBS Binding Agent(s).

☞ ***Some products can be invoked in more than one way, therefore to gain complete control you must specify an association for each method that can be used to invoke the product.***

For batch there is only one section type:

TYPE(BATCH_PGM)

TYPE Statements for SAC Control

TYPE(JES2_NAMES)

This optional section maps the JES2 system name into a meaningful name to be used when expanding the message inserts &CURRENT and &SYSTEMS. For example, if users refer to a system with the JES2 name SYS1 as the PRODUCTION system, you can map SYS1 into the name PRODUCTION.

The format is:

```
jes2_name(user_name)
```

jes2_name

Is the 1-4 character JES2 name for the system to be mapped.

user_name

Is 1-24 alphabetic, numeric, or national characters specifying the new name. You can include as many name mappings as you need, but each JES2 system name can be specified only once.

TYPE(EXEMPT_USERS)

This optional section defines a list of TSO userids, separated by commas, that are exempt from SAC checking. The syntax is:

```
userid1[,userid2,...,useridn]
```

Masking characters are allowed.

TSO Access Control

The general syntax for TSO is simple:

```
TYPE(section-type)
```

section-type

Specifies the type of invocation in TSO to which SAC control is applied, and must be one of:

```
ISPF_PANEL
ISPF_PGM
ISPF_COMMAND
READY_COMMAND
COMMAND
```

The TYPE statement must be followed by one or more invocation descriptions, which specify which product is to be controlled and what actions to take when the product is invoked. The descriptions follow this format:

```
invocation_name PRODUCT(name)
                [FAIL[(msgid1[,msgid2,..])] |
                 WARN[(msgid1[,msgid2,...])] |
                 NOT_ALLOWED(msgid1[,msgid2,...])]
                [DISPLAY_ONLY]
                [NOTES('Up to 50 characters of text')]
                [LOG[(msgid1[,msgid2,...])]]
```

invocation_name

The identification panel name or command name. For example: IMS@USER. Masking characters are allowed.

PRODUCT(name)

Is the name of the product, as previously defined in the TYPE(PRODUCT) section.

PARM(parms) | PARMFLD(nn,parms)

Is the parameter field from the EXEC statement. PARM has the same meaning as PARMFLD(1,parms).

Note: This description applies only to TYPE(ISPF_PGM)

parms

Is a string to be matched against a parameter field from the EXEC statement. Masking characters are allowed.

nn

Is the field reference number of the parameter field and must be specified as decimal digit(s) 1-50.

FAIL[(msgid1[,msgid2,...])]

If none of the Binding Agents are active on the ambient system, this keyword causes the request to fail. The message IDs are optional and override the default messages defined in the TYPE(PRODUCT) section. They are displayed (using PUTLINE) before denying the request. You can specify up to 24 message IDs, as defined in the TYPE(MSGID) section. This keyword is mutually exclusive with NOT_ALLOWED and WARN.

WARN[(msgid1[,msgid2,...])]

If none of the Binding Agents are active on the ambient system, this keyword allows the request to proceed, but with a warning. The message IDs are optional and override the default messages defined in the TYPE(PRODUCT) section. They are displayed (using PUTLINE) before the request is allowed. You can specify up to 24 message IDs, as defined in the TYPE(MSGID) section. This keyword is mutually exclusive with NOT_ALLOWED and FAIL.

NOT_ALLOWED(msgid1[,msgid2,...])

This keyword restricts access to any user that is not exempted. This keyword is mutually exclusive with FAIL and WARN.

☞ ***NOTE: You must specify at least one of FAIL, WARN, NOT_ALLOWED, or LOG.***

DISPLAY_ONLY

Use this keyword to indicate that access control is not exercised with SAC, but you want the information about this product to be available to the administrator for display using ISPF dialogs (available in a future release). If this keyword is included, no other SAC action is taken.

NOTES('Up to 50 characters of text')

Use this keyword for annotations, which can be displayed in ISPF dialogs (available in a future release).

LOG[(msgid1[,msgid2,...])]

This keyword can be used for logging information to evaluate access patterns, product usage, etc. in SYSLOG. The message IDs are optional and override the default messages defined in the TYPE(PRODUCT) section.

Notes:

- The entries in each section are sorted alphabetically, most specific first. Entries containing masking characters follow entries without masking.
- A CLIST or REXX that is implicitly invoked with a proceeding "%" is not eligible for JBS Agent verification for any of the "COMMAND"table sections.
- EXEC (or EX) commands are not eligible for JBS Agent verification.
- TSOEXEC, CALL and TEST are special cases for TSO commands. The command after TSOEXEC and the member name of a CALL or TEST command are the values that are checked against the SAC table.

Batch Access Control

Batch access control requires that you indicate that SAC binding applies to the job. To do this, you must use the JAL statement:

```
JBS BIND FROM_SAC
```

This causes ThruPut Manager to use the batch section of the SAC table. See the full syntax description in the ***JAL Reference Guide***.

The syntax for the statement that indicates the start of the batch section is:

```
TYPE(BATCH_PGM)
```

This statement must be followed by one or more invocation descriptions that specify which product is to be controlled, and what actions to take when the product is invoked.

The descriptions follow this format:

```
program_name PRODUCT(name)
             [ACCT(acctfld) | ACCTFLD(nn,acctfld)]
             [DDNAME(ddname)]
             [DISPLAY_ONLY]
             [DSNAME(dsname)]
             [EXEC_ACCT(acctfld) | EXEC_ACCTFLD(nn,acctfld)]
             [EXEC_PARMDD]
             [JAL_GLOBAL(global)]
             [JOBLOG_MSG(msgid1,[msgid2,...])]
             [JOBNAME(jobname)]
             [LIBRARY(libname) | LINKLIST)]
             [NOT_ALLOWED(msgid1[,msgid2,...])]
             [NOTES('Up to 50 characters of text')]
             [PARM(parms) | PARMFLD(nn,parms)]
             [PROCSTEP(procstepname)]
             [SCHENV(envname | $NO_SCHENV)]
             [STEPNAME(stepname)]
             [SYSAFF_ANY]
             [SYSLOG_MSG(msgid1[,msgid2,...])]
             [SYSMSGS_MSG(msgid1[,msgid2,...])]
```

All matches specified represent “and” conditions. For example, if DDNAME and DSNAME are specified, both must be matched to satisfy the identification of a particular program.

program_name

Is the 1-8 character name of the program that is being executed, for example: LC370B. Masking characters are allowed. If more than one mask creates a match, the ***most specific*** mask is used.

PRODUCT(name)

Is the name of the product, as previously defined in the TYPE(PRODUCT) section.

ACCT(acctfld) | ACCTFLD(nn,acctfld)

Is the accounting information from the JOB statement. ACCT has the same meaning as ACCTFLD(1,acctfld).

acctfld

Is a string to be matched against a accounting related field from the JOB statement. Masking characters are allowed.

nn

Is the field reference number of the accounting related field and must be specified as decimal digit(s) 1-68.

DDNAME(ddname)

Is the 1-8 name of a DD statement in the step that is attempting to run the product. Masking characters are allowed.

DISPLAY_ONLY

Use this keyword to indicate that access control is not exercised with SAC, but you want the information about this product to be available to the administrator for display using ISPF dialogs (available in a future release. This might apply, for example, when detection of a product in batch requires logic that is more complex than is available in SAC. You can make the determination in DAL/JAL instead.

☞ ***If this keyword is included, no other SAC action is taken.***

EXEC_ACCT(acctfld) | EXEC_ACCTFLD(nn,acctfld)

Is the accounting information coded on the EXEC statement for the product. EXEC_ACCT has the same meaning as EXEC_ACCTFLD(1,acctfld).

acctfld

Is a string to be matched against a accounting related field from the JOB statement. Masking characters are allowed.

nn

Is the field reference number of the accounting related field and must be specified as decimal digit(s) 1-68.

Is the PARMDD information coded on the EXEC statement for the product. It is used to recognize programs that have PARMDD= specified. (New with z/OS 2.1.)

DSNAME(dsname)

Is the 1-44 dataset name of a dataset referred to in the step that is attempting to run the product. Masking characters are allowed.

JAL_GLOBAL(global)

Specifies the name of a GLOBAL variable that can be set in TM DAL and tested by SAC. This provides SAC with the same ability to test a GLOBAL

variable that exists in JAL. The variable must be set to TRUE to satisfy the identification of the program.

JOBLOG_MSG(msgid1[,msgid2,...])

This keyword can be used to direct messages to the JOBLOG during job analysis. Note that the listed messages are issued for every matching reference to the product.

JOBNAME(jobname)

Is the 1-8 character job name from the JOB statement. Masking characters are allowed.

LIBRARY(libname)

Is a 1-44 dataset name to be matched with either a JOBLIB or STEPLIB dataset name, if present in the step that is attempting to run the product. If there is both a JOBLIB and a STEPLIB, the STEPLIB takes precedence. Masking characters are allowed. This keyword is mutually exclusive with LINKLIST.

LINKLIST

Tests whether a JOBLIB or STEPLIB is present in the step that is attempting to run the product. This keyword is mutually exclusive with LIBRARY.

NOT_ALLOWED(msgid1[,msgid2,...])

This keyword indicates that any batch job referring to this product is not allowed to execute.

NOTES('up to 50 characters of text')

Use this keyword for annotations, which can be displayed in ISPF dialogs (available in a future release).

PARM(parms) | PARMFLD(nn,parms)

Is the parameter field from the EXEC statement. PARM has the same meaning as PARMFLD(1,parms).

parms

Is a string to be matched against a parameter field from the EXEC statement. Masking characters are allowed.

nn

Is the field reference number of the parameter field and must be specified as decimal digit(s) 1-50.

PROCSTEP(procstepname)

Is the 1-8 character step procedure name on the EXEC statement. Masking characters are allowed.

SCHENV(envname | $NO_SCHENV)

This keyword allows the setting of a scheduling environment when an entry is located, or ignore a SCHENV keyword coded on the JOB statement.

envname

A valid scheduling environment name, which can be 1-16 alphanumeric, national ($, #, @) characters, or the underscore (_). Underscores (_) must be imbedded.

$NO_SCHENV

A reserved word that indicates that a SCHENV keyword coded on the JOB statement is to be ignored.

STEPNAME(stepname)

Is the 1-8 character stepname of the statement that is attempting to run the product. Masking characters are allowed.

SYSAFF_ANY

Specifies that SAC should duplicate the effect of a JBS SET SYSAFF(ANY) statement in JAL.

SYSLOG_MSG(msgid1[,msgid2,...])

This keyword can be used to direct messages to the SYSLOG during job analysis. Note that the listed messages are issued for every matching reference to the product.

SYSMSGS_MSG(msgid1[,msgid2,...])

This keyword can be used to direct messages to SYSMSGS during job analysis. Note that the listed messages are issued for every matching reference to the product.

Note:

- The entries are sorted alphabetically, most specific first. Entries containing masking characters follow entries without masking.

Sample SAC Table

The following example shows how a SAC table for TSO might look for your installation.

In this hypothetical installation, they have the following situation:

- Three LPARS.
- They are known to JES2 as SYS1, SYS2, and SYS3.
- For historical reasons, users know these systems as the OLD system, the PRODUCTION system, and the TEST system, respectively.
- The COBOL compiler is licenced on SYS1 .
- New implementations of COBOL are tested on SYS2. If SYS1 is not available then SYS2 runs as backup for SYS1. When this happens, no testing is allowed.
- Licenced product PRODX is to run on SYS1 and SYS2 for batch, and only on SYS1 for TSO.
- Licenced product PRODY is only available on SYS3.
- Product PRODZ is licenced for all LPARS, but the administrator wants to see its usage pattern to see if he can reduce the number of licences.
- Product GONE is to be replaced by a PC-based offering and will not be available after November 30th.
- There are a number of user IDs for administration purposes that are to be exempted.
- This list for SYS3, the TEST system, is expanded to include the user IDs for the development team.

The following example shows an implementation of the SAC table to handle the above requirements.

```
                    /* SOFTWARE ACCESS CONTROL TABLE */

TYPE(MSGID)
      MSG1('USR001I &PRODUCT not licensed for system &CURRENT')
      MSG2('USR002I &PRODUCT cannot be used on system &CURRENT')
      MSG3('USR003I Please use one of the following systems: &SYSTEMS')
      MSG4('USR004I COBOL testing is available on: &SYSTEMS')
      MSG5('USR005I Licenced Software routing in effect for: &PRODUCT')
      MSG6('USR006I Invocation via TSO command not allowed')
      MSG7('USR007I This product will not be available after Nov 30th')
      MSG8('USR008I For a PC-based replacement contact your LAN admin')
      MSG9('USR009I &JOBNUMBER &JOBNAME REFERS TO &PRODUCT')

   /* PRODUCT DEFINITION SECTION - MUST FOLLOW THE MESSAGE SECTION */
TYPE(PRODUCT)
  COBOL /* IDENTIFIES PRODUCTION COBOL */ +
      NOTES('COBOL PRODUCTION')           +
      AGENT(COBOL.PRIMARY,COBOL.SECOND)   +
      FAIL_MSG(MSG1) /* MESSAGE FOR TSO USERS */
  COBOL_NEW /* IDENTIFIES NEW VERSION OF COBOL TO TEST */ +
      NOTES('COBOL TEST')                                 +
      AGENT(COBOLT.PRIMARY)                               +
      FAIL_MSG(MSG2,MSG4)
  PRODXB /* IDENTIFIES PRODUCT X RULES FOR BATCH */ +
      NOTES('PRODUCTX BATCH')                       +
      AGENT(PRODXB.PRIMARY,PRODXB.SECOND)
  PRODXTSO /* IDENTIFIES PRODUCT X RULES FOR TSO */ +
      NOTES('PRODUCTX FOR TSO')                     +
      AGENT(PRODXTSO.PRIMARY)                       +
      FAIL_MSG(MSG2,MSG3)
  PRODY +
      NOTES('PRODUCTY')                             +
      AGENT(PRODY.PRIMARY)                          +
      FAIL_MSG(MSG1)
  PRODZ /* FOR THIS PRODUCT ONLY LOGGING FOR USAGE IS IN EFFECT */ +
      NOTES('PRODUCTZ')                                            +
      AGENT($DUMMY)                                                +
      LOG_MSG(MSG9)
```

Sample Software Access Control (SAC) Table

```
  GONE              +
      AGENT($DUMMY) +
      WARN_MSG(MSG7,MSG8)

                /* THIS SECTION PROVIDES SYSTEM NAME MAPPING */

TYPE(JES2_NAMES)
     SYS1(OLD)
     SYS2(PRODUCTION)
     SYS3(TEST)

  /* THIS SECTION ASSOCIATES PANELS, COMMANDS AND PROGRAMS WITH PRODUCTS */
               /* THE TYPE STATEMENTS CAN BE IN ANY ORDER */
              /* EACH PARTICULAR TYPE CAN APPEAR ONLY ONCE */

TYPE(BATCH_PGM)
  IGYCRCTL PRODUCT(COBOL) JOBLOG_MSG(MSG5)
  IGYNEW   PRODUCT(COBOL_NEW) JOBLOG_MSG(MSG5)
  STATS*   PRODUCT(PRODXB) JOBLOG_MSG(MSG5) +
           DDNAME(STAT*)
  RMS*     PRODUCT(PRODY) JOBLOG_MSG(MSG5) +
           DDNAME(RMS*)                    +
           DSNAME(RMS*.*)
  AAWW*    PRODUCT(PRODZ) SYSLOG_MSG(MSG9)
  GONE*    PRODUCT(GONE) JOBLOG_MSG(MSG7,MSG8)

TYPE(ISPF_PANEL)
  IGYFPO2  PRODUCT(COBOL) FAIL
  IGYFPO2T PRODUCT(COBOL_NEW) FAIL
  GONE*    PRODUCT(GONE)
  STATPO2  PRODUCT(PRODXTSO) FAIL
  RMS?PO1  PRODUCT(PRODY) FAIL
  AAWW?PO3 PRODUCT(PRODZ) LOG

TYPE(ISPF_PGM)
  IGYCRCTL PRODUCT(COBOL) FAIL
  IGYNEW   PRODUCT(COBOL_NEW) FAIL
  STATS*   PRODUCT(PRODXTSO) FAIL
```

Sample Software Access Control (SAC) Table (Continued)

```
  RMS*     PRODUCT(PRODY) FAIL
  AAWW*    PRODUCT(PRODZ) LOG

TYPE(ISPF_COMMAND)
  STATCALC PRODUCT(PRODXTSO) FAIL

TYPE(READY_COMMAND)
  STATCALC PRODUCT(PRODXTSO) NOT_ALLOWED(MSG6)

TYPE(COMMAND)
  COBOL    PRODUCT(COBOL) FAIL
  COBOLNEW PRODUCT(COBOL_NEW) FAIL
  RMSCALL  PRODUCT(PRODY) FAIL
  AAWWLOAD PRODUCT(PRODZ) LOG

TYPE(EXEMPT_USERS) /* IDENTIFY SPECIAL USER ID'S THAT ARE EXEMPT */
  FOR JES2_NAME(SYS3)
    DEV*
  ENDFOR
    USER01,BACKUP*,APP??XY
```

Sample Software Access Control (SAC) Table (Continued)

Chapter 5. JBS: Environment Services

This chapter describes TM/JBS Environment Services. It explains basic concepts and discusses how JBS Environments can replace and extend the function provided by IBM's Resource Affinity Scheduling.

Prerequisites

This chapter assumes that you are already familiar with the latest version of the IBM document:

- ***z/OS Planning: Workload Management***

JBS Environments are intended to provide extended capabilities to users of IBM scheduling environment (SCHENV) support. If you are not already using scheduling environments, TM/JBS Binding Agents are more suitable and more flexible.

What is a JBS Environment?

A ***JBS Environment,*** similar to an IBM scheduling environment, is a list of Resource Elements and their desired State. When all Resource Elements for an Environment are set to the desired state, the JBS Environment is available and jobs requiring that Environment are eligible to run.

A job is assigned a JBS Environment through JAL or JECL. JBS Environments are roughly equivalent to scheduling environments, but provide more flexibility.

JBS Environments vs. Scheduling Environments

The scope of SCHENV differs from that of JBS. The IBM implementation makes a scheduling environment and its constituent Resource Elements known throughout the SYSPLEX. The scope of JBS Environment Agents and Resource Elements is that of the JESplex.

Users either have to specify SCHENV or accept the single default that can be associated with the job's class. Users do not have to specify JBS Environments, since they are normally applied to the job through JAL directives. JBS Environments can be requested through a JECL statement, but the request is overridden if an Environment is provided by JAL.

Only one SCHENV can be associated with a job. You can specify up to four JBS Environments, which are treated like an OR condition, therefore any of the specified Environments can satisfy the requirement.

What Is a Resource Element?

A ***Resource Element*** is a logical entity that can represent any characteristic of the environment that you wish to use to schedule work. This is the same basic definition that a Resource Element has in a scheduling environment (SCHENV).

JBS Resource Elements vs. SCHENV Resource Elements

IBM Resource Elements in a scheduling environment can have one of three states: ON, OFF, and RESET. The significance of the ON and OFF states is arbitrary, and the only means of providing some context is by your choice of the Element name.

JBS Resource Elements allow you to name up to 8 states. For example, you could name a state for each day of the week. Because these are all states of a single Resource Element, there is no chance of inadvertently setting more than one state on at a time. This simplifies definitions and environment management, as well as providing a meaningful name for the state. Like SCHENV Resource Elements, a JBS Resource Element can be set to a different state on each one of multiple systems, but only one specific state on any one system.

JBS Resource Elements can be combined in an Environment using both AND and OR conditions. SCHENV Resource Elements are limited to just the AND condition.

SCHENV Resource Elements let you define a resource list that has conflicting combinations of resources and is therefore impossible to implement. ThruPut Manager checks for conflicts in JBS Resource Elements and if any are found, prompts you to correct such a list.

JBS Resource Elements provide flexibility not found with SCHENV Resource Elements. Operators control SCHENV Resource Element states on each system, but JBS Resource Elements also let jobs set the state, either at step or job termination or as the result of a message by using the Application Program Interface (API). SCHENV Resource Elements automatically revert their state to RESET after an IPL, but JBS Resource Elements can either be RESET or, optionally, maintain the state that was set before the IPL.

Components to Support JBS Environments

The components provided to support JBS Environments are listed below.

JBS Environment Definition ISPF Dialog

JBS Environments are defined, managed, and installed using an ISPF dialog, described later in this document.

JAL for JBS Environments

JAL is used to apply JBS Environments, while Resource Elements are manipulated with operator commands and JECL. JAL includes the following facilities to support JBS Environments:

- The SET SCHENV Action statement supports the $NO_SCHENV reserved word. When $NO_SCHENV is specified, the SCHENV keyword on the JOB statement is nullified.

  This allows conversion from SCHENV to JBS Environments without any JCL changes.

- A JAL statement allows you to request one or more JBS Environments for a job:

  ```
  JBS NEEDS ENVIRONMENT(environ1[,environ2,environ3,environ4])
  ```

  Multiple Environment requests are treated as a logical OR, that is, any one of the Environments can satisfy the request.

  A request for an environment can be negated by the statement:

  ```
  JBS NEEDS NO_ENVIRONMENT
  ```

  If an Environment has been requested through JECL, any setting made in JAL takes precedence.

- A JAL statement allows you to control the action taken when a job is submitted that attempts to request a non-existent Environment:

  ```
  JBS HOLD UNDEFINED_ENVIRONMENTS(YES)
  ```

  If this statement is present, jobs requesting unavailable Environments are requeued in the Job Analysis Class and placed in the MHS_TM hold category ENVIRONMENT, as described below.

- JAL Descriptors allow you to determine information about the JBS Environment of a job:

- $JBS_NEEDS# is a range Descriptor that allows you to check for the presence of a /*JBS NEEDS JECL statement.
- $JBS_SET# is a range Descriptor that returns the number of /*JBS SET JECL statements.
- $JBS_NEEDS is a unique Descriptor that allows you to check whether a specific JBS Environment has been requested.
- $JBS_SET is a unique Descriptor that allows you to check whether s specific state has been set.
- $LIST_JBS_NEEDS is a Display Variable that allows you to display in a message the requested JBS Environment.
- $LIST_JBS_SET is a Display Variable that allows you to display in a message the states that have been set.

Additionally, the Language Processor parameter VERIFY performs a check to determine whether the JBS Environments referenced in the JAL have been defined on the system on which the Language Processor is running.

JECL for JBS Environments

ThruPut Manager JECL statements are used to manipulate Resource Elements, although you can also use JECL to *request* a JBS Environment:

```
/*JBS NEEDS environ1[,environ2,environ3,environ4]
```

☞ ***If JAL assigns a JBS Environment, any JECL request is ignored.***

A request for multiple Environments is treated as a logical OR, that is, any of the requested Environments can satisfy the request.

You can use JECL to set the state of Resource Elements:

```
/*JBS SET resource-name state [AT=timing]
```

These statements can also be triggered through the API interface support for JBS. A complete syntax description of these statements is provided later in this document. For more information about the API interface, see “Chapter 10. Application Program Interface (API).”

MHS for JBS Environments

MHS supports a new type of MHS_TM hold category, qualified by ENVIRONMENT(environment-name).

When an existing set of Environment definitions is replaced, some jobs might have /*JBS NEEDS statements that refer to Environments or Resource Ele-

ments that have been removed. ThruPut Manager identifies such jobs and automatically requeues them to the Job Analysis class, but in MHS_TM hold category, qualified by ENVIRONMENT(environment-name). MHS_TM commands allow you to display and/or release these jobs as a group.

Operator Commands for JBS Environments

Operator commands are provided to control the state of Resource Elements, verify that the definition is correct, and to display the state of Environments:

- /JBS SET sets the state of a Resource Element.
- /JBS RESET lets you set a Resource Element to the RESET state.
- The /JBS DISPLAY ENV command displays JBS Environment details and Resource Element details.
- The /MHS_TM DISPLAY command can now display jobs placed in MHS_TM hold and qualified by Environment name.
- The /MHS_TM RELEASE command can release jobs that have been placed in MHS_TM hold and qualified by Environment name.

  ☞ ***Note that if a job is placed in the TM_HOLD category ENVIRONMENT and the JBS Environment for which it is held is then defined, the job is released automatically.***

- The /JAL VERIFY command can verify that JBS Environments that are referenced in JAL are defined.

Implementation

To implement JBS Environments, you must:

1. Define Environments.
2. Define Resource Elements.
3. Install the JBS Environment definition.
4. Add JECL to jobs requesting specific Environments, and/or add JAL to add Environment requests to jobs. Optionally, add a JBS HOLD statement for errors.
5. Implement operating procedures to ensure that Resource Elements are always set to the desired state.

6. Optionally, use the JAL statement SET SCHENV($NO_SCHENV) to remove the SCHENV keyword if it is coded on the JOB statement.

Defining JBS Environments and Resource Elements

JBS Environments and their Resource Elements are defined using an ISPF dialog. The definition is then installed from the dialog. These definitions exist until a new definition is installed, or until JBS is *cold started*.

JBS Environment definitions are created using ThruPut Manager ISPF services. To invoke ISPF Services, use the command:

```
TMISPF
```

For a complete description of TMISPF, see the chapter "ThruPut Manager ISPF (TMISPF) Services" in the ***Base Product: System Programming Guide.***

```
-  GoTo                                                                Help
------------------------------ ThruPut Manager -------------------------------
                                  Main Lobby
Command ===> 2

1  Applications          - TM Application Dialogs

2  Systems Programmer    - TM Systems Programmer Main Menu

3  Operator Commands     - TM Operator Commands Interface

4  Automation Services   - TM Automation Services Dialog

X  Exit                  - Exit TM Main Lobby
```

The TMISPF Main Lobby

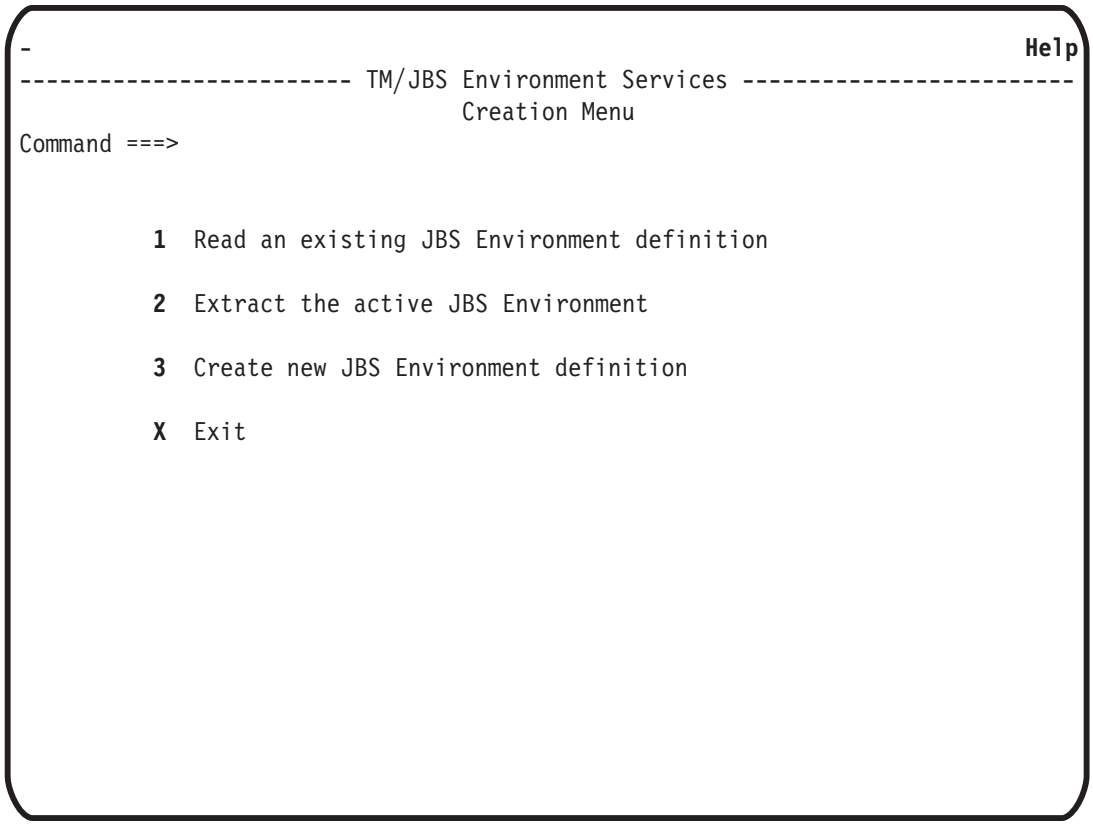
The TMISPF command takes you to the Main Lobby, as shown above.

To define JBS Environments, select **2**, the TM Systems Programmer Main Menu, which looks like this :

```
-  GoTo                                                                Help
------------------------------ ThruPut Manager -------------------------------
                              Systems Programmer Menu

Command ===> 3

   1  CFMU                - Control File Management Utility

   2  JAL Test            - Job Action Language Test Facility

   3  JBS Environments    - JBS Environment Definition and Management

   X  Exit                - Exit Systems Programmer Menu
```

The Systems Programming Services Menu

Selecting **3**, JBS Environment Definition and Management, opens a screen that provides you with three choices, as shown below.



The JBS Environment Services Creation Menu

From the Creation Menu:

- Option **1** reads an existing JBS Environment definition from a sequential dataset or PDS member. You are prompted to enter the name of a sequential dataset or PDS member.
- Option **2** extracts the current JBS Environment from the ambient system.
- Option **3** allows you to create a new JBS Environment from scratch.

The following descriptions are intended to demonstrate one way to create and maintain a JBS Environment and associated Resource Elements. They therefore do not attempt to describe the complete JBS Environment Services dialog. For full details, consult the online help.

Also, you should review the section following this example for notes and considerations concerning JBS Environment definitions.

Creating Resource Elements

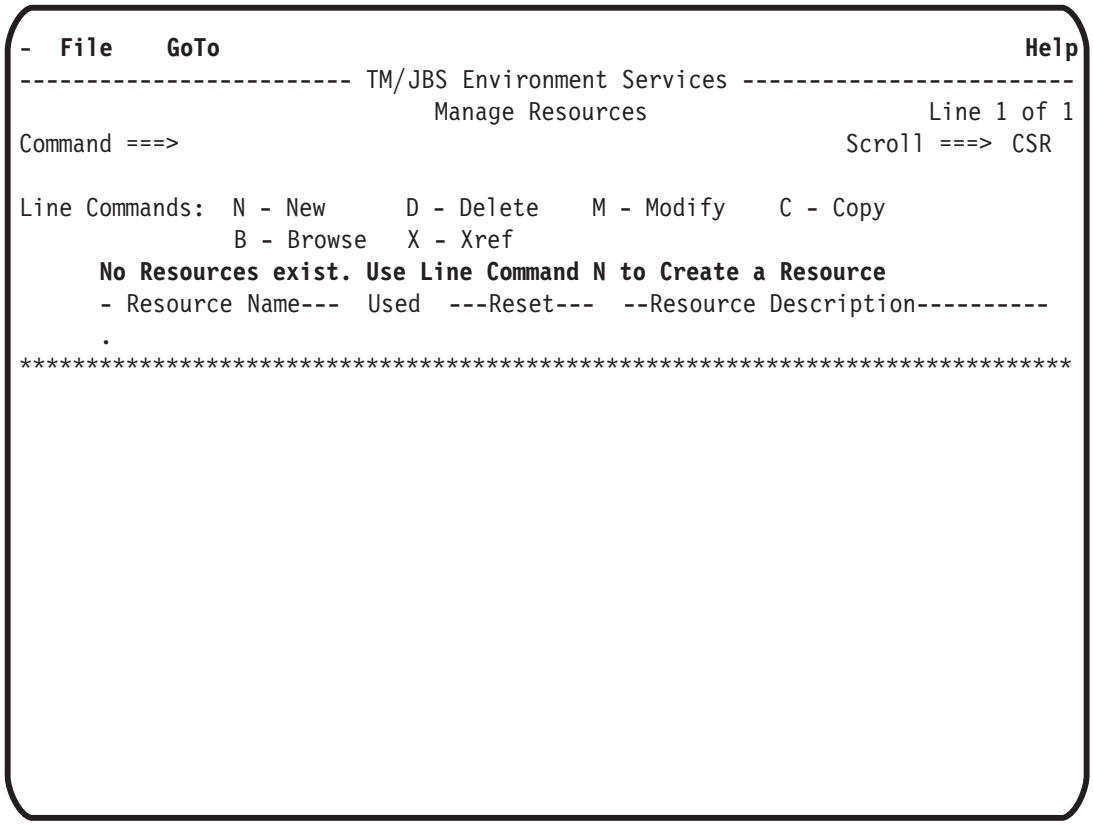
When you select **3** from the Creation Menu, you are presented with two choices, as shown below:

```
-  File    GoTo                                                          Help
------------------------- TM/JBS Environment Services -------------------------
                                    Main Menu
Command ===>

    Environment definitions created from scratch

    No Environments Defined
    No Resources Defined


                1  Manage Environments

                2  Manage Resources
```

The JBS Environment Services Creation Menu

You can choose to define either a JBS Environment or a Resource Element first. Although an Environment requires Resource Elements, the JBS Environment dialog provides the ability to define them when they are needed. Usually, however, it is easier to determine which Resources you want to manage first, then decide how to combine them to form an Environment. We therefore begin this example by selecting item **2** Manage Resources.

The first time you manage Resources, the JBS Environment dialog prompts you to add a Resource Element, as shown here:



The JBS Environment Services Manage Resources Window

Use the **N** line command to add a Resource Element. You can add as many Resource Elements as you need to manage your JBS Environment.

Name the Resource Element with 1 to 16 alphabetic, numeric, or special (@, $ and #) characters . The underscore character (_) is also permitted, but the name cannot start or end with an underscore.

You can provide up to 32 characters of description for your Resource.

A Resource Element retains its state across IPLs, JES2 restarts, and ThruPut Manager restarts unless you indicate otherwise. To have the Resource Element placed in the RESET state after one or more of these events, place an X in the appropriate field(s).

A Resource Element must have states defined. Simply enter a state name of 1 to 8 alphabetic, numeric, or special (@, $ and #) characters . The underscore character (_) is also permitted, but the name cannot start or end with an underscore. Here is a sample dialog with several states defined:

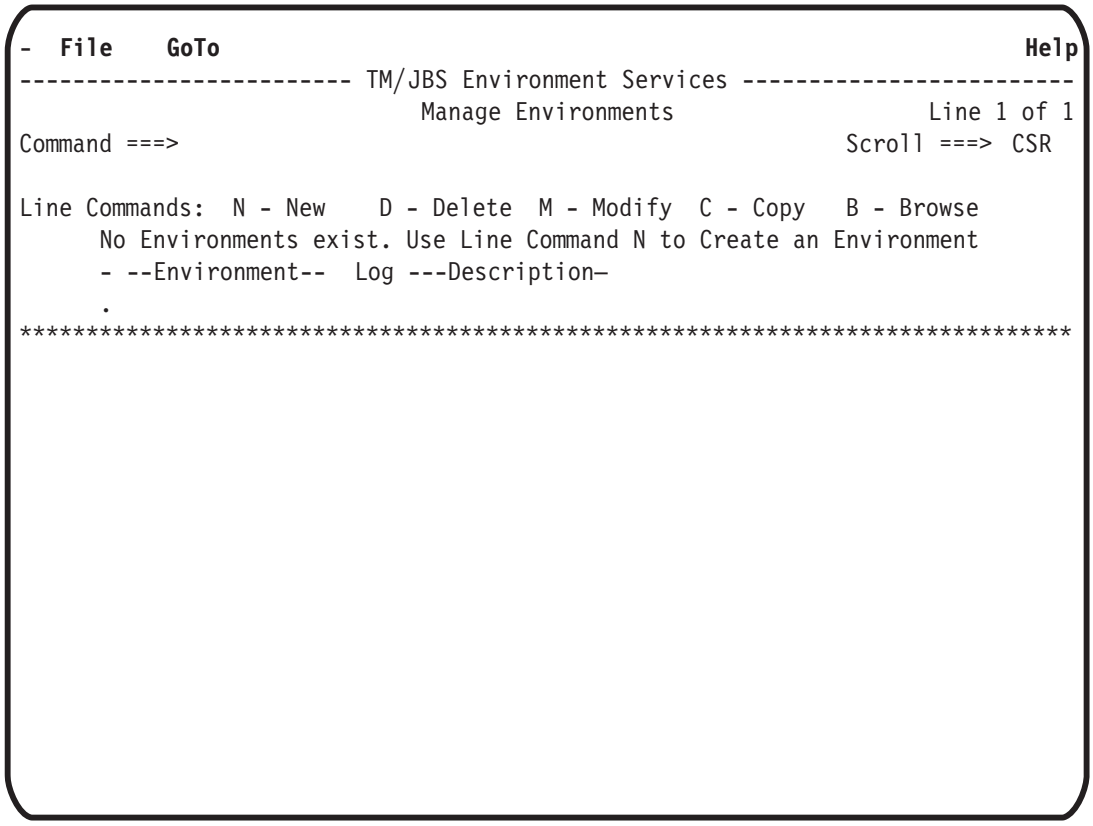
```
-                                                                        Help
------------------------ TM/JBS Environment Services ------------------------
                              Create a Resource      Line 1 of 6 State added
Command ===>                                                 Scroll ===> CSR

Line Commands:  D - Delete  X - Xref

Resource    : DAY_____________
Description : Day of week_____________________

Reset Resource at  IPL: _   JES: _   TM: _

To ADD a State to this Resource
Type State name  Sun_____  and Press Enter

Press END to confirm modifications
Press CANCEL to cancel all modifications made

      - State    In Use
      . FRI
      . MON
      . SAT
      . THU
      . TUE
```

Adding a Resource Element

Once you have defined a Resource Element and its states, you can use it in a JBS Environment definition.

Creating JBS Environments

If there are no JBS Environments defined when you select item **1** Manage Environments from the Main Menu of the JBS Environments dialog, you are prompted to create one with the **N** line command:



The JBS Environment Services Manage Environments Window

Name the Environment with 1 to 16 alphabetic, numeric, or special (@, $ and #) characters. The underscore character (_) is also permitted, but the name cannot start or end with an underscore.

You can provide up to 32 characters of description for your Environment.

You can request that ThruPut Manager log events relating to this Environment in jobs that refer to the Environment. Log entries are sent to the system log by entering an **X** to select the LOG attribute.

```
-                                                                          Help
------------------------- TM/JBS Environment Services -------------------------
                            Modify Environment                    Line 1 of 2
Command ===>                                                 Scroll ===> CSR

Line Commands:  D - Delete          T - Toggle operator
                A - Insert After    B - Insert before

Environment : WEEKLY_BACKUP
Description : Runs Friday nights_____________

Attributes - LOG: _

Press END to confirm modifications
Press CANCEL to cancel all modifications made

      - Op  Resource         RqState  Resource Description
      a
*******************************************************************************
```

The JBS Environment Services Modify Environments Window

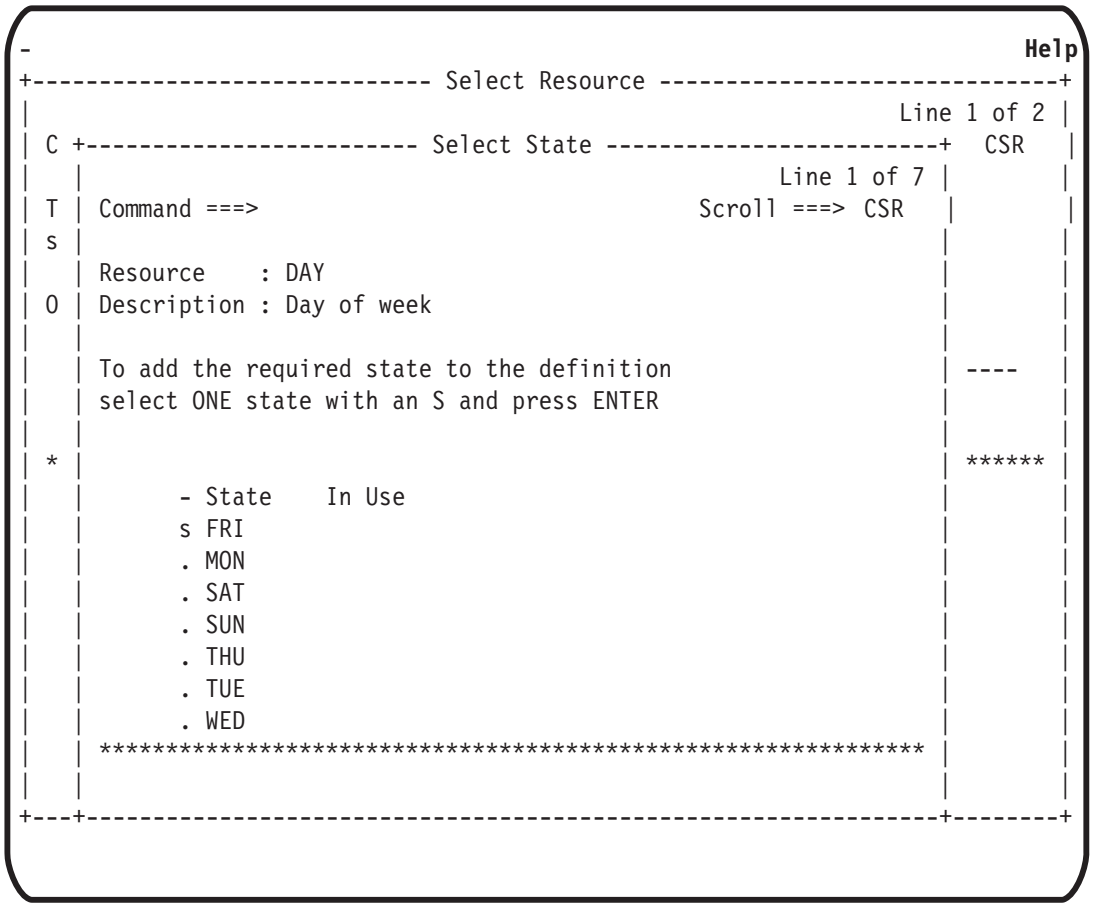
An Environment requires one or more Resource Elements to be set to a specific state. You can add Resource Elements with the **A** or **B** line command.

When the Select Resource window opens, you are presented with a list of the defined Resource Elements to choose from. Note that the **N** line command allows you to define a new Resource Element if that is necessary.

```
-                                                                          Help

+----------------------------- Select Resource ------------------------------+
|                                                                Line 1 of 2 |
| Command ===>                                             Scroll ===> CSR   |
|                                                                            |
| To add a resource to the environment definition                            |
| select ONE resource with an S and press ENTER                              |
|                                                                            |
| OTHER Line Commands:  N - New      D - Delete    M - Modify                |
|                                                                            |
|   - Resource Name---  Used  ---Reset---  --Resource Description----------  |
|   s DAY                                 Day of week                        |
|   . SHIFT             YES   IPL -   -    Operating schedule                |
| ************************************************************************** |
|                                                                            |
|                                                                            |
|                                                                            |
|                                                                            |
|                                                                            |
|                                                                            |
|                                                                            |
|                                                                            |
|                                                                            |
|                                                                            |
+----------------------------------------------------------------------------+
```

Selecting a Resource Element

Once you have selected a Resource Element, you must select a state for it. The dialog opens the Select State window for this purpose:



Selecting a State

You are then returned to the Select Resource window, from which you can select more Resource Elements if desired.

The example below shows that two Resource Elements are used to define the Environment:

```
-                                                                          Help
------------------------ TM/JBS Environment Services ------------------------
                           Modify Environment                  Line 1 of 2
Command ===>                                                Scroll ===> CSR

Line Commands:  D - Delete          T - Toggle operator
                A - Insert After    B - Insert before

Environment : WEEKLY_BACKUP
Description : Runs Friday nights_____________

Attributes - LOG: _

Press END to confirm modifications
Press CANCEL to cancel all modifications made

      - Op  Resource         RqState  Resource Description
      .     DAY              FRI      Day of week
      . and SHIFT            OVERNITE Operating schedule
*******************************************************************************
```

The JBS Environment Services Modify Environments Window

Note that the default operator (shown in the **Op** column) is AND, indicating that both Resource Elements must be set be set to their required state (shown in the **RqState** column) before jobs needing this Environment are eligible to run. You can toggle the operator to OR with the **T** line command. All ORs are evaluated before any AND is applied.

Once you have added all the Resource Elements and set the operators correctly, the END command returns you to the Manage Environments window. Save your Environment using the **File** drop-down menu and selecting item **1**. You are prompted to provide a dataset name or PDS membername for saving:

```
-  File    GoTo                                                          Help
 +--------------------------+ S Environment Services -------------------------
 | 1  1. Save to Dataset    | anage Environments                  Line 1 of 1
 |    2. Install Environment |                                  Scroll ===> CSR
 |    3. Exit               |
 +--------------------------+ Delete  M - Modify  C - Copy   B - Browse
      No Environments exist. Use Line Command N to Create an Environment
      - --Environment--  Log ---Description-
      . WEEKLY_BACKUP    NO  Runs Friday nights
*******************************************************************************


    +---------------------- Specify Output Dataset -----------------------+
    |                                                                      |
    | Command ===>                                                         |
    |                                                                      |
    | Enter OUTPUT Data Set Information                                    |
    | Press ENTER to accept, END or CANCEL to return                       |
    |                                                                      |
    | Data Set Name ===> ENVIRONS                                          |
    |   Member Name ===> WKLYBKUP                                          |
    |                                                                      |
    |                                                                      |
    +----------------------------------------------------------------------+
```

Saving Your JBS Environment

Installing a JBS Environment

To install your JBS Environment, you also use the **File** menu. When a JBS Environment definition replaces an existing definition, the existing definition is compared to the new definition. The number of added, replaced, or deleted definitions is displayed, as well as the number of jobs that are affected by the change:

```
-  File    GoTo                                                          Help
 +--------------------------+ S Environment Services -------------------------
 | 2  1. Save to Dataset    | anage Environments                  Line 1 of 1
 |    2. Install Environment |                                Scroll ===> CSR
 |    3. Exit               |
 +--------------------------+ Delete  M - Modify  C - Copy   B - Browse
     No Environments exist. Use Line Command N to Create an Environment
     - --Environment--  Log ---Description—
     . WEEKLY_BACKUP    NO  Runs Friday nights
*******************************************************************************


       +---------- Confirm Environment Definition Install -----------+
       |                                                             |
       | Command ===>                                                |
       |                                                             |
       | Environments:     1 Added     0 Replaced    46 Deleted      |
       | Resources   :     1 Added     1 Replaced    19 Deleted      |
       | Jobs        :     0 MHSHeld     0 Held     0 Released       |
       |                                                             |
       | Please Confirm installation of environment                  |
       |                                                             |
       | Press ENTER to confirm the request                          |
       | Press CANCEL or EXIT to cancel the request                  |
       |                                                             |
       +-------------------------------------------------------------+
```

Installing Your JBS Environment

At this point you are given the option of continuing with the definition. The display shows how many jobs the new definition causes to be held and/or released.

- ***Added*** is the number of new Environments or Resources introduced by the new definition.

- ***Replaced*** is the number of Environments or Resources that are in the new definition and also in the old definition.
- ***Deleted*** is the number of Environments or Resources that existed in the old definition, but do not exist in the new definition.
- ***MHSHeld*** is the number of jobs that referred to Environments that will be deleted if the new definition is accepted. These jobs will be requeued in the primary Analysis class and placed in MHS HOLD.
- ***Held*** is the number of jobs that will be held for Environment considerations if the new definition is accepted. These jobs were not held for Environments prior to this definition, but might have been held for other reasons.
- ***Released*** is the number of jobs that are held for Environment considerations that will be released if the new definition is accepted. These jobs might still be held for other reasons.

Notes and Considerations

Defining JBS Environments

- The JBS Environment is available on the ambient system when the combination of Resource Elements and their desired states evaluates to TRUE.
- If multiple Resource Element states are specified, they are handled as follows:
  - All ORs are evaluated first. The result of an OR is then treated as a separate element for purposes of resolving AND operators.
  - All AND pairs (which now include the results of any OR pairs) are then resolved.
- JBS Environments are ignored in JECL when ThruPut Manager is deciding whether the job is eligible for transmission to another node.

Undefined JBS Environments

Normally, when a job is submitted that requests a JBS Environment that is not defined, it is canceled; however, you can change this behavior with the JAL statement:

```
JBS HOLD UNDEFINED_ENVIRONMENTS(YES)
```

If this statement is present, jobs requesting undefined JBS Environments are requeued in the Job Analysis class and placed in the MHS_TM hold category and qualified by the missing Environment name, as described below.

Deleted JBS Environments

When you install a new JBS Environment definition, you might omit some Environments that existed in the previous definition. If a job that has already been through Job Analysis refers to a deleted Environment, then ThruPut Manager requeues the job in the Job Analysis Class and places it in the MHS_TM hold category, qualified by the missing Environment name. See below for further information.

JBS Environments and MHS

A new JBS Environment definition can implicitly remove existing Environments that have been requested by jobs awaiting execution. When a new Environment definition is installed, ThruPut Manager detects any missing Environments and requeues the affected jobs in the Job Analysis Class in MHS_TM hold, qualified by the missing Environment name. When you are installing a new JBS Environment, the message DTM6463A informs you how many jobs (if any) will be placed in MHS_HOLD as a result of the new definition.

While jobs are in MHS_TM hold for missing Environments, you can install a corrected definition that revives the missing Environment. When this happens, the MHS_TM hold for the restored Environment is removed automatically.

Defining Resource Elements

- If your Resource Element definition includes a state that is already set in the active definition, that state persists.
- If your Resource Element definition eliminates a state that is currently set in the active definition and your new definition does not set a state for the Resource Element, it is set to RESET.
- Any Resource Element for which no initial state is defined assumes the RESET state, unless it already exists in an active definition, in which case it will retain its current setting.

Undefined Resource Elements or States

If a job is submitted that includes the JECL statement /*JBS SET with a reference to a Resource Element or state that is not defined, the job is failed with a JECL error. Note that the effect of the JAL statement JBS HOLD does ***not*** extend to undefined Resource Elements and states.

Deleted Resource Elements or States

When you install a new JBS Environment definition, you might omit some Resource Elements or states that existed in the previous definition. If a job that has

already been through Job Analysis refers to a deleted Element or state, the reference is effectively removed from the job. ***This removal is permanent***, that is, installing a new definition that contains the referenced Resource Element or state does not revive the reference. To restore the reference, the job must be re-analyzed.

Converting from IBM Scheduling Environments

As described in the introductory chapter, JBS Environments can be considered equivalent to a scheduling environment. Conversion to JBS Environments therefore require that you remove or negate the SCHENV keyword on the requesting job's JOB statement, and substitute a JBS Environment. To make this change transparent to your users, the JAL Action statement SET SCHENV supports the reserved word $NO_SCHENV:

```
SET SCHENV($NO_SCHENV)
```

This statement nullifies the SCHENV keyword on the JOB statement. You can then use JAL to insert an Environment request that replicates the scheduling environment:

```
IF ($IN_SCHENV(PRIME)) THEN
  JBS NEEDS ENVIRONMENT(PRIME)
  ...
ENDIF
```

In most cases, it is likely that you can use the same name for the JBS Environment that you were using for the scheduling environment.

You can use the same names for JBS Resource Elements that you used for the SCHENV Resource Elements and name the states ON and OFF. While this provides compatibility, it does not take advantage of the ability to use your own names for JBS Resource Element states.

For example, assume you wanted to distinguish weekdays from weekends. Using SCHENV, you might choose Resource Elements WEEKDAY and WEEKEND. These can only have the states ON, OFF, or RESET.

With JBS Environments, you might define a single Resource Element named TYPE_OF_DAY with the states WEEKDAY and WEEKEND. If desired, you could also add the third state HOLIDAY, for example.

Using JAL with JBS Environments

Jobs can be assigned a JBS Environment through JAL by specifying the JBS NEEDS statement:

```
JBS NEEDS ENVIRONMENT(WEEKLY_BACKUP)
```

A JBS Environment assigned in JAL overrides any Environment request made by the job's JECL.

Descriptors have been added to allow your JAL to determine whether a JBS Environment has been assigned and what its characteristics might be.

You can also use the JAL statement JBS HOLD to control the way in which jobs requesting non-existent Environment Agents are handled.

For syntax descriptions, see the ***JAL Reference Guide***.

Using JECL with JBS Environments

To request a JBS Environment for a job, you use the /*JBS NEEDS JECL statement:

```
/*JBS NEEDS environ1[,environ2,environ3,environ4]
```

An additional JECL statement allows you to control the state for a Resource Element:

```
/*JBS SET resource-name state [AT=timing]
```

For syntax descriptions, refer to the chapter “JECL Services” in the companion document ***Base Product: System Programming Guide*** .

Chapter 6. Job Limiting Services (JLS) Function

This chapter describes the capabilities and services provided by the Job Limiting Services (JLS) Function.

Description

JLS is designed to assist datacenters with the problem of limiting the parallel execution of certain types of work without having to resort to separate classes. The proliferation of classes makes deploying initiators difficult and potentially confusing. With JLS you can greatly reduce the "alphabet soup" required to control different types of work in a standard z/OS JES2 environment. JLS delivers your installation a set of powerful tools to manage Job Limiting. This includes:

- The ability to associate jobs with Limiting Agents with JAL.
- Facilities to create Limiting Agents dynamically in JAL. This greatly simplifies the administrative requirements.
- Facilities to provide a JECL LIMIT statement with a job, or with the DD SUBSYS.
- The ability to associate jobs with a particular Agent but with different weights so one job can, if desired, count as several jobs.
- A *system level* limiting capability that allows you to define Agents that limit only the system for which they are defined.
- Facilities to request that a job must run in *exclusive mode*, that is, no other job associated with the Limiting Agent can be executing at the same time.
    - A further refinement of exclusive mode is provided with the DRAIN/NODRAIN facility. When DRAIN is associated with a job requesting exclusive control, other jobs that use the resource in non-exclusive mode are not to be selected. This mode expedites the availability of the resource for exclusive usage.
- A comprehensive set of operator commands, giving a higher degree of operational control. With these, the limits set in JAL can be overridden. This permits rapid response to unusual situations that might be creating shortages of computing resources.
- Addition of a JLS Display to the User Display Facility.

Implementation Summary

Before implementing the JLS Function you should become familiar with its facilities and purpose. The ***TM/JBS Concepts and Facilities*** publication provides an overall description of this function. Some examples of usage are included.

To implement the JLS Function you must:

- Determine the type of work your installation want to LIMIT.
- Determine the groupings and Limiting values.
- Insert the necessary JAL statements to associate each particular workload with its Limiting Agents(s).
- Put in place the new JAL.
- Define the operational procedures to handle situations that may warrant the overriding of JAL Limiting Values by operations.

Job Limiting Agents

Limiting Agents provide the control mechanism for the scheduling of jobs associated with Limits.

They are similar in naming to Binding Agents but they are totally independent.

What Is a Limiting Agent?

A Limiting Agent is a logical element that controls the number of associated jobs that are executing concurrently.

Since this mechanism is provided to "limit" the execution of jobs, users are not expected to provide the association:

☞ ***The installation, using JAL with all its identification capabilities, detects the jobs and automatically associates them with Limiting Agents.***

Limiting Agents have a name that is chosen by your installation. The names follow dataset naming conventions. They can have two levels. For example:

```
CLASSH.GROUP1
```

is a valid Limiting Agent Name.

```
PAYROLL
```

is also a valid Limiting Agent.

```
CLASSH.GROUPA.USER1
```

is not a valid name because it has three levels.

☞ ***Limiting Agents are dynamically created in JAL. The JLS_LIMITDEF statement, together with the character string capabilities of JAL, give you all the facilities needed for that purpose. For a detailed explanation, refer to the publications DAL/JAL User Guide and JAL Reference Guide.***

Limiting Agents can have both JESplex and System level scopes:

- ***JESplex,*** which applies to all systems in the JESplex that share the Control File. An upper limit is assigned to this type of Limiting Agent. The limit set in JAL can be overridden with operator commands.
- ***System level,*** which applies only to the system for which it was defined. This type of Limiting Agent is distinguished by a name beginning with '+' (plus sign). It can be defined through JAL, but the system level limit defaults to 0 and cannot be set or changed through JAL. Jobs referencing this type of Limiting Agent will not run until a non-zero limit has been set by an operator command on one or more systems.

Activating Limiting Agents

Limiting Agents are dynamic. The Agents exist only when there are jobs in the system associated with Limiting Agents.

With JAL, a job is associated with a Limiting Agent name. If no other job is associated with that Agent at that time, then Job Limiting Services creates the necessary entry. If another job(s) is already under the control of the Agent, the new arrival is placed in that group.

If all the jobs associated with a Limiting Agent are executed without any new job arriving, the entry for the Limiting Agent is removed.

The JLS_LIMITDEF Statement in JAL

This statement is fully documented in the publication ***JAL Reference Guide***, hence this discussion represents some duplication. For a proper discussion of the operational considerations associated with the naming of Limiting Agents, however, the capabilities of JLS_LIMITDEF are central. As a result, it is included here.

The format of JLS_LIMITDEF is as follows:

```
JLS_LIMITDEF name  [LEVEL1(first-level-name)]
                   [LEVEL2(second-level-name)]
                   LIMIT(value)
```

Where:

LEVEL1

Indicates that a level one name for the Limiting Agent is provided. It is optional for Global Limiting Agents and mandatory for Local Limiting Agents. If not present, the JLS_LIMITDEF "name" is used.

first-level-name

It can be one of the following:

1. A hard coded string, such as 'SETUP'.
2. A request to create the name dynamically from a character Job Descriptor. For example:

   ```
   JLS_LIMITDEF DEVELOP LEVEL1($RACFU) ...
   ```

3. One of the execution time string variables. For example:

   ```
   JLS_LIMITDEF DEVELOP LEVEL1(%FLNAME) ...
   ```

4. The special execution time variable $JXCLASS. This variable represents the execution class assigned to the job (so far).

The level one name can begin with a '+' (plus sign), indicating that the Agent being defined will be a ***system level*** Limiting Agent.

LEVEL2

Indicates that a level two name for the Limiting Agent is provided. This keyword is optional.

second-level-name

The same options as documented above for "first-level-name."

LIMIT

This keyword is used to indicate the limit to be placed for the selection of jobs that are associated with this Limiting Agent.

value

A numerical value such as LIMIT(10).

It can also be a numeric User Job Descriptor. For example:

```
LIMIT($USERN1)
```

JAL Action Statements

A number of action statements are provided to facilitate the association of jobs and Limiting Agents. They are:

JLS ADD LIMIT

Associates an Agent with the job.

JLS REPLACE LIMIT

Replaces the Limiting Agent previously added to a job.

JLS DELETE LIMIT

Deletes the Limiting Agent previously added to a job.

Job Limiting Considerations

The JLS_LIMITDEF provides the installation with the means to cover almost any situation in terms of creating and naming Limiting Agents. JLS_LIMITDEF is a highly flexible tool. A word of caution, though. We should remember the old saying: "*it is so flexible that it cannot stand up*". When designing Limiting Agents you have to decide what role you want Operations to play. If you do not intend to have Operations adjust limits with commands, then you can be as free with your naming conventions as you want. Otherwise, you must plan the naming conventions so the operational procedures are simple.

Since variable substitution is allowed for each level of the name you must decide how dynamic you want the names to be.

For example:

```
JLS_LIMITDEF DEV LEVEL2($RACFU) LIMIT(3)
```

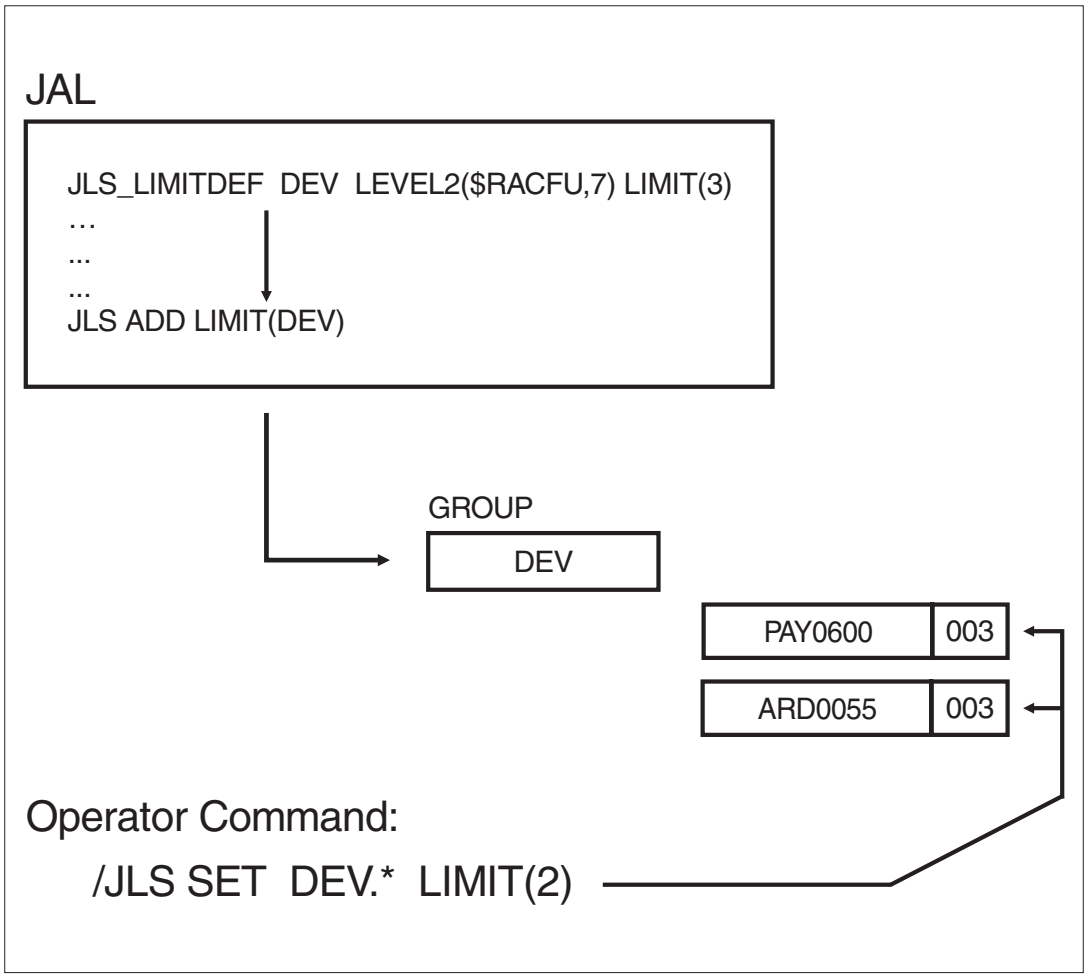
Since the keyword LEVEL1 is omitted, the above definition creates Limiting Agents with a fixed first level qualifier of DEV. If, for whatever reason, Operations has to reduce the limit for all the Agents created under that JLS_LIMITDEF the following JLS command does it:

```
/JLS SET DEV.* LIMIT(2)
```

The example shows that the name DEV identifies the JLS_LIMITDEF definition for the SET LIMIT action statement. DEV is also the first level of the name of any Limiting Agent constructed under this JLS_LIMITDEF statement. In this

case the second level of the name is dynamically constructed based on the RACFU field of the job. Two Limiting Agents, 'DEV.PAY0600' and 'DEV.ARD0055', are shown. Both are grouped under DEV and have the same limit of 3.

The figure below shows the relationship between the definition of the Limiting Agent in JAL, the association of the JLS_LIMITDEF and JLS ADD LIMIT in JAL, and any reference to the Limiting Agent via operator commands.



Limiting Agent Naming Relationships, Example 1

The operator command '/JLS SET DEV.* LIMIT(2)' changes the LIMIT value for any Limiting Agent under DEV, regardless of how many, or the name of the second level qualifier, to 2. This new value overrides the LIMITDEF LIMIT value until a RESET command is issued.

Now with the definition power of JLS_LIMITDEF you can do the following:

```
JLS_LIMITDEF DEV LEVEL1($RACFU) LEVEL2($JXCLASS) LIMIT(3)
```

Both levels are created dynamically, so there can be as many first level Agent names as you have unique RACF User IDS. Clearly, if operations needs to reset the limit value to handle an emergency, it is almost impossible.

A more manageable approach has to be used. One suggestion is the following:

- First, determine how to group the different users. Let's say that you want to separate them into three groups. Two of the groups may represent your IS developers, the third group may represent your "end-user" computing community.
- Using the string definition capabilities provided with STRINGDEF, you can create a character string to be used as your LEVEL1 parameter that allows you "to have your cake and eat it too". You can create a predefined grouping of jobs for operational control while maintaining the flexibility of the $RACFU variable substitution. This can be done as follows:
  - Assign a letter to each of the three groups. For example A for the first group, B for the second group, and C for the third group. Operations is made aware of this simple nomenclature.
  - When you construct the name for the first level qualifier do the following: To the letter designating the group concatenate the $RACFU Job Descriptor. The statements to do this are as follows.

    ```
    STRINGDEF %NAMEA ('A') || ($RACFU)
    STRINGDEF %NAMEB ('B') || ($RACFU)
    STRINGDEF %NAMEC ('C') || ($RACFU)
    ```

  - The above definitions create the required character string for use with JLS_LIMITDEF. The definitions for the Limiting Agents are:

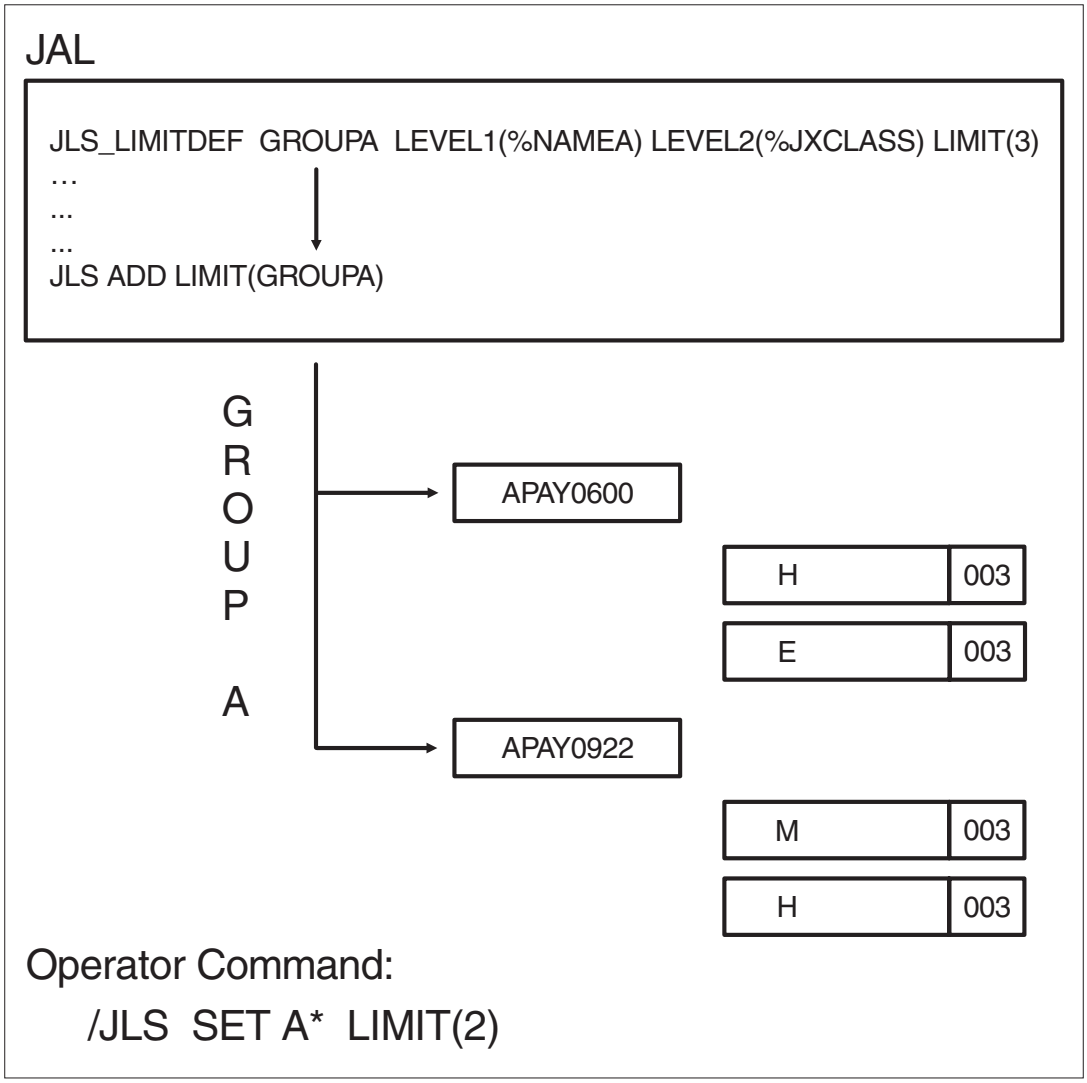
    ```
    JLS_LIMITDEF GROUPA LEVEL1(%NAMEA) LEVEL2($JXCLASS) LIMIT(3)
    JLS_LIMITDEF GROUPB LEVEL1(%NAMEB) LEVEL2($JXCLASS) LIMIT(3)
    JLS_LIMITDEF GROUPC LEVEL1(%NAMEC) LEVEL2($JXCLASS) LIMIT(3)
    ```

- Now the Limiting Agents are grouped into three segments that operations can control. With the JLS commands, operations can reset the limit for a particular group. For example, if they want to restrict group A to only two jobs, they can do it with the following command:

  ```
  /JLS SET A* LIMIT(2)
  ```

  This command alters the limit for any job associated with "group A" Limiting Agents. It stays as 2 until reset with another command.

Again, the relationship between the JLS_LIMITDEF, the Limiting Agents created, and the operator command can be seen in the diagram below:



Limiting Agent Naming Relationships, Example 2

In this case you see that the following Agents have been created with JLS_LIMITDEF GROUPA:

```
APAY0600.H    APAY0600.E    APAY0922.M    APAY0922.H
```

The sample command sets the new limit to 2 for all the Agents whose first level qualifier starts with "A". This new value overrides the JLS_LIMITDEF value until a RESET command is issued.

☞ ***This example illustrates both the power and flexibility of the JLS_LIMITDEF mechanism, as well as the need to plan the naming conventions for Agents.***

System Level Limiting

As previously explained, a Limiting Agent can have its scope restricted to only those systems that have been set to non-zero limit. An Agent that is to have system level scope must be defined with a first level name that begins with a '+' (plus sign). For example, in JAL:

```
JLS_LIMITDEF  SYS1_LIMIT  LEVEL1(+MYAGENT) LIMIT(10)
```

Note that this definition sets a limit of 10. This limit is a ***JESplex*** limit that applies to the total of all jobs using this Agent across the JESplex. You cannot specify the ***system level*** limit in JAL, it must be set by using the JLS SET operator command. When the system level Limiting Agent is initially defined by an operator JLS SET command, the default for its JESplex limit is 1. Only JLS SET command allows you to set limits that apply to individual systems.

When the Agent is defined using JAL the default for all system level limits is 0. This means that jobs associated with the Agent ***will not run until a system level limit is set with the JLS SET command:***

```
/JLS SET +MYAGENT LIMIT(5) SYSTEM(SYS1)
```

This command sets the limit for +MYAGENT to 5, but only on SYS1. The limit on other systems remains at 0 until specifically set, and the JESplex limit remains unchanged (10 in our example above). Of course, when the limit for +MYAGENT is set on another system, it can have a value different from the one set for SYS1.

Because you cannot set a system level Limit in JAL, the JLS RESET command has a slightly different effect on system level Limiting Agents. You can specify that a specific system level Limiting Agent be reset:

```
/JLS RESET +MYAGENT SYSTEM(SYS1)
```

Since there cannot be a JAL-specified limit, this command resets the system level Limit to 0, the default. Omitting the SYSTEM keyword applies the command to the JESplex limit for the Agent, therefore the JESplex limit is reset to whatever was specified when the Agent was defined. If no limit was specified at definition, the JESplex limit is reset to 1, the default.

For all other purposes, system level Agents can be treated exactly like JESplex Agents. You can associate jobs with system level Limiting Agents in JAL, just as with JESplex Limiting Agents. The JLS ABANDON command lets you remove

jobs from the control of a system level Limiting Agent, and you can display them with the JLS DISPLAY command.

JLS and JES2 Exit 14 (Job Queue Work Select)

JLS extends the JES2 QGET logic by using JES2 Exit 14, the JES2 Job Queue Work Select exit. This allows JLS to resolve the "initiator race" problem.

This problem has been eliminated by extensions to the JES2 QGET logic through the use of JES2 Exit 14, the Job Queue Work Select exit.

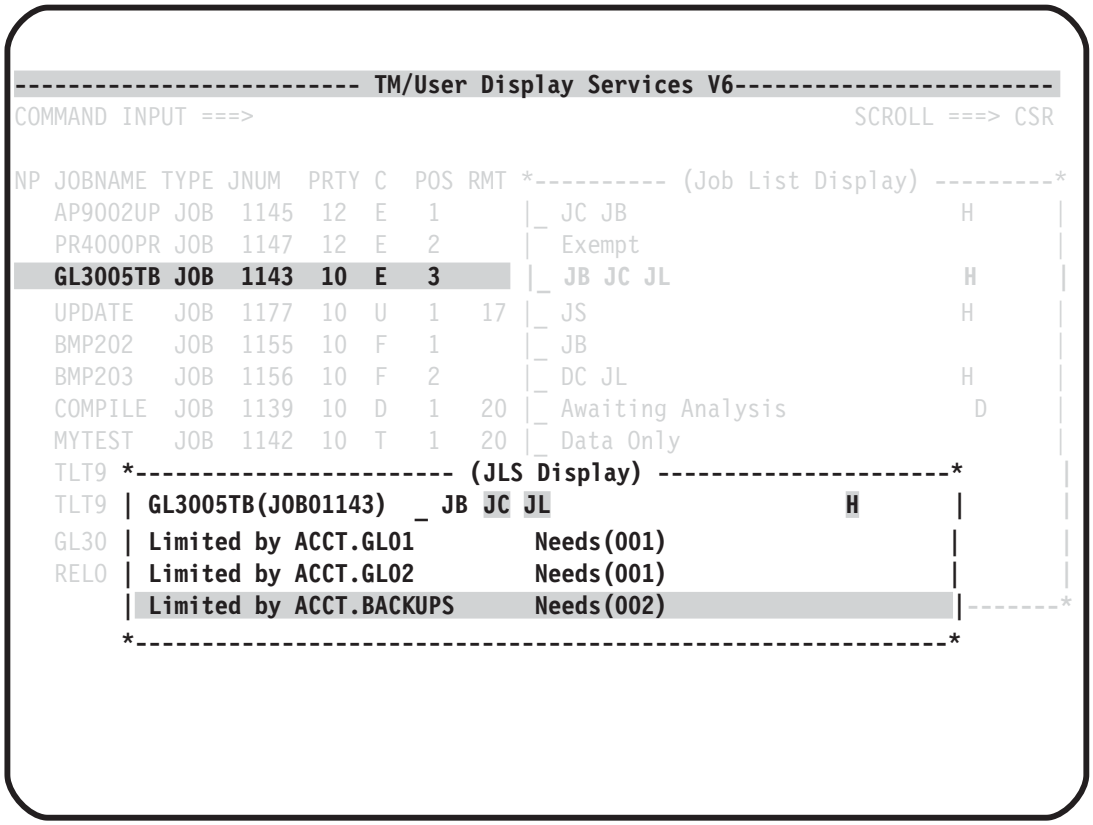
☞ ***If your installation uses, or plans to use, JES2 Exit 14 for the purpose of selecting jobs for execution, please contact ThruPut Manager Customer Support. We will help you determine whether there is a conflict and if so, what can be done about it.***

If the job selection process is left entirely to the standard z/OS JES2 logic, JES2 does not ensure that the next selected job is actually the first one eligible for a Limit.

With the ThruPut Manager changes to JES2 Exit 14, ***QGET and JLS co-operate to ensure that the next job to get the Limit really is the first available one.*** The logic in Exit 14 eliminates "initiator races" with their unpredictable results.

UDF Extensions For JLS

If the User Display Facility shows the JLS acronym for a job, a user can select that acronym and open the JLS Window. A sample JLS Window is shown here.



The JLS Display Window

The first line of the JLS Window is the Information Summary Line for the job.

Subsequent lines list each Limiting Agent affecting the selected job. Each line lists a Limiting Agent, showing the weight assigned to that job (default weight is 1), or, for ENQ Limiting Agents, whether the Agent is exclusive (EXC) or shared (SHR).

Highlighted lines indicate that the Limiting Agent described on that line causes the selected job to be held. Remember that other causes could also result in the job being held.

Displaying Installation-defined Information

UDF allows you to define an ISPF panel that is displayed when a user places the cursor on a JLS Limiting Agent name in the JLS Display Window and then hits ENTER. The panel name is associated with the Agent by the PANEL keyword of the JLS_LIMITDEF DAL/JAL statement:

```
JLS_LIMITDEF DEVELOP … PANEL(OURPANEL)
```

OURPANEL must be a valid ISPF selection panel, since it is displayed using the ISPF SELECT service.

UDF initializes several ISPF variables that you can use in your panel, as shown in the table below.

ISPF Variables Initialized for Installation-defined UDF JLS Display

Name	Type	Length	Description
DTMJOBNM	Char	8	JES2 job name.
DTMJOBID	Char	8	JES2 job ID.
DTMRACGR	Char	8	RACF group (RACF only).
DTMRACUS	Char	8	RACF user ID (RACF only).
DTMTSSUS	Char	8	TSS user ID (TSS only).
DTMACFSI	Char	8	ACF2 Source ID (ACF2 only).
DTMACFLI	Char	8	ACF2 logon ID (ACF2 only).
DTMACFUI	Char	24	ACF2 user ID (ACF2 only).
DTMJLNAM	Char	17	JLS Limiting Agent name.
DTMJLVRB	Char	5	JLS verb (LIMIT, ENQ).
DTMJLLIM	Char	3	JLS Limit value (when LIMIT).
DTMJLWEI	Char	3	JLS Limit weight (when LIMIT).
DTMJLTYP	Char	3	JLS ENQ type (when ENQ).

Facilities Summary

JLS Operator Commands	
Command	**Description**
JLS ABANDON	Unconditionally removes a job from Job Limiting Services control.
JLS DISPLAY	Displays information about Limiting Agents, jobs, and their relationships.
JLS RECONCILE	Analyzes active Limiting Agents and ensures synchronization with the Control File.
JLS RESET	Resets Limiting Agents to the normal status of accepting Limiting values from JAL.
JLS SET	Overrides Limiting Values from JAL. The SET value stays until a RESET command is issued.

JLS JECL Statements Refer to *Base Product: System Programming Guide*		
Statement	**Description**	**Chapter**
/*JLS ENQ	This statement is provided for the Mellon Bank Compatibility Mode.	7
/*JLS LIMIT	Sets up a Limiting Agent for the job.	7

Chapter 7. Job Chaining Services (JCS) Function: Before & After

This chapter describes the Job Chaining Services or Before & After facility of TM/Job Binding Services, provided for Mellon Bank compatibility purposes.

Introduction

The TM/Job Binding Services Component includes Job Chaining Services (JCS). It is active by default. If you do not want to use it, you can deactivate it with the OPTIONS keyword of TMPARM. This section describes the capabilities of JCS.

Requirements

Job Chaining Services requires the Control File.

Job Chaining Services—Before & After

This mode replicates the behavior of the BEFORE & AFTER (B&A) support of Mellon Bank, with some enhancements:

- With JCS, a job is known only from the time it arrives in the system until it terminates (normally or abnormally) or is purged, whichever occurs first.
- With the Mellon Bank, the B&A ***applies to all the jobs present in the system*** at the time the job is to be selected for execution. This scope can create problems because one group can inadvertently affect another group. JCS provides you with control over the scope of B&A statements.

For a job to be part of the B&A domain, it must have a ***Job Chaining BATCH Name.*** Jobs without a BATCH Name associated with them are not "seen" by the B&A processor. Jobs with different BATCH Names are unrelated to one another.

The BATCH Name can be assigned either:

- With the JECL statement /*JCS BATCH.
- In JAL, using the JCS SET BATCH action statement.

The BATCH Name allows your installation to group jobs into separate domains so the scope of BEFORE and AFTER statements can be controlled.

BATCH Name Conventions

The BATCH Name is a one or two level name following dataset naming rules. For example:

```
PAYROLL      PAYROLL.GROUP1
```

are both valid BATCH Names.

In JAL you can verify:

- If a JECL BATCH statement is present in the job.
- The actual BATCH Name so you can enforce naming conventions.

Implementation Summary

To implement JCS to behave like the Mellon Bank BEFORE & AFTER:

- During the JAL processing, assign each job the same BATCH Name. This way all jobs in the system belong to the same B&A domain. The name can be an arbitrary 17 character string that conforms to the naming conventions described above.
- You might want to create more than one domain. Using different BATCH Names you can, for example, separate your production jobs from your on-demand jobs to eliminate the potential of accidental interference:
  - Assign all production jobs the same BATCH Name of "PROD".
  - For the other type of work, assign a different BATCH Name, for example "TEST".
  - With the above implementation, the BEFORE and AFTER statements for a job with a BATCH Name of TEST ***do not "see"*** jobs with a BATCH Name of PROD.

JECL Statements

Job Chaining Services accepts the JECL statements /*BEFORE and /*AFTER with the same syntax and meaning as the Mellon Bank JECL statements. Users do not have to make any changes. The following JECL statements are supported:

```
/*BEFORE or /*JCS BEFORE
/*AFTER or /*JCS AFTER
/*JCS BATCH
```

JCS also supports extensions to these statements. The actual format is described in the ***JECL Reference Guide***.

☞ ***If you are not a previous user of the Mellon Bank usermod BEFORE & AFTER you should be aware that the relationships of jobs do not carry any history. If the job you are referring to in a BEFORE or AFTER statement is not in the system, the condition is considered to have been satisfied.***

Batch Job Sequencing

The /*JCS BATCH statement supports the SEQUENCE keyword. When coded, this keyword specifies that, within the named BATCH, jobs must be executed in the same sequence in which they were read in. Sequencing applies as long as there are at least two jobs from the named BATCH still in the system.

☞ ***Do not use the SEQUENCE keyword for a job that includes any of the /*JCS AFTER, the /*JCS BEFORE, the /*AFTER, or the /*BEFORE statements. Doing so requests conflicting processing sequences with unpredictable results.***

Adding a BATCH Name to TSUs and STCs

Three methods are provided:

1. A //*+JCS BATCH statement can be added to the TSU or STC procedure.
2. The DD Subsystem Interface can be used to insert a JCS BATCH statement in the TSU or STC procedure.
3. A BATCH Name can be added dynamically to the STC or TSU by using the JECL keyword of the TMPARM JES2 initialization statement.

The first two methods need no further explanation other than those given elsewhere in this manual. The following discussion therefore deals only with the dynamic addition of the BATCH Names to TSUs or STCs.

Implementation

The TMPARM JECL keyword has been extended so that you can use it to indicate what BATCH Name should be added and which STCs or TSUs are eligible for this process. Different TSUs or STCs can be associated with different JCS BATCH Names.

To add a BATCH Name dynamically to a TSU or STC, code the following JECL keywords on the TMPARM JES2 initialization statement.

For TSUs:

```
TMPARM … JECL=($JCSTSO,name,IGNORE,XLATE(level1,level2))
```

For STCs:

```
TMPARM … JECL=($JCSSTC,name,IGNORE,XLATE(level1,level2))
```

Where:

$JCSTSO

Is a subparameter indicating that this JECL keyword is to apply to TSUs.

$JCSSTC

Is a subparameter indicating that this JECL keyword is to apply to STCs.

name

Is the jobname of the TSU or STC.

You can use the wildcard character '*' (asterisk) to make the name generic. For example, "DFP*".

☞ ***Do not use the mask "*" for STCs. Any started task ahead of ThruPut Manager will not be able to "get going".***

You can use "*" to indicate the BATCH Name is to be applied to all TSUs.

☞ ***Under no circumstances should you include a specific or generic name to STCs that includes the name of the ThruPut Manager address space.***

level1

Is the 1st level of the BATCH Name.

level2

Is the 2nd level of the BATCH Name.

Considerations

You can choose to exclude a STC or TSU from this process. Use the IGNORE keyword instead of the XLATE keyword to accomplish this:

```
JECL=($JCSTSO,name,IGNORE,IGNORE)
JECL=($JCSSTC,name,IGNORE,IGNORE)
```

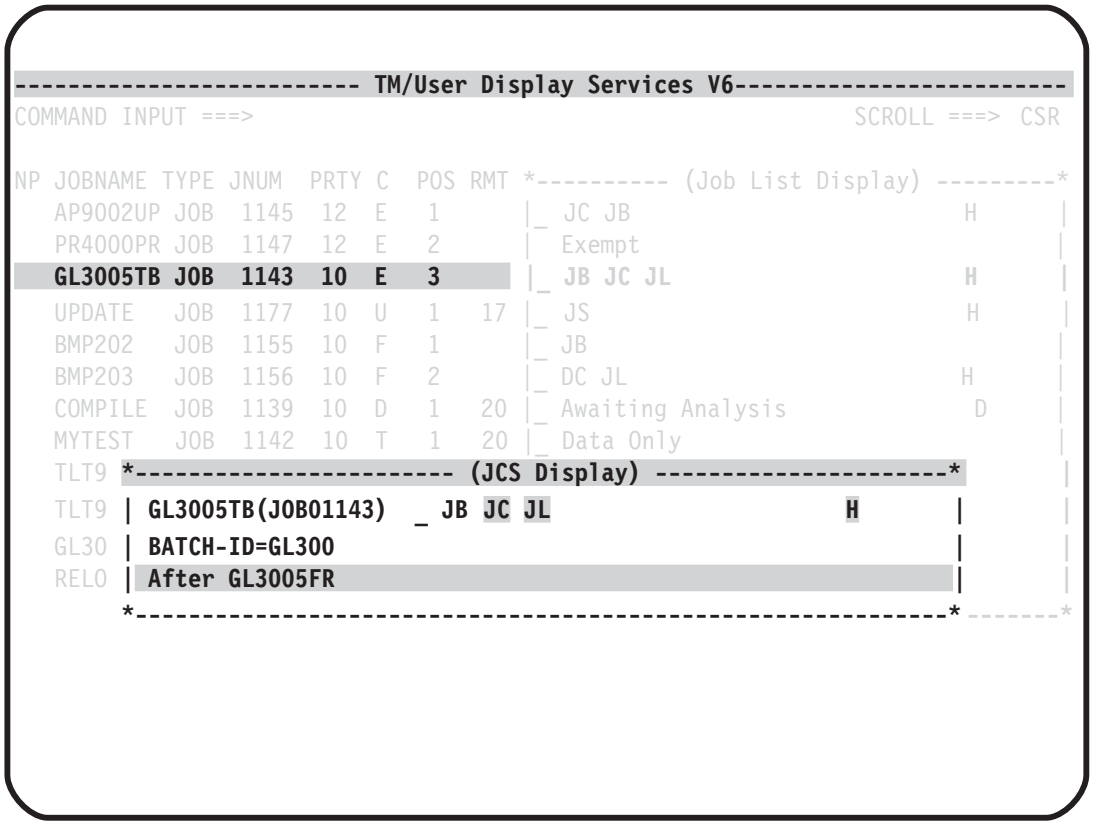
Example

```
JECL=($JCSTSO,*,IGNORE,XLATE(TSO,BATCH))
JECL=($JCSSTC,IMS*,IGNORE,XLATE(IMS,BATCH))
```

These statements would cause the equivalent of a //*+JCS BATCH TSO.BATCH statement to be added to ***all*** TSUs and the equivalent of a //*+JCS BATCH IMS.BATCH to be added only to STCs with a name starting with "IMS".

UDF Extensions for JCS

If the User Display Facility shows the JCS acronym for a job, a user can select that acronym and open the JCS Window. A sample JCS Window is shown here.



The JCS Display Window

The first line of the JCS Window is the Information Summary Line for the job.

BATCH-ID identifies the JCS Batch to which the job belongs.

Subsequent lines show whether the selected job must run BEFORE or AFTER other jobs in the BATCH, one line per job.

Highlighted lines indicate that the job whose AFTER request is described on that line is causing the selected job to be held. Satisfying this condition does not assure that the job will be released, however, since there might be additional reasons why the job is being held.

Facilities Summary

JCS Operator Commands	
Command	**Description**
JCS DISPLAY	Displays information about jobs that are under the control of JCS.
JCS RELEASE	Allows you to unconditionally release a job from Job Chaining Services Control.

JCS JECL Statements Refer to *Base Product: System Programming Guide*		
Statement	**Description**	**Chapter**
/*AFTER	Indicates to JCS a job is to run after a named job.	7
/*BEFORE	Indicates to JCS a job is to run before a named job.	7
/*JCS AFTER	Indicates to JCS a job is to run after a named job.	7
/*JCS BATCH	Assigns a BATCH Name to the job.	7
/*JCS BEFORE	Indicates to JCS a job is to run before a named job.	7

Chapter 8. Mellon Bank Compatibility Services

This chapter describes the integrated services provided by TM/Job Binding Services for compatibility with the Mellon Bank Shared Spool Mods.

Introduction

This section presupposes that you are familiar with TM/Job Binding Services and the Mellon Bank Shared Spool Mods.

The TM/Job Binding Services Component provides integrated facilities that are compatible with the following JES2 User Mods:

1. Resource Routing (/*ROUTE XEQ resource-name)
2. Resource Serialization (/*CNTL)
3. Job Dependency (/*WITH)
4. Job Chaining (/*BEFORE and /*AFTER)

This chapter describes how TM/Job Binding Services relates to the first three facilities listed above. Job Chaining (Before & After) is described in "Chapter 7. Job Chaining Services (JCS) Function: Before & After."

Enabling Mellon Bank Compatibility

After you have installed TM/Job Binding Services, the Mellon Bank Compatibility Option status is "installed/inactive". You use the OPTIONS keyword in the TMPARM JES2 statement to enable this option.

The Resource Routing Facility

The Resource Routing Services provided by the Mellon Bank JES2 Mod is supported by the Job Binding Services function of TM/JBS. ***Before attempting to activate this compatibility option you should have the JBS function fully operational.***

Resource Definition

With the Mellon Bank User Mods, resources are defined using the $RESTABL macro. They are arbitrary 8-character names that represent whatever the installation wants them to represent.

With TM/JBS, you define Binding Agents using the DEFINE command. These Agents can be used to represent your current "resources". TM/JBS allows two-level names; however, if you want to have identical names you can use a single level 8-character name.

TM/JBS provides several types of Binding Agents. ***Permanent*** is the type of Agent that corresponds to the $RESTABL resources.

Resource Activation/Deactivation

$QA Command

With the Mellon Bank Mods, you attach a resource to a particular CPU using the $QA command.

With TM/JBS, you use the ACTIVATE command.

$QD Command

With the Mellon Bank Mods, you remove a resource with the $QD command.

With TM/JBS, you use the DEACTIVATE command.

The ACTIVATE and DEACTIVATE commands for Permanent Binding Agents provide all the necessary options to completely replicate the attaching and detaching of resources to MAS systems.

Display Facilities

The Mellon Bank Mods provide display commands to show jobs that are associated with resources that are not attached to any CPU.

With TM/JBS, you use the JBS DISPLAY command with the following keywords:

```
JBS DISPLAY HELD
```

The display output shows all the jobs that are in HOLD status because they have requested Binding Agents that are not Active. This is the equivalent of "resources" that are not attached.

JECL Control Statements

The Mellon Bank Mod uses the "/*ROUTE XEQ name" JECL statement as the means for users to request resource routing.

The equivalent service is provided by TM/JBS with the JECL statement:

```
/*JBS  BIND name
```

With the Mellon Bank User Mods, the name of the resource determines how the ROUTE statement is to be treated using the following search algorithm:

- **The $RESTABL is searched first**.
- If a name match is found, the ROUTE statement is treated as a resource association request.
- If not, it is treated as an NJE routing request.

To make the transition as simple as possible, TM/JBS accepts the /*ROUTE XEQ JECL statements.

☞ ***When you activate the Mellon Bank compatibility option, the mechanism to convert the statements is activated automatically.***

The only difference between the Mellon Bank Mods and TM/JBS is in the name validation process:

- **TM/JBS searches the JES2 NJE node name table first.**
- If the name is a valid NJE node, then the ROUTE statement is not converted to a BIND statement. As a result, the system treats it as a normal JES2 NJE routing request.
- If the name is **not** a node name, then the ROUTE statement is converted to a BIND statement to be processed by the Binding Services processor at Job Analysis Time.

Special Cases

There are two special cases of resource routing:

```
/*ROUTE XEQ HERE
/*ROUTE XEQ CPUn
```

If you are using the above facilities, then the following must be done:

1. Define Permanent Binding Agents with the names CPU1 to CPUn, where n represents the highest possible system number in your MAS complex.

2. Associate each one of them with the corresponding JES2 system at ACTIVATION. These Agents must always be active.

Note that HERE is automatically converted to CPUn, where n is the corresponding CPU number to the system where the job was submitted.

Additional Considerations

If you have renamed the /*ROUTE XEQ control statement for Resource Routing to a ***user name***, for example /*RESOURCE, you have to "inform" ThruPut Manager of the change. You can do that with the keyword JECL in the TMPARM initialization statement. This facility is explained in ***JECL Reference Guide***

The CNTL Facility

The Control Facility for the compatibility mode is supported by the Job Limiting Services function of TM/JBS. JLS is fully documented in the standard ThruPut Manager publications. ***Before attempting to activate this compatibility option, you should have the JLS function fully operational.***

Implementation

With the Mellon Bank Mods, a facility is provided that allows users to single-thread jobs by using JECL statements.

The JECL statement is:

```
/*CNTL resource-name,EXC
  or
/*CNTL resource-name,SHR
```

TM/JBS provides two methods to replicate this facility.

1. You can continue using the Mellon Bank Mod control statements. TM/JBS automatically invokes the JLS function to accomplish the same result.
2. You can automatically insert the needed control statement, using the JAL statements JCS_CNTLDEF and JLS SET CNTL.

The WITH Facility

The WITH facility is provided for compatibility for users converting from the Mellon Bank modifications. If you do not already use WITH but have a requirement for co-requisite jobs, use the facilities of Job Binding Services.

The WITH facility allows you to ensure that a given job executes on the same JES2 complex member as a specified job. By including the appropriate JECL statement, you can specify that the job cannot run unless the job named on the JECL statement is already running.

ThruPut Manager supports the WITH facility through JCS.

Implementation

The Mellon Bank modifications implement the WITH facility through the WITH JECL statement:

```
/*WITH jobname
```

ThruPut Manager honors this format of the WITH statement, and also supports the WITH facility with the following JCS JECL statement:

```
/*JCS WITH jobname[,NOSYSAFF]
```

The NOSYSAFF keyword allows you to specify that the jobs can run on different systems in the complex.

Facilities Summary

Mellon Bank Compatibility Commands	
Command	**Description**
JBS DISPLAY	Displays information for jobs with ROUTE XEQ control statements.
JCS DISPLAY	Display information about jobs that have WITH dependencies.
JLS DISPLAY	Displays information for jobs with a CNTL statement.

JECL Statements Refer to *Base Product: System Programming Guide*		
Statement	**Description**	**Chapter**
/*CNTL	This JECL statement is accepted for compatibility purposes. It is internally converted to a JLS ENQ statement.	7
/*JCS WITH	Specifies that a job depends on the concurrent execution of another job.	7
/*JLS ENQ	Allows the serialization of jobs, whenever it is necessary.	7
/*ROUTE XEQ	This JECL statement is accepted for compatibility purposes. It is internally converted to a JBS BIND statement.	This Chapter Page 109
/*WITH	This JECL statement is accepted for compatibility purposes. It is internally converted to a JCS WITH statement.	7

Chapter 9. DD Subsystem Interface

This section discusses started tasks and their use of the DD SUBSYS to supply the equivalent of JECL to ThruPut Manager.

Introduction

The DD SUBSYS interface was originally provided because started tasks could not use the services of JES2 control statements. This was not a ThruPut Manager restriction, it was the way that z/OS JES2 worked. Current versions of z/OS allow ThruPut Manager to provide JECL services by using the JCL comment form for statements:

```
//*+JBS BIND AGENT1
```

While the DD SUBSYS interface is still supported, we recommend the use of the simpler JCL comment form.

DD SUBSYS Interface

Started tasks can request ThruPut Manager services using the SUBSYS keyword of the JCL DD statement. This is the equivalent of JECL statements for batch jobs. The format is shown on the next page.

Started tasks are not processed by the Job Analyzer. As a result:

- The ThruPut Manager installation exit 19 is not invoked.
- JAL Verification of JECL control statements is not applicable.

Since started tasks are under the control of the datacenter, the above services are normally not required.

☞ ***The DD SUBSYS facility is only available to started tasks. It is not operational for batch jobs.***

DD SUBSYS

JECL for started tasks

Provides ***started tasks*** with a mechanism to include the equivalent of JES2 control statements in their JCL.

☞ ***This mechanism is not operational for batch jobs.***

```
//ddname DD SUBSYS=($$TM,'JBS ACTIVATE ....')
//ddname DD SUBSYS=($$TM,'JBS DEACTIVATE ...')
//ddname DD SUBSYS=($$TM,'JBS BIND ....')
//ddname DD SUBSYS=($$TM,'JBS MESSAGE....')
//ddname DD SUBSYS=($$TM,'JLS LIMIT....')
```

ddname

Can be any arbitrary DD name.

SUBSYS

Indicates that this DD statement is to be processed by the named subsystem.

$$TM

Is the name of the ThruPut Manager subsystem.

JBS ACTIVATE/JBS DEACTIVATE

Conform to the /*JBS ACTIVATE and /*JBS DEACTIVATE JES2 control statements format described in ***JECL Reference Guide***.

You can use JBS API services to activate and deactivate Agents. The JBS MESSAGE facility can also be used. It is described below.

JBS BIND

Can be used to ensure that the started task is being initiated in the correct processor, or when a particular GLOBAL Agent is active:

- If the Agent is not active the started task is not initiated.
- If the Agent is active but in a different processor the started task is not initiated.

In both cases, an appropriate message is issued to indicate the problem.

JBS MESSAGE

This conforms with the JECL 'JBS MESSAGE' statement described in ***JECL Reference Guide***. The only consideration is the syntax requirements resulting from JCL conventions for the use of apostrophes. The best way to show the required syntax is with an example.

Here is a JECL statement:

```
/*JBS MESSAGE *'IEA123I'*'CICSPROD'*,API=10
```

The equivalent DD SUBSYS request is:

```
//anyname DD SUBSYS=($$TM,'JBS MESSAGE *''IEA123I''*''CICSPROD''*,API=10')
```

The JCL syntax rule is simple (even though the resulting statement looks awkward): The character string to be passed to the subsystem must be enclosed in apostrophes. This explains the opening and closing apostrophes. If the string to be passed contains apostrophes, then you must code two apostrophes for each apostrophe to be passed.

You might ask: what if the message string I want to match has apostrophes? Again an example is the best way to show the syntax. The message to match is IEA123I DAY'S TEST.

Here is the JECL statement:

```
/*JBS MESSAGE *'IEA123I DAY''S TEST'*,API=10
```

The equivalent DD SUBSYS request is:

```
//anyname DD SUBSYS=($$TM,'JBS MESSAGE *''IEA123I DAY''''S TEST''*,API=10')
```

A bit tedious, but this follows the consistent rule of coding two apostrophes for each one to be passed to the subsystem.

JLS LIMIT

Can be used to associate a started task with a Limiting Agent.

If the limit has been reached the started task is not initiated.

This can be used to prevent the accidental starting of a task. For example, you may want to have ***only one task of a given type*** active in your MAS complex. Placing a Limiting Agent with a Limit of 1 prevents another similar task from starting.

Notes:

Note that all the facilities described in ***JECL Reference Guide*** for the DD SUBSYS supported statements are applicable when using the DD SUBSYS interface.

Chapter 10. Application Program Interface (API)

This chapter provides a brief description of the Application Program Interface.

Description

There are situations where you want to activate or deactivate Binding Agents from a program, or as a result of an application having done something. A typical example is a CICS region that dynamically allocates and deallocates resources. In this case, no step initiation or step termination takes place. The region might simply deallocate a database dynamically for maintenance purposes. Prior to doing that, the Binding Agent associated with that service should be deactivated.

There are also situations in which you want to determine the status of an Agent either in your own code or in a CLIST or REXX script. For example, a CLIST that submits jobs might test the status of an Agent to determine whether a particular job should be submitted.

APIs are provided to address both situations.

Activating/Deactivating Using the API

There are two methods that can be used:

- The first method applies to situations where you have control of the software that dynamically allocates and deallocates the resources. In this case, you can add code to issue the required message:

  ```
  DTM6999A JBSAPI=nn
  ```

- The second method covers two common situations:
  - You cannot insert your own "code" in the application that activates/deactivates the resources; therefore, you cannot issue the required message. This situation is typical of CICS R2, where commands are now provided to allocate and deallocate particular resources.
  - There is a delay between the time an Agent is activated and the time the actual resource is available. As a result, jobs start to execute before the resource is truly available. This occurs even though the Agent is activated at the step level.

API Method 1

WTOs are trapped for jobs that are submitted with the JES2 statements:

```
/*JBS ACTIVATE Agent-name,API=xx
   or
/*JBS DEACTIVATE Agent-name,API=xx
```

This method expects a fixed message format. The message format is as follows:

```
DTM6999A  JBSAPI=xx
```

Where:

DTM6999A

A message identifier to go in SYSLOG.

JBSAPI=xx

The identification mechanism that links this request to the actual /*JBS ACTIVATE or /*JBS DEACTIVATE JES2 statement.

A route code 11 (write-to-programmer) is recommended.

For high level languages such as COBOL or PL/I, all that is needed is a DISPLAY statement with the text:

```
DTM6999A   JBSAPI=xx
```

☞ ***The message must begin with DTM6999A in column 1.***

API Method 2

In this situation you do not have a fixed format message. You are "watching" for a particular text in a message or messages from the address space. A facility is provided to specify the message text that is to trigger the activation of the Agent. Several messages can be associated with the activation of an Agent. This covers cases where different messages could indicate when a resource becomes available. ***The message facility allows you to specify either a precise text match or a pattern match.***

To support the above facility, a JECL statement is available for JBS. The format is as follows:

```
/*JBS MESSAGE msgmask,API=id
```

For a detailed description of this statement, refer to ***JECL Reference Guide***.

Some examples of the use of this facility are:

```
/*JBS ACTIVATE RESOURCE.A,API=10
/*JBS MESSAGE *'IST009A'*'STARTED'*,API=10
```

The above statement matches any message with IST009A ***and*** STARTED in its text. The message must occur within the address space running the job with the Message statement. When a match occurs, RESOURCE.A is activated.

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

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

```
/*JBS ACTIVATE CICS.PROD,API=10
/*JBS MESSAGE *'IEA123I'*'CICSPROD'*,API=10
/*JBS MESSAGE *'IEA125I'*'CICSPROD'*,API=10
```

In the previous example, a match on either message triggers the activation of CICS.PROD

☞ ***You can associate multiple messages with one ACTIVATE or DEACTIVATE statement. They are connected with the API id number. You can also associate one message with multiple ACTIVATE or DEACTIVATE statements.***

Testing Agent Status Using the API

To allow you to test the status of an Agent, JBS provides an API using the DTMJBAPI program. This program can be invoked in several ways. It can be:

- Invoked directly through JCL.
- Called as a TSO command from REXX or a CLIST.
- Called using a CALL or LINK from your own code.

DTMJBAPI accepts a single Agent name or name mask (e.g. ABC*), and sets a return code reflecting the status of the Agent(s). Possible return codes are shown in the table below.

DTMJBAPI Return Codes	
Return Code	**Meaning**
0	Agent is active on the local system.
4	Agent is active on another system in the JESplex.

DTMJBAPI Return Codes	
Return Code	**Meaning**
8	Agent is active on another node.
12	Agent is inactive.
16	Agent is not defined.
20	System error.

If DTMJBAPI is invoked from another program, the return code is found in register 15. Additional information is returned in register 0 as follows:

- If the return code is 0 or 4, register 0 contains the 4-byte affinity mask.
- If the return code is 8, register 0 contains the 2-byte node number.
- If the return code is 20, register 0 contains the error reason code.

If the request to DTMJBAPI was an Agent name mask, the result is the lowest return code for any Agent that matches.

DTMJBAPI accepts a standard parameter list or a TSO CPPL. The VL bit ***must*** be on for a standard parameter list. The format is:

```
R1 -> x'8aaaaaaa'

      aaaaaaaa
```

Represents the parameter address.

```
Parm -> x'llll',c'agent.name'

      llll
```

Represents the length of the Agent name.

```
      agent.name
```

Represents the Agent name.

Sample Assembler Code

```
LA    R1,PARMLST
OI    PARMLST,=X'80'    SET VL BIT
LINK  EP=DTMJBAPI
```

```
LTR   R15,R15
     ...
PARMLST  DC    A(PARM)
PARM     DC    Y(L'AGENT)
AGENT    DC    C'TEST.AGENT'
```

JCL Example

```
//CHK EXEC PGM=DTMJBAPI,PARM='DBASE.UP'
/* RUN STEP1 IF AGENT ACTIVE HERE
//STEP1 EXEC PGM=DBASE1,COND=(0,LT,CHK)
// ...
//* RUN STEP2 IF AGENT ACTIVE ON ANOTHER SYSTEM
//* BUT NOT IF ACTIVE HERE
//STEP2 EXEC PGM=DBASE2,COND=((4,LT,CHK),(0,EQ,CHK))
// ...
```

CLIST Example

```
PROC 0
DTMJBAPI DBASE.UP
SET &CC = &LASTCC
IF &CC = 0 THEN DO
   CALL 'SYS1.LINKLIB(DBASE1)'
   END
IF &CC = 4 THEN DO
   CALL 'SYS1.LINKLIB(DBASE2)'
   END
IF &CC > 4 THEN WRITE 'DATABASE NOT AVAILABLE, CODE=&CC'
...
```

Facilities Summary

API JECL Statements Refer to *System Base Product: Programming Guide*		
Statement	**Description**	**Chapter**
/*JBS MESSAGE	Allows you to specify the text pattern to match with messages to trigger the API mechanism.	7

Index

Index

Special Characters

$$DELETE
- Special Incompatible Agent 38

$QA Command
- Mellon Bank Compatibility 110

$QD Command
- Mellon Bank Compatibility 110

$RESTABL Macro
- Mellon Bank Compatibility 110

$RESTABL Search
- Mellon Bank Compatibility 111

&AGENT
- SAC Message Insert 48

&CURRENT
- SAC Message Insert 48

&ENTITY
- SAC Message Insert 49

&JOBNAME
- SAC Message Insert 49

&JOBNUMBER
- SAC Message Insert 49

&NAME
- SAC Message Insert 49

&PRODUCT
- SAC Message Insert 49

&SYSTEMS
- SAC Message Insert 49

A

AFTER
- JECL Statement 108

AGENT Keyword
- TYPE(PRODUCT) Statement 50

ALTER Subparameter
- TYPE(PRODUCT) Statement 50

API
- Activating and Deactivating With 119
- Binding Agent 23
- COBOL 120
- Considerations 25
- **See Also** DTMJBAPI
- Description 119
- Facilities Summary 123
- Fixed Message 23,120
- Introduction for JBS 16
- JBS Message 123
- Message Format 119
- Message ID 120
- Message Pattern 23,120
- PL/I 120
- Route Code 120
- Testing Agent Status With 121
- Using DTMJBAPI 121

Access Control
- Software 43

Activating
- Binding Agents 22
- Limiting Agents 89
- Using the API 119

Agents
- Incompatible 35

B

BATCH Name
- Before & After 103
- Conventions 104
- For TSUs and STCs 105
- JCS 103

BEFORE
- JECL Statement 108

Batch Control
- Software Access Control 56

Before & After
- BATCH Name 103
- Facilities Summary 108
- Implementation 104
- JCS DISPLAY Command 108
- JCS RELEASE Command 108
- JECL Statements 104,108
- Job Chaining Services 103
- Operator Command 108
- Scope of Control 103

Binding Agent
- API 23
- Activating 22
- Activation Attributes 20
- Attributes 19
- Binding Request 23
- Busy 25
- DD SUBSYS 22
- Defining 20
- Definition Attributes 19
- Deletion Considerations 25
- GLOBAL Attribute 20
- Group Definition 20
- Introduction 18
- JAL VERIFY Command 27
- JECL 22
- Job-related 19
- LOCAL Attribute 20
- LOG Attribute 21
- MULTIPLE Attribute 21
- Naming 18
- OPER Attribute 22
- PERMANENT Attribute 19
- Reserving 25
- UNIQUE Attribute 21
- Verifying 26
- WARN Attribute 21
- XMIT Attribute 20,21

Busy State
Binding Agents 25

C

CLASS Keyword
TYPE(PRODUCT) Statement 51
CNTL
JECL Statement 114
Mellon Bank Compatibility 114
SET in JAL 112
CNTL Facility
Mellon Bank Compatibility 112
CNTLDEF in JAL
Mellon Bank Compatibility 112
CONTROL Subparameter
TYPE(PRODUCT) Statement 50
Commands
Mellon Bank Compatibility 113
Components
TM/Job Binding Services 11
Considerations
API 25
JAL 26,92
JBS Function 24
JECL 24
Job Limiting 93
Controlling
Limiting Agents 95

D

DD SUBSYS
Binding Agents 22
Description 115
JAL Verification 115
JBS ACTIVATE 116
JBS BIND 116
JBS DEACTIVATE 116
JBS Function Exit 19 115
JBS MESSAGE 116
JLS LIMIT 117
Syntax 116
DD Subsystem
Introduction 15
DISPLAY_ONLY Keyword
TYPE(PRODUCT) Statement 51
DTMJBAPI
Sample Assembler Linkage 122
Sample CLIST Invocation 123
Sample JCL Invocation 123
TSO CPPL Format 122
Using 121
Deactivating
Using the API 119
Definition
Attributes for Binding Agents 19
Binding Agent 20
Limiting Agents 90
Deletion Considerations
Binding Agents 25
Description
API 119
DD SUBSYS Interface 115
JBS Function 17
JLS Function 87
Limiting Agents 88

E

EXC Keyword
Mellon Bank Compatibility 112
EXIT 19
DD SUBSYS 115
Job Binding Services 33
Enabling Components
Mellon Bank Compatibility 109
Examples
Limiting Agents Naming 94
Exits
JBS Extensions 16
Job Binding Services 33

F

FAIL_MSG Keyword
TYPE(PRODUCT) Statement 51
Facilities Summary
API 123
Job Binding Services 32
Job Chaining Services 108
Job Limiting Services 100
Mellon Bank Compatibility 113
Functional Summary
TM/Job Binding Services 11

G

GLOBAL Attribute
Binding Agent 20
Group Definition
Binding Agent 20

H

HERE Keyword
Mellon Bank Compatibility 111

I

IBM Scheduling Environment
Converting From 85
Implementation Summary
Before & After 104
JBS Function 17

JLS Function 88
Job Chaining Services 104
Incompatible Agents
$$DELETE 38
Example 39
Introduction 35
Usage 36
Installation-defined Information
Displayed by UDF 31,99
Introduction
API for JBS 16
DD Subsystem 15
JBS Environments 65
JECL Support 15
Job Binding Services 12
Job Chaining Services 14,103
Job Limiting Services 13
Mellon Bank Compatibility 15,109
Software Access Control 43
Started Task Support 15

J

JAL
Considerations 26
Environments 86
JAL Considerations 92
JAL Extensions
JBS 12
JAL VERIFY
Binding Agents 27
JAL Verification
DD SUBSYS 115
JBS
Incompatible Agents 35
JBS ABANDON
Operator Command 32
JBS ACTIVATE
DD Subsystem Interface 116
Operator Command 32
Started Tasks 116
JBS API
Introduction 16
JBS BIND
DD Subsystem Interface 116
Started Tasks 116
JBS DEACTIVATE
DD Subsystem Interface 116
Operator Command 32
Started Tasks 116
JBS DEFINE
Operator Command 32
JBS DELETE
Operator Command 32
JBS DISPLAY
Mellon Bank Compatibility 113
Operator Command 32
JBS Environments
Defining 76
Definition Considerations 83
Deleted 84
Implementation 69
Installation 82
Introduction 65
MHS Considerations 84
Resource Element 66
Supporting Components 67
Undefined 83
Using JAL 86
Using JECL 86
JBS Files
Control File 16
Introduction 16
JBS Function
API JECL Statement 123
Considerations 24
Description 17
JBS Function Interface
Started Tasks 115
JBS MESSAGE
DD Subsystem Interface 116
JECL Statement 123
Started Tasks 116
JBS Message
API 123
JCS AFTER
Before & After 108
JECL Statement 108
JCS BATCH
Before & After 108
JECL Statement 108
JCS BEFORE
Before & After 108
JECL Statement 108
JCS DISPLAY
Mellon Bank Compatibility 113
Operator Command 108
JCS RELEASE
Operator Command 108
JCS WITH
JECL Statement 114
Mellon Bank Compatibility 114
JECL
Binding Agents 22
Considerations 24
Environments 86
JECL Statements
Before & After 104
Job Binding Services 32
Job Chaining Services 104,108
Job Limiting Services 100
Mellon Bank Compatibility 111,114
JECL Support
Introduction 15
JES2 Exit 14
JLS 97
JLS
JES2 Exit 14 97
QGET 97
JLS ABANDON
Operator Command 100
JLS DISPLAY
Mellon Bank Compatibility 113
Operator Command 100

JLS ENQ
JECL Statement 100,101,114
Mellon Bank Compatibility 114
JLS Function
Description 87
Implementation Summary 88
Limiting Agents 88
JLS Function Interface
Started Tasks 115
JLS LIMIT
DD Subsystem Interface 117
JECL Statement 100,101
Started Tasks 117
JLS RECONCILE
Operator Command 100
JLS RESET
Operator Command 100
JLS SET
Operator Command 100
JLS_LIMITDEF
Job Limiting Services 89
Limiting Considerations 93
Job Binding Services
API Fixed Message 120
API Message Pattern 120
Component 11
Exit 13 33
Exits 16,33
Facilities Summary 32
Fixed Message API 23
Functional Summary 11
Introduction 11,12
JAL Extensions 12
JBS ABANDON Command 32
JBS ACTIVATE Command 32
JBS DEACTIVATE Command 32
JBS DEFINE Command 32
JBS DELETE Command 32
JBS DISPLAY Command 32
JECL Statements 32
Message Pattern API 23
Operator Command 32
UDF Extensions 30
Job Chaining Services
AFTER Statement 108
BATCH Name 103
BATCH Name Conventions 104
BEFORE Statement 108
Before & After 103
Facilities Summary 108
Implementation 104
Introduction 14,103
JCS BATCH Statement 108
JCS BEFORE Statement 108
JCS DISPLAY Command 108
JCS RELEASE Command 108
JECL Statements 104,108
Operator Command 108
Requirements 103
UDF Extensions 107
Job Limiting Services
Facilities Summary 100
Introduction 13
JECL Statements 100
JLS ABANDON Command 100
JLS DISPLAY Command 100
JLS ENQ Statement 100,101
JLS LIMIT Statement 100,101
JLS RECONCILE Command 100
JLS RESET Command 100
JLS SET Command 100
JLS_LIMITDEF in JAL 89
Operator Command 100
UDF Extensions 99
Job-related Attribute
Binding Agent 19

K

Keywords
AGENT in TYPE(PRODUCT) Statement 50
CLASS in TYPE(PRODUCT) Statement 51
DISPLAY_ONLY in TYPE(PRODUCT) Statement 51
FAIL_MSG in TYPE(PRODUCT) Statement 51
LOG_MSG in TYPE(PRODUCT) Statement 51
NOTES in TYPE(PRODUCT) Statement 51
RESOURCE in TYPE(PRODUCT) Statement 51
SAF_AUTHORITY in TYPE(PRODUCT) Statement 50
WARN_MSG in TYPE(PRODUCT) Statement 51

L

LOCAL Attribute
Binding Agent 20
LOG Attribute
Binding Agent Definition 21
LOG_MSG Keyword
TYPE(PRODUCT) Statement 51
Limiting Agents
Controlling 95
Defining 90
Description 88
Example of Naming 94
JESplex 96
JLS Function 88
Naming Relationships 93
Operational Control 95
Scope 96
System Level 96
Use of STRINGDEF for Naming 95
Limiting Considerations 93
JLS_LIMITDEF JAL Statement 93

M

MULTIPLE Attribute
Binding Agent Definition 21
Mellon Bank Compatibility
$QA Command 110
$QD Command 110

$RESTABL Macro 110
$RESTABL Search 111
CNTL Facility 112
CNTL Statement 114
CNTLDEF in JAL 112
Command 113
Component Enabling 109
EXC Keyword 112
Facilities Summary 113
HERE Keyword 111
Introduction 15,109
JBS DISPLAY Command 113
JCS DISPLAY Command 113
JCS WITH JECL Statement 114
JECL Statements 111,114
JLS DISPLAY Command 113
JLS ENQ JECL Statement 114
ROUTE XEQ JECL Statement 114
Resource Routing 109
Resource Routing Definition 110
SHR Keyword 112
SYS Keyword 111
WITH Facility 112
WITH JECL Statement 114

Message Id
API 120

Message Pattern
API 120

N

NOTES Keyword
TYPE(PRODUCT) Statement 51

Naming
Binding Agents 18
Limiting Agents 89

O

OPER Attribute Definition
Binding Agent 22

Operational Control
Limiting Agents 95

Operator Commands
Job Binding Services 32
Job Chaining Services 108
Job Limiting Services 100

P

PERMANENT Agents
Activation Considerations 28
Deactivation Considerations 28

PERMANENT Attribute
Binding Agent 19

Packaging
TM/Job Binding Services 11

Q

QGET
JLS 97

R

READ Subparameter
TYPE(PRODUCT) Statement 50

RESOURCE Keyword
TYPE(PRODUCT) Statement 51

ROUTE XEQ
JECL Statement 114
Mellon Bank Compatibility 114

Requirements
Job Chaining Services 103

Reserving
Binding Agents 25

Resource Element
Defining 73
Definition Considerations 84
Deleted 84
JBS Environments 66
Undefined 84

Resource Routing
Defining 110
Mellon Bank Compatibility 109

Route Code
API 120

S

SAC
Associating Tables With Systems 47
Batch Control 56
Introduction 43
Mandatory TYPE Statements 48
Message Inserts 48
Sample Table 61
TSO Control 53
TYPE(BATCH) 56
TYPE(COMMAND) 52
TYPE(EXEMPT_USERS) 53
TYPE(ISPF_COMMAND) 52
TYPE(ISPF_PANEL) 52
TYPE(ISPF_PGM) 52
TYPE(JES2_NAMES) 53
TYPE(MSGID) 48
TYPE(PRODUCT) 50
TYPE(READY_COMMAND) 52
Table Coding Syntax 45
Table Structure 46

SAF_AUTHORITY Keyword
TYPE(PRODUCT) Statement 50

SHR Keyword
Mellon Bank Compatibility 112

STRINGDEF
Limiting Agents Naming 95

SUBSYS Keyword
In DD Statement 116
SYS Keyword
Mellon Bank Compatibility 111
Scope
Limiting Agents 96
Sequencing
Batch Jobs 105
Software Access Control
See SAC
Started Task Support
Introduction 15
Started Tasks
Control Statements 115
EXIT 19 115
JAL Verification 115
JBS ACTIVATE 116
JBS BIND 116
JBS DEACTIVATE 116
JBS Function Interface 115
JBS MESSAGE 116
JCS BATCH Names 105
JLS Function Interface 115
JLS LIMIT 117
Subparameters
ALTER in TYPE(PRODUCT) 50
CONTROL in TYPE(PRODUCT) 50
READ in TYPE(PRODUCT) 50
UPDATE in TYPE(PRODUCT) 50
Subsystem Services
$$TM 116
ThruPut Manager 116
Syntax
DD SUBSYS 116
System Level
Limiting Agents 96

T

TM/JBS
Component 11
TSO Control
Software Access Control 53
TSO Session
JCS BATCH Names 105
TYPE(BATCH)
SAC TYPE Statement 56
TYPE(COMMAND)
SAC TYPE Statement 52
TYPE(EXEMPT_USERS)
SAC TYPE Statement 53
TYPE(ISPF_COMMAND)
SAC TYPE Statement 52
TYPE(ISPF_PANEL)
SAC TYPE Statement 52
TYPE(ISPF_PGM)
SAC TYPE Statement 52
TYPE(JES2_NAMES)
SAC TYPE Statement 53
TYPE(MSGID)
SAC TYPE Statement 48
TYPE(PRODUCT)
AGENT Keyword 50
ALTER Subparameter 50
CLASS Keyword 51
CONTROL Subparameter 50
DISPLAY_ONLY Keyword 51
FAIL_MSG Keyword 51
LOG_MSG Keyword 51
NOTES Keyword 51
READ Subparameter 50
RESOURCE Keyword 51
SAC TYPE Statement 50
SAF_AUTHORITY Keyword 50
UPDATE Subparameter 50
WARN_MSG Keyword 51
TYPE(READY_COMMAND)
SAC TYPE Statement 52

U

UDF
Displaying Installation JBS Information 31
Displaying Installation JLS Information 99
JBS Extensions 30
JCS Extensions 107
JLS Extensions 99
UNIQUE Attribute
Binding Agent Definition 21
UPDATE Subparameter
TYPE(PRODUCT) Statement 50

V

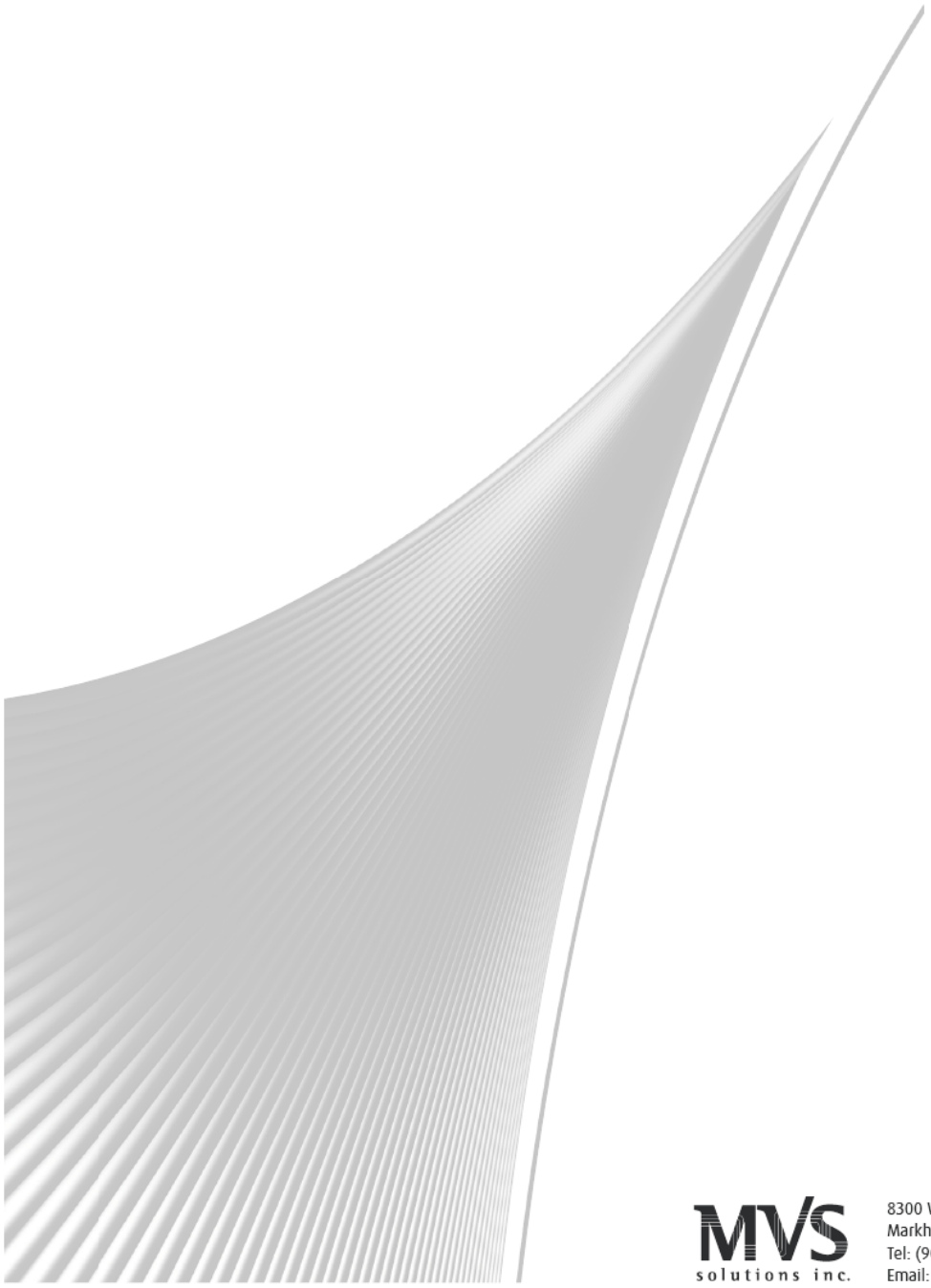
Verifying
Binding Agents 26

W

WARN Attribute
Binding Agent Definition 21
WARN_MSG Keyword
TYPE(PRODUCT) Statement 51
WITH
JECL Statement 114
Mellon Bank Compatibility 114
WITH Facility
Mellon Bank Compatibility 112

X

XMIT Attribute
Binding Agent 20
Binding Agent Definition 21



MVS solutions inc.

8300 Woodbine Avenue, 4th Floor
Markham ON Canada L3R 9Y7
Tel: (905) 940-9404 Fax: (905) 940-5308
Email: marketing@mvssol.com www.mvssol.com