IBM

JES3 to JES2 Migration Considerations

Differences between JES2 and JES3

Preparation and planning for a migration

Migration considerations

Frank Kyne
Karan Singh
Bruce Dennis
Brad Habbershaw
John A. O'Leary
John Unterholzner
Tom Wasik

Redbooks

ibm.com/redbooks

International Technical Support Organization

JES3 to JES2 Migration Considerations

December 2014

SG24-8083-00

Note: Before using this information and the product it supports, read the information in “Notices” on page xi.

First Edition (December 2014)

This edition applies to Version 2, Release 1, Modification 0 of z/OS (product number).

Contents

Notices

Trademarks

AS/400®	NetView®	RMF™
CICS®	OS/390®	System i®
DB2®	Parallel Sysplex®	System z®
GDPS®	POWER®	Tivoli®
Global Technology Services®	Print Services Facility™	VTAM®
IBM®	RACF®	z/OS®
IMS™	Redbooks®	z/VM®
iSeries®	Redbooks (logo) ®	z/VSE®
MVS™	RETAIN®	zEnterprise®

Preface

This book deals with the migration from JES3 to JES2. Part One describes this decision. Part Two describes the steps and considerations of this migration.

This IBM® Redbooks® publication provides information to help clients that have JES3 and would like to migrate to JES2. It provides a comprehensive list of the differences between the two job entry subsystems and provides information to help you determine the migration effort and actions.

The book is aimed at operations personnel, system programmers, and application developers.

Authors

This book was produced by a team of specialists from around the world working at the IBM International Technical Support Organization (ITSO), Poughkeepsie Center.

Frank Kyne is an Executive IT Specialist at the IBM International Technical Support Organization, Poughkeepsie Center. He writes extensively and teaches IBM classes worldwide on all areas of IBM Parallel Sysplex® and high availability. Before joining the ITSO 17 years ago, Frank worked in IBM Ireland as an IBM MVS™ system programmer.

Karan Singh is a Project Leader at the ITSO Poughkeepsie Center, responsible for producing various IBM System z® and IBM z/OS® IBM Redbooks publications. A generalist by nature, with over 15 years of working with various clients and IBM internal systems, he has acquired expertise in core z/OS technologies and a broad familiarity with System z. Karan holds a Diploma in Computer Programming, a Bachelors Degree in Liberal Arts, and a Masters Degree in Teaching.

Bruce Dennis is an Infrastructure System Engineer with Nationwide Insurance in Columbus, Ohio. He has 30 years of experience in various z/OS areas. He has held several roles at Nationwide including the z/OS Infrastructure Architect. His areas of expertise include Sysplex planning, IBM CICS® systems, and Application design. He has been a speaker at technical conferences and is a steering committee member of Central Ohio Mainframe Users Group (COMUG). Bruce holds an Associate Degree in Computer Processing and a Bachelors Degree in Business Administration.

Brad Habbershaw is an I/T Specialist in IBM Canada as the system programmer team lead for the Infrastructure Services, Securities Industry Services for IBM Global Technology Services®, Canada. He has over 27 years of experience in the I/T field specializing in z/OS, Parallel Sysplex, IBM RACF®, and many other IBM products. He holds a B.Sc. Degree from Western University in Ontario, Canada. He has written extensively on parallel sysplex operations and z/OS high availability topics.

John A. O’Leary is an MVS System Programmer in the IBM East Fishkill, NY Global Delivery Center. He has 34 years of experience in System z operations and system programming.

John Unterholzner is a Software Engineer for z/OS Core Technologies at IBM Rochester, MN. He has 13 years experience working with z/OS JES3 and IBM iSeries® Clustering. He holds bachelor degrees in Electrical Engineering and Computer Science from the University of Wisconsin Platteville and a Masters degree in Computer Science from the University of Minnesota. His areas of expertise include communication protocols, groups services infrastructure, and z/OS job processing. He has published an article in the IBM Systems Magazine.

Tom Wasik is a Software Engineer for z/OS Core Technologies at IBM Rochester, MN. He has 28 years of experience working in JES2 design and development. He holds degrees in Chemistry and Computer Science from LeMoyne College and Syracuse University. His areas of expertise include all aspects of JES2 processing, performance, security, NJE networking, and general z/OS processing. He has written and presented extensively on JES2 processing.

Thanks to the following people for their contributions to this project:

Mike Ebbers
Bob Haimowitz
International Technical Support Organization, Poughkeepsie Center

Chad Gavorcik
Nationwide Insurance, US

Dagmar Harms
SIGNAL IDUNA Gruppe, Germany

Lutz Kuehner
Credit Suisse Group, Switzerland

Jean-Louis LaFitte
Consultant, Switzerland

Ituriel Do Nascimento Neto
Bradesco Bank, Brazil

Mike Cox
Don Goldie Jr.
John Hutchison
David Jones
Peter Kania
Kevin Kelley
John Kinn
Tom Nisonger
Gary Puchkoff
Greg Thompson
IBM US

Now you can become a published author, too!

Here’s an opportunity to spotlight your skills, grow your career, and become a published author—all at the same time! Join an ITSO residency project and help write a book in your area of expertise, while honing your experience using leading-edge technologies. Your efforts will help to increase product acceptance and customer satisfaction, as you expand your network of technical contacts and relationships. Residencies run from two to six weeks in length, and you can participate either in person or as a remote resident working from your home base.

Find out more about the residency program, browse the residency index, and apply online at:

`ibm.com/redbooks/residencies.html`

Comments welcome

Your comments are important to us!

We want our books to be as helpful as possible. Send us your comments about this book or other IBM Redbooks publications in one of the following ways:

- Use the online **Contact us** review Redbooks form found at:

  `ibm.com/redbooks`

- Send your comments in an email to:

  `redbooks@us.ibm.com`

- Mail your comments to:

  IBM Corporation, International Technical Support Organization
  Dept. HYTD Mail Station P099
  2455 South Road
  Poughkeepsie, NY 12601-5400

Stay connected to IBM Redbooks

- Find us on Facebook:

  `http://www.facebook.com/IBMRedbooks`

- Follow us on Twitter:

  `https://twitter.com/ibmredbooks`

- Look for us on LinkedIn:

  `http://www.linkedin.com/groups?home=&gid=2130806`

- Explore new Redbooks publications, residencies, and workshops with the IBM Redbooks weekly newsletter:

  `https://www.redbooks.ibm.com/Redbooks.nsf/subscribe?OpenForm`

- Stay current on recent Redbooks publications with RSS Feeds:

  `http://www.redbooks.ibm.com/rss.html`

Part 1

The decision to migrate from JES3 to JES2

This book provides information for organizations that have both JES3 and JES2 and would like to consolidate onto JES2. It will also be beneficial to organizations that have only JES3 and are considering migrating to JES2.

In Part 1, we cover the information that you need to make a migration decision. In Part 2, we cover the information that you need to plan your migration.

Perhaps one of your first questions is “Why would an enterprise want to convert to a different job entry subsystem”? There are a number of possible reasons why you might be considering such a move:

- Because of mergers or acquisitions, you now have both JES3 and JES2 and do not want to have to maintain skills in both products.
- You have both JES3 and JES2, and would like to have consistent JCL and procedures across all your z/OS systems.
- Because there are more JES2 than JES3 installations, it might be easier in your area to find personnel with JES2 experience.
- It is possible that products you would like to use do not support JES3.
- Perhaps a certain product is better tested with JES2.
- You might find that there are features of JES3 that you no longer use.
- New functions often appear in JES2 before they appear in JES3, and you would like to remain current in your product levels.
- You have done a financial analysis and found that costs could be reduced by consolidating systems and converting them to JES2.
- You are working to improve your availability and JES2 appears to provide more flexibility to make dynamic changes than JES3 does.

Not all of these apply in every case. As with any IT strategy, your decision will be based on a thorough analysis of the costs and benefits of migrating, based on the latest information.

This book helps you identify what the migration effort would be. For some JES3 installations, the migration might be relatively easy, but for others it will be time-consuming and complex. It depends on the extent to which you exploit the capabilities that are unique to JES3.

Positioning for migration

Many of the issues that need to be addressed when performing a JES3 to JES2 migration pertain to the use of facilities that, at one time, were only provided by JES3. Over time, many of these facilities were provided by the operating system. However, naturally, people continue using things that they are familiar with.

Even if you never migrate to JES2, it is a good idea to ensure that any *new* applications or new jobs avoid the use of facilities that are unique to a particular Job Entry Subsystem (JES). Most enterprises take a number of years to deliberate over whether they will perform the migration or not. During that time, you could potentially be creating many more things that will subsequently need to be addressed as part of the migration.

In the opinion of the authors, it is a very good investment of your time to put tools, documentation, and education in place now to ensure that your users (including operators, production schedulers, application developers, and system programmers) stop using mechanisms that are unique to JES3. If you have both JES2 and JES3 today, it is worth considering to stop using mechanisms that are unique to any JES.

6.1.1, “Positioning moves” on page 90 provides information about changes you can start making in advance of any migration. Additionally, throughout this book we point out changes that can be made now that will not impact current operations under JES3, but that will make the migration easier if you decide to go down that path in the future.

Information provided in this book

The book is divided into two parts. Part One gives you information to make your migration decision. Part Two gives you the information that you need to plan for your migration.

The next chapter provides background information about how JES2 works.

1

How JES2 works

When making a migration decision from JES3, you will want to know what JES2 provides. This chapter is intended for people who are familiar with JES3 and want to understand how JES2 works. It helps you to understand situations where the same function has a different name in JES2 than in JES3.

In many ways, JES2 provides many of the same functions that JES3 does. However, it generally does them in a completely different way. These differences arose over the years, partly because of different functional requirements, but also because of a fundamental difference in philosophy. JES3's basic philosophy is one of centralized control where one global system controls some number of local systems, freeing up the locals to do what is most important to them: run jobs. JES2 however, is a collection of peer systems that independently process and manage work on the input, execution, and output queues. Spool-related work is done, as much as possible, by the process requesting a function, or by the corresponding JES2 address space.

1.1 What is a job entry subsystem (JES)?

The concept of a job entry system originated around 1960. Computers were constantly getting faster, but unit record devices (printers, card readers, and card punches) with their mechanical processes were getting slower relative to processor speeds. When systems performed an input/output (I/O) operation to one of these devices, they would wait and do nothing until the I/O completed. In an effort to improve the utilization of large computer systems, IBM designed a way to use a separate peripheral processor to perform the relatively slow I/O to card readers, printers, and punches. This meant you could run more jobs and make more efficient use of the expensive system.

From this beginning, JES2 and JES3 evolved. The two products were originally designed to address different parts of the mainframe market, but over time their surface functions have been converging.

1.2 JES2 overview

At its most basic level, JES2 provides the same functions as JES3 (or any other batch management system). It manages job and output work queues, implements a SPOOL to temporarily store data (job streams and SYSOUT data sets), and deals with peripheral devices (card readers, card punches, and printer). Jobs are generally submitted to JES from TSO using the **`SUBMIT`** command that writes card images to an internal reader. An internal reader is just a logical card reader. Job control language (JCL) is written in 80-character card images with room for sequence numbers in columns 73-80 (a carry-over from the days when you could drop your JCL card deck and had to get it back in the correct sequence).

JES2 also supports input and output using network job entry (NJE) protocols and remote job entry (RJE). These are ways to connect other systems to JES2 either as peers (NJE) or as workstations (RJE).

JES2 has an initialization deck that defines how JES2 will operate, operator commands that allow users and operators to interact with JES2, and programming interfaces such as the subsystem interface (SSI) that allow JES2 to communicate with other processes.

Up to this point, the same description applies to both JES2 and JES3. So what is different about JES2? One of the big differences is philosophy (centralized control versus distributed). Another is the scope of the function provided. The two have a different command and initialization deck philosophy. It is these subtle items that make a world of difference in how the two JESs do the same basic processing so differently.

The purpose of this chapter is to help you understand how JES2 does things and, where applicable, contrast that with how JES3 does things. If you want to learn even more about JES2, refer to *z/OS JES2 Introduction*, SA22-7535[1] and *z/OS JES2 Initialization and Tuning Guide*, SA22-7532[2].

1.2.1 JES2 environment

In JES2, the work queue resides in a construct called the *checkpoint*. The checkpoint can be placed in a DASD data set or in a Coupling Facility structure. A JES2 member accesses this

[1] http://www-01.ibm.com/support/knowledgecenter/SSLTBW_2.1.0/com.ibm.zos.v2r1.hasa800/abstract.htm?lang=en

[2] http://www-01.ibm.com/support/knowledgecenter/SSLTBW_2.1.0/com.ibm.zos.v2r1.hasa300/abstract.htm?lang=en

checkpoint and adds work to, or takes work from, one of the work queues. JES2 can have 1 - 32 members accessing the checkpoint. All members must also have access to a set of data sets that contain the SPOOL data. A MAS (also referred to as a JES2 JESplex) is therefore the collection of JES2 members that access the same checkpoint and SPOOL data sets. All members in a JES2 MAS must be in the same z/OS sysplex. You can have multiple JESplexes in a single sysplex, including JES2 and JES3 JESPLEXes in the same sysplex. See Figure 1-1.

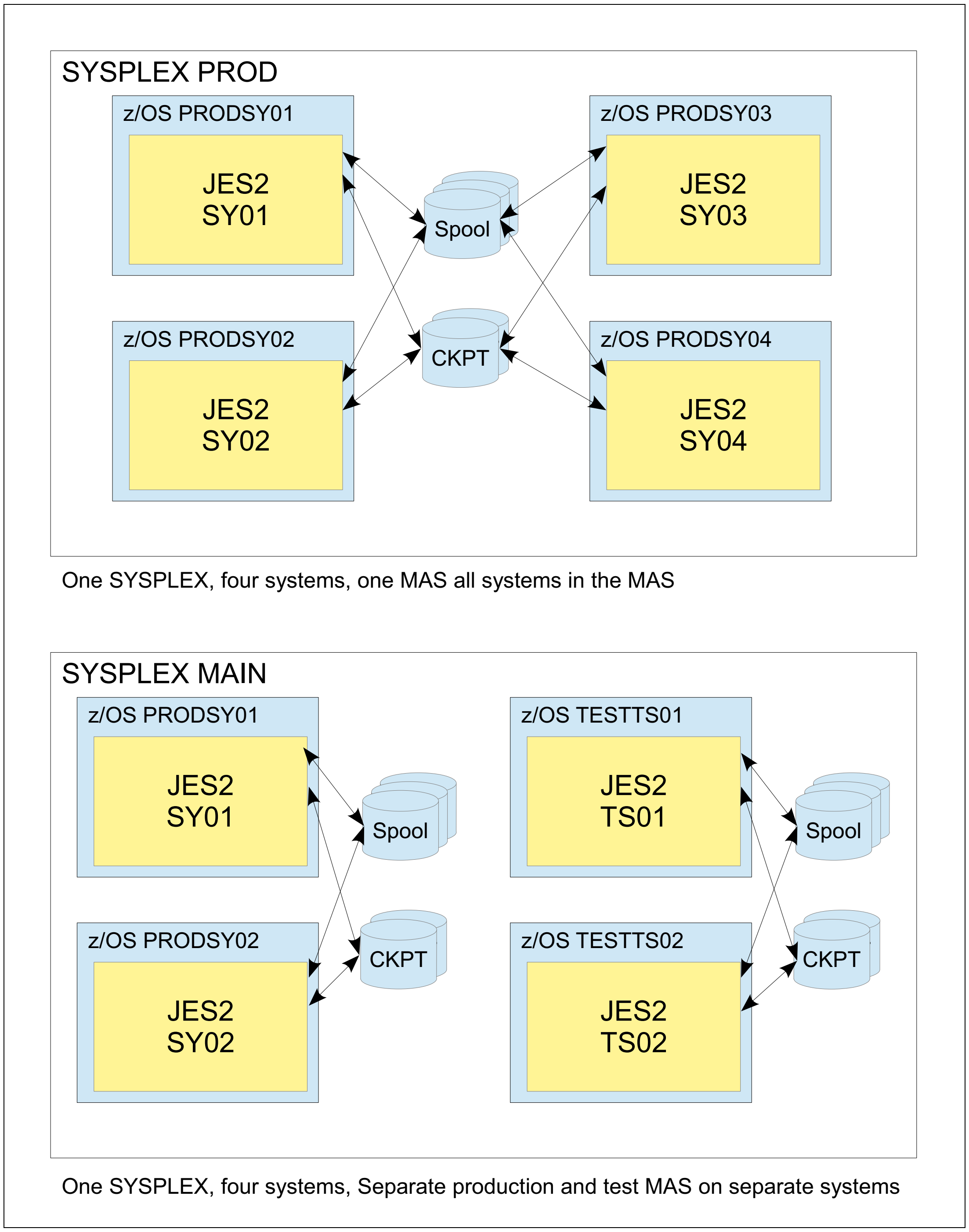


Figure 1-1 JES2 systems sharing checkpoint data sets

JES2 supports a concept called *poly-JES*. This is also known as *secondary subsystems*. In z/OS, when subsystems are defined to the SSI[3] using the IEFSSN*xx* Parmlib member, one subsystem is defined as the primary job entry subsystem and others are considered secondary subsystems. z/OS, by default, routes requests (for example START commands) to the primary subsystem. However, z/OS supports any number of secondary subsystems to be defined. JES2 supports being defined as a primary or a secondary subsystem. This implies that you can define a primary and a secondary JES2 on the same z/OS image. See Figure 1-2.

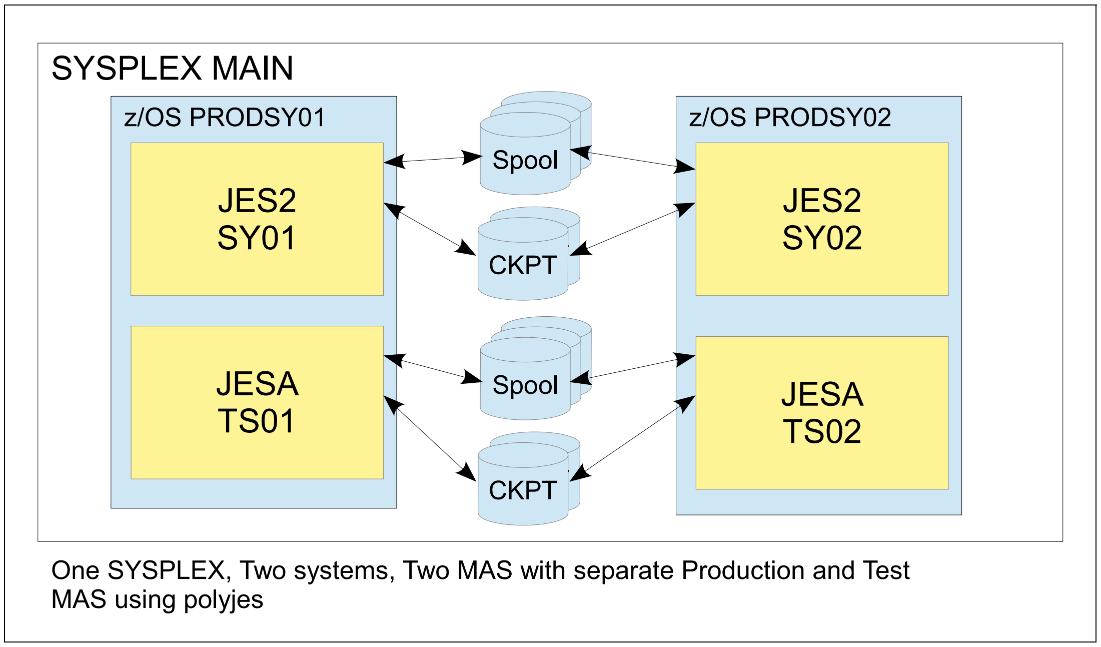


Figure 1-2 JES2 systems with poly-JES

You can also define a JES3 as the primary and JES2 as a secondary subsystem (however running JES2 and JES3 on the same z/OS image is not currently supported, though there are no known problems). If JES3 is running on a system, it *must* be the primary job entry subsystem. You cannot run JES2 as the primary subsystem and JES3 as a secondary subsystem.

A secondary JES2 can be in the same MAS as the primary or in a separate MAS. There are multiple reasons to run a secondary JES2. The first is for testing purposes. Some customers test user exits or JES2 service in a secondary subsystem that is in a separate MAS from the primary but with access to the production environment. Other customers use secondary JES2s with a separate MAS to move work off the production MAS due to capacity issues. One case of this was a customer that had a flood of SYSOUT data sets created in their production MAS. The number of SYSOUT data sets was overwhelming their production archiver, so they used NJE to transfer most of the SYSOUT data sets to a secondary MAS using NJE and then brought them back to the production MAS a few thousand at a time to be archived.

A secondary JES2 in the same MAS as the primary has also been used to isolate functions (for reliability or performance). One example of this is placing SNA networking connections in a secondary JES2. Networking is CPU intensive for the JES2 main task but does not require access to the JES2 checkpoint. Moving it to a secondary JES2 allows frees up resources in the primary JES2 for other work. Of course, moving to NJE over TCP/IP achieves the same

[3] For a discussion of SSI, see ftp://ftp.software.ibm.com/s390/jes2/Share102/NewInter.pdf.

thing, but by doing all the work in a separate address space, without the need for a secondary JES2.

Each JES2 address space operates in a generally autonomous way. Most communication between JES2 address spaces traditionally occurs through the checkpoint data set. More recently, JESXCF is used to broadcast notifications and request specific functions from a member, and multi-system Event Notification Facilities (ENFs) are used to communicate job and outputs transitions. But the checkpoint is still the primary way JES2 members communicate.

All members in JES2 are capable of performing all JES2 functions. There are some functions for which a single member acts as coordinator (such as a checkpoint reconfiguration or SPOOL configuration manipulation) and others that only get performed on one member (such as priority aging). In these cases, if the member coordinating or performing the function fails, the processing moves to another member automatically.

Functions like RJE and NJE can be performed on any member of a MAS (including secondary JES2s). From an NJE point of view, the MAS is a single NJE node (just like JES3). However, a single z/OS image can have multiple JES2s in different MASes. So a single z/OS image can have multiple NJE node names (one for each MAS). NJE can be done between any two JES2s that are in separate MASes, including between a primary JES2 and a secondary one, or between two secondary JES2s. Of course, NJE can also be done between JES2 and JES3 or any other operating system that supports the NJE protocol.

The implementation of NJE over TCP/IP in JES2 and JES3 is implemented by a common set of services in using a separate NETSERV address space. Though the externals that manage NJE are different between JES2 and JES3, the underlying code is the same.

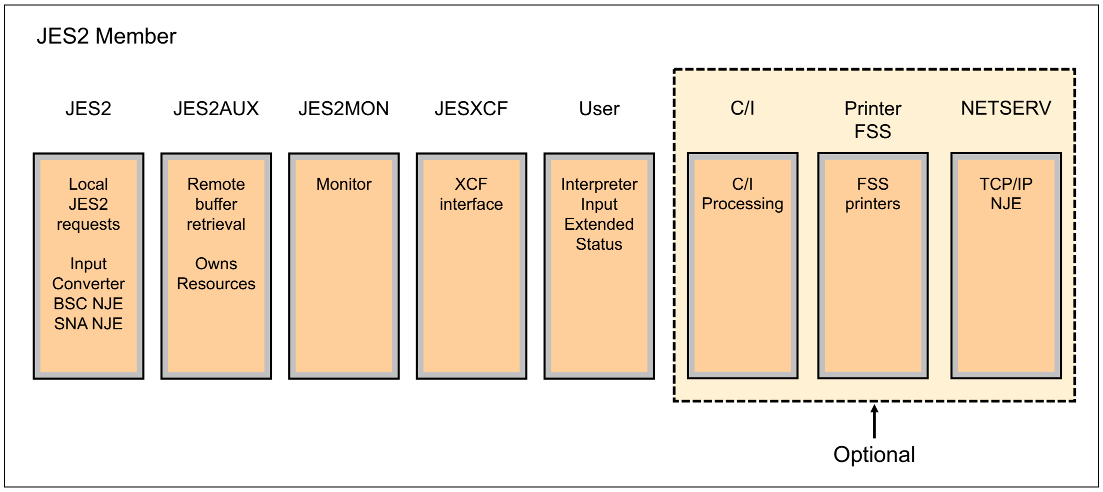


Figure 1-3 Address spaces associated with JES2

A JES2 member is composed of a number of address spaces and data areas. The JES2 address space is the started task that is created by the `S JES2` command. As part of the initialization of JES2, two additional address spaces are created, JES2AUX and JES2MON (note that “JES2” is replaced with the actual subsystem name specified on the start command). The JES2AUX address space is the owner of various resources associated with JES2. It also runs some code to process requests from other MAS members. It cannot be canceled or forced. The JES2MON address space monitors the health of JES2 and ensures that it is processing normally.

JES2 uses XCF messaging to communicate between members and to determine member status. Similar to JES3, JES2 uses JESXCF to manage this communication. The JES2 messages consist of smaller messages between all members of the MAS. This differs from JES3 where messages can be larger and are primarily between the global and the local. JES2 also uses messaging to transport unwritten buffers between address spaces for spool data set browse.

A number of other optional address spaces are associated with JES2 based on functions used by JES2. Print processing can be done using the Functional Subsystem Interface (FSI). This interface, similar to the JES3 FSI, is used in JES2 to interact with products such as PSF. JES3 also uses the FSI for other processing, but JES2 only uses it for printing. NJE over TCP/IP is implemented using NETSERV address spaces. Finally, as of z/OS 2.1, JES2 supports invoking both the MVS converter and interpreter during the conversion phase of job processing. If this option is active, the processing is done in a JES2CI address space.

In addition to these address spaces, there is code that implements the JES2 side of the SSI and code that implements the SPOOL access method (known as HAM or HASP Access Method). These interfaces connect running address spaces and z/OS functions with JES2. Even when the JES2 address space terminates (such as when it ABENDs), these interfaces are still active. They continue to function so long as they do not require functions from the JES2 address space. If the requests require services from the JES2 address space (on that member), they will either wait for JES2 to be restarted, or fail the request.

Whether an interface can continue to function without the JES2 address space depends on where the code that implements the function runs. This differs greatly between JES2 and JES3. For example, creating, opening, and writing to a SYSOUT data set does not require any processing in the JES2 address space. However, creating and writing to a SYSOUT data set does require spool space. Each address space has some spool space available from its last allocation of space and JES2 maintains a cache of spool space (called the *BLOB*) for use by running address space. If there is available space, address spaces can continue to create and write to SYSOUT data. But once that space is exhausted, the address space waits for spool space.

Other functions always require the local JES2 address space to be active. For example, submitting a job, placing a job into execution, taking a job out of execution, processing a start command, or logging on to TSO all require services from the JES2 address space. But it is only the JES2 address space on the member where the request originated that must be active. Loss of a JES2 address space only impacts the work associated with that JES2, not work associated with other systems.

When shutting down JES2, you can either stop just the JES2 address space or try to completely shut down JES2. Completely shutting down JES2 is done using a **`$P JES2`** command with no operands, or with the **`QUICK`** operand. The command is only accepted if there are no address spaces connected to JES2 and if there are no active JES2 processes. If a **`$P JES2`** command is successfully processed, not only will the JES2 address space be stopped, but the JES2AUX, JES2MON, and any JES2CI address spaces are stopped. JES2 also will disconnect from the SSI. This returns z/OS to a state similar to what it was had JES2 never been started. This is called a clean JES2 shutdown (or just a **`$P JES2`**).

Often a clean shutdown of JES2 is not practical for the primary subsystem due to the number of address spaces that connect to JES2. But a secondary JES2 can more easily be shut down cleanly, allowing more possibilities for how secondaries are restarted.

After a clean shutdown of all JES2 members of a MAS, the systems are in a state where an all-member warm start or even a cold start could be performed. So strictly speaking, a cold start does not require any systems to be IPLed. However, in the case of a primary subsystem, it is often the most practical thing to do.

1.2.2 JES2 PROC JCL

JES2 (and in fact any JES subsystem) runs as a started task under the master subsystem of z/OS. As a started task, it requires a PROC that is accessible to the master subsystem (in other words the PROC must be in a concatenation defined in the **`IEFPDSI DD`** in the MSTJCL*xx* member of Proclib). There are two styles you can use when defining the JES2 PROC. The traditional style has Data Definition (DD) cards pointing to the initialization deck, PROCLIB concatenations, and often a STEPLIB DD. Then there is the minimalist approach, where you only have a `PROC` and `EXEC` card. Both are described here. However, you can also use some middle ground between the two approaches.

To STEPLIB or not to STEPLIB

One of the first questions you need to decide is how to package your JES2 load modules. JES2 supports placing all its load modules (and your exit modules) into a single data set. You can then place that data set in a STEPLIB in the JES2 PROC or add it to link list. This is useful if you want to run a secondary JES2 that has a different level of code from the primary JES2. Each JES2 can point to separate data sets that they will use to load the JES2 code.

Alternatively, you can place the load modules whose names start with HASC into LPA and place the remaining modules in the LINKLST concatenation. There is a small advantage to placing the HASC* modules into read-only LPA because that protects them from being overlaid. However, depending on your setup, it might be less error prone to just put all the JES2 modules into a single data set. If that is your choice, placing it in STEPLIB in the JES2 PROC with a parameter on the PROC that allows you to start from an alternate STEPLIB can give you a back-out path in case a bad fix is placed in the library.

You can also use a combination of these two methods by placing the code for the primary JES2 into the LINKLST or LPA, and use a STEPLIB for your secondary JES2.

JES2 initialization statements

JES2 initialization statements can be obtained from either the `HASPPARM DD` in the JES2 PROC or from the logical PARMLIB concatenation defined to z/OS. The `HASPPARM DD` points to a sequential data set (it can be a concatenation of sequential data sets or PDS members) that is read to obtain the JES2 initialization statements. The logical PARMLIB is a z/OS facility that allows programs to read initialization statements from a logical concatenation of PARMLIB data sets without having to have a DD statement pointing at the library.

Most installations do not place their JES2 initialization decks in the normal PARMLIB data set, preferring instead to keep them in a separate JES2PARM data set. However, in an effort to simplify your JES2 PROC, you can create a single member in the PARMLIB data set that has a single statement to include additional initialization streams from the JES2PARM data set. This allows a separate data set for the JES2 initialization statements but does not require a `HASPPARM DD` in the JES2 PROC.

PROCLIB concatenations

JES2 supports multiple PROCLIB concatenations, each of which can be associated with particular job classes (including TSO logons and started tasks) or specified in JCL. Traditionally, these are allocated as DDs in the JES2 PROC. The DD name was generally of the form PROC*xx* where *xx* is a 2-digit concatenation number.

However, PROCLIB data sets can now be allocated using the `PROCLIB` statement in the JES2 initialization deck or by using the **`$ADD PROCLIB`** command. The advantage is a simplified JES2 PROC and less chance of JES2 encountering a JCL error when you try to start it. You can also update PROCLIB concatenations using the **`$T PROCLIB`** command.

Putting it all together (sample JES2 PROCs)

Given all this, the most basic JES2 PROC would be what is shown in Example 1-1.

Example 1-1 Simplified JES2 PROC

```
//JES2     PROC
//IEFPROC  EXEC PGM=HASJES20,TIME=1440,REGION=0M,PARM=(WARM,NOREQ)
```

The PROC in Example 1-1 assumes that JES2's load modules are in LINKLST, the PROCLIBs are defined in the JES2 initialization deck, and the JES2 initialization parameters are contained in the HASJES2 member of the logical PARMLIB. There is not much that can go wrong with this PROC (in terms of JCL errors with the PROC). And there is nothing that ever needs to be changed.

On the other extreme, Example 1-2 contains a JES2 PROC that specifies all the possible parameters that are traditionally in the JES2 PROC.

Example 1-2 Traditional JES2 PROC

```
//JES2     PROC DSN1='SYS1.PROCLIB',     * STANDARD PROCLIB.      *
//         DSN2='SYS1.PROCLIB2',         * SYSTEM SUPPORT PROCS.  *
//         DSN3='SYS1.PROCSPP',          * PROG. PRODUCT PROCS.   *
//         DSN4='SYS2.PROCLIB',          * USER PROCLIB.          *
//         STEPLIB='SYS1.JES2.SHASLNKE', * JES2 PGM LIBRARY       *
//         MBR=HASIPARM                  * DEFAULT PARMLIB MEMBER *
//*                                                               *
//IEFPROC  EXEC PGM=HASJES20,TIME=1440,REGION=0M,
//         PARM=(WARM,NOREQ) SPECIFY START OPTIONS
//STEPLIB  DD   DSN=&STEPLIB,DISP=SHR (MUST BE AUTHORIZED)
//*                                                               *
//******** DEFAULT PROCEDURE LIBRARIES LIST **********************
//*                                                               *
//PROC00   DD   DSN=&DSN1,DISP=SHR
//         DD   DSN=&DSN2,DISP=SHR
//         DD   DSN=&DSN3,DISP=SHR
//         DD   DSN=&DSN4,DISP=SHR
//*                                                                *
//******** INDIVIDUAL PROCXX CARDS FOR EACH PROCLIB. **************
//*                                                                *
//PROC01   DD   DSN=&DSN1,DISP=SHR      SPECIFIC DECLARATIONS
//PROC02   DD   DSN=&DSN2,DISP=SHR      FOR USE WITH JOBPARM PROC=
//PROC03   DD   DSN=&DSN3,DISP=SHR      PARAMETERS.
//PROC04   DD   DSN=&DSN4,DISP=SHR                                 *
//HASPPARM DD   DSN=SYS1.PROCLIB(&MBR),DISP=SHR
//PARMBACK DD   DSN=SYS1.JESPARM(&MBR),DISP=SHR     ALTERNATE
//HASPLIST DD   DDNAME=IEFRDER                      LISTING
```

As you can see, the PROC in Example 1-2 has defined all the various types of DDs that can be specified in a JES2 PROC. PARMBACK is an alternate DD name to get the JES2 initialization deck from (this can be selected via **`PARM=`** when JES2 is started). HASPLIST is a DD name that JES2 sends a listing of the processed initialization deck to. This is very useful if your JES2 initialization deck uses system symbols and you want to see how they were resolved.

Because of the number of DDs in the traditional JES2 PROC, there are more opportunities for JCL errors that could prevent JES2 from starting. The use of symbols allows you to fix the problem by simply overriding one of the statements. But diagnosing the problem and the correct override can be complicated if JES2 is not active.

The two styles of JES2 PROC can be combined based on your requirements. You can pick some DDs from the traditional JES2 PROC and use them in the simplified JES2 PROC. The two that are the least risky and most useful are the HASPLIST and the STEPLIB DDs.

You can use the HASPLIST DD to get a listing of the initialization statements that were processed with the symbol substitution resolved. If you have never used IEFRDER, it is a DD created by Start processing that picks up any DD keywords specified on the **`Start`** command. If you use the traditional JES2 PROC and issue the following command, the HASPLIST DD will be assigned to the specified data set name:

```
S JES2,DSN=SYS1.JESPARM.LISTING,DISP=OLD
```

Starting with z/OS 2.1, you can display information about how JES2 was started using the **`$D INITINFO`** command. This displays the command used to start JES2, the initialization decks read, and the STEPLIB specified in the JES2 PROC.

Example 1-3 Sample output of the $D INITINFO command

```
$HASP825 INITINFO  --- Command used to start JES2
$HASP825           S JES2,SUB=MSTR
$HASP825           --- HASPPARM data sets read
$HASP825           DSN=SYS1.JESPARM(JES2Z7TS),VOLSER=SYSAQ4,
$HASP825           CARDS=1763,
$HASP825           DSN=SYS1.JESNODE(CONNECTS),VOLSER=SYSAQ4,
$HASP825           CARDS=4041,
$HASP825           DSN=SYS1.JESNODE(NODES),VOLSER=SYSAQ4,
$HASP825           CARDS=4058
$HASP825           --- STEPLIB Concatenation
$HASP825           DSN=SYS1.SHASLNKE,VOLSER=PRIPKB
```

1.2.3 MAS members

In a JES2 MAS, all members are peers and can perform all JES2 functions. You can choose to run certain functions on one member and not another (for example NJE or printing). Any members can join or leave a MAS at any time without impacting other members. Every MAS is defined as having 32 members even if only one member actually exists. Members do not have to be predefined before initializing. A new member can define itself to a MAS as part of its initialization.

One implication of this processing is there is no such thing as a single system environment in JES2. Even when only one member is active, it is considered a single member MAS. As such, JES2 processing is the same if there is one or multiple members in a MAS.

Each member of a JES2 MAS starts independently, processing the initialization statements read from the initialization deck. The member joins the JESXCF XCF group using group and member information from the initialization deck. This is to inform any other MAS members of its existence and to open a communication path to other members.

At this point, the address space is not yet considered part of the MAS because it has not been assigned a member number. Using the JESXCF group, the member sends a message to other members, asking where the checkpoint data set is located. This is in case the checkpoint had been moved but the initialization deck was not updated to reflect the change. The starting member then obtains the checkpoint data set lock and reads the checkpoint into memory. It is at this point that the JES2 address space assigns itself a member number and joins the MAS. The initialization parameters are validated against the checkpoint and updates made based on the start type.

All start types are processed in the same way until after the checkpoint is initially processed (including cold starts). Only after the checkpoint is initially read does the code start to distinguish the start type. This allows the initializing member to determine the status of other members of the MAS and perform the appropriate processing. On a cold start, if other active members are detected, the cold starting member fails initialization.

There are only two external values specifiable for the JES2 start type: `COLD` and `WARM`. A cold start clears the work queues, the checkpoint, and the SPOOL. A warm start is the normal way JES2 is initialized. The various types of warm starts that JES2 performs is based on the environment JES2 discovers after reading the checkpoint data set. If the SSI environment is active from a previous instance of the JES2 address space (the JES2 address space ABENDed), JES2 performs a hot start. Otherwise, the start is based on the status of other members and the state of the work queues. If there are no other JES2s active (no other JES2 address spaces or SSI environments in this MAS), an all-member warm start is performed. If there are other address spaces active, the start is a either a single system warm start or a quick start. The difference is that for a quick start, there are no residual traces of a previous instance of the member in the job queue (such as jobs in the job queue indicating they are active on the member initializing).

1.2.4 SPOOL

The SPOOL is the bulk data repository in JES2. It primarily contains JES-managed data sets that contain job input (instream data) and output (SYSOUT). This includes job-oriented data sets like the JCL, the output of the converter (internal text), and restart information (the job journal). It also contains a number of job-related control blocks that are used to manage the characteristics of a job and the data sets that it owns.

The SPOOL is composed of up to 253 standard z/OS data sets. Each data set can only have a single extent (this is due to the way the data set is managed). In a MAS, JES2 can only access one SPOOL data set per volume. This is because JES2 uses the volume serial number (VOLSER) that the SPOOL data set resides on to manage the SPOOL data set. Because of this, JES2 uses the terms “SPOOL volume” and “SPOOL data set” interchangeably when referring to a SPOOL extent.

The SPOOL configuration is established when JES2 is cold started, but can be altered via operator command. The SPOOL configuration on a warm start must match the configuration that other MAS members are using. The exception is an all-member warm start, where, in certain situations, a missing SPOOL volume can be removed from the configuration.

There are two ways that JES2 determines the SPOOL configuration on a cold start:

- The traditional method uses the SPOOL data set name and prefix (`DSNAME=` and `VOLUME=` on the `SPOOLDEF` initialization statement) to locate all volumes that match the specified prefix and have the required SPOOL data set on it. All data sets found this way are initialized as SPOOL data sets.
- If all your SPOOL data sets and volumes do not conform to the same naming convention, you can use `SPOOL` initialization statements to define each SPOOL data set you want to

place in your configuration. **SPOOL** initialization statements must be used if the SPOOL prefix specified on the **SPOOLDEF** initialization statement contains the generic specification characters “*” or “?”.

SPOOL volumes can be added after a cold start by using the **$S SPOOL** command. The actual data set can already exist on the volume, or it can be dynamically created on the specified volume by the command. The SPOOL configuration is stored in the JES2 checkpoint data set and restored when JES2 warm starts. Because of this, you are not *required* to update the JES2 initialization data set when a new volume is started. However, it is a good practice to update the initialization deck in case a cold start is ever needed.

You can also use the **$P SPOOL** operator commands to delete a SPOOL volume from the configuration. JES2 tracks at a job level what SPOOL volumes a job has ever used. Before a SPOOL volume can be deleted from the configuration, all jobs that ever allocated space on the volume need to be purged from the system. This can take a number of IPLs plus waiting for the jobs to age off the system. In many installations that do not IPL frequently, this can takes weeks or months.

Alternatively, there is a function called SPOOL migration that can move data from one volume to another using the **$M SPOOL** command. It can also merge SPOOL volumes together. This process does not remove the logical volume from the configuration. Rather, it allows you to stop using a physical volume and delete the SPOOL data set from that volume. The logical volume remains until it can be purged (with the same rules as the **$P SPOOL** command).

It is not required that All SPOOL data sets have the same data set name. However, at the time the volume is defined (either on the **SPOOL** initialization statement or on a **$S SPOOL** command) the data set name must match the current value on **DSNMASK=** on the **SPOOLDEF** statement. Using this mask (with generic characters) limits the format of SPOOL data set name, preventing typing errors from creating or using an unintended data set for SPOOL.

The VOLSER of a SPOOL volume must match the value of **VOLUME=** on the **SPOOLDEF** statement, either as a prefix if there are no generic characters in **VOLUME=** on **SPOOLDEF**, or match using the generic characters. This restriction prevents an unintended volume from being used for SPOOL.

If you have multiple JESplexes in your environment, it is recommended that the VOLSER or data set name for SPOOL clearly indicate what JESplex the SPOOL data set is intended to be used in.

The SPOOL data sets are formatted with records that match the buffer size specified on the **BUFSIZE=** parameter on the **SPOOLDEF** statement. The buffer size can be 1944 - 3992 bytes (though most customers use the default size of 3992). The data set can be formatted before its first use, or JES2 formats it before adding it to the configuration. All volumes must have the same buffer size, and the buffer size cannot be altered without performing a cold start.

SPOOL space is allocated in units called “track groups”. A track group consists of multiple of tracks on the volume, however it is specified in terms of records using the **TGSIZE=** parameter on the **SPOOLDEF** statement. The number of tracks per track group on a SPOOL volume is determined when the volume is started (either by command or cold start). Different volumes can have a different number of tracks per track group. The value of **TGSIZE=** is set based on the type of jobs and data sets that are typically created. If most SYSOUT data sets are large, a higher value for **TGSIZE=** performs better but could result is less efficient use of SPOOL space. If most SYSOUT data sets are small spin data sets, a lower value for **TGSIZE=** will make more efficient use of SPOOL space but could slow the performance when creating larger SYSOUT data sets.

Another setting on **SPOOLDEF, TRKCELL=**, specifies the number of records per track cell. When a SYSOUT data set is written to a SYSOUT class that specifies **TRKCELL=YES**, space is assigned to the data set in track cell size increments. This improves the efficiency of reading in the data sets later when they are printed (or accessed by applications via SAPI or spool data set browse). The number of records per track will be divided into an even number of track cells. Tracks cells are generally full or half tracks.

SPOOL space ownership is tracked in internal control blocks and using data stored in record zero of the first track in each track group of SPOOL. Record zero in DASD architecture is an 8-byte data record that exists at the beginning of every track. JES2 stores the job number and job key (unique identifier) of the job that owns the SPOOL space. This ensures the integrity of ownership of every track group of SPOOL space. It also allows for a garbage collector to reclaim any SPOOL space whose owning jobs have lost track of it.

One of the disadvantages of using record zero is that JES2 is one of the few functions that uses the record. As a result, there might be applications that copy or move data sets or volumes that might not correctly preserve record zero. Before using any process to clone or move a SPOOL volume or data set, you must ensure that the process preserves the contents of record zero.

The following SHARE presentation provides more information about how to manage the JES2 SPOOL:

https://share.confex.com/share/118/webprogram/Session10844.html

1.2.5 Checkpoint data set

The JES2 checkpoint is used to store various control blocks that represent the various work queues that JES2 manages. It serves two purposes:

- It is used when JES2 restarts to restore the work queues to their previous state.
- It serves as the primary communication between members of a MAS.

Because of this dual role, it needs to be stored on a non-volatile medium, and it will have very short access times.

Each member of the MAS has a complete copy of the checkpoint data in memory. This allows rapid access to the work queues that are placed in the checkpoint. When designing JES2 enhancements, one issue is whether to place data on SPOOL or in the checkpoint. This is a trade-off between the delay created by reading data from SPOOL and the overhead of placing the data in the checkpoint where every member has to maintain a current copy of it. The deciding factor is often if the data is needed for an operator command. To ensure commands can complete quickly, the data needed is generally placed in the checkpoint. Also, data in the checkpoint is available in extended status terse requests and displayed in the SDSF main view. Data on SPOOL is placed in the verbose section in extended status and often only displayed in the alternate view in SDSF.

The checkpoint is shared by MAS members in a time-sliced manner. Each member gets a lock on the checkpoint data set, reads in the changes made by other members, processes the queues, writes updated control blocks back to the checkpoint, and finally releases the lock. It then waits before trying to get the checkpoint again. The amount of time that a member can hold the checkpoint for, and the time it waits before trying to reacquire the checkpoint is controlled by the **HOLD=** and **DORMANCY=** parameters on the **MASDEF** statement.

HOLD= specifies the time (in hundredths of a second) that a member holds the checkpoint. It will be set long enough that a member can select or add items to the work queues. However, specifying too large a value can keep other members from accessing the checkpoint.

DORMANCY= has two parameters (also specified in hundredths of a second). The first parameter is the minimum amount of time a member waits after releasing the checkpoint lock before trying to acquire it again. This is called "the politeness interval". Once the politeness interval ends, the member will only access the checkpoint if it needs it. If there is no process that needs the checkpoint, the member reacquires the checkpoint after the time specified on the second parameter passes. Since new work for a member is only discovered by reading the checkpoint, the second parameter controls the latency of the MAS (the delay between when work is added to the queue and when it is processed).

The settings of these parameters can greatly affect the performance of JES2. When first setting up JES2, there are rules of thumb that can be used to establish initial values, but they will be examined periodically (weekly or monthly at first, and later quarterly or yearly) to see if adjustments are needed. They will also be examined when workloads change.

The checkpoint can be placed on DASD or in a Coupling Facility structure. There are two advantages to placing the checkpoint in a Coupling Facility: better performance and better lock management. When the checkpoint is on DASD, a hardware reserve is used to serialize access to the data set. Some customers running GRS star mode have converted this reserve into a SYSTEMS ENQ. But this lock is often in contention, which increases GRS overhead. When the checkpoint is in a Coupling Facility structure, a lock element on the list structure is used to serialize access, meaning that no RESERVE (or ENQ) is issued. The list structure lock has less overhead and is more "fair". When the checkpoint is on DASD, a RESERVE request will either be granted or will fail (because someone else is holding the RESERVE). If it fails, that system will try again later, with the same chance of success. As a result, it is possible for a large system to dominate access to the checkpoint. When the checkpoint is in a Coupling Facility, if another system is holding the checkpoint when this system looks for it, this system will be placed in a queue. This means that small systems cannot get constantly locked out of the checkpoint by a much larger one.

In general, if there are more than three members in a MAS, placing the checkpoint in a CF structure has advantages. A three-member MAS is a toss up and a two-member MAS generally runs better when checkpoint is on DASD.

JES2 supports two checkpoint data sets: **CKPT1** and **CKPT2**. Depending on the mode the checkpoint is running in (duplex mode or dual mode), these are either primary and backup (duplex mode) data sets, or alternating primary data sets (dual mode). If the checkpoint is in a CF structure, then you *must* use duplex mode. You can use the **$TCKPTDEF MODE=** command to alter the mode as needed.

When introduced, dual mode provided a significant performance advantage by trading off extra CPU overhead for a reduced I/O size. But with modern DASD, the benefit of the reduced I/O size is minimal. Because of the extra CPU cost of dual mode, duplex mode would be the better option for most customers.

IBM strongly recommends always running with two checkpoint data sets in use, and with **CKPT2** always on DASD. Further, IBM recommends running with **DUPLEX=YES** on all members whenever running in duplex mode.

When the checkpoint is in a CF structure, JES2 uses the structure in much the same way it uses a checkpoint on DASD. It defines list elements for each of the checkpoint records in the structure. The amount of space needed by JES2 in the structure does not change based on load or number of jobs in the system. Also, the data in the structure is not used as a staging area to improve performance. The structure is used as a permanent storage medium for the checkpoint data. Loss of a structure that contains current checkpoint data results in some amount of data loss (the difference between what is in CKPT1 and CKPT2). Before a coupling facility is serviced, the checkpoint data must be moved to another coupling facility or back to a DASD checkpoint.

JES2 also supports two backup checkpoints that can be used in the event of an error on the primary checkpoint data sets. These are specified as `NEWCKPT1` and `NEWCKPT2`. When an error occurs, JES2 performs a process called *checkpoint reconfiguration*. For I/O errors, this can either be a manual process or an automated process. If `OPVERIFY=NO` and the appropriate `NEWCKPT1` or `NEWCKPT2` is specified, JES2 will automatically move the checkpoint data sets to the backup locations in the event of an I/O error.

You can also use the checkpoint reconfiguration process to manually move the checkpoint from its current location to a new one. This is done either to relocate the checkpoint (for example, to move it back to its primary location after the I/O error has been fixed) or to move it to a larger data set for expansion purposes.

Operations staff will familiarize themselves with the reconfiguration process to avoid potential delays or outages if it is ever started unexpectedly.

Note: The JES2 checkpoint will not be confused with checkpoint restart processing or printer checkpoints. All are often referred to a *checkpoint processing*.

1.2.6 Functional subsystem interface (FSI)

The functional subsystem interface (FSI) was originally created to address the problem of JES having to provide specific support whenever a new printer was created. In a sense, it was the print driver code of that time. Before the FSI, JES wrote directly to the printer, using the appropriate Channel Command Words (CCWs). JES had to be aware of all the capabilities of the printer and provide support for them.

The FSI addresses this issue by moving the direct communications to the printer into a separate product. JES provided a formal interface to access SYSOUT data sets and for control of the device.

The FSI is what is known as a JES-centric interface. There are definitions for the interface in JES, the commands to start, stop and alter the devices are JES commands, and JES decides (via JES printer parameters) which SYSOUT to process on the printer. It is designed to lock and act like a JES printer.

In contrast, interfaces like process SYSOUT (PSO) and the SYSOUT application programming interface (SAPI) are application-centric interfaces. JES has no definitions or control of the processes that use these interfaces.

JES2 has never used the FSI for anything but printers (whereas JES3 has used the FSI for Converter/Interpreter processing). In the FSI, there are functional subsystems (or FSS address spaces) and functional subsystem applications (FSAs). An FSS is defined in JES2 using the `FSS` initialization statement. This defines the MVS PROC used to start the address space and the characteristics of the address space.

The FSAs are printers within an FSS address space. Each printer that will be running in an FSS specifies the name of the owning subsystem on the `FSS=` statement in the printer definition.

Starting an FSS mode printer will start the FSS address space and activate the FSA within that FSS to process requests. Stopping an FSS mode printer deactivates the FSA and potentially stops the FSS if it is the last active printer and `AUTOSTOP=YES` was specified on the FSS definition.

1.2.7 Remote job entry (RJE)

Remote job entry (RJE) in JES2 works in the same way as remote job processing (RJP) in JES3. It was created to support unit record devices (card readers and punches and printer) outside the traditional data center. In RJE, the JES2 subsystem acts as the server, receiving jobs and commands from the remote workstation, and sending command output and SYSOUT to the remote printers and punches attached to the workstation. Both JESs support BSC, BSC multi-leaving, and SNA protocols when using RJE.

JES2 RJE consoles do not receive all the messages that JES3 RJP consoles do. The JES2 consoles can only receive JES2 messages routed to the console. JES3 RJP consoles can receive other operating system messages in addition to the JES3 messages.

JES2 supports a concept of remote pooling for RJE output. This allows a pool of remotes that sign on as individual RJE devices to be pooled together when printing output. For example, if remotes 2, 3, and 4 are pooled, any SYSOUT routed to one of the remotes can print on any of the remotes.

Since all JES2 address spaces can perform all JES2 functions, RJE devices can be attached to any member of the MAS (including secondary JES2 subsystems).

1.2.8 Network job entry (NJE)

Where RJE is a connection of workstations to servers with the intent to support unit record devices outside the data center, network job entry (NJE) is a connection between peer systems for the purpose of exchanging jobs and SYSOUT.

An NJE network is a collection of nodes. In JES2, a MAS is a single NJE node. NJE connections can be established by any member of a MAS. JES2 supports BSC, SNA, and TCP/IP protocols when connecting to other nodes. Unlike JES3, JES2 is not limited to a single protocol when connecting to other nodes in the network. NJE can be used to send jobs and SYSOUT between JES2, JES3, IBM z/VM® (using RSCS), IBM z/VSE® (using IBM POWER®), and IBM System i® systems.

NJE supports both directly-connected and indirectly-connected nodes. If a node is indirectly connected, NJE jobs are sent to a directly connected intermediate node and then forwarded to the destination node. These intermediate nodes are called *store and forward nodes*.

Network routing is the process of determining how to send NJE jobs to other nodes, potentially through intermediate store and forward nodes. To facilitate this, JES2 has a function called the *network path manager*. In an NJE network of JES2 nodes, there are connection status records that are shared across NJE connections. These connection records allow JES2 to dynamically determine the routing needed to reach a destination node. Connections that involve non-JES2 nodes are called *non-path manager connections*. Connections between JES2 nodes can also be defined as non-path manager connections. JES2 must have `CONNECT` statements defined to represent connections between non-path manager nodes.

The JES2 path manager uses the path manager connection records combined with static `CONNECT` statements to determine the path (or routing) to get to nodes in the NJE network.

When defining an NJE network, you must ensure that the proper connect statements and other network parameters are set to ensure consistent flow of data through the network.

The following SHARE presentation provides more information on various JES2 NJE parameters:

`https://share.confex.com/share/119/webprogram/Session11757.html`

1.2.9 JESXCF

JES2 uses the checkpoint data set as the primary way to communicate between members in a MAS. However, some messages need a more direct and immediate mechanism. For this, JES2 uses XCF messaging managed by the JESXCF component. JESXCF extends the standard XCF messaging model to be more compatible with the subdispatching needs of JES2. It also provides member and system status updates to JES2.

JESXCF is started automatically during z/OS initialization processing. All JES2 address spaces running on a system use the same JESXCF address space. There are multiple JESXCF/XCF groups that JES2 members join to perform different functions. The main group defaults to the NJE node name of the MAS the member is in. However, this can be altered using the **`XCFGRPNM=`** parameter on the **`MASDEF`** statement.

JES2 uses JESXCF to process the following:

- Member and system status tracking. The status information is used to control processes like resetting failed members and to managing processes such as starting and stopping SPOOL volumes.
- The checkpoint reconfiguration process.
- SPOOL migration coordination.
- Cross member data retrieval (such as in storage buffers from other members).
- Posting members for various queue status changes.

JES2 also uses XCF indirectly when it issues multi-system ENFs 58, 70, and 78. These do not go through JESXCF, but they do use similar XCF resources. ENFs 58 and 70 are used by JES2 instances to update each other about what jobs or SYSOUT groups are active on devices reported in the device status SSI. They can also be used by other products to track jobs and SYSOUT as they move through various phases of processing.

1.2.10 JES2 initialization deck

Whenever a JES2 address space initializes, regardless of the start type, it reads the JES2 initialization decks. The initialization statements are read from either the **`HASPPARM DD`** in the JES2 PROC, or from the logical PARMLIB concatenation defined to z/OS.

As each statement is read, JES2 performs system symbol substitution on each initialization statement. This allows installations to customize the initialization of JES2 on a system by system basis, but without having to have unique JES2 parm members for each system.

JES2 supports an **`INCLUDE`** initialization statement that can embed additional initialization statements from either another member of the current input data set, a member from the logical PARMLIB, or a fully specified data set name. Because JES2 does system symbol substitution, the data set or member name that is being included can contain a system symbol. For example:

```
INCLUDE DSN=SYS1.JES2PARM(&SYSNAME)
```

This would include a member from SYS1.JES2PARM that matches the name of the system being initialized.

If the member you want to refer to resides in the logical PARMLIB, you would specify the member name as **MEMBER=** in the **PARM=** for JES2. Otherwise, JES2 attempts to read from the **HASPPARM DD** for initialization statements. If there is no **HASPPARM DD** (or it cannot be opened) and **MEMBER=** was not specified in the **PARM=**, JES2 defaults to reading from the member HAS*jesx*, where *jesx* is the name of the subsystem being started (normally JES2).

As part of streamlining the JES2 PROC, you might not want to have a **HASPPARM DD** in the JES2 PROC and instead use the logical PARMLIB concatenation to start JES2. But you might not want to keep your JES2 initialization decks in the SYS1.PARMLIB data set. This can be accomplished by placing an HASJES2 member in your PARMLIB concatenation that does an include of the appropriate data set and member that contains the JES2 initialization parameters (potentially using system symbol substitution as needed to select a data set or member). An alternative to this is to add the data set with the JES2 initialization decks to the default PARMLIB concatenation and then process initialization statements from there. This is a matter of installation preference.

This is all implying that there can be more than one member that contains your JES2 initialization statements. Generally, that is how customers do things. There are members that have general settings (such as SPOOL and checkpoint definitions, MAS-wide resource limits, MAS NJE parameters, and so forth), a network-wide definition of NJE nodes (shared with other MASes), and a MAS definition of DESTIDs. They then have local (member level) definitions of parameters, printers (and other devices), RJE workstations, and so forth. The number of members generally depends on both style and scope of management (perhaps one person manages NJE nodes so all that data is in a single member).

Besides organization, there is also the question of whether you want to have a general initialization deck that does an **INCLUDE** of the local definitions or a local initialization deck that does an **INCLUDE** for the general definitions. Again this is a question of style and preference.

All these member includes can cause some confusion about exactly which data sets and member were used during initialization. One suggestion to help improve understanding and help to debug problems is to include Display commands in your initialization decks. At the top of every initialization deck member, you can add a **D INCLUDE** statement. This will display (via WTO) where the **D INCLUDE** statement was read from (either PARMLIB member or full data set name and member). This lets you see what initialization decks were used. Another common practice is to display parameters like **CKPTDEF** or **MASDEF** at the end of the main initialization deck. This helps you see what was actually set by the initialization decks.

Beginning with z/OS 2.1, you can also use the **$D INITINFO** command to display where initialization statements were read from. Example 1-3 on page 11 has a sample of the output of this command.

Depending on the type of start that is being performed, some statements that cannot be processed on this type of start will be ignored. The type of start required to alter each JES2 parameter is listed after each keyword in the *z/OS JES2 Initialization and Tuning Reference*, SA22-7533. Statements that try to alter one of the MAS resources limits (like the number of jobs) on an inappropriate start will get a message similar to this:

Example 1-4 Message for ignored limit change

```
$HASP442 INITIALIZATION STATEMENTS CONFLICTING WITH SAVED VALUES FOLLOW:
$HASP496 JOBDEF JOBNUM=500 SAVED VALUE OF 410 WILL BE USED
```

Other initialization statements, such as altering characteristics of a job class, are ignored without a message.

1.2.11 JES2 monitor

To help you determine if JES2 is functioning properly, JES2 has a separate address space (one per JES2 address space) that monitors the JES2 main task. Its purpose is to detect problems in the JES2 main task that could prevent it from performing its normal functions. It does this both by monitoring activity of the JES2 main task and collecting data about resources that JES2 uses. It can also examine status indicators set by the main task that indicate that abnormal conditions exist.

The monitor is started as part of normal JES2 initialization and terminated on a clean shutdown of JES2. It issues messages when certain conditions are detected (such as the JES2 main task looping or waiting). It also supports commands to display abnormal conditions. All monitor commands start with the command prefix followed by the character J. For example, **`$JD MONITOR`** displays the status of the monitor address space.

The command **`$JD STATUS`** is the primary command to determine if JES2 is operating properly. It displays alert messages about abnormal conditions that the monitor has detected, resources that are above warning levels, and conditions that could affect normal JES2 processing.

Information that the monitor collects is also available from an SSI request and also displayed in SDSF (as prefix commands in the MAS panel and the RM panel).

1.3 JES2 internal components

At their most basic, JES2 and JES3 have the same structure. There is a single task within the JES2 address space that performs most of the processing. This is called the *JES2 main task*. There are subtasks that perform work that could possibly result in a WAIT or that need to be offloaded from the main task. JES2 connects to the SSI to handle requests from z/OS or other applications. There is also an interface that provides application access to JES data sets on SPOOL called the *HASP access method* (HAM).

However, the way that JES2 works internally is very different from how JES3 works internally. JES2 has no equivalent to a dynamic service program (DSP) or to the concept of a job segment scheduler (JSS). There is no way to directly specify which phases a job in JES2 will pass through. All jobs move through the same processing phases, affected only by their success or failure of each particular phase.

1.3.1 Processor control element (PCE)

JES2 processing centers around a control block called the *processor control element* (PCE). This is the unit of work (or thread) that is dispatched under the JES2 main task TCB by the JES2 dispatcher. A PCE can represent a device such as a printer. It can represent a phase of processing such as output processing or purge processing. Or it can represent some work process such as process SYSOUT (PSO) or the SYSOUT API (SAPI). All work done by the JES2 main task that is outside the JES2 dispatcher runs under the control of a PCE.

Because JES2 (and JES3) subdispatches a single TCB to perform much of its work, functions running in the main task cannot enter an MVS wait. If this were to happen, it stops all JES work until the wait is resolved. This can cause deadlocks if a JES function is needed to resolve the wait.

Most PCEs are created during JES2 initialization. However some, like PCEs that help perform warm start processing, can be created and deleted as needed. PCEs are defined in a table using the $PCETAB macro. But they can also be defined in installation exits using dynamic tables and the same $PCETAB macro.

1.3.2 Daughter task element (DTE)

Similar to how a PCE represents a subunit of work running in the JES2 main task, a DTE represents an MVS subtask that is running in the JES2 address space. Subtasks are created to perform work that either cannot be done under the JES2 main task or for performance reasons is best done in a separate task.

Operating system functions like WTO, allocation, and conversion can all result in an MVS WAIT. Because of this, these functions must be performed in a subtask. Some subtasks in JES2 are associated with a particular PCE, such as the conversion subtask. Others are associated with a function such as the WTO subtask. Still others are general-purpose subtasks that can perform any work that must be run outside the JES2 main task.

With the introduction of JES2CI address spaces that perform the conversion and interpretation process during the JES2 conversion phase, there can now be subtasks (DTEs) in other address spaces. When the JES2CI address space is being used, the converter subtask is started in the JES2CI address space instead of the JES2 address space. Otherwise, it functions in much the same way as a subtask in the JES2 address space.

1.3.3 Device control table (DCT)

JES2 does have various devices associated with it, though this is not as extensive as JES3 managed devices. The devices that JES2 manages are represented by DCT data areas. Devices include the physical devices like a printer, or a BSC TP line (CTC) and also logical devices like SNA lines or offload devices. The DCT manages the settings and status for the device. Many devices have a direct relationship with a PCE (such as a printer). In other cases, multiple devices can be associated and managed by a single PCE (such as lines). And there is also a concept of subdevices.

A subdevice is either a logical or physical device that is part of a larger device. An example would be a line that is used for NJE can have various transmitter and receiver devices associated with it. These manage the various job and SYSOUT streams that can be active on an NJE connection at any time. Another example is an RJE line can have various printers and readers associated with the RJE workstation. Each RJE device has its own DCT to manage it.

Subdevices are named as part of the primary device. So an NJE line called LINE15 could have a SYSOUT transmitter named L15.ST1. Similarly, remote 6 can have a printer named R6.PR1.

1.3.4 HASP communication tables (HCT and HCCT)

There are two anchor control blocks in JES2. In the JES2 address space, there is the HASP communication table (HCT). This table is pointed to by register 11 when code is running in the main task. It points to the major data areas in the JES2 address space and has fields that reflect the options specified during initialization. The HCT is divided into two sections: Set of local data areas and a set of checkpoint data areas. Any change to the checkpointed section is automatically propagated to other members using the JES2 checkpoint.

In common storage, the HASP common storage communication table (HCCT) anchors the common storage data areas used by JES2. It also has pointers to the data spaces that JES2 used to store some data areas. Code running in the SSI points to this area using register 11. The HCCT also has copies of some of the options that are in the HCT.

1.3.5 Checkpointed data areas

The JES2 checkpoint holds the primary job and output queues, data needed to manage the spool, and other areas JES2 needs to keep members synchronized. It contains data that is needed to start or restart a member. The checkpoint is divided into control information (such as the portion of the HCT that is shared between members), and data blocks called CTENTs. Each CTENT has an array of homogeneous control blocks.

For example, there is a CTENT that tracks what member an RJE device is signed on. Each defined remote has an element in this array. To find the entry for a particular RJE device, you index into the array to find the corresponding entry.

All CTENTs can be thought of as arrays of control blocks. However, most are not directly accessed like the remote sign-on table, but instead contain various chains of array elements. There are queue heads and chain pointers used to run the elements in the section using the array index.

Each CTENT is mapped to 4 K records in the checkpoint data set. The size of the array (and thus the number of records in the checkpoint) is established when JES2 is cold stared. Many of the arrays can have their size altered via operator command. Most arrays can be increased or decreased in size. Increasing the size of an array depends on the availability of space in the checkpoint data sets (or structures).

The size of many of the arrays (CTENTs) in the checkpoint constrains the number of objects JES2 can manage. These limits are monitored by JES2 and reported using the $HASP050 message. You will understand what is stored in the various CTENTs and how reaching the limits can affect your system.

Job queue element (JQE)

The job queue element (JQE) represents a job, started task, or TSO logon. These elements constitute the job queue in the checkpoint. Each active JQE is on one of the phase chains in the checkpoint (or on a free chain). JQEs are in no particular order, but another CTENT, the job index (JIX), can be used to locate the JQE for a given job number. If you run out of JQEs, you cannot submit any jobs, log on to TSO, issue a start command, or receive jobs or SYSOUT over NJE. The number of JQEs is controlled by the **`JOBDEF JOBNUM=`** specification. It can be increased or decreased via operator command.

Job output element (JOE)

When a job created output, it is represented by the Peripheral Data Definition Block (PDDB) on spool. Each instance of a data set has its own PDDB to describe the properties of that instance. When a job completes execution (or when a spin data set is spun), the PDDBs are gathered into job output elements (JOEs) for printing. All SYSOUT data sets that can be accessed by processes like printers, SAPI, and NJE are represented in JOEs.

The number of JOEs is controlled by the **`OUTDEF JOENUM=`** specification. It can be increased or decreased via operator command. If JES2 runs out of JOEs, it cannot make PDDBs available for processing. However, the SYSOUT data sets will still exist on spool. As JOEs are freed up, waiting PDDBs will be processed and placed into the available JOEs. For normal job output (not spin data sets), this has little impact on the system (except for the delay in making the data sets available for processing). But if a job cannot get a JOE for a spin data set, JES2 must go through an I/O intensive process to locate the spin data sets that need to be processed. This can impact the performance of a running system.

Block extension reuse table (BERT)

One of the major limitations of the checkpoint structure is that the CTENTs are arrays of fixed size element. This presents two problems. First, it is difficult to expand the size of the elements in the array. This is due to how the data is accessed by JES2. Second, if the element size is expanded, all elements of a particular type get larger. This requires JES2 to maintain space in the elements for the maximum size of all possible data that could be associated with an element. This is expensive if the data is variable in size (such as the accounting string) or only applies to a small subset of jobs (such as SYSLOG jobs).

The block extension reuse tables (BERTs) were created to address both these problems. Using BERTs, JES2 can add extensions to other existing data areas (such as JOEs and JQEs). These extensions can contain variable and optional data. This allows JES2 to maintain data for some jobs (such as the system name associated with a SYSLOG job) that do not consume space in the checkpoint for jobs that do not need it.

In addition, BERTs can be used to create other control blocks that do no warrant their own section in the checkpoint. This is either because there are too few instances of the data areas, or the number of data areas needed is not predictable. Examples are data areas that track duplicate job names (DJBs) or data areas that represent job classes (CATs).

Since their introduction, BERTs have become more integral to normal JES2 operation. However, this also implies that running out of BERTs (they are still an array in the checkpoint) can have varied and serious impact. So the question is always asked, how many BERTs is enough? The number of BERTs is specified on `CKPTSPACE BERTNUM=`. The number can be increased or decreased via operator command. But the number you need is very dependent on an installation’s work stream. As stated, BERTs contain variable data that is stored with various control blocks. One example is accounting strings. When a job goes through input processing, the accounting string is associated with the JQE and stored in BERTs. They remain with the job until it completes execution. At that point, the data is removed and the BERTs freed. So the number of BERTs needed to store the accounting string depends on the length of the accounting string at your installation and the number of jobs that are typically queued for execution.

Another example is tracking duplicate job names. When multiple jobs with the same job name are submitted, JES2 tracks them from after conversion until they complete execution. This is done to reduce the overhead of ensuring that two jobs with the same name are not executing at the same time. The tracking is done regardless of whether duplicate jobs matter (DUPL_JOB= parameter). The overhead of this then depends on the number of job names that have duplicates awaiting execution.

The bottom line is that installation must monitor the usage of BERTs over time and ensure that there are an adequate number defined. The cost of over defining BERTs is low, so often customers do that. A general rule of thumb is one BERT per JQE plus one quarter the number of JOEs.

Job numbers

JES2 assigns job numbers from a pool that is limited using the `JOBDEF RANGE=` specification. This can be changed at any time with an operator command. There is no direct relationship between the job number pool and the JQE pool that represents jobs. The range is typically greater than the number of JQEs defined to the MAS, reducing how often the job numbers wrap. The job index (JIX) is used to determine what job numbers are available for a new job.

Jobs arriving from NJE are assigned their original job number if it is available. If the original job number is outside the current range but available, it will be assigned to the job if **`JOBDEF RASSIGN=YES`**. If the job number is not available, the new job will be assigned a number within the range. If **`RASSIGN=NO`**, all jobs must be assigned a job number within the current range.

The job unidentified is a character representation of the job number. In JES2, the job identifier also indicates the type of job, batch job, started task, or TSO logon. The exact format of the JES2 job ID depends on the job number range. If the upper limit of the range is less than 100,000, then job identifiers start with a three character prefix (JOB, STC, TSU) followed by the job number. For example, a batch job assigned the number 1234 would have a job ID of JOB01234. If the upper level of the range is 100,000 or greater, the job ID prefix is reduced to a single character (J, S, T). In this case, the job assigned job number 1234 would have a job ID of J0001234. Jobs arriving via NJE that have an original job number of 100,000 or greater and with **`RASSIGN=YES`** will keep their original job number and have a job ID that starts with a single character regardless of the range setting.

Spool space

Spool space is represented in the checkpoint by two main data areas (CTENTs). First the Direct access spool block (DAS) represents each spool volume (data set). These are defined in 32 volume increments using the **`SPOOLDEF SPOOLNUM=`** specification. This value can be increased by operator command but not decreased. The value limits the number of spool volumes that can be defined.

The track group map (TGM) keeps track of what spool space is available. Each track group has 2 bits that represent it, one tracks the availability of the track group and the other tracks whether there were any problems accessing the track group. The number of tracks in the track group map is controlled by **`SPOOLDEF TGSPACE=MAX=`** specification. Track group space is specified in increments of about 16,000. The maximum number of track groups can be increased by operator command but not decremented.

1.4 JES2 start types

When JES2 is started, the value specified in **`PARM=`** can control the start type and the options associated with the start. There are two specifiable types of starts:

- **`COLD`** starts clear the entire checkpoint and logically clears the spool. All data that was stored on spool is logically lost. There is another option **`FORMAT`** that can be specified with **`COLD`** or in place of it to request that all spool volumes are formatted. On a cold start, JES2 detects whether a spool volume is formatted for use by JES2. This is done by inspecting specific tracks on the volume to see if they are formatted. If you are cold starting and are not certain that all volumes are properly formatted, you can specify the **`FORMAT`** option. Formatting large volumes can take up to an hour depending on the environment. Because of this, the **`FORMAT`** option will only be used when needed. An alternative to the **`FORMAT`** option would be to use IEBDG to format the spool volumes. Though this is not necessarily faster, it can be done before the IPL.

- `WARM` starts read in the existing checkpoint information and the initializing member joins the MAS. Warm starts take various forms based on the environment when the member is started. These forms are:
  - **All member warm start**. This form of warm start occurs when this member is the first member to join a MAS (there are no other active members). The member being initialized is also not resuming after an ABEND of JES2 without an IPL (a hot start, described below). There are certain parameters that can only be changed on an all member warm start. In addition, there is a more complete validation of all jobs and SYSOUT data structures that only occur on an all member warm start.
    On an all member warm start, you can request that the spool track group be rebuilt by specifying `SPOOL=VALIDATE` with `WARM` on `PARM=`. This causes JES2 to read each jobs spool allocation data area and rebuild the track group map based on what tracks jobs are actually allocated on. This can be a time consuming process and is generally only done if there are serious problems with the spool data structures.
  - **Single member warm start**. This form of warm start occurs when a member is joining an existing MAS (there is at least one member active or hot startable in the MAS). The member was active previously and did not terminate cleanly (implying that there still might be work associated with the member in the job or output queue). The member being initialized is also not resuming after an ABEND of JES2 without an IPL (a hot start, described below).

    Single member warm starts require the member to examine the job and output queue to determine if there are any jobs that are associated with the restarting member. If so, processing is required to process the incomplete processing of this work. This process takes time and can delay the JES2 start. To avoid this, when a member fails and the image it is running on is IPLed, there is an option to perform this restart processing on an active member of the MAS. This can be done using the `$E MEMBER` command or automatically using the `MASDEF AUTOEMEM` combined with the `MASDEF RESTART` specification. `AUTOEMEM` cleans up failed members when JES2 detects via XCF (JESXCF) that the member has terminated.
  - **Quick start**. This is the same as a single member warm start except that the member is either new to the MAS (this is the first time it is starting) or it was in the MAS previously but came down cleanly (via a `$P JES2`), or the work was reset after an IPL using `$E MEMBER` or `AUTOEMEM` processing. As its name implies, this is the quickest start of JES2.
  - **Hot start**. A hot start is the restarting of a JES2 address space after it has ABENDed. The subsystem interface (SSI) is still active. JES2 restarts the address space, deals with any work that was active in the address space when JES2 ABENDed, and then reconnects to the existing SSI. The intent of this type of start is to be as minimally disruptive as possible.
  - **All member hot start**. This is referenced in $ACTIVATE processing as a case where a starting member can hot start and change the activation level with the start. In this type of start, there are no JES2 address spaces active anywhere in the MAS (they are either not active or the JES2 address space on the member has been ABENDed). There is certain processing that can be done on this type of start.

As you read the JES2 documentation, there are descriptions of the types of starts needed to make certain changes. As you can see, the warm start types are not options on the start command (or `PARM=`) but rather environments that must be created before starting JES2.

1.5 Activation levels

To avoid cold starts and also enable various functions, JES2 uses a function called **$ACTIVATE**. What this command does is alter what is referred to as the checkpoint level, checkpoint mode, or activation level (different terms for the same concept) to enable or disable functions. The process involves verifying that all members have the appropriate level of code, that certain functions are properly enabled or disabled, and that the checkpoints are large enough for the new checkpoint level. Performing an activation often involves reformatting data areas in the checkpoint or adding section (CTENTs) to the checkpoint. When a new level of code is activated, this generally prevents older level of code from joining the MAS. This ensures that before a function can be used, all members have the correct code to support that function. Also, the use of new functions can create data areas that prevent activation to a lower level.

Generally, there are two levels that any release of JES2 can run in. One is a compatibility level that supports the older releases. The other is the activated or full function level that enables the new function. After older releases fall off the coexistence support, compatibility mode is generally dropped and customers must activate full function mode before migrating. The design does support more than two levels of activation at a time, but this has never been needed.

Not all releases include a new activation level. Generally, they are only needed every few releases.

In some instances, full function mode might require changes to JES2 exits. This is because an exit might be accessing the data areas that are changing in format. All of the JES2 code supports dynamically changing modes using operator commands. However, your exits might not have been written to support this, or can only support moving in one direction.

In preparation to change your activation level, you will issue a **$D ACTIVATE** command. This displays the current checkpoint level and any requirements that have not been met to go to the other checkpoint level (assuming that there are only 2 allowed for this release). In some cases, an option might need to be turned on or off (depending on the direction of activation). In other cases, the checkpoint data set might need to be enlarged. There also might be a requirement to increase the limit on a parameter (for example the number of BERTs). The command also ensures that there are no errors in the checkpoint queues. All these must be set before issuing the **$ACTIVATE** command.

The most common way to alter the checkpoint level is to use the command:

```
$ACTIVATE,LEVEL=Z11
```

This will update the activate level to the Z11 level (the full function level for z/OS 1.11 to z/OS 2.1). To go back to compatibility mode, the command would be:

```
$ACTIVATE,LEVEL=Z2
```

This restores the activate level to the Z2 level (the compatibility level for z/OS 1.11 to z/OS 2.1).

On a cold start, JES2 by default starts in the full function mode for the current release. However, you can use the **OPTSDEF COLD_START_MODE=** initialization statement to cause JES2 to cold start in a particular mode. This parameter is only used on a cold start. The default is to start in full function mode of the current release. The valid values are the same as the **LEVEL=** keyword on the **$ACTIVATE** command.

Another way to affect the activation level is to use the **UNACT** parameter on the **PARM=** when starting JES2. This parameter allows you to cold start in compatibility mode or to switch to compatibility mode on an all member warm start or an all member hot start.

1.6 JES2 job processing

The life of a job starts when it is submitted to JES2 via one of its input sources. From there, it progresses through a series of phases that include the running of the job, processing of the output, and ultimately in the purging of the job. There are two primary ways jobs are processed by JES2:

- Pre-execution jobs that include JCL that describes how to run the job.
- Post-execution jobs that arrive via NJE and contain essentially SYSOUT data sets to be processed.

NJE also includes the concept of store and forward job streams that can be pre or post executions. These are jobs that are transitioning through a node on the way to their final destination.

1.6.1 Pre-execution job phases

Jobs submitted to JES2 though an input device (such as a reader or job receiver) go through the phases shown in Figure 1-4. Jobs (JQEs) move from phase to phase by moving from one work queue to another. Members of the MAS select work from a queue based on their ability to process the job, and for some phases, the affinity associated with the job.

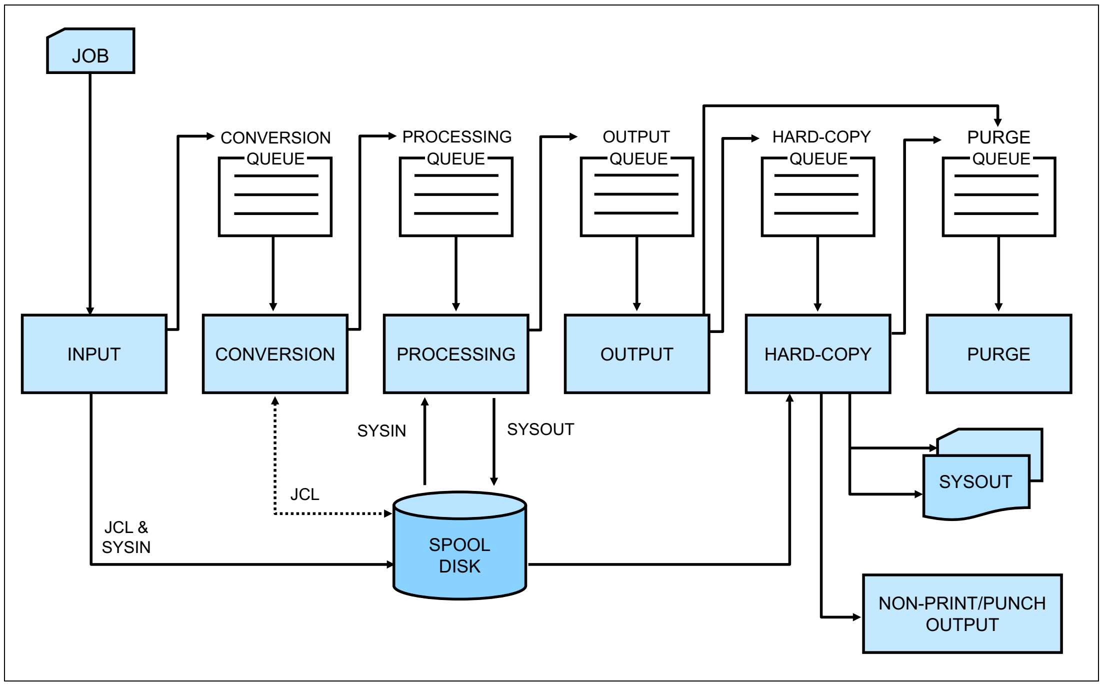


Figure 1-4 Standard JES2 job phases

Input phase processing

The first phase that a job goes through is the input phase. This phase performs the following tasks:

- Creates the control blocks needed for the job in the checkpoint (JQE) and spool (JCT and IOT)
- Parses the JOB card extracting fields needed by JES2 to process manage the job
- Parses, validates, and builds the data areas associated with the JES2 JECL cards
- Separated the instream data from the JCL and created the needed instream data sets on spool
- Passes the security information to the security product (via SAF VERIFYX call) to determine the authority that the job will run under

In addition, if this is an internal reader (INTRDR) then symbols processing occurs for symbols passed on the internal reader allocation and to extract and return symbols related to the job submission.

Where the actual processing for the input phase occurs depends on the type of input device. Internal readers and NJE over TCP/IP processing occurs in the address space that is writing the JCL. So a TSO user issuing a SUBMIT has the bulk of the input phase processing occur as the records are written to the internal reader. This spreads the overhead (and billing) for this work to the address spaces that submit the jobs. There is still code in the JES2 address space that creates the job in the job queue and then moves it to the next phase. But this is relatively little overhead compared to the rest of input processing.

Input processing for all other devices (RJE readers, SNA, BSC and offload job receivers, and so on) occurs in the JES2 main task under a separate PCE for each device.

Any JES2 main task processing for the input phrase occurs on the member where the job is being input.

If the job is being routed to another node due to an XMIT or a ROUTE XEQ card in the JCL, the job is then queued to the XMIT phase/queue for processing by NJE. Otherwise, it is queued to the conversion phase. There are a few errors that can cause the job to be queued directly to purge processing. Also, before z/OS 2.1, there were cases where the job could be queued to the output phase.

Conversion phase processing

The conversion phase is responsible for calling the z/OS converter to parse the various JCL statements and convert them to internal text. Also, with z/OS 2.1, the conversion phase can also invoke the z/OS interpreter to convert the internal text into SWA control blocks (and detect additional JCL errors). This phase performs the following tasks:

- Determines the PROCLIB concatenation to be used by the job
- Passes the job to the conversion subtask
- Under the conversion subtask, the following occurs:
  - JES2 opens the appropriate PROCLIB
  - Reads in any symbols passed via the internal reader
  - Calls the z/OS converter
  - Processes any instream data creation requests from the converter
  - Checks if the interpreter needs to be called and invokes it if needed
- A service class is assigned to the job based on WLM classification rules
- Duplicate jobname processing is performed
- The job is queued based on the return codes from the services called

Conversion processing can be done on any member of the MAS. There is one PCE for every conversion subtask. The conversion subtasks can be in the JES2 or JES2CI address space (depending on whether the interpreter is being called).

At the end of conversion processing, the job can be queued for the processing (execution) phase or the output phase if errors are detected.

Processing (execution) phase processing

The processing (or execution) phase is where the job runs. Jobs are queued to this phase by job class (one queue per class). In addition, if this is a batch job class that is managed by WLM, it is also added to a service class queue. Started tasks and TSO logons (also known as demand select jobs) are queued to the STC and TSU classes only.

The batch job classes have properties that can help an installation manage where and how many jobs execute in a given class. These control for both WLM and non-WLM managed job classes. There are limits on what members a class can run on, the number of jobs that can execute in the class anywhere in the MAS, and the number of jobs that can execute on each member of the MAS. In addition, if the job is in a WLM managed class, then there are controls to determine what members that WLM service class is active on.

If a batch job class is WLM managed, every job in that class is also queued to a service class queue in JES2. Each JES2 member will register the service class queue with WLM if the service class is active on that member. However, there is a restriction in WLM that only one address space per system can register to process a particular service class queue. The registration includes a MAS identifier, so the restriction is one service class queue manager per system in each MAS. So if there are 2 JES2s (JES2 and JESA) on the same system in the same MAS, only the JES2 will register with WLM and run WLM managed jobs. The system that registers the WLM service queue is referred to as the *boss system*.

Jobs in WLM managed classes are run in initiators that are started and stopped by WLM. These initiators are started and stopped based on the importance of work in the service class, the service class goals, the size of the queue of jobs waiting to execute and the capacity of the various systems in the MAS to absorb more initiators. The size of the queue of work can be influenced by the job class limits previously discussed. The size is also limited by the affinity associated with the jobs, and the systems where the job class is active.

JES2 is responsible for selecting the next job to execute. Even with WLM management, all WLM controls is the number of WLM managed initiators active on a member. JES2 selects the specific job to be placed in an initiator.

An alternative to WLM managed initiators is the traditional JES2 managed initiators. These must be defined in the JES2 initialization deck and are started and stopped by operator commands. Where the initiators are started and the number active is controlled by operator command.

There is also the capability to start batch jobs via operator command. This sends a request to WLM (even if the job is in a JES managed job class) to create an address space to run the job in. WLM will select a system (within the affinity limits for the job) that has capacity to run the job and then start an initiator to process that job. JES2 will then assign the job to that initiator.

After a job is assigned to an initiator and starts running, the JES2 address space does not get involved in running the job. It will process specific SSI requests that might be made by the address space (such as a SAPI or PSO SSI request) and provide spool space to the job (via the CSA space cache called the BLOB). The JES2 address space does not get involved when data sets are allocated or when SYSOUT is created. However, if a spin SYSOUT data set is created, a request is queued to the JES2 address space to build the JOE for the data set. This is an asynchronous request that the address space makes to the owning JES2 address space.

When a job completes running, JES2 examines how the job terminated and queues it to either:

- The output phase under normal conditions
- The spin phase if the job could still have unprocessed spin requests
- Back into execution if the job needs to be rerun

Spin phase processing

The spin phase exists to ensure that there are no outstanding spin SYSOUT data set requests for the address space. The spin phase is processed by the same set of PCEs that processes the spin SYSOUT data set requests. For each job in this phase, the spin work queue is examined. If there are no requests pending, the job is queued to the output phase. If any requests are found, the job remains in the spin phase until those requests are processed.

Output phase processing

This phase takes the non-spin SYSOUT data sets that were created during job execution and groups them into job output elements. Each instance of a data set has had a PDDB built for it. PDDBs with like characteristics are grouped into a JOE. The JOEs are then placed on the SYSOUT queues for processing by the various output processes (such as SAPI and printers).

It is during the output phase that job completion notification is issued. It is done at this point because all jobs (except for certain error scenarios) go through the output phase. This phase also reads the spool data areas, which are not read in the JES2 address space during the processing phase.

It is also during this phase that JES2 cleans up data areas that are no longer required for the job. Data stored in BERTs that is not longer required is deleted.

After a job completes the output phase, it will either be queued to the hardcopy phase (if there are any JOEs associated with the job) or the purge phase (if there are no JOEs for the job).

Hardcopy phase

Jobs that have completed the output phase and have any JOEs associated with them are placed on the hardcopy phase while their JOEs are processed. From the jobs standpoint, nothing happens to the job during this phase.

As JOEs are selected, processed, and purged, a check is made to see if there are any JOEs left for the job. When the last JOE for a job is purged, the job is moved to the purge queue by the process that eliminated the last JOE.

Purge phase

This is the last phase in the life of a job. Before a job is queued to this phase, all the JOEs have been removed and all that remains is the JQE and the spool space. The job's control blocks that represent the spool space for the job are read in and the spool space purged. The security product is called to audit that the job has been deleted. Finally, the JQE is removed from the job index (JIX) freeing the job number and then the JQE is placed on the free queue.

1.6.2 Post-execution job phases

Post-execution jobs arrive via NJE or offload SYSOUT receivers. These jobs are assigned a job number and JQE in much the same way a pre-execution job does. However, these jobs are placed on the receive queue instead of the input queue. The processing in the receive queue is done in the NETSERV address space for NJE over TCP/IP and in the JES2 address space for NJE over BSC or SNA. The primary difference is that there is no JCL being processed for these jobs as they come in. Instead, NJE job and data set headers are used to build the needed job structures and SYSOUT PDDBs. The SYSOUT data sets are written to spool by the receiver processing.

When a job completes the receive phase, it is queued to the output phase similar to jobs completing execution. The only difference is that if the SYSOUT data sets are spin data sets, the jobs is also processed by the spin PCEs to create the JOEs for the spin data sets. Though the spin PCEs are processing the jobs, they are never actually placed on the spin phase queue.

After the output phase, the job progresses through the hardcopy and purge phases as normal.

2

Terminology differences

This chapter provides a description of some of the terminology you are likely to encounter during the rest of this book. We focus in particular on cases where the same term is used in JES2 and JES3 but with different meanings.

2.1 JES3 terminology

In JES3 terminology, a processor is defined as a hardware unit that contains software to interpret and process instructions. Figure 2-1 shows a typical JES3 complex.

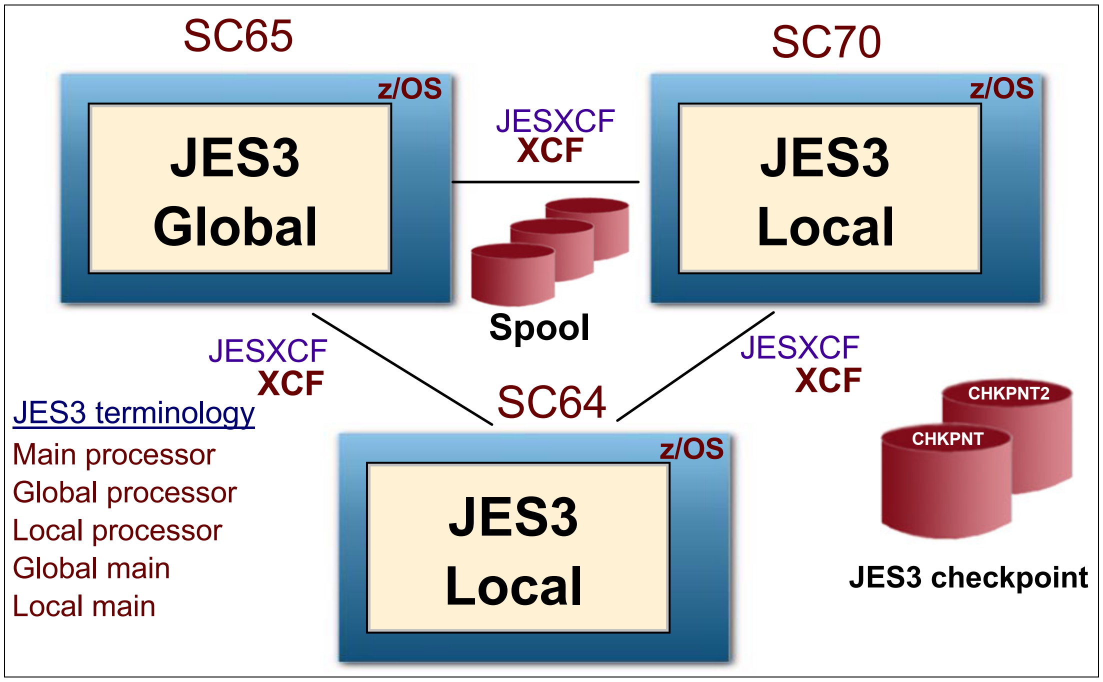


Figure 2-1 JES3 complex showing three LPARs

This complex is described as follows:

Global processor	The processor that controls job scheduling and device allocation for a complex of processors. See also local processor.
Local processor	In a complex of processors under control of JES3, a processor connected to the global main by JESXCF services, for which JES3 performs centralized job input, job scheduling, and job output services by the global main.
Main	A processor named by a JES3 MAINPROC initialization statement, on which jobs can execute; represents a single instance of MVS. The two types of mains are global main, and local main.
Global main	The global main controls job scheduling and device allocation for a complex of JES3 processors. Each local main in the complex exists under control of the JES3 global main and is connected to the global main by JESXCF services. The JES3 on the global main can perform centralized job input, job scheduling, and job output services. Only the global main performs scheduling functions, although scheduled work executes on the local mains.
Local main	In a complex of processors under control of JES3, a processor connected to the global main by JESXCF services, for which JES3 performs centralized job input, job scheduling, and job output services by the global main.

Note: The JES3 global is the system where JES3 executes and schedules work to the JES3 locals. The JES3 address on the locals is there for the possibility that the global might fail and a DSI process can be used to make any of the locals the new global. Of course, a local can take over the global function when it is decided to switch the global for any wanted reason.

2.2 Different use of terms

Because of the long and somewhat independent histories of JES2 and JES3, there are some situations where the use of terminology might cause some confusion when moving from one JES to the other. There are some cases where the same term is used to mean different things. And other cases where two different terms are used to describe what might be the same thing.

This section brings all these cases together into one place, and will help avoid confusion as you read the remainder of this book.

2.2.1 Non-specific JES2 and JES3 references

Often it is useful to refer to JES2 and JES3 in ways that are non-specific. There are a number of ways that this is done:

JES	This is a non-specific reference to a JES. It is used when speaking of concepts that apply to both JESs. For example, “Each z/OS image must have a JES subsystem to process jobs.”
JES-neutral	There are functions that perform the same in both JES2 and JES3. These functions are often referred to as *JES neutral*. For example, security processing is performed by the security product in a way that is JES-neutral.
JES-agnostic	Much like JES neutral, this is something that uses JES2 service in a way that does not care about the type of JES you are running. For example, “One of the goals in preparing for a migration to JES2 is to make your JCL JES-agnostic.”

2.2.2 Collections of JESes

Both JES3 and JES2 have the concept of a collection of JES address spaces that share a single work queue. However, the terms used to refer to the collection varies with the JES:

Complex	This is the term that JES3 uses to refer to its collection of a single JES3 Global and up to 31 Locals that share a single JES3 work queue.
MAS	Multi-Access SPOOL: This is the term that JES2 uses to refer to the collection of up to 32 JES2 address spaces that share a single JES2 work queue.
JESplex	JES complex: This is a JES-agnostic term coined to refer to either a JES2 MAS or a JES3 complex. You can also modify this with the type of JES such as a JES2 JESplex.

The term used to refer to one of the JES address spaces in the collection also differs by JES:

Member	JES2 address spaces in a MAS are referred to as members of the MAS. This is also the term used to refer to members of a JESplex. In JES2, the member name defaults to the SMF ID of that system. You can override that with a name of your choice, but the name is limited to 4 characters.
System	JES3 address spaces in a complex are often just referred to as systems. This terminology was developed because there is only one JES3 address space per z/OS image.
Main	Another name for a member of a JES3 complex. This can be a Local or a Global. In JES3, the name of a JES3 Global or Local can be up to 8 characters long.

2.2.3 JES startup processing

Operationally, there are two ways to start JES2. You can specify `PARM=COLD` or `PARM=WARM` (the default). A cold start will clear all JES2 SPOOL and checkpoint data areas (delete all jobs) and like a JES3 cold start, is rarely done. Warm starts are the normal way to start JES2. How JES2 processes a warm start depends on the environment JES2 discovers during initialization. Depending on the environment, a JES2 warm start will do different processing.

Regardless of the type of start, JES2 always reads its parameters from the members pointed to on the HASPPARM DD statements when it is starting. It then compares the information found in the parms with the information it gets from the checkpoint and from the other members of the MAS. If it encounters a parameter that requires a more disruptive type of start, it might issue an HASP442 message, informing you that the parameter was ignored.

The types of start that JES2 can perform are as follows:

Cold start	This start occurs when you specify `PARM=COLD` when JES2 is started. The JES2 spool will be cleared of all contents. This type of start requires that all the members of the MAS are stopped. On the system that is performing the cold start, you either must perform an IPL, or you can completely stop JES2 and then start it up again, specifying that it will perform a cold start. Given that all work on the system must be stopped before you do this, there is little difference between stopping and restarting JES2 to do the cold start, and IPLing that system.
Warm start (single system)	This start occurs when you specify `PARM=WARM` when JES2 is started and the starting JES2 member is joining a MAS with other active members. The JES2 checkpoint is read in and processed. Any work that might have been associated with this member from a previous instance will be reset (marked as no longer actively being processed).
Quick start (single system)	This start occurs when you specify `PARM=WARM` when JES2 is started. This is the same as a warm start except that no work is associated with this member from a previous instance. This occurs if the member: ► Was shut down cleanly via a `$P JES2` command, or,

- Is starting after an all member warm start or a cold start, or,
- Has had its work reset by a `$E MEMBER` command or via the `AUTOEMEM` process

Warm start (MAS-wide)	This start occurs when you specify `PARM=WARM` when JES2 is started and the starting member is the first (only) active member of the MAS. Like all other warm starts, the checkpoint is read in and processed. If any entry in the work queue indicates it is active, it is reset at this time. In addition, certain operating parameters can only be reset on this type of start.
Hot start	This start occurs when you specify `PARM=WARM` when JES2 is started and a previous instance of the JES2 address space had ABENDed and no intervening IPL has occurred. Like all other warm starts, the checkpoint is read in and processed. Work in the job queue that is associated with processes that were terminated when the JES2 address space was ABENDed are reset, but work associated with active address spaces (running jobs, internal readers, and so on) is not reset. That work continues normal processing.

JES3 uses similar terminology, but the impact can be different. For JES3, the start type is set in the response to message IAT3011:

Cold start	During a cold start, the initialization deck is read to determine the configuration. The SPOOL is initialized, and any jobs that were in the system are lost. A JES3 cold start requires that every z/OS in the JES3 complex is IPLed.
Warm start	A warm start also requires an MVS IPL before it is allowed. The configuration is determined from the initialization stream. Most job processing resumes after this less intrusive restart. Like a cold start, a JES3 warm start also requires that every z/OS in the JES3 complex is IPLed.
Hot start	During a hot start, the initialization stream is *not* read. Instead, the configuration information is read from control blocks stored in the spool. A JES3 hot start does not require that z/OS be IPLed. If a system is IPLed and then JES3 hot starts, job processing will resume. Job processing can continue *across* a JES3 hot start if the system is not IPLed.
Hot start with refresh	Hot start with refresh is similar to a hot start except that the initialization stream *is* re-read. This allows for initialization deck statements to be altered without requiring a warm or cold start and the associated complex-wide IPL.
Restart with analysis	Hot and warm JES3 restarts allow for an additional “analysis” option to be specified. When analysis is requested, additional verification is performed for jobs on the job queue and invalid jobs can be purged.
Warm start with replace	This start performs the same function as a warm start. In addition, it allows you to replace a spool data set.

2.2.4 JES parameter statements

Both JESs have parameter statements that define objects to JES and set operating parameters. These are referred to as:

Inish deck	JES3 initialization steam. This is read when JES3 first initializes on a system and on a hot start with refresh.
Init deck	JES2 initialization stream. This is read in on every start of a JES2 address space. The format of statements in the JES2 init deck is the same as the corresponding operator command and the display command for the statement.

2.2.5 SYSOUT processors

Both JESs support sending SYSOUT from SPOOL to physical or logical devices for processing. These are referred to as:

Writer	In JES3, a printer (JES-controlled or FSS-managed) or an application that uses the *SYSOUT API* (SAPI) is referred to as a writer. SYSOUT is commonly referred to as being placed on the writer queue.
Printer	In JES2, a printer (JES-controlled or FSS-managed) is referred to as a printer. Applications that use SAPI are referred to as SAPI applications or SAPI threads. SYSOUT is commonly referred to as being placed on the print queue or the ready queue.

2.2.6 Remote workstations

A remote work station that connects to JES is referred to as:

RJP	Remote Job Processing (RJP) in JES3
RJE	Remote Job Entry (RJE) in JES2

2.2.7 JES threads

Both JESs have a main task that is shared by multiple threads or processes. There are also externals that control the number of particular types of these threads in both JESs. The names for these threads are:

DSP	JES3 dynamic support program. A DSP represents code that performs a small piece of job processing. Most JES3 job processing is performed by IBM written DSPs. These units of work, in conjunction with FCT entries, provide the basis for JES3 subsystem multitasking.
FCT	JES3 function control table. The JES3 main task processing scans a priority-ordered chain of FCT entries, looking for any FCT entries that represent active work to-do. Each FCT entry points to a DSP that is called to perform the work. Because FCT entries reference DSPs, the two terms are sometimes used interchangeably. Multiple FCTs that reference the same DSP can be inserted into the DSP chain. Some FCT entries reside permanently in the FCT chain, and some are added and removed only as needed.

PCE	JES2 processor control element. This is a control block that represents a thread or process running under the JES2 main task. Each PCE performs a function (such as execution services), processes a job phase (such as the purge phase), or manages a device (such as a printer). Some PCEs are created at initialization based on keywords on `PCEDEF` or other internal constants. Others can be created via commands (such as `$ADD PRINTER`). Exit code or installation load modules can also define and create PCEs as part of their processing.

2.2.8 Multiple JES2 images

JES2 supports running multiple instances of JES2 running on a single z/OS. The additional instances can be in a separate MAS to the primary JES2, or they can be in the same MAS as the primary JES2. There are a number of reasons why you might want to have more than one JES2 instance. Some common examples are:

- To test new functions on a production system, but separate from the production MAS.
- To offload functions such as NJE or printing from the primary member to a secondary.
- To provide easy access to a secondary MAS on a production image.

There have been numerous terms used when describing this concept. They include:

Primary JES	This refers to the JES that is the primary subsystem on a particular z/OS image. There is only one primary JES. JES2 can run as either a primary or a secondary JES. JES3 can *only* run as a primary.
Secondary JES	This refers to a JES2 subsystem that is not the primary JES2 subsystem. There can be many secondary subsystems on a z/OS image. These are always JES2 subsystems since JES3 does not support running as a secondary subsystem.
PolyJES	This refers to the process of running multiple JES instances on a single z/OS.
Alternate JES	This is another name for a secondary JES2 subsystem. Alternate is often used when there are more than two JES instances on a single z/OS image. For example, “This system has three JES2 address spaces; one primary and two alternates.”

2.2.9 JES initialization statements

Because JES2 and JES3 provide fundamentally the same function (batch job handling), and they both depend on a set of initialization statements to define the configuration to them, it is not surprising that JES2 and JES3 will have initialization statements that are similar. In fact, in some cases they use identical keywords. Sometimes these keywords have the same meaning, and other times they have subtly different meanings. See 7.3, “Initialization statements” on page 101.

3

Differences between JES2 and JES3 are becoming smaller

This chapter provides information about functions and features that are included in JES3 that are not provided by JES2 (although the function might be available in other products). We also discuss that in recent years, surface functions have been converging between the two.

This chapter will be used as part of the discovery phase of a migration project to identify the functions that you must find a replacement for. It also provides information to help you determine whether you are using a given function or not.

3.1 JES3 functions

This chapter provides a list of functions or features provided in JES3 that do not have a direct equivalent in JES2. A brief explanation of "*Where this is enabled*" is included for most features to help you know how you would determine if you are using that feature in your installation. Some of the functions presented are part of the fundamental design of JES3 so it might not be so easy to determine if you are using them or not. These features might take some more investigation and time to convert.

For all of the functions, you need to determine the extent of its usage and whether you still need it or not. For example, some features like Main Device Scheduling are less relevant now (especially for installations that use tape virtualization) than they were 40 years ago. So, while you might still be *using* them, you might find that you do not actually *need* them.

Tip: As you read this chapter, we suggest that you start creating a checklist of the JES3 functions that you use. Later, when you get to the stage of deciding to migrate to JES2, that list will be one of the inputs. To help with that task, Table 3-1 on page 54 could be used as a template for such a checklist.

3.1.1 Dependent Job Control (DJC)

Dependent Job Control (DJC) was originally provided as a JES3 function for installations that required a basic batch job networking capability and found that the use of conditional JCL (using COND codes) was cumbersome. Over the years, most installations found that they required a more robust batch planning, control, and monitoring capability with less manual intervention so the use of batch scheduling products such as IBM Tivoli® Workload Scheduler is now prevalent.

Where this is enabled: //*NET Control statement card in JCL. A list of all DJC networks that are currently in use can be found using the `*I N` command. Be aware that DJC networks are constantly being created and deleted as jobs start and end, so you will also check the log for messages that start with `IAT73xx` (DJC messages) to build a list of DJC networks.

Any installation that still uses DJC will consider replacing it with a batch scheduler because:

- It has more function, including tailored interfaces for production personnel.
- Batch schedulers are far more powerful than DJC. They also provide job planning, tracking, and reporting functions that are not provided by DJC.
- At present, JES2 can only provide DJC function if you use a third-party product.

Replacing DJC with a batch scheduler is one of the "Positioning Moves" that are referred to in 6.1.1, "Positioning moves" on page 90.

3.1.2 Deadline scheduling

Deadline scheduling is a function in JES3 that can be used to provide a user with the ability to have a job submitted at a certain priority level at a certain time or day. It was also intended for jobs that needed to run at a designated time or period of time, for example weekly. Although its functions worked without a scheduling package or manual operator intervention, it is becoming obsolete because the functions it provides are better handled and controlled by a batch scheduling product such as IBM Tivoli Workload Scheduler and using some of its features for critical path processing and Event Triggered Tracking.

Where this is enabled: DEADLINE subparameter on the //*MAIN card in JCL. To display if deadline scheduling is currently being used for any jobs, enter the **`*I L`** command. If any jobs are queued, you can get a list of them using the **`*I A D=DEADLINE`** command. Be aware that jobs that use DEADLINE might start and end at any time, so just because the **`*I L`** command does not show that no jobs are using deadline at the moment does not mean that no jobs use it. Therefore, you will also check the log for messages that start with **`IAT74xx`** (DEADLINE messages).

As processing capacity increased over the years, users have come to expect that their jobs will run as soon as they submit it, so this function is not as critical. If the job has specific resource dependencies, or needs to run at a certain time for charge back reasons, that is generally controlled by using different job classes. Note that using DEADLINE scheduling does not guarantee the job would execute at the exact time you want. Some installations might find this function manually intensive to replace.

This is another one of those Positioning Moves referenced in 6.1.1, “Positioning moves” on page 90 that can be completed in advance of the cutover to JES2.

3.1.3 Priority aging

Jobs are selected to run based on jobclass and the available initiators in that class as well as based on a priority in that particular jobclass queue. Priority aging is used to help jobs that were submitted on a system with an insufficient number of initiators. Periodically, as defined by the relevant parameter, if the job was still on the job queue, the priority of the job would be increased. This would potentially give it a better chance of being selected by an initiator and was intended to ensure that low priority jobs would not languish in the job queue forever, while higher priority jobs were continually selected ahead of them. It is only used for JES3-managed initiators.

Where this is enabled: SAGER/SAGEL and MAGER/MAGEL keywords on the SELECT INIT statement in the JES3 init deck.

In JES2, there is a similar function for changing the priority of delayed jobs. You can use this by:

- Specifying a priority on the /*PRIORITY JECL statement for JES2-managed initiators.
- Using the PRTYHIGH=, PRTYLOW=, and PRTYRATE= keywords on the JOBDEF initialization statement.

For more information about this function, refer to the section titled “Job priority aging” in *z/OS JES2 Initialization and Tuning Guide*, SA22-7532.

However, many installations now use WLM-managed initiators. With WLM-managed initiators, WLM determines which job will be selected to run based on the job’s service class and the Performance Index of that service class. In that environment, the JES priority of the job is irrelevant once it is selected for processing. Before that, the JES priority can be changed, which might change the designated service class for that job.

Converting to WLM-managed initiators is a Positioning Move that can be completed before the move to JES2.

3.1.4 HSM early recall

JES3 recalls non-SMS-managed data sets that have been migrated by DFSMShsm as part of the job setup processing for SMS-managed data sets. One disadvantage is that if you recall during setup, by the time you execute the job, the data set might be migrated again.

JES2 only recalls migrated data sets at the time that they are referenced. This might have an impact on the elapsed time for the job if it has to wait for large data sets (particularly those resident on tape) to be recalled. But remember that this difference only applies to non-SMS-managed data sets. If most of your data is SMS-managed, there will be no change in recall behavior between JES2 and JES3.

z/OS 1.11 added the ability to control whether migrated data sets that are deleted in an IEFBR14 step will be recalled before they are deleted.

In z/OS V2R1, there is a parm in the ALLOCxx member of PARMLIB that allows HSM recalls in serial or parallel. The behavior is the same in JES2 and JES3:

```
BATCH_RCLMIGDS(SERIAL | PARALLEL)
```

3.1.5 Main Device Scheduling (MDS)

Main Device Scheduling is a feature of JES3 that verifies that all the resources (devices, volumes, and data sets) needed by a job are available before that job goes into execution. It can be disabled at a system level by using SETUP=NONE. It can still be overridden in jobs that specify //*MAIN SETUP= in their JCL.

Pre-execution setup (JOB setup)

This is the basic feature of JES3 for pre-allocation of all devices, including DASD and tapes. Job setup reserves all devices and mounts all volumes needed by a job before job execution. Job setup can be requested on a job-by-job basis by specifying SETUP=JOB on the //*MAIN statement or on the JES3 initialization statement STANDARDS. SETUP=JOB is the default setting for the STANDARDS statement. Also, the resources are only reserved from a JES3 setup perspective; no actual ENQs or RESERVEs are issued. More information about Main Device Scheduling is available in the chapter titled “Main Device Scheduling” in *ABCs of z/OS System Programming Volume 13*, SG24-7717.

Where this is enabled: If your STANDARDS statement includes SETUP=JOB or has no SETUP parm referenced (JOB is the default parm). It can also be specified at a job level by specifying SETUP=JOB on the //*MAIN JCL statement.

Job setup also does data set awareness. It prevents an initiator from being assigned if the data set is currently allocated OLD. See 3.2.1, “Data Set Name disposition conflict resolution” on page 47.

High water mark setup

This feature reduces the number of resources that will be reserved for a job by determining the maximum, or high water mark, number of devices that will be used by any step in the job. The data set awareness feature is of significant benefit in JES3, especially if you have limited the initiators to a group. It stops a job from consuming an initiator then waiting for data sets.

Where this is enabled: The STANDARDS statement SETUP=xHWS (most likely THWS for tape) for a default system setting and a **`HWSNAME,TYPE=`** entry in the inish deck for each type of device that can be controlled. Also, overrides to this can be specified via //*MAIN card SETUP= parm in JCL.

For example: A job with three steps that requests two tape drives in the first step, followed by three tape drives in the second step, and one tape drive in the last step, would reserve, or allocate a total of three tape drives for the job since the maximum number of tape drives used by the job would be three. Without this feature enabled, JES3 could attempt to allocate the total of seven devices for the job. This is probably not what was intended because JES3 would view those devices as not being available to other jobs. This could especially be an issue in the case of a long-running job, where it is only the last step that requires many drives.

Because JES2 does not provide an equivalent function to MDS, you will take actions before any migration to JES2 to eliminate its use in JES3. It is likely, if you *do* use MDS, that it is only used for tape. If you use tape virtualization, it is reasonable to assume that you have more virtual tape drives than you ever use at one time, so disabling MDS would probably have no visible impact on job throughput. Nevertheless, it is prudent to make this change before the migration, so that if it does cause a problem, you can re-enable MDS while you investigate ways to address the problem.

Removing the use of MDS is one of the Positioning Moves described in 6.1.1, “Positioning moves” on page 90.

3.1.6 JES3 device control and device fencing

The original default was for JES3 to control device allocation, including tape and DASD. Device fencing, also known as device pooling, was a feature that could be used to isolate or reserve a certain set of devices for a certain set of jobs or groups, for example, if you wanted a certain group of jobs to use DASD at a remote location only.

For DASD device allocation, it is now recommended to remove all devices from JES3 management by removing their definition from the JES3 inish deck. This feature was most commonly used for tape drive allocations. However, with the combination of SMS-managed tape and tape virtualization, this is now less of a concern and many customers no longer use JES3 to control their tape allocations.

Where this is enabled: The JES3 inish deck GROUP statement (part of generalized main scheduling or GMS), using the EXRESC subparameter or the DEVPOOL subparameter.

Removing tape and DASD from JES3 control would be a positioning move and is referenced in 6.1.1, “Positioning moves” on page 90.

3.1.7 Inish deck checker

The JES3 inish deck checker is used to validate the format of the JES3 initialization statements. This is more of an issue in JES3 than in JES2 because of the complexity of the JES3 initialization deck, especially if you use MDS. The IATUTIS program takes input from the IODF and uses that to validate the syntax of the inish deck. An example of this, including the JCL used to run it, can be found in 7.3.2, “Verifying the JES initialization deck” on page 104.

Where this is enabled: JCL using program IATUTIS.

In JES2, syntax checking on the initparm can be performed using a secondary JES2, a capability that does not exist in JES3. A second JES2 address would be started using the new parameters to verify the syntax. This is not necessarily a recommended solution.

Also, JES2 tends to be more forgiving of syntax errors in the JES2 initialization statements, providing the operator with an option to resolve any errors during initialization.

Moving from JES3 to JES2 requires a change in the process that you use to validate the JES initialization statements. However, this will only impact a few people.

3.1.8 JES3 Monitoring Facility

The JES3 Monitoring Facility (JMF) provides a number of reports that can be used by the system programmers or software support if there are any specific performance or tuning concerns in JES3. The reports provided are specific to JES3 and are not applicable to a JES2 environment and would not necessarily require an equivalent option.

Where this is enabled: Invoked by entering a form of the `*X JMF` command.

If you do find that you encounter performance issues within JES2 following a migration, JES2 has a similar function called the JES2 health monitor. While the concept is similar, the details will obviously be completely different for JES2 than they were for JES3. More information can be found in the appendix titled "The JES2 health monitor" in *z/OS JES2 Initialization and Tuning Guide*, SA22-7532.

3.1.9 Disk reader

The disk reader function provides the ability to submit JCL from a PDS to the internal reader using a JES3 command. There are many parameters available to control the set of jobs submitted.

Where this is enabled: By placing a //JES3DRDS DD card in the JES3 PROC or by specifying DYNALLOC,DDN=JES3DRDS,DSN=dsn in the JES3 inish deck. It is invoked by using the `*X DR M=` command, so a search in SYSLOG might be required to determine if this facility is being used.

In JES2, you could provide the equivalent capability by using IEBGENER to copy JCL from a data set or PDS member to the internal reader or have it implemented in a batch scheduling product.

3.2 JES3 features

Following are fundamental features or characteristics of JES3 that cannot be enabled or disabled, but that are different from how JES2 handles the same situation.

3.2.1 Data Set Name disposition conflict resolution

JES3 performs Data Set Name (DSN) conflict resolution before execution of a job. If a job is submitted and will request access to a data set that is inconsistent with another job that is already using that data set, the newly submitted job will not be selected for execution until the data set is freed by the currently executing job. Instead, the job is placed in the JES3 allocation queue. For example, if the new job requests exclusive access to a data set (DISP=OLD or DISP=MOD), and that data set is already in use by another job, the new job will not start executing.

Operators can display the JES3 queues by entering an **`*I S`** command. If there are jobs in the allocation queue, you can determine why they are there by entering a form of the **`*I S A J=nnnn`** command.

During job execution, if a job requests dynamic allocation of a data set that is already in use, the behavior of the JESs is the same and you will get messages similar to those shown in Example 3-1.

Example 3-1 Message when there is a DSN conflict

```
IEF861I FOLLOWING RESERVED DATA SET NAMES UNAVAILABLE TO jobname
IEF863I DSN=test.dsname jobname RC = 04
IEF099I Job jobname waiting for datasets
```

In JES2, the data set needs of a job are not considered when JES2 decides if a job is selected for execution or not. As a result, if you migrate to JES2, you will expect to see the "waiting for data set" message more often. Some tuning of your batch schedules might help reduce the incidence of jobs contending over data sets. After you migrate to JES2, the IBM RMF™ Enqueue Delay Report might help you identify data sets that are experiencing contention. You might also find that you need to increase the number of initiators in JES2 to allow for the fact that some initiators might be occupied by jobs that are waiting for another job to release some required resource. Additionally, any automation that is triggered on this message would need to be reviewed and possibly changed.

You also have the option to issue a SYSDSN ENQ downgrade[1]. This reduces control from exclusive to shared, allowing access by other jobs.

3.2.2 Job class groups

A job class group is a named set of resource assignment rules to be applied to a group of job classes. System programmers define job class groups on JES3 initialization statements. They establish a link between a job class group and a job class by specifying a job class group name when they define the job classes. The job class group definitions in the initialization deck provide information about the resources that can be used by the set of jobs that are currently running.

Job class groups act differently on JES2, so conversion is needed.

[1] `http://pic.dhe.ibm.com/infocenter/zos/v2r1/index.jsp?topic=%2Fcom.ibm.zos.v2r1.e0za100%2Fmvssysdsn.htm`

3.2.3 Single point of control

The design of JES3 is that there is a primary, or single, point of control in a multi-system environment, which is known as the JES3 Global system. All other systems are designated as JES3 Locals. The behavior and processing of the JES3 Locals is controlled by the Global.

In a JES2 configuration, each system operates independently. Manipulation of spool files can be performed by any system in the sysplex. However, functions such as selecting a job for execution are performed independently by each system.

Both the operators and the system programmers require some time to get familiar with this different behavior. To ease the migration, you might decide to connect all JES2-managed peripheral devices (printers) to one system in the JESplex, so that all printer control can still be performed from just one system.

3.2.4 Printer naming conventions usage

In JES3, there are no restrictions on printer naming conventions. In JES2, however, there are conventions that must be followed related to the names that you can assign to your printers. As a result, it possible that the printer names you use in JES3, while being more meaningful to a human, will not be acceptable to JES2.

You might be able to circumvent this issue by using JES2 destination IDs that match your old printer names. This issue probably has more impact on the operators because they would need to become familiar with the new printer names.

3.3 On-the-surface convergence

As years pass, one can notice a convergence between JES2 and JES3. However, internal structures are still quite different. Hence the writing of this book to help plan a migration.

Figure 3-1 on page 49 shows how things have been evolving in terms of functions available in JES3 and JES2.

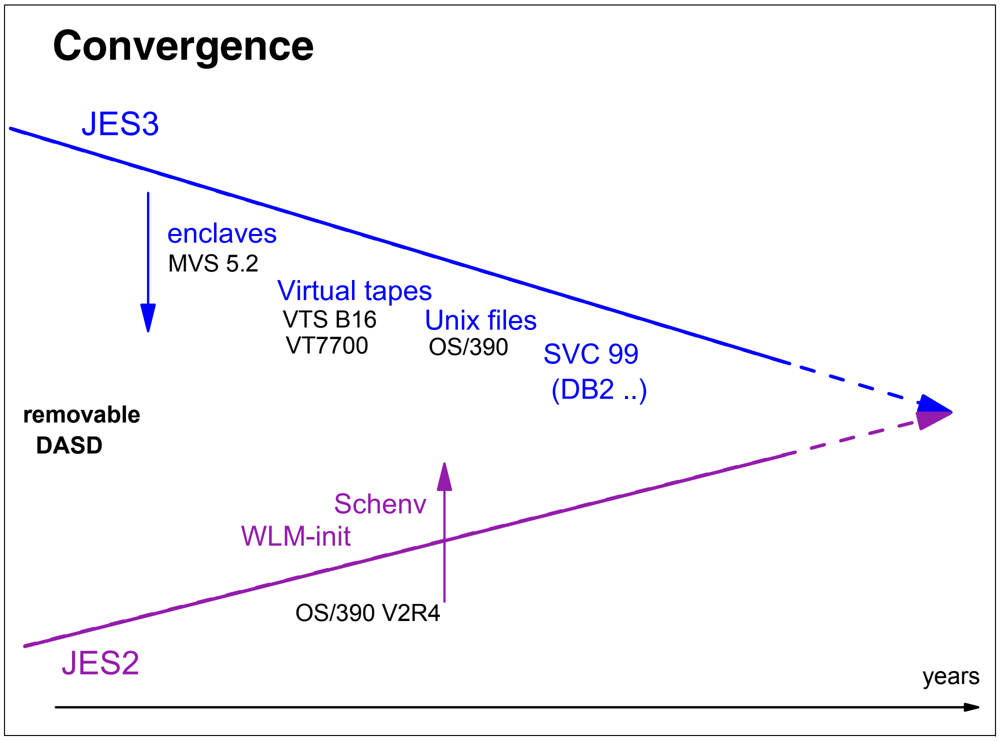


Figure 3-1 JES2 - JES3 surface convergence

Here is some history behind the diagram:

- At very the beginning, JES3 was mainly accomplishing its functions based on removable disks and tapes volumes. Nowadays, removable media belong to the past century. Tapes are more virtual; in consequence, there are no more operators to ask for a tape volume to be dug out of the basement. For a discussion of the TS7700 Virtualization Engine support that is available today, refer to *IBM Virtualization Engine TS7700 R3.0*, SG24-8122. In general, the support for the TS7700 Virtualization Engine is simplified with JES2 since the INISH deck setup with JES3 (the special library and device-related esoterics) is not used with JES2. Instead, the allocation requests are driven entirely through the SMS ACS routines.
- With MVS 5.2, new dispatchable units (enclaves) have been introduced which are not handled by any job entry subsystem but still have priorities assigned and so on, by WLM.
- Dynamic allocation (SVC 99) is the highly preferred way of allocating resources for subsystems such as IBM DB2®. No more DD cards to handle by any job entry subsystem.
- In IBM OS/390® V2R4, WLM was assigned the direct management of initiators in a way close to JES3 principles. JES2 took the benefit to fill a little bit the gap between the two converging job entry subsystems.
- UNIX System Services successful integration into MVS opened up a large avenue of applications for which the natural access to data is the UNIX hierarchical file structure, for which no JES is involved.
- Recent availability of hybrid platforms embodied in zBX boxes offers a new workload point of view provided by Unified Resource Manager. This is outside the domain of a function like JES.

As an example of this on-the-surface convergence, in the next section we describe JES2 support for virtual tape server VT7700. It was made available in z/OS V1R13, and in a slightly different manner for JES3 in z/OS V2R1.

3.3.1 JES3 role in today's zWorld

We can recap the JES3 role as follows:

- Originally in charge of keeping track of resource availability, JES3 knows about how to assign resources to job batch jobs, based on an image of IOCDS contents that a tool can automatically provide.
- Based mostly on MVS catalog, JES3 can know the availability and the shareability of MVS data sets.
- SMS-managed data sets availability can be indirectly known of JES3 through an interaction with DFSMS.
- USS files are out of its range and their allocation cannot be followed, among other things, because USS files cannot be allocated as MVS data sets have been.
- Starting with z/OS V2R1, JES3 can now interact with a VT7700, providing more adequate tape allocation.
- Following its traditional load management, JES3 can exploit JES and WLM initiators in order to exploit the hardware resources according to the SLA.

Batch parallel recall

To facilitate interaction between JES and SMS in z/OS V2R1, DFSMShsm can help determine whether data sets to be allocated, have been migrated.

For DFSMShsm-migrated data sets, allocation can now be planned to:

- Issue recall requests during step initiation
- Or issue recalls for all data sets in the job and wait for the recalls to complete

There is a new ALLOCxx keyword to enable and SETALLOC support.

3.3.2 Job correlator

With JES2 in z/OS V2R1, a correlator can be specified in the JOB card. It opens a unique 64-byte token, the correlator, for each job in a sysplex. This job correlator:

- Provides a larger name space for jobs (in addition to classical jobname).
- Helps relating jobs to output and other records.
- Provides a simple way for applications to determine the Job ID of a job that was just submitted.
- Is available with the z/OSMF REST API.

In JES3 environments, this UJOBCORR parameter is accepted but ignored.

The job correlator (UJOBCORR parameter on the JOB card) is a 64-byte token that uniquely identifies a job to JES. The JOBCORR value is composed of a 32-byte system portion, which ensures a unique value, and a 32-byte user portion, which helps identify the job to the system. The UJOBCORR parameter specifies this 32-byte user portion of the job correlator. The UJOBCORR value can be overridden when the job is submitted by using the appropriate JES2 exits.

The job correlator is used to identify the job in multiple interfaces, including:

- JES operator commands
- ENF messaging

- Subsystem interfaces such as extended status and SAPI
- SMF records

In the following example, the user portion of the job correlator is set to JMAN_COMPILE:

```
//TEST JOB 333,STEVE,UJOBCORR='JMAN_COMPILE'
```

Subsequently, this value will be combined with the system portion of the correlator to form a job correlator similar to the following example:

```
J0000025NODE1...C910E4EC.......:JMAN_COMPILE
```

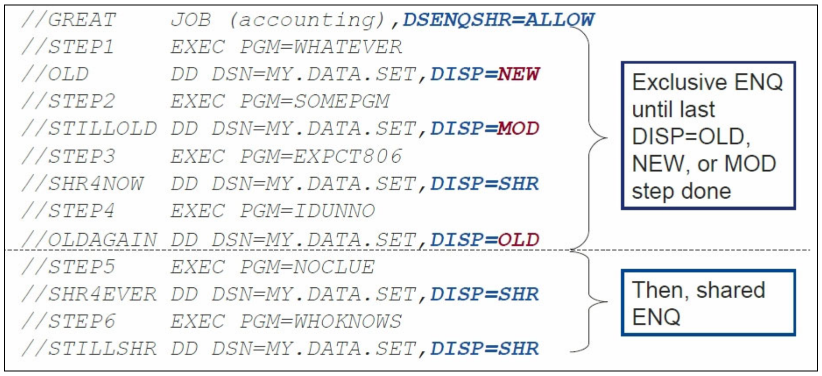
|<-system portion----------------------->||<-user portion--------------->

3.3.3 MVS resource serialization: JCL examples

Exploiting these enhancements in MVS resource serialization, not only has the ENQ interface been enhanced, but JCL support has been provided. New functions like these generally appear first in JES2 and then later on in JES3.

As depicted in Figure 3-2, starting with z/OS V2R1, a multiple step job can change an exclusive ENQ to shared ENQ for a given data set after the last job step with DISP=OLD, MOD, or NEW has ended. A new JES2 job class parameter can be specified: `DSENQSHR=AUTO ¦ ALLOW ¦ DISALLOW`

There is a new JOB statement parameter, DSNENQSHR=ALLOW to use with ALLOW.

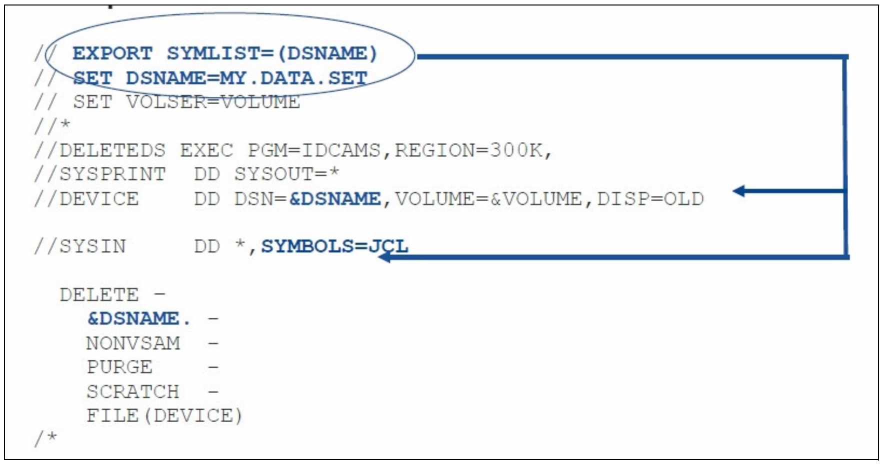


```
//GREAT     JOB  (accounting),DSENQSHR=ALLOW
//STEP1     EXEC PGM=WHATEVER
//OLD       DD DSN=MY.DATA.SET,DISP=NEW
//STEP2     EXEC PGM=SOMEPGM
//STILLOLD  DD DSN=MY.DATA.SET,DISP=MOD
//STEP3     EXEC PGM=EXPCT806
//SHR4NOW   DD DSN=MY.DATA.SET,DISP=SHR
//STEP4     EXEC PGM=IDUNNO
//OLDAGAIN  DD DSN=MY.DATA.SET,DISP=OLD
//STEP5     EXEC PGM=NOCLUE
//SHR4EVER  DD DSN=MY.DATA.SET,DISP=SHR
//STEP6     EXEC PGM=WHOKNOWS
//STILLSHR  DD DSN=MY.DATA.SET,DISP=SHR
```

Figure 3-2 Dynamic ENQ downgrade

3.3.4 JES2 symbols for instream data

z/OS V2R1 introduces new support for JES2 symbols within instream data. As depicted in Figure 3-3 on page 52, this new capability is externally provided by a new step-level EXPORT statement to list system and JCL symbols available to be resolved. It has a new symbols keyword for DD * and DD DATA to control substitution.



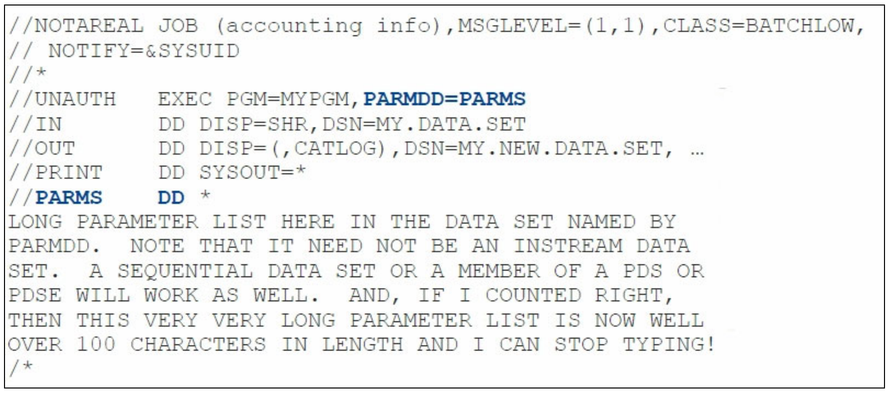
```
// EXPORT SYMLIST=(DSNAME)
// SET DSNAME=MY.DATA.SET
// SET VOLSER=VOLUME
//*
//DELETEDS EXEC PGM=IDCAMS,REGION=300K,
//SYSPRINT  DD SYSOUT=*
//DEVICE    DD DSN=&DSNAME,VOLUME=&VOLUME,DISP=OLD

//SYSIN     DD *,SYMBOLS=JCL

  DELETE -
    &DSNAME. -
    NONVSAM  -
    PURGE    -
    SCRATCH  -
    FILE(DEVICE)
/*
```

Figure 3-3   JES2 symbols for instream data

3.3.5 New PARMDD EXEC keyword

z/OS V2R1 introduces a new PARMDD EXEC keyword in order to support longer parameter strings. Shown in Figure 3-4, this new capability has these characteristics:

- Mutually exclusive with PARM keyword.
- No other changes required for unauthorized programs.
- Authorized programs must be link-edited using LONGPARM or the system will terminate the job at step initiation.
- Supports parameter lists 1 - 32760 bytes long.



```
//NOTAREAL JOB (accounting info),MSGLEVEL=(1,1),CLASS=BATCHLOW,
// NOTIFY=&SYSUID
//*
//UNAUTH   EXEC PGM=MYPGM,PARMDD=PARMS
//IN       DD DISP=SHR,DSN=MY.DATA.SET
//OUT      DD DISP=(,CATLOG),DSN=MY.NEW.DATA.SET, …
//PRINT    DD SYSOUT=*
//PARMS    DD *
LONG PARAMETER LIST HERE IN THE DATA SET NAMED BY
PARMDD.  NOTE THAT IT NEED NOT BE AN INSTREAM DATA
SET.  A SEQUENTIAL DATA SET OR A MEMBER OF A PDS OR
PDSE WILL WORK AS WELL.  AND, IF I COUNTED RIGHT,
THEN THIS VERY VERY LONG PARAMETER LIST IS NOW WELL
OVER 100 CHARACTERS IN LENGTH AND I CAN STOP TYPING!
/*
```

Figure 3-4   New PARMDD EXEC keyword

3.3.6 JES new functions in z/OS V2R1

Both JES2 and JES3 are enhanced in z/OS V2R1 in a similar manner (at least externally).

Expansion to 8-character job class name

With z/OS V2R1, both JES2 and JES3 add support for 8-character alphanumeric job class names on the JOB JCL statement.

JES3 supports 8-character job classes via JECL:

```
//*MAIN CLASS=xxxxxxxx
```

JES3 continues to override CLASS from JOB statement when CLASS is coded on the //*MAIN statement.

Note: A migration task would be to convert to having all CLASS= statements on the JOB card.

JES2 also supports creating up to 512 job classes. JES3 continues to support a maximum of 255 job classes.

In-stream data support

With z/OS V2R1, JES3 adds support for in-stream data sets in PROC and INCLUDE statements. This support is similar to what was introduced in z/OS V1R13 for JES2.

New keyword on the JOB JCL statement

With z/OS V2R1, JES2 and JES3, both add for new SYSTEM and SYSAFF keywords on the JOB JCL statement for directing jobs to specific JES3 main systems.

For the JES3 environment, the following parameters must be consistent with the SYSTEM or SYSAFF parameter, or JES3 will terminate the job:

- For the CLASS parameter on the JOB or //*MAIN statement, the requested processor must be assigned to execute jobs in the specified class.
- All devices specified on DD statement UNIT parameters must be available to the requested processor.
- The TYPE parameter on the //*MAIN statement must specify the system running on the requested processor.
- Dynamic support programs requested on //*PROCESS statements must be able to be executed on the requested processor.
- If any DD statement UNIT parameter in the job specifies a device-number, either a SYSTEM or SYSAFF parameter must be coded or the JES3 //*MAIN statement must contain a SYSTEM parameter.

MVS SSI 80 64-bit storage requests

With z/OS V2R1, JES2 and JES3 both support requests for 64-bit storage. MVS JES-neutral SSI request 80 (IAZSSST) includes new 64-bit fields.

Access controls on job classes

With z/OS V2R1, JES2 and JES3 both add support for SAF control over job class usage, using new profiles in the JESJOBS class. Many users had an exit for this function, which is no longer needed.

3.4 Checklist

This section contains a simple sample checklist that you can use to keep track of which JES3 functions you are using. You will probably extend this list with functions that you provide in any JES3 user exits that you exploit.

Table 3-1 JES3 unique functions checklist

Function	**Being exploited?**
Dependent Job Control	
Deadline Scheduling	
Priority aging	
Early HSM recall	
JES3 Device Control and Device Fencing	
Main Device Scheduling	
Inish Deck Checker	
JES3 Monitoring Facility	
Disk Reader	
Data Set Name Disposition Conflict Resolution	
Job class groups	
Single Point of Control	
Printer naming rules	

4

JECL and JCL differences

JES2 and JES3 have evolved over time by introducing new functions that address the needs of their specific customer sets. As a result, there are job entry control language (JECL) and job control language (JCL) statements that are unique to JES2 or JES3. There are also some JCL statements that are common to both, but that are processed slightly differently by the two JESs. This chapter lists those differences and recommends ways to migrate JES3-specific JCL streams to equivalents that will work with JES2.

After you read this chapter, you will see that this is a great time to avoid converting some of your old JCL or JECL statements over to their JES2 equivalent. Instead:

- Eliminate statements that are no longer required.
- Put controls in place to ensure that any NEW jobs do whatever they have to do in a JES-neutral way.

4.1 JCL processing

JCL is defined and managed by z/OS and is supposed to be JES-neutral. Most JCL is not affected by the JES that is running on a system. However, because some functions were or were not implemented by the individual JES (or were implemented differently by each JES), it is possible that some of your JCL that works with JES3 will not work with JES2 or that will not behave in the same way. Table 4-1 lists and briefly describes JCL constructs that are impacted by whether the JCL is processed by JES2 or JES3.

Table 4-1 Differences in JES3 and JES2 handling of JCL statements

JCL statement and parameter	JES3 system	JES2 system
`// command`	No difference.	No difference.
`// COMMAND`	No difference.	No difference.
`//*comment`	No difference.	No difference.
`// CNTL`	No difference.	No difference.

`// DD DDNAME`	Duplicate ddnames in a job step:
	Duplicate ddnames in the same step for JES3 or jointly managed devices are not allowed causing job to fail.
	If only one or neither DD statement requests a JES3- or jointly-managed device, the processing is the same as JES2.
	Reserved ddnames:
	`JCBIN JCBLOCK JCBTAB JESJCLIN`
	`JESI`*`nnnn`* `JESJCL JESMSGLG JOURNAL`
	`JESYSMSG JST JS3CATLG J3JBINFO`
	`J3SCINFO J3STINFO STCINRDR TSOINRDR`
		Duplicate ddnames in a job step:
		The system performs device and space allocation and disposition processing for both DD statements, however, it directs all references to the first DD statement in the job step.
		Reserved ddnames:
		`JESJCLIN JESJCL JESMSGLG JESYSMSG`

`// DD *`
`// DD DATA`
	In general, instream data set LRECL limited to the JES3 buffer size minus 46. For DD *, the maximum record length is 80.
	NJE transmission of instream data is truncated to a record length of 80.
	For SNA RJP the allowable parameters for input devices are BLKSIZE and LRECL.
	When DCB=MODE=C is specified with // DD DATA, it is the only parameter that can be specified.
		Instream data sets LRECL is limited to 32756.
		Record format (RECFM) is determined based on the length of the individual records.
		NJE transmission to/through non-JES2 nodes is limited to a record length of 254.
		DCB=MODE=C is ignored.

`// DD COPIES=`	nnn: 0-254 Group value nnn: 1-254 COPIES=0 is processed the same as COPIES=1.	nnn: 1-255 Group value nnn: 1-255 COPIES=0 is ignored (processed as if COPIES was not specified)
`// DD DCB=`	DCB=MODE=C is supported for SYSIN data sets.	DCB=MODE=C is not supported for SYSIN data sets.

JCL statement and parameter	JES3 system	JES2 system
`// DD DEST=`	Supports name (an individual local or remote device or workstation) and group-name (a group of local or remote devices or workstations), as well as node name, node name and user ID, or default ANYLOCAL.	Supports name (an individual local or remote device or workstation) as well as node name, node name and user ID, or default ANYLOCAL or LOCAL. JES2 also supports DESTID to identify destinations. For more information about destinations and DESTIDs, see "SYSOUT destination types" on page 176.
`// DD DISP=`	If DISP=MOD is coded for a multivolume data set and any of the volumes are JES3-managed, the job is not executed until all volumes are allocated.	JES2 does not manage volumes or data sets.
`// DD DLM=`	On either a DD * or DATA statement, only the assigned delimiter ends the input data set. The delimiter is only treated as a delimiter. In particular, if the delimiter is //, any data on the delimiter is ignored.	On DD * statement, either the assigned delimiter or // ends the input data set. The // is processed as a JCL statement. On a DD DATA statement, only the assigned delimiter ends the input data set. Data on the delimiter (even a // delimiter) is ignored.
`// DD HOLD=`	JES3 treats HOLD=YES as operator hold (that is, the data set is to be printed but is held until released by the operator). If the SYSOUT data set is destined for another NJE node, JES3 holds the data set at the execution node until released; it is then printed (or whatever) at the destination node. Refer to 4.1.1, "SYSOUT HOLD processing" on page 61 for translation considerations. JES3 ignores HOLD=YES when: 1. DEST=(node,userid) is coded 2. The SYSOUT data set is placed on the JES3 HOLD queue "(SYSOUT=(,writer-name)"	JES2 treats HOLD=YES as non-selectable (that is, the data set can be viewed by a TSO user but it is not to be printed). If the SYSOUT data set is destined for another NJE node, JES2 holds the data set at the destination node for TSO.
`// DD MODIFY=`	When trc is not specified, JES3 uses the first character arrangement table named in the CHARS parameter. When the trc value is greater than the number of table names in the CHARS parameter, JES3 uses the last character arrangement table named in the CHARS parameter.	When trc is not specified or the trc value is greater than the number of table names in the CHARS parameter, JES2 uses the first character arrangement table named in the CHARS parameter.
`// DD OUTLIM=`	JES3 ignores OUTLIM when specified on a SYSABEND or SYSUDUMP DD statement.	OUTLIM applies to all SYSOUT DDs.
`// DD SEGMENT=`	Not supported.	Supported.
`// DD SYMBOLS=`	Not supported.	Supported.
`// DD SYMLIST=`	Not supported	Supported.

JCL statement and parameter	JES3 system	JES2 system
`// DD SYSOUT=`	SYSOUT in JES3 has a TYPE= on the SYSOUT definition in the INISH deck, defining how output in this class is to be processed. When a SYSOUT data set is destined for another NJE node, and the SYSOUT class is a held class, the SYSOUT is held for TSO at either the execution or destination node, depending on the source of the destination specification. Refer to "// OUTPUT DEST=" on page 61 for more information. NJERDR is a reserved writer name in JES3.	In JES2, there is a connection between the SYSOUT class and the type (print or punch) of the data set. JES2 does not have a "hold at execution node" function. JES2 optionally uses the third positional parameter (form name) as a reference to a JES2 /*OUTPUT JECL statement.
`// DD UCS=`	JES3 does not support the FOLD subparameter.	JES2 does not support the FOLD subparameter, but does support the fold function via the UCS for JES2-managed printers. JES2 uses UCS as CHARS if CHARS is not specified.
`// DD UNIT=`	If DD UNIT parameter specifies a device-number for a JES3-managed or jointly JES3/MVS managed device, the JES3 //*MAIN statement must contain a SYSTEM parameter.	JES2 does not manage devices or data sets.
`// DD VOLUME=`	If you want JES3 to manage the allocation and a volume serial is not coded, PRIVATE must be coded. Otherwise, MVS manages the allocation. JES3 ignores IBM RETAIN®.	JES2 does not manage volumes or data sets.
`//SYSCHK DD ...`	If the checkpoint data set volume is to be mounted on a JES3-managed device, do not code the DEFER subparameter of the UNIT parameter on the SYSCHK DD statement.	JES2 does not manage volumes or data sets.
`/*`	Ignored and treated as a comment.	JECL statement prefix. If column 3 is non-blank, it must be the start of a valid JES2 JECL statement. If column 3 is blank, treated as a comment.
`// ENDCNTL`	No difference.	No difference.
`// EXEC COND=`	JES3 processes all jobs as though all steps will execute; therefore, JES3 "allocates[a]" devices for steps that are bypassed as part of the SETUP process. It is an issue when converting to JES3. Utilities such as IDCAMS allocate data sets with control cards that JES3 does not know about. When these data sets are referenced later in the JCL with DISP=OLD, JES3 fails the job with "data set not found."	JES2 does not do any setup processing.
`// EXEC PGM=`	PGM=JCLTEST and PGM=JSTTEST cause JES3 to a job looking for JCL or resource usage errors. These are only supported in JES3. For more information about translating this for JES2, refer to 4.1.2, "Scanning jobs for JCL errors" on page 62.	The JES2 TYPRUN=SCAN function is analogous to PGM=JCLTEST and JSTTEST.

JCL statement and parameter	JES3 system	JES2 system
`//IF/THEN /ELSE/ENDIF`	Supported with the same concerns that exist with data set allocations for conditional and mutually exclusive steps. JES3 SETUP processing assumes all steps will run and thus can flag SETUP errors that are not really a problem.	Supported.
`// INCLUDE`	Supported.	Supported.
`// JCLLIB`	Supported.	Supported.
`// JCLLIB PROCLIB=`	Not supported.	Supported.

`// JOB Accounting information`	First positional parameter supported.	First positional parameter supported. JES2 has a default parser that supports the following accounting information:
		`PANO ROOM TIME LINES`
		`CARDS FORMS COPIES LOG`
		`LINECT`

`// JOB BYTES=`
`// JOB LINES=`
`// JOB PAGES=`
	Supported.	Supported.

`// JOB CARDS=`	Maximum value supported is 6,500,000.	Maximum value supported is 99,999,999.
`// JOB CLASS=`	Default depends on STANDARDS initialization statement.	Default is based on the source of the job (card reader, work station, or TSO user that submitted the job).
`// JOB COND=`	JES3 processes all jobs as though all steps will execute; therefore, JES3 allocates devices for steps that are bypassed as part of SETUP processing.	JES2 does not allocate devices for steps that are bypassed.
`// JOB DSENQSHR=`	Not supported.	Supported.
`// JOB NOTIFY=`	NOTIFY= on the JOB statement sends the message to the system where it ran or to BROADCAST. The ACMAIN parameter on the //*MAIN statement allows the notify message to be directed to the named processor. Does not support an NJE node qualifier.	Notification is always directed to the member where the job was submitted by a TSO user or a started task. Member used for notification is propagated when a batch job submits another batch job. No external to control where notification is sent. Notify can be NJE node qualified.
`// JOB PRTY=`	nn: 0-14	nn: 0-15
`// JOB TYPRUN=`	Does not support TYPRUN=JCLHOLD or TYPRUN=COPY. Specifying //*MAIN HOLD=YES has the same effect as TYPRUN=HOLD. If // EXEC PGM=JCLTEST or JSTTEST is used, the result is the same as specifying TYPRUN=SCAN on the JOB statement.	Supports TYPRUN=JCLHOLD and TYPRUN=COPY.

JCL statement and parameter	JES3 system	JES2 system
`//`	The NULL statement ends a job. It isolates that job from any JCL errors in a subsequent job.	JES2 ignores a NULL statement when it is included in a job's JCL. JES2 processes JES2 control statements following a NULL statement as part of the job, until the next JOB statement or EOF is encountered.

`// OUTPUT`	In JES3, the // OUTPUT statement interacts with the //*FORMAT statement, possibly producing two distinct instances of a SYSOUT data set with attributes derived from the // OUTPUT or //*FORMAT statement. The FORMS or COPIES parameters (and possibly others) specified on the OUTPUT JCL statement might be lost if a SYSOUT data set is requeued from the JES3 HOLD queue to the WTR queue (Q=HOLD to Q=WTR). The system uses the default values.
	The following keywords are supported only on JES3 systems:
	`OVFL     THRESHLD`
	Refer to 4.1.3, "OUTPUT statement parameters" on page 63 for translation considerations.
		The following //OUTPUT keywords are supported only on JES2 systems:
		`DDNAME   GROUPID  INDEX    LINDEX`
		`LINECT   MERGE    OUTDISP`

`// OUTPUT ADDRESS=`
`// OUTPUT BUILDING=`
`// OUTPUT DEPT=`
`// OUTPUT NAME=`
`// OUTPUT ROOM=`
`// OUTPUT TITLE=`
	Supported.	Supported.

`// OUTPUT CHARS=`
`// OUTPUT FCB=`
`// OUTPUT FLASH=`
`// OUTPUT FORMS=`
	JES3 allows the specification of STD for these parameters. Refer to Table 4-4 on page 63 for translation considerations.	There is no standard value for these parameters.

`// OUTPUT CKPTLINE`
`// OUTPUT CKPTPAGE`
`// OUTPUT CKPTSEC`
	When CKPTPAGE and CKPTSEC are both specified, JES3 uses the value on CKPTPAGE. JES3 supports these parameters only via PSF.	When CKPTPAGE and CKPTSEC are both specified, JES2 uses the value on CKPTSEC, unless this is overridden by the installation.

`// OUTPUT COPIES=`	nnn: 0-255 Group value nnn: 1-254 Refer to Table 4-4 on page 63 for considerations of translating COPIES=0.	nnn: 1-255 Group value nnn: 1-255 COPIES=0 is ignored in JES2 (processed the same as not specifying COPIES=).
`// OUTPUT DDNAME=`	Not supported.	Specifies the DDNAME (within the scope of the of the OUTPUT statement) that the OUTPUT statement applies to. This is similar in function to the DDNAME= on the JES3 //*FORMAT statement.

JCL statement and parameter	JES3 system	JES2 system
`// OUTPUT DEST=`	Supports name (an individual local or remote device or workstation) and group-name (a group of local or remote devices or workstations), as well as node name, node name and user ID, or default ANYLOCAL. When a SYSOUT data set is destined for another NJE node, and the SYSOUT class is a held class, the SYSOUT is held for TSO at the destination node when DEST= is specified on the OUTPUT statement. This contrasts with the processing when DEST= is specified on a //*FORMAT statement, but is consistent with JES2 processing.	Supports name (an individual local or remote device or workstation) as well as node name, node name and user ID, or default ANYLOCAL or LOCAL. JES2 can also use DESTIDs to create aliases for destinations that can be specified on DEST=.
`// OUTPUT MERGE=`	Not supported.	Specifies that the characteristics on this OUTPUT statement are to be merged, as defaults, for all SYSOUT generated by this job. This is similar to the function provided by the JES3 non-specific //*FORMAT statement.
`// OUTPUT MODIFY=`	When trc is not specified, JES3 uses the first character arrangement table named in the CHARS parameter. When the trc value is greater than the number of table names in the CHARS parameter, JES3 uses the last character arrangement table named in the CHARS parameter.	When trc is not specified or the trc value is greater than the number of table names in the CHARS parameter, JES2 uses the first character arrangement table named in the CHARS parameter.
`// OUTPUT NOTIFY=`	Supported by PSF and JES3.	Supported by PSF and JES2.
`// OUTPUT PRTY=`	JES3 ignores this parameter for NJE SYSOUT.	Supported.
`// OUTPUT TRC=`	Supported.	JES2 supports this parameter only via PSF.
`// OUTPUT WRITER=`	Reserved names in JES3: INTRDR STDWTR NJERDR	Reserved names in JES2: INTRDR STDWTR
`// PEND`	No difference.	No difference.
`// PROC`	No difference.	No difference.
`// SET`	Supported.	Supported.
`// XMIT`	Supported.	Supported except for the SUBCHARS= keyword. Specifying SUBCHARS= will generate a JCL error.

a. JES3 does not do actual allocation; this is done by MVS.

4.1.1 SYSOUT HOLD processing

Hold processing for SYSOUT in JES3 is different from hold processing for SYSOUT in JES2. JES3 can indicate that a data set is held when it is queued for specific processing. For example, JES3 indicates output is held for XWTR when it is to be processed by an external writer application. Similar holds exist for TSO and NJE. JES3 also does not apply certain characteristics (such as //*FORMAT statements) to output held for TSO until that output is released for processing.

In JES2, there is ready output, held output, and NJE output. Held output has an OUTDISP of HOLD or LEAVE. Ready output has an OUTDISP of WRITE or KEEP. NJE output has a destination that is not the local node.

Held output in JES2 is generally not available for any process to access. The two exceptions to this are the TSO OUTPUT command (using the PSO interface) and the SYSOUT API (SAPI) requesting held output. Any output that was created as held is available to the TSO OUTPUT command (and PSO held data set support), so long as it has never been on the ready queue. SAPI can select any held output. JES2 applies all characteristics (//OUTPUT or /*OUTPUT statements) to the output when it is placed on the held queue. See Table 4-2.

Ready output in JES2 is available to any device (including SAPI and external writers) provided it matches the selection criteria of the device. This implies JES2 managed printers and printers using the FSI (for example PSF) can select output with an external writer name. Also, external writer programs can select output that does not have a writer name specified.

NJE bound SYSOUT is kept separate from all other output. It cannot be held, nor can it be processed by anything except the NJE process.

Operator commands can be used to alter the hold status or NJE destination of SYSOUT at any time, so SYSOUT can be moved between any of the JES2 processing queues.

Understanding the difference in how JES2 and JES3 identifies what output is held will not only help in migration but also in understanding how to manage the output once it is created, such as knowing what output appears on the held versus output display in SDSF.

Table 4-2 Translation of parameters on the // DD statement

Parameter	Value	Translation	Comments
HOLD=	YES	None	JES2 does not provide a means to hold a data set (at the execution node), for release by the operator before printing. However, you might want to remove HOLD=YES from the DD statement, so the data set will still be printed (instead of held for TSO) when run under JES2.

4.1.2 Scanning jobs for JCL errors

It is often desirable to scan jobs for JCL errors before running them. Both JES2 and JES3 support the JOB statement keyword TYPRUN=SCAN and the /*SCAN internal reader statement to process a job through the conversion phase and then stop. In JES3, the same function can also be achieved by using PGM=JCLTEST on one of the steps in the job.

This scan processing in JES3 will run both the MVS converter and interpreter for the job submitted. This identifies most JCL errors that typically exist in a job. However, some runtime errors (such as data set not found) might not be identified.

JES2 does not always run the MVS interpreter during the conversion phase. The specification of INTERPRET= on the JOBDEF statement controls this. If INTERPRET=JES, both the converter and interpreter are run when TYPRUN=SCAN is specified. JES2 does not support the PGM=JCLTEST keyword.

JES3 has additional function that identifies the resources that are required by a job. This is done by coding PGM=JSTTEST on one of the steps. In addition to the functions performed by scan, a list of resources the job requires is included in the output. Since JES2 does not do device or data set management, it does not collect this data and does not support this function. See Table 4-3 on page 63.

Table 4-3 Translation of parameters on the // EXEC statement

Parameter	Value	Translation	Comments
PGM=	JCLTEST	TYPRUN=SCAN on the JOB JCL statement	The MVS interpreter is only called if JOBDEF INTERPRET=JES is specified in JES2. Otherwise, only converter error messages are returned.
PGM=	JSTTEST	TYPRUN=SCAN on the JOB JCL statement	In addition to verifying the JCL, the JSTTEST facility provides information about resources required for the job to execute. The latter function does not exist in JES2.

4.1.3 OUTPUT statement parameters

The OUTPUT JCL statement was designed to replace the JES3 //*FORMAT statement and the JES2 /*OUTPUT statement. As a result, both JES3 and JES2 originally implemented all operands on the statement. However, there are some differences that resulted from the function implemented in each JES. See Table 4-4.

In JES3, certain characteristics of a SYSOUT data set can be defaulted based on installation specification on the OUTSERV statement or based on the SYSOUT class associated with the SYSOUT. If a parameter is not specified on the DD or OUTPUT statement, the value is obtained from the appropriate default. To force the default value for a characteristic, JES3 supports setting the value to STD. This gets the standard (or default) value for the parameter.

JES2 does not allow for installation defaults for parameters. When converting JCL to run on JES2, any parameter that is specified as STD or is set based on installation defaults must be explicitly set in the JCL.

Table 4-4 Translation of parameters on the // OUTPUT statement

Parameter	Value	Translation	Comments

CHARS=
FCB=
FLASH=
FORMS=
	STD	Keyword=value on the OUTPUT JCL statement	When STD is specified on these keywords, you can replace STD with the appropriate value from the JES3 OUTSERV initialization statement, or the default value if none is coded on OUTSERV, as follows:
			`OUTPUT       OUTSERV   Default`
			`------       -------   -------`
			`CHARS        CHARS     GS10`
			`FCB          CARRIAGE  6`
			`FLASH        FLASH     NONE`
			`FORMS (pr)   FORMS     1PRT`
			`FORMS (pu)   CDSTOCK   5081`

COPIES=	0 causes the output to be removed from spool without going anywhere	Easiest conversion is to replace with OUTDISP=PURGE	There are some alternatives in JES2 to obtain the JES3 COPIES=0 function. You could change the DD to a DUMMY statement, make use of the OUTDISP=PURGE function, or use a DUMMY SYSOUT class.
THRESHLD=	limit	None	JES2 does not support this parameter.

4.1.4 Command statements

JCL can contain commands that are processed as the job goes through input or conversion processing. In the case of MVS commands, these will work as expected. However, if JES3 specific commands are included, you need to determine the purpose of the command and convert them to the appropriate JES2 or MVS commands.

For example, there might be commands to set up the system to run the job (such a varying a device online). With JES2, you might no longer need to do this or perhaps the setup needs to be done using MVS commands instead of JES3 commands. See Table 4-5.

Table 4-5 Translation of parameters on the JES3 command statement

Parameter	Value	Translation	Comments
command	command parameters	Equivalent JES2 commands	You will have to manually convert JES3 commands to JES2 commands. The main use of this statement is to enter JES3 commands via the JES3 DISKRDR facility. JES2 has no equivalent to the JES3 DISKRDR facility.

4.2 JECL statements

Unlike JCL statements, which are generally intended to be independent of which job entry subsystem is being used, Job Entry Control Language (JECL) is specifically designed to be used to communicate with a specific job entry subsystem. Statements starting with "//*" followed by a keyword (with no space) identify JES3 JECL. Statements starting with "/*" followed by a keyword (with no space) identify JES2 JECL statements.

This section provides information about the various JES3 JECL statements, and how you would address the functions that these statements deliver when you migrate to JES2. In many cases, the recommended approach is to replace the JES3 JECL statements with standard (JES-neutral) JCL statements. Often, this can be done in advance of the migration.

One thing to note when converting JECL statements to JCL is the placement of the statements within the job. Though many JECL statements have recommended placements within a job, only a very few are required to be in a specific place. Because of this, replacing JECL with JCL might require the JCL to be placed in a specific context.

For example, you decide to replace a //*FORMAT statement with a // OUTPUT statement. The //*FORMAT statement can be placed anywhere in the job. Generally they are placed after the JOB statement and before the first EXEC statement. But they could be placed after an EXEC statement. Where the //*FORMAT statement is located has no effect on the function of the statement. Replacing the //*FORMAT with a // OUTPUT statement at the same place in the JCL does make a difference. If the // OUTPUT statement is placed before the first EXEC statement, it applies to the entire job. Placed after an EXEC statement limits the scope of the statement to that step.

4.2.1 PROCESS and DATASET JECL

JES3 has a set of JECL to identify specific DSPs that are to be used when processing a job. The DSPs can be standard DSPs or installation defined DSPs. The //*PROCESS and //*ENDPROCESS JECL statements identify the DSPs. The //*DATASET and //*ENDDATASET JECL is used to pass data into the DSPs specified.

The JES2 equivalent of a JES3 DSP or FCT is the PCE. Though these can be added by installations and dynamically created and deleted, there is no facility in JES2 to select which PCEs will process a job. Any job that uses the //*PROCESS and related statements needs to be reworked to not use these.

One common use of the //*PROCESS statement is for output-only jobs. This type of job creates a data set and uses the OUTSERV DSP to place the job on the SYSOUT queue for processing. No application program is executed when this process is used; the “job” just creates output. In JES2, this would need to be replaced with an IEBGENER job that would copy the selected data to an appropriate SYSOUT data set. See Tables Table 4-6 and Table 4-7.

For other more complicated processes, a PROC and appropriate program could be created to process any input data appropriately. The //*PROCESS and //*DATASET statements would then be replaced with the execution of the PROC that causes the wanted processing to be performed with the job executes.

In all these cases, the job must execute to achieve the wanted results.

*Table 4-6 Translation of parameters on the //*PROCESS statement*

Parameter	Value	Translation	Comments
(parameters on statement following PROCESS)	(parameters on statement following PROCESS)	None	JES2 does not have an equivalent to the //*PROCESS function. You will be able to convert some types of //*PROCESS OUTSERV jobs into IEBGENER jobs.

*Table 4-7 Translation of parameters on the //*ENDPROCESS statement*

Parameter	Value	Translation	Comments
N/A	N/A	None	The //*ENDPROCESS statement indicates the end of the input for the //*PROCESS statement. JES2 does not have an equivalent to the //*PROCESS facility.

The DATASET statements are a way to pass data into a DSP. JES2 does not have an equivalent function. See Tables Table 4-8 and Table 4-9.

*Table 4-8 Translation of parameters on the //*DATASET statement*

Parameter	Value	Translation	Comments
(All)	(All)	None	The //*DATASET statement is used to supply input to JES3 Dynamic Support Programs (DSPs) named on //*PROCESS statements. JES2 has no equivalent to the //*PROCESS function.

*Table 4-9 Translation of parameters on the //*ENDDATASET statement*

Parameter	Value	Translation	Comments
N/A	N/A	None	The //*ENDDATASET statement indicates the end of the input to the DSP. JES2 has no equivalent to the //*DATASET or //*ENDDATASET function.

4.2.2 FORMAT JECL

The FORMAT statement performs three basic functions in JES3. It allows for the specification of SYSOUT characteristics not defined on the DD statement. It provides a mechanism for getting multiple instances of a data set (using multiple specific FORMAT statements for the same DD). It can provide job level defaults for SYSOUT characteristics. All of these functions are available with JES2 using the OUTPUT JCL statement.

When the original OUTPUT JCL statement came out, it was intended as a replacement for both the JES3 FORMAT and JES2 OUTPUT statement. It did provide the needed characteristics and has been expanded over the years with many new characteristics. But the original statements did not have all of the functions of the FORMAT statement. However, the

OUTPUT statement was enhanced with new features that make it easier to convert FORMAT statements to OUTPUT statement.

One feature of the //*FORMAT statement is the non-specific format statement. This provides job-wide default characteristics for the all SYSOUT that is created by the job. The // OUTPUT statement had DEFAULT=YES and JESDS=ALL to provide default OUTPUT statements for all SYSOUT, but if one data set needed a special characteristic to print and thus a specific OUTPUT statement, the default OUTPUT statement would not apply. In other words, if a SYSOUT data set had an OUTPUT statement associated with it, all the needed characteristics had to be specified on that OUTPUT statement.

To address this, the MERGE=YES keyword was added to the traditional OUTPUT statement. These merge OUTPUT statements provided default characteristics for all OUTPUT within the scope of that OUTPUT statement (at the JOB or STEP level). Similar to the non-specific FORMAT statement, any characteristic specified on a merge OUTPUT statement would apply to every instance of a SYSOUT data set created, unless the same characteristic was specified on a specific/default OUTPUT statement. As with the non-specific FORMAT statement, only one merge OUTPUT statement applies. However, because it is JCL and not JECL, the merge OUTPUT statement applies to the environment it is in. So if placed at the job level, it applies to the entire job. If placed at the step level, it applies to SYSOUT in that step. If there are merge OUTPUT statements at the job and step level, the step level merge OUTPUT statement applies to SYSOUT data sets in the step and the job level one applies to all SYSOUT not in that step. Two merge OUTPUT statements never apply to the same SYSOUT data set.

Another addition to the OUTPUT statement is a DDNAME= parameter. Much like the specific FORMAT statement, this allows the OUTPUT statement to point to the SYSOUT DDs to which it applies (instead of having to point the DDs to the OUTPUT statement). This will simplify the migration of //*FORMAT statements to // OUTPUT JCL.

One thing that does not exist with OUTPUT statements is a distinction between print and punch data sets. If there is a distinction that needs to be made (for example, a job with non-specific print and punch FORMAT statements that have different characteristics), you will need to code the appropriate OUTPUT statements directed to the specific print or punch data sets.

Common FORMAT PR and PU keywords

Table 4-10 lists the parameters common to FORMAT for print and punch SYSOUT data sets.

*Table 4-10 Translation of parameters common to //*FORMAT PR and //*FORMAT PU*

Parameter	Value	Translation	Comments

DDNAME=	,
	ddname
	stepname.ddname
	stepname.procstepname.ddname
	JESYSMSG
	SYSMSG
	JESJCL
	JESMSGLG
	JESMSG
		//xxx OUTPUT ....DDNAME=
		JESDS=MSG on the OUTPUT JCL statement
		JESDS=JCL on the OUTPUT JCL statement
		JESDS=LOG on the OUTPUT JCL statement
			For specific FORMAT statements, use DDNAME= on the OUTPUT statement. For non-specific code, use MERGE=YES on the OUTPUT statement.

Parameter	Value	Translation	Comments
CHNSIZE=	DS (nnn,mmm)	None	There is a CHNSIZE parameter on the JES3 DEVICE Initialization statement. User jobs will not specify the number of records to be transmitted to an SNA work station, or when output checkpoints are to occur. The installation-specified values will be used.
COMPACT=	compaction-table-name	COMPACT=compaction-table name on the OUTPUT JCL statement	Direct translation.
COPIES=	nnn (nnn,(group-value,...)) (group-value,...)	COPIES= on the OUTPUT JCL statement	JES2 does not support the specification of COPIES=0 on the OUTPUT statement. Use OUTDISP=PURGE to not print SYSOUT.
DEST=	destination	DEST= on OUTPUT JCL statement	If DEST is specified on a FORMAT statement, and a file that is affected by this FORMAT statement is in a class defined to be held (that is, HOLD=TSO or HOLD=EXTWTR on the SYSOUT JES3 initialization statement for the class), the file will be held at the execution node. Contrasted with this, if DEST is specified on an OUTPUT statement, the file will be held at the destination node.
ERDEST=	destination	DEST= on OUTPUT JCL statement	ERDEST= is a undocumented synonym for DEST=. Any uses of ERDEST will be replaced with DEST=.
EXTWTR=	name	This keyword is similar in function to WRITER=.	EXTWTR is the writer to use for this output when it reaches the destination node. However, processing is different if the output is not destined to a known NJE destination. See “OUTPUT WRITER= versus FORMAT EXTWTR=” on page 68 for details.

Parameter	Value	Translation	Comments
FORMS=	STANDARD form-name	FORMS=value on the OUTPUT JCL statement	When STANDARD is specified with //*FORMAT PR, the value specified on the FORMS parameter on the OUTSERV JES3 initialization statement is used. The default value is '1PRT'. When STANDARD is specified with //*FORMAT PU, the value specified on the CDSTOCK parameter on the OUTSERV JES3 initialization statement is used. The default value is '5081'.

OUTPUT WRITER= versus FORMAT EXTWTR=

There is a subtle difference between the workings of WRITER= on the OUTPUT JCL statement and EXTWTR= on the JES3 FORMAT JECL statement.

The WRITER=name on the OUTPUT statement identifies an external writer to process the SYSOUT data set. If the output is destined for a known NJE node, JES3 places the SYSOUT on the appropriate queue for the destination node. Then, when the output reaches the destination node it is placed on Q=HOLD for WRITER=name. If the output is not destined to a known NJE node, JES3 places the output on Q=HOLD for WRITER=name (held for WRITER=name).

The EXTWTR=name on the JES3 FORMAT JECL identifies an external writer at a destination node that is to process the SYSOUT data set. If the output is destined for a known NJE node, JES3 places the SYSOUT on the appropriate queue for the destination node. Then, when the output reaches the destination node it is placed on Q=HOLD for EXTWTR=name. If the output is not destined to a known NJE node, JES3 places the output on Q=WTR (not held for EXTWTR=name).

FORMAT PR keywords

Table 4-11 lists the parameters that are unique to the FORMAT print statement.

*Table 4-11 Translation of parameters on the //*FORMAT PR statement*

Parameter	Value	Translation	Comments
CARRIAGE=	STANDARD carriage-tape-name	FCB=value on the OUTPUT JCL statement	Archaic keyword for 3211 devices. When STANDARD is specified, the value specified on the CARRIAGE parameter on the OUTSERV JES3 initialization statement is used. The default value is “6”.
CHARS=	STANDARD table-name	CHARS=value on the OUTPUT JCL statement	When STANDARD is specified, the value specified on the CHARS parameter on the OUTSERV JES3 initialization statement is used. The default value is “GS10”.
FCB=	image-name	FCB=value on the OUTPUT JCL statement	When STANDARD is specified, the value specified on the CARRIAGE parameter on the OUTSERV JES3 initialization statement is used. The default value is “6”.

Parameter	Value	Translation	Comments
FLASH=	STANDARD overlay-name (overlay-name,count)	FLASH=value on the OUTPUT JCL statement	When STANDARD is specified, the value specified on the FLASH parameter on the OUTSERV JES3 initialization statement is used. The default value is "NONE".
MODIFY=	module-name (module-name,trc)	MODIFY=value on the OUTPUT JCL statement	
OVFL=	ON OFF	OVFL=ON|OFF on the OUTPUT JCL statement	Not used by JES2.
PRTY=	nnn	OVFL=nnn on the OUTPUT JCL statement	

STACKER=	STANDARD
	S
	C
		BURST=YES|NO on the OUTPUT JCL statement	STACKER=S is BURST=Y (separate sheets)
			STACKER=C is BURST=N (continuous fanfold)
			When STANDARD is specified, the value specified on the STACKER parameter on the OUTSERV JES3 initialization statement is used. The default value is "C" (BURST=N).

THRESHLD=	limit	THRESHLD=limit on the OUTPUT JCL statement	Not used by JES2.
TRAIN=	STANDARD train-name	UCS=value on the OUTPUT JCL statement	When STANDARD is specified, the value specified on the TRAIN parameter on the OUTSERV JES3 initialization statement is used. The default value is "PN".

FORMAT PU keywords

Table 4-12 lists the parameters that are unique to the FORMAT punch statement. Since there is no distinction between print and punch on OUTPUT statements, these parameters are translated to a standard OUTPUT statement.

*Table 4-12 Translation of parameters on the //*FORMAT PU statement*

Parameter	Value	Translation	Comments
INT=	YES NO	None (delete keyword)	Option unique to a 3525 card punch.

4.2.3 MAIN JECL

The JES3 //*MAIN JECL statement specifies job level processing parameters. Many of these options are now available on the job statement or other JCL statements. Some of these keywords represent functions that do not exist in JES2 and thus cannot be translated. Others represent concepts that are considered archaic because they no longer apply to modern systems and will be deleted (for example MSS or TYPE).

The USER= keyword on //*MAIN provides a function that will be obsolete but might still be in use. On the JCL JOB statement, USER=userid identifies to the system the user ID under whose authority the job will run. However, the USER= keyword on the JES3 MAIN JECL statement is used to identify a TSO user that can issue the TSO/E OUTPUT, STATUS, or

CANCEL commands for the job if normal RACF checking is not being done to verify access to the job. USER= on the MAIN JECL predate the RACF checking using profiles in the JESSPOOL and JESJOBS classes. It also requires the use of JES3 user exits IATUX29 and IATUX30. So though the keyword on the MAIN JECL matches the keyword on the JOB statement, the function provided is very different.

*Table 4-13 Translation of parameters on the //*MAIN statement*

Parameter	Value	Translation	Comments
ACMAIN=	processor-id	None	Archaic. This statement will no longer be used. Enhancements in the handling of SYS1.BRODCAST data sets obviate the need to specify the processor ID for notify messages.

BYTES=	(nnnnnn,WARNIN G,mmm)
	(nnnnnn,W,mmm)
	(nnnnnn,CANCEL)
	(nnnnnn,C)
	(nnnnnn,DUMP)
	(nnnnnn,D)
		BYTES=(nnnnnn,WARNING) on the JOB JCL statement
		BYTES=(nnnnnn,CANCEL) on the JOB JCL statement
		BYTES=(nnnnnn,DUMP) on the JOB JCL statement
			If BYTES= is already specified on JOB, nothing needs to be done because this statement on the JOB statement overrides MAIN.
			The mmm value lets you control the frequency of warning messages being issued (WARNING,mmm). This capability is not supported on the JOB statement (the default is 50, in other words 50% of nnnnnn).

CARDS=	(nnnn,WARNING, mmm)
	(nnnn,W,mmm)
	(nnnn,CANCEL)
	(nnnn,C)
	(nnnn,DUMP)
	(nnnn,D)
		CARDS=(nnnnnnnn,WARNING) on the JOB JCL statement
		CARDS=(nnnnnnnn,CANCEL) on the JOB JCL statement
		CARDS=(nnnnnnnn,DUMP) on the JOB JCL statement
			If CARDS= is already specified on the JOB statement, nothing needs to be done because this parameter on the JOB statement overrides MAIN.
			On MAIN, CARDS is specified in hundreds, but on the JOB statement it is specified in ones. When MAIN CARDS is to be translated to JOB CARDS, the number of cards value on the MAIN must be multiplied by 100.
			The frequency of warning message issued (WARNING,mmm) is not supported on the JOB statement (the default is 50, in other words 50% of nnnnnnnn).

CLASS=	class-name	CLASS=class-name on the JOB JCL statement	Convert to having your CLASS= statement on the JOB statement (8-character class names are supported on the job statement). Note that on the MAIN statement, the character set that can be used for the job class is virtually unlimited. The JOB statement limits it to A-Z and 0-9.
DEADLINE=	(time,type,date) (time,type,rel,cycle)	None	No JES2 equivalent currently exists.
EXPDTCHK=	YES NO	None	No JES2 equivalent exists.
FAILURE=	RESTART CANCEL HOLD PRINT	RESTART=Y or N on the /*JOBPARM statement.	FAILURE=CANCEL or HOLD or PRINT have no equivalent in JES2.

Parameter	Value	Translation	Comments
FETCH=	ALL NONE SETUP (ddname,ddname,...) /(ddname,ddname,...)	None	No JES2 equivalent exists.

HOLD=	YES
	
	NO
		TYPRUN=HOLD on the JOB JCL statement
		TYPRUN=HOLD **not** on the JOB JCL statement
			Translation to TYPRUN=HOLD must be bypassed if TYPRUN is already specified, since the TYPRUN specifications are mutually exclusive.

IORATE=	MED HIGH LOW	None	Archaic. No JES2 equivalent exists.

JOURNAL=	YES
	NO
		RD=R on the JOB JCL statement
		RD=NR on the JOB JCL statement
			If RD is already specified on the JOB statement, do not change it to match the value on the JOURNAL parameter. Making such a change could change existing specifications (for example, if RD=RNC is already specified, translation to RD=R would change the no checkpoint specification).

LINES=	(nnnn,WARNING, mmm)
	(nnnn,W,mmm)
	(nnnn,CANCEL)
	(nnnn,C)
	(nnnn,DUMP)
	(nnnn,D)
		LINES=(nnnnnn,WARNING) on the JOB JCL statement
		LINES=(nnnnnn,CANCEL) on the JOB JCL statement
		LINES=(nnnnnn,DUMP) on the JOB JCL statement
			If LINES= is already specified on the JOB statement, no action is required because the value on the JOB statement overrides the LINES value on the MAIN statement. The frequency of warning message issued (WARNING,mmm) is not supported on the JOB statement (the default is 50, in other words 50% of nnnnnn).

LREGION=	nnnnK	None	Archaic. No JES2 equivalent exists.
MSS=	JOB HWS	None	Archaic. No JES2 equivalent exists.
ORG=	groupname nodename.remote	destname on the /*ROUTE PRINT and /*ROUTE PUNCH statements.	The groupname function can be handled in JES2 via appropriate DESTID statements in the initialization stream. One /*ROUTE PRINT and one /*ROUTE PUNCH approximate the function of ORG=, but do not cover all the possibilities. For strictly output routing, a merge OUTPUT statement with DEST= might do what is needed.

Parameter	Value	Translation	Comments

PAGES=	(nnnnnnnn,WARNING,mmm)
	(nnnnnnnn,W,mmm)
	(nnnnnnnn,CANCEL)
	(nnnnnnnn,C)
	(nnnnnnnn,DUMP)
	(nnnnnnnn,D)
		PAGES=(nnnnnnnn,WARNING) on the JOB JCL statement
		PAGES=(nnnnnnnn,CANCEL) on the JOB JCL statement
		PAGES=(nnnnnnnn,DUMP) on the JOB JCL statement
			If PAGES= is already specified on the JOB statement, no action is required because the value on the JOB statement overrides the PAGES value on the MAIN statement. The frequency of warning message issued (WARNING,mmm) is not supported on the JOB statement (the default is 50, in other words 50% of nnnnnnnn).

PROC=	ST xx	PROCLIB=ddname on the JCLLIB JCL statement.	You need to determine the equivalent concatenation in JES2 (most logically it would be PROCxx). ST would most logically be PROC00. Could also use PROCLIB= on the JOBPARM JECL statement.
RINGCHK=	YES NO	None	Archaic. No JES2 equivalent exists.
SETUP=	JOB HWS THWS DHWS (stepname.ddname,...) (stepname.procstepname.ddname,...) /(stepname.ddname,...) /(stepname.procstepname.ddname,...)	None	No JES2 equivalent exists.
SPART=	spool partition name	None	No JES2 equivalent exists.

SYSTEM=	ANY
	JGLOBAL
	JLOCAL
	(main-name,...)
	/(main-name,...)
		SYSTEM=ANY on the JOB JCL statement
		SYSTEM=(cccc,...) on the JOB JCL statement
		SYSTEM=(-cccc,...) on the JOB JCL statement
			You use the SYSTEM= keyword on the JOB statement with MVS system names or the SYSAFF=keyword on the JOB statement with JES2 member names. With either keyword, "*" implies the input system/member. JES2 does not have a global and locals so there is no equivalent to JGLOBAL and JLOCAL.

THWSSEP=	IGNORE PREFER REQUIRE	None	No JES2 equivalent exists.
TRKGRPS=	(primary-qty,second-qty)	None	Archaic. No JES2 equivalent exists.
TYPE=	ANY VS2	None	Archaic. No JES2 equivalent exists.
UPDATE=	(dsname,...)	None	No JES2 equivalent exists.
USER=	user ID	None	No JES2 equivalent exists.

4.2.4 NET JECL

The NET JECL statement implements dependent job control in JES3. This allows applications submitting JCL to cause jobs to run in a particular sequence. It also allows some jobs to be skipped based on the results of other jobs. In JES2 installations, this is either done with a single job that uses IF/THEN/ELSE JCL or by using an external job scheduler. Many customers implemented job networks as a solution for the problems with JES3 SETUP processing and conditional steps. See Table 4-14.

*Table 4-14 Translation of parameters on the //*NET statement*

Parameter	Value	Translation	Comments
(All)	(All)	None	No equivalent function currently exists in JES2.

4.2.5 NETACCT JECL

The JES3 NETACCT JECL provides a way to specify typical accounting string values to associate with a job when it is sent over NJE. These values typically appear in SMF records to manage billing for a job. In some installations, the value specified might have nothing to do with the keyword. For example, BLDG= could be used as an employee number or part of the accounting location.

The JES3 NETACCT JECL will not be confused with the JES2 NETACCT JECL statement.

The JES3 NETACCT allows for the specification of a number of attributes that are often extracted from the accounting string on the job statement. It allows for different values on the local and destination NJE node.

The JES2 NETACCT JECL statement only supports an 8-character accounting string to be used at the destination node. It does not support the other parameters.

In translating these values, you need to understand the billing process that will be used on the JES2 system versus what was done on the JES3 system or at the destination for the job or SYSOUT. If a job is being sent (versus SYSOUT), the accounting information can be included on the JOB statement sent to the destination node. See Table 4-15.

*Table 4-15 Translation of parameters on the //*NETACCT statement*

Parameter	Value	Translation	Comments
PNAME	Programmer-name	Programmer name field on the JOB JCL statement	Specify the programmer name following the accounting information on the JOB statement.
ACCT	network-account-number	Network account number on the /*NETACCT JECL statement	Could also use the accounting field on the job statement if that is not currently being used.
BLDG	building	None	No equivalent exists at the job level in JES2. Building can be specified on the OUTPUT JCL.
DEPT	department	None	No equivalent exists at the job level in JES2. Department can be specified on the OUTPUT JCL statement.

Parameter	Value	Translation	Comments
ROOM	room	ROOM=room on the /*JOBPARM JECL statement	
USERID	userid	User ID on the /*NOTIFY JECL statement	This user ID is used in JES3 as the notify user ID

4.2.6 OPERATOR JECL

The OPERATOR JECL is used to communicate with the “operator” information about the job being submitted. Often, in modern systems, this is actually communicating with an automation package that triggers off the message to accomplish some task needed to run the job. This can be translated to the JES2 /*MESSAGE statement; however, the text length in JES2 is limited to 61 characters (JES2 does not send the traditional sequence number fields). See Table 4-16.

*Table 4-16 Translation of parameters on the //*OPERATOR statement*

Parameter	Value	Translation	Comments
message	N/A	message on the /*MESSAGE JECL statement	The message length on OPERATOR statement can be up to 68 characters, but is limited to 61 characters on MESSAGE. You can either truncate the message, or intelligently split it into two lines if greater than 61 characters.

4.2.7 PAUSE JECL

The PAUSE JCL causes JES3 to stop reading from the input device that the command was sent from. A message must be replies to or a command received from a remote device in order to resume reading data. The main use for this was with physical card readers on remote workstations. JES2 does not have this function. See Table 4-17.

*Table 4-17 Translation of parameters on the //*PAUSE statement*

Parameter	Value	Translation	Comments
N/A	N/A	None	No JES2 equivalent exists.

4.2.8 ROUTE XEQ JECL

There are two ways to route a job to another NJE node. The // XMIT JCL statement is designed to be compatible with both JES3 and JES2. If that is being used, no translation is required (except possibly removing the usage of the SUBCHARS= keyword). The other way to route jobs in JES3 is to use the //*ROUTE XEQ JECL statement. Though similar in syntax to the JES2 /*ROUTE XEQ JECL statement, the function is not the same.

In JES2, there are two methods used to transmit jobs to NJE nodes. The first uses the // XMIT JCL or the /* XMIT JECL. These, like the JES3 //*ROUTE XEQ JECL, use one JOB statement before the XMIT and one job statement (or similar if not sending to z/OS) after the XMIT statement. In this case, the first JOB statement is processed locally on the submitting system and everything after the XMIT is sent to the destination NJE node without any processing on the submitting node. This allows any stream of records to be sent to a destination NJE node, even if it is not JCL.

The second method JES2 uses involves the use of the /*ROUTE XEQ JECL statement. In this case, there is only one JOB statement. The job goes through input processing normally in JES2 and must successfully pass input processing (including any validation of JCL and JECL statements on the input node). Then, once the job completes input processing, it is routed to the destination node. This limits the input to JCL that is understood by the input system. The advantage to this method is that some validity checking is done on the input node and the job is not set over NJE if the checking does not pass.

There is also a difference if jobs are submitted with passwords specified. Both JES2 and JES3 cannot look at anything beyond the XMIT statement. However, in JES3, there is an option to take a password off the first job statement, encrypt it, and send it with the job to the other node. In JES2, the password can only be encrypted in the one job statement case (when /*ROUTE XEQ is used). However, none of this is an issue if you use encrypted TCP/IP connections.

*Table 4-18 Translation of parameters on the //*ROUTE XEQ statement*

Parameter	Value	Translation	Comments
(All)	(All)	Equivalent to // XMIT JCL statement. Add DEST= before the nodename specification.	Although the JES3 ROUTE XEQ is syntactically closer to the JES2 ROUTE XEQ, it is functionally closer to the JES2 XMIT JECL or the XMIT JCL statement. You are able to convert to either statement. When converting be sure to change any “NJB” on the JOB statement to “JOB”.

4.3 Other JCL considerations

In addition to translating individual JCL/JECL statements, there are techniques that are used in JCL that are specific to JES3 processing. Some of them are discussed in this section.

4.3.1 Allocation differences

JES3 processing can get involved in many aspects of allocation processing. However, there are aspects of JCL and allocation processing that JES3 might not be able to anticipate. This can result in unexpected errors when the job is processed by JES3. Over the years, techniques have been developed to allow these jobs to be processed as expected. When translating these jobs to run on JES2, some of these JES3 accommodations can be deleted.

Conditional step processing

If a data set is uncataloged or deleted in a step with the COND parameter on the EXEC statement, JES3 ignores the uncatalog/delete disposition during CI. When the same data set is requested later in the job with disposition SHR or OLD, JES3 assumes that the two steps are mutually exclusive. If both steps run, the job will fail during execution. By contrast, JES2 processes the requested disposition based on the COND parameter and the steps run as planned.

Dynamic allocation

When a data set is allocated in step one by dynamic allocation and the second step accesses this data set with DISP=SHR/OLD, in JES3 this job fails in setup. JES2 does not do setup processing, which avoids this error. The job only fails when the data set is not available at execution time.

4.3.2 JCL standards

IBM stated direction is that JECL statements will not be enhanced and any future job control language changes will be made only to JCL statements (such as the //JOB and //OUTPUT statements). Those changes will be implemented in the same manner in both JES2 and JES3 (though not always in the same release).

Therefore, if there are functions that you use or plan to use that are supported using both JECL and JCL (such as on both the //*MAIN and //JOB statements), you will use the //JOB statement version of the function.

4.3.3 Retaining tapes

The RETAIN keyword works the same in JES2 and JES3. A tape remains mounted if it is used again in the same job.

4.4 JES2 options

In the long term, we suggest that the best solution is to convert your JES3 JCL and JECL to the JES2 equivalent. This avoids confusion, eliminates the need to maintain skills in both types of JECL, and provides you with a clean base for moving forward. Nearly every JES3-provided function can be re-created in a JES2 environment through a combination of standard JES2 functions, z/OS functions, and (if necessary) user exits.

While we generally encourage the use of as few user exits as possible, if you encounter a situation where a user exit is required, you can investigate the so-called “Mellon Mods” for JES2 to see if you can find an existing exit that provides the functionality you need. The Mellon Mods and other JES2 exits and usermods can be downloaded from the following web site:

`http://www.cbttape.org/jes2down.htm`

The CBT tape is not owned or operated by IBM, and anything you download from that site is provided on an as-is basis. Considering the criticality of the migration being successful, you will carefully consider if you want to be reliant on an unsupported function.

Another possibility is a non IBM software product called zOSEM from a company called Trident Services. That product provides functions that might allow you to run JES3 JECL and JCL unchanged on a JES2 system. Also, there is a product called Thruput Manager AE from MVS Solutions.

4.5 Identifying jobs that require changes

Having identified the types of JECL and JCL statements that could potentially cause an issue, the next challenge is finding the jobs that use those statements so that you can fix them.

There are hundreds of potential sources of batch jobs in a z/OS environment. An obvious one is the batch job scheduler. Another one is private JCL libraries belonging to TSO users. You might have home-written tools that run under ISPF that build JCL dynamically. There are products that will create JCL dynamically. Chapter 10, “Related products” on page 135 contains a description about the types of products that might be affected by a JES migration.

To help you identify jobs that contain potentially problematic JCL or JECL statements, you could use JES3 exit 33. A sample exit is contained in Appendix A, “Sample JES3 exit to analyze JECL usage” on page 227.

4.6 Reference information

For more information about the differences between JES2 and JES3 JECL, refer to *z/OS MVS JCL Reference*, SA22-7597. There is also valuable information in the following SHARE presentation:

`https://share.confex.com/share/119/webprogram/Handout/Session11710/JES2-JES3%20JCL-JECL%20Differences.pdf`

5 Migration considerations

This chapter describes considerations and methods associated with migrating from JES3 to JES2. The first part of this chapter discusses some of the considerations that you might use in the assessment of migration. The second part of this chapter discusses the discovery process in terms of gathering data for analysis to build a business case.

In this chapter the following topics are discussed:

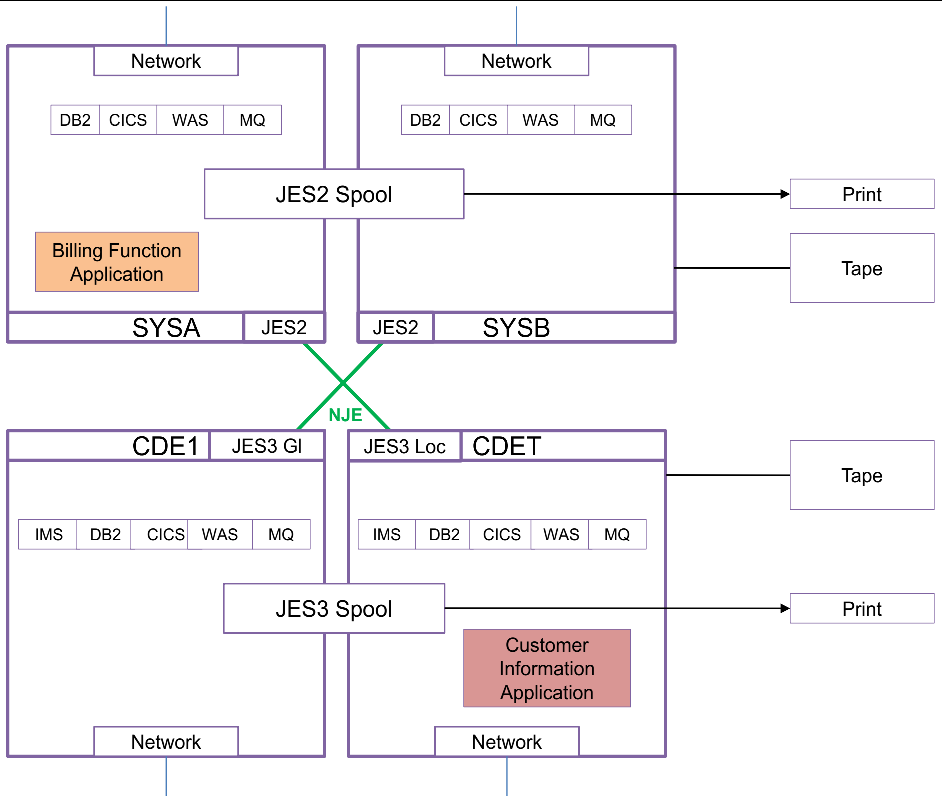
- Making an informed decision
- Discovery process
- Building a business case

5.1 Making an informed decision

Each installation analyzes their needs regarding JES management. One factor that might help with the decision to migrate from JES3 could be to consider your current configuration with a goal in mind. For example, today your configuration might have separate JESPLEXs on multiple LPARs on separate CPCs. These could have vastly different or similar workloads on them. A goal might be to combine some of that workload to run together based on business needs or capacity exploitation. That being said, it would be simpler to run that new mixed workload in its entirety within either JES2 or JES3.

Through mergers and acquisitions, installations end up with multiple JES systems in their environments. Consider an example of four distinct environments from four different companies who merge. Three of them utilize JES2 and one uses JES3. Based on certain processing requirements it might be desirable to combine batch workloads for business reasons. A need to combine jobs might be focused on capacity workload balancing or system affinity to a particular LPAR. Being able to relate workloads under the same JESPLEX would be of benefit from a system and application management standpoint.

Figure 5-1 shows an example of a billing application that is normally executed in a JES2 environment. But it needs to also run a batch function in the Customer Information Application that executes in the JES3 environment.



Here, a billing application running on a JES2 system could benefit from a single JESPLEX rather than having to do part of its processing on a JES3-based system.

Figure 5-1 Sysplex containing both JES3 and JES2

5.2 Discovery analysis process

To create a solid business case to migrate, you need to do some analysis to set the scope of the effort. In order to accomplish this, there needs to be an understanding, at an application level, as to the magnitude of impact. There are several things that you might want to do in the discovery phase of this migration effort, examples of which are outlined in this chapter. The more you can find out about the systems targeted for migration, the higher the probability of success you will have. This step is vital in figuring out the scope of change for planning purposes.

One of the first things is create an inventory of the JES3 systems that are the subject of migration. This becomes important for later analysis. Review the JES3 functions and features currently being used today so that a corresponding action can be implemented in the JES2 target system. Some things to consider are:

- JES3 functions used such as Dependent Job Control (DJC), Main Device Scheduling (MDS) setup, and high water mark setup.
- Study the characteristics of the initialization decks commonly used operator commands (local and remote).
- Review SMF data on accounting metrics.
- Inventory JES3 exits.
- Dynamic support programs (DSP) and user mods.
- ISV products.

In addition, identifying actual JECL in use for JES3 would be beneficial to analyze as well. Using third-party ISV products or creating a user exit can yield good results for finding this type of information. This helps quantify the scope of changes that might be needed for the target system.

5.2.1 Identifying JCL and JECL changes

In creating an exit, you want it to be called early in the routine to capture JCL statements before them being modified by any other part of the exit. Here is an example of an exit:

See Appendix A, “Sample JES3 exit to analyze JECL usage” on page 227 for example code of this exit.

IATUX33 was used for this exit and provides a basic scanning for JES3 JECL being called. It tracks events using the existing CNXTRKR component of z/OS. The output of which is not highly formatted due to limitations of 28 bytes. The //*DATASET and //*ENDDATASET do not call IATUX33; therefore, these are not tracked by this exit. A good reference to more information about the tracker component is in *z/OS V1R13.0 MVS Planning: Operations*, SA22-7601.

Note: If someone else activates or deactivates the tracker for other z/OS based information for example, this would also activate or deactivate the tracking function for others as well.

5.2.2 Activating the exit

To turn on the exit:

```
SETCON TRACKING=ON
```

To turn if off:

```
SETCON TRACKING=OFF
```

You can clear the Tracker by turning if off and then on.

You can display the data with:

```
DISPLAY OPDATA, TRACKING
```

Things to note are the sysid, job name, job number, user ID, and JECL statements.

Here is some sample output from an IATUX33 exit code. As you can see, this can be a useful tool to help identify JECL that could need changing.

Here is what it looks like if it found five //*MAIN cards, three //*PROCESS cards, and one //*ENDPROCESS card.

Example 5-1 JES3 exit sample output

```
00- D OPDATA,TRACKING
    SY1           CNZ1001I 10.42.43 TRACKING DISPLAY 415                    C
    STATUS=ON       NUM=3    MAX=1000 MEM=n/a EXCL=0     REJECT=0
    ----TRACKING INFORMATION---- -VALUE-- JOBNAME  PROGNAME+OFF-- ASID NUM
    JES3 JECL: EPRC                    00 JES3     IATNUC      00   1D   1
    JES3 JECL: MAIN                    00 JES3     IATNUC      00   1D   5
    JES3 JECL: PROC                    00 JES3     IATNUC      00   1D   3
    ----------------------------------------------------------------------
```

5.2.3 Understanding the output

We have 28 bytes to put into INFORMATION and four bytes for VALUE.

At the time of this writing, the format is to start each entry with “JES3 JECL: “, then write out four characters to identify the JECL called. See Table 5-1 for mapping.

Table 5-1 Output mapping of JECL to 4 Character ID

JECL	4 Character ID
//*ROUTE	ROUT
//*FORMAT	FORM
//*PROCESS	PROC
//*ENDPROCESS	EPRC
//*MAIN	MAIN
//*OPERATOR	OPER
//*NETACCT	NETA
//*NET	NET

Trying to find out this information without an exit is error prone and time consuming but not insurmountable. Other methods to try to track down JES3 functions comprise multiple steps. The first thing that you can do is review the Initialization deck for JES3 and see if certain functions are defined. MDS or high water setup are examples of this. The problem is even if you identify functions active within a JES3 system, not all applications might be using them. In this case, you must find other avenues to locate information. An additional way is to review the syslog or schedulers for information. Then, use compare software to identify JECL statements. Finally, since there is not an easy automated way to do this analysis without an exit of some kind, it might be necessary to meet with the application teams individually asking them to help inventory what they are using. This can take a long time especially if there are hundreds or thousands of applications to review.

We cover some of the more common JES3 functions from the JECL that will need attention for migration. There will be some items that are not supported by JES2 and other items that might need to be changed to work in JES2. For more information, see Chapter 6 in *ABCs of z/OS System Programming Volume 13*, SG24-7717:

http://www.redbooks.ibm.com/redbooks/pdfs/sg247717.pdf

Also, see the SHARE presentation *z/OS Basics: JES2/JES3 JCL/JECL Differences*, SHARE Anaheim, Aug. 2012:

https://share.confex.com/share/119/webprogram/Session11710.html

See Chapter 4, "JECL and JCL differences" on page 55 of this book for more details about the JECL differences.

5.2.4 JES JECL cards

MAIN versus JOBPARM and JOB statements

One of the more prominent areas to deal with in a migration from JES3 to JES2 centers around the JES3 //*MAIN card. While many parameters might transfer to a JES2 /*JOBPARM or the //JOB statement, some have no corresponding parameter. So it is necessary for the installation to perform a mapping of these parameters to JES2 and JCL. In addition, some items have no equivalent. Some more common parameters that need to be addressed are:

- CLASS: Both JES2 and JES3 //*MAIN support eight-character job classes in z/OS V2R1. In earlier releases, JES2 supported only one character.
- USER: While these appear on the //*MAIN and //JOB statements they have slightly different meanings. The //JOB USER identifies to the system the person submitting the job. Whereas, with the //*MAIN USER identifies the job with the TSO/E user.
- SETUP: This is typically used in MDS for JES3 managed devices. This is done before job execution.
- THWSSEP: This indicates the minimum number of JES3 devices to allocate. This is typically used of TAPE management under JES3.
- ORG: This is for the JES3 default routing. Be advised that some installations might use this parm for multiple purposes.
- PROC/PROCLIB: Compatible in z/OS V2R1. In earlier releases, there was no true JCL equivalent for this. Thus under JES2, you would need to use the JCLLIB to specify PROC data sets for a job to use.
- SYSAFF/SYSTEM: Compatible in z/OS V2R1. In earlier releases, there was no JCL equivalent. For those, you might use a scheduler or use the SCHENV= parm on the //JOB statement. This specifies the name of the WLM environment to associate with a job.

Table 5-2 provides a summary of the parameters.

Table 5-2 Summary of parameters

//*MAIN	/*JOBPARM	//JOB	Usage notes
CLASS	none	CLASS	JES2 and JES3 support 8 char in V2R1. In earlier releases, JES2 supported 1 char
HOLD	none	TYPRUN	Holds job. Really same function
USER	none	USER	These have slightly different meanings
SETUP	none	none	JES3 setup capability
THWSSEP	none	none	JES3 high water mark for setup processing
ORG	none	none	JES3 default routing similar to //OUTPUT DEST

Main device scheduling

If you use this in JES3, you will probably have to set up your ALLOCxx member to allow for the fact that JES2 does not provide equivalent functions. For example, how will the system react when a job looks for an offline device?

Dependent Job Control

This function uses the //*NET cards to define dependencies between jobs in a Dependent Job Control (DJC) network. JES3 sets up a network of dependent jobs and executes them in a specific order. If there needs to be a change in one of the network jobs, they all have to be resubmitted. These jobs are also not eligible to be registered with Automatic Restart Manager (ARM).

There is no corresponding function within JES2; therefore, action will be taken to set these jobs up within your installation's job scheduling package, TWS, CA-7, ESP, with all the triggering, dependency information consolidated in one place. Also, there are products that assist with migration activities. See 10.16.1, "Migration assistance products" on page 142.

Format statement

The //*FORMAT card is used to tell JES3 what SYSOUT data sets are to be printed. There is special processing with this statement for SYSOUT data sets. The PR suffix of the Format statement applies to only SYSOUT data sets printed by JES3. The statement is ignored for data sets set to a TSO user ID or processed by an external writer.

In place for //*FORMAT statements for JES2, simply code an //OUTPUT statement additionally supplying the destination of the output.

Process statement

The //*PROCESS statement is used by JES3 to control how a job processes. The job that specifies this only receives the JES3 processing specified on the //*PROCESS statement. It calls a JES3 Dynamic Support Program (DSP) from the DSP dictionary where that process needs to reside. This is a function that is probably best handled by means of an enterprise scheduler product.

Route statement

The //*ROUTE XEQ statement is used to send input streams to network node for execution on that node. In JES2, //XMIT, /*XMIT, or /*ROUTE XEQ can be used. Under JES2, there might be other job definition statements recognized by the destination node as well. The //XMIT JCL statement transmits a job input stream or other job definition statements. It transmits all records following the XMIT JCL statement to one of the following:

- Two character delimiter specified by the DLM= parameter
- /* if DLM= is not specified
- Or end of input stream

The /*XMIT transmits a job input stream or other job definition statements. This is very similar to the //XMIT statement.

The /*ROUTE XEQ transmits the entire job input stream to execute at the destination. This transmits all records in the job.

For more information about JECL and JCL, refer to Chapter 4, “JECL and JCL differences” on page 55.

5.2.5 Other components for discovery analysis

Additional components might need to be analyzed to create a plan on how they will be retrofitted into the new target JES2 environment. Review items will include but are not limited to:

- Printers: Naming standards are different under JES3 if they are actual JES3 printers.
- Local readers, printers, and punches if any exits.
- FSS printers.
- Review of any RACF rules created especially for JES3. There will then need to be a plan set in place for handling these in JES2.
- Remote Job Entry (RJE).
- RJP in JES3.
- Network Job Entry (NJE).
- Identify your installations scheduler and check its readiness for running under JES2. Are there any exits/usermods that need refactored to make sure they work with JES2?
- Check with other subsystem groups such as CICS or DB2, to validate if there are any ties currently with JES3. There is a CICS interface to JES spool, for example.
- Inventory of your current JES3 Exit and what functions they currently perform. This can be the basis for analysis in figuring out how the exits translate into JES2. Keep in mind some functions might be supported in other ways or does not apply anymore.
- Review exits/usermods with ISV products.
- Inventory of the applications that will be then running on the new target system.

While this is not a complete list, it will help in the discovery analysis assessment. This information can be evaluated in helping to determine if you move to the next phase of planning. This discovery analysis process will help you scope the migration effort if you choose to move forward. Again, a business case will be set forth as the reason why a migration effort is in the best interest of the organization. Use the sources listed in “Related publications” on page 235 for more information about JES2 and other documentation to assist in your evaluation.

5.2.6 The use of a Parallel Sysplex

The development of both JES2 and JES3 took place decades earlier than the IBM Parallel Sysplex. While both work in or with a Parallel Sysplex, JES2 is aligned more with the sysplex natural design of symmetry. JES3 has a global (master) architecture that is not as aligned with the symmetric view that a sysplex has. This becomes more evident in an IBM GDPS® sysplex where the functions are spread across physical sites; in JES2 you do not have to worry where the global is.

5.3 Building a business case

There are different reasons that an installation might want to consider migrating to JES2. Regardless, there will be a sound business case to merit the migration. Among the reasons might be to simplify systems software management. By doing so, there would not be a need to maintain two sets of software components (SMPE data sets, applying APARs, PTFs, and other maintenance, including ISV products) for similar JES functionality. This not only consolidates system products in what seems an ever increasing inventory, but helps in the reduction of technical debt. In this case, it reduces dual maintenance.

The term *technical debt* is a metaphor for the eventual consequences of poor or evolving software architecture or software development. A classic example might be hardcoding functions in programs, which tightly couples them within the application and makes it more expensive and harder to add new functionality. Whereas, if the function was loosely coupled, and certain functions externalized, there would be less work to do and thus less expense in performing the change. Perhaps a user could just make changes themselves via a user interface and the new function is available in significantly less time. This reduces the time to market scenario, which is very important to businesses today. By reducing technical debt, the overall effectiveness and operational excellence of the installation can be improved.

In addition, there might be an associated cost reduction because JES3 is an optional extra cost item with z/OS. This cost varies from customer to customer based on MSUs among other criteria. With TCP NJE as a direct replacement for SNA NJE, this mitigates the use of BDT as well since BDT performed NJE for SNA to a JES3 platform.

Also, there might be other JES3-related products that are not relevant or needed. Alternatively, there might be the need to purchase products to offset functionality that is lost in either JES3 or associated products. In this case, an installation needs to find other means to accomplish similar function. An example might be perhaps using FTP/SFTP or XCOM as alternatives. Evaluation of the functions provided by some of the ISV products would need to be reviewed and validated. This is true since SDSF is now available with JES3 in later releases of z/OS.

Part 2

Details of the migration

In this part, we describe the migration steps and considerations. These chapters address the numerous things that you need to consider if you are interested in replacing JES3 with JES2. An obvious difference is that the JES3 JECL is different from JES2 JECL. Therefore, you need to consider how you would identify and then replace all the JES3 JECL statements with alternatives that would provide the same net effect in a JES2 environment. Appendix A, “Sample JES3 exit to analyze JECL usage” on page 227 includes a JES3 exit routine that you can run to find these statements.

Other less obvious differences include things such as:

- Chargeback considerations because JES2 and JES3 have different ways of accounting for the CPU time used by applications for JES-related functions.
- Functions that are provided by JES3 that are not included in JES2. You will need to identify which ones you use, whether you would still need that function in a JES2 environment, and if so, how would you provide that function.
- User exits and user mods. Because the source code for both JES2 and JES3 is shipped with the product, it is not unusual to find that modifications have been made. You need to identify these changes and determine if they are still required and how you would provide that capability in a JES2 environment.

We begin with a summary, Chapter 6, “Planning for a JES3 to JES2 migration” on page 89. Then, we provide chapters on specific topics and components. To make it easier for you to make an objective assessment of the effort to migrate to JES2, some chapters conclude with a checklist of the items covered in that chapter. We suggest that you complete those checklists as you proceed through the book.

6

Planning for a JES3 to JES2 migration

This chapter focuses on the planning aspects of a migration, including positioning activities that would make the transition easier.

In this chapter, the following topics are discussed:

- Planning
- Choosing a target system
- Migration process
- Functional testing
- System and integration testing
- Implementation

If you are not familiar with the details of JES2 and JES3, the rest of the chapters in this book discuss them so that you can see the differences. You might prefer to read those chapters first to get familiar with the terminology before reading this chapter.

Note: While we cannot cover all possibilities in this chapter, it will serve as a guideline for migration.

6.1 Planning

After you decide to proceed with migrating from JES3 to JES2, you will want to use the information gained from the discovery analysis as input to the planning phase. There are several ramifications to converting to JES2. Some are easily managed while others could be more challenging.

One of the initial activities in setting up a migration process is to form the project team. This is important for the standpoint of having continuity and representation for certain areas so the project has a higher degree of success. This starts with the management commitment as mentioned before. Suggestion for appropriate personnel are as follows:

- Project sponsor: Usually an executive that has overall accountability for the project
- Project manager: Help manage project scope and deliverable
- System programmers
  - z/OS (both JES3 and JES2)
  - ISV system programmer
  - Subsystem technology towers
    - IBM IMS™
    - CICS
- Storage engineer
- Security person (RACF, ACF2, or Top Secret)
- Performance engineer
- Operations staff
  - Remote
  - Systems automation
  - Print shop

At some point, there is the need to align this project with the business areas of your company that will be impacted by this migration. Even before the project starts, it would be a good idea to review the process out with several business areas to gauge their interest and allow them time to make ready for system and integration testing.

In addition, you want to create a checklist of component items that need to be validated once the initial target JES2 system is available. This checklist is used at each phase of the migration as new target JES2 systems are built in each environment such as Sys Prog test, Test/Dev, and Production.

6.1.1 Positioning moves

Where possible, position the installation to use more JES2 compatible facilities such as the following:

- Use // OUTPUT JCL instead of //*FORMAT.
- Plan to move to z/OS V2R1. Earlier versions of JES2 were limited to one-character job classes, while JES3 supported eight-character job classes (as does JES2 in V2R1).
- Use multiple console support (MCS) instead of JES3-managed consoles.

- Try running without JES3 deadline scheduling, device pooling, MDS, Generalized Main Scheduling (GMS), and DJC. Use a scheduler such as IBM Tivoli Workload Scheduler (TWS), CA-7, or ESP, to help manage scheduled workload.
- Try getting rid of as many JES3 mods as possible.
- Use SDSF within JES3 to start to familiarize everyone with how it interacts with output.
- Remove “Tape and Disk set up” high water mark thresholds:
  - Use WLM and job schedulers to manage this instead
  - In the JES3 inish deck, under the STANDARDS section, replace SETUP=THWS with SETUP=NONE. This helps with making the JES3 more JES neutral.
- Convert from JES3 managed volumes to SMS-managed volumes.
- Minimize JES3 mod, exits, and user DSPs.
- Migrate from JES3 DLOG to OPERLOG.
- Migrate SNA/NJE to TCP/IP/NJE.
- Stop all spool browsing and modify spool job control from z/VM. In most cases, this is handled using SDSF in JES2 along with other ISV output processing and archiving products.

Another benefit to do implementing the items mentioned, is it can help in managing your installation to become more JES neutral. This is especially true if it appears that a migration will take a significant amount of time to implement. Perhaps the funding will not be there until the following year for example. In addition, moving to a JES-neutral configuration as much as possible can contribute to operational excellence by simplifying over all systems management.

6.2 Choosing a target system

One of the first items to decide is what the target JES2 system will look like. If the installation is not running any system currently with JES2, you will need to plan for installing it separately. Fortunately, JES2 ships as part of the base z/OS product. Therefore, it will be fairly easy to create.

The process to identify applications slated for the target system is straightforward to figure out. Choose whichever applications are currently running on the JES3 system that is the subject of the migration. At a high level of the planning process, there are a couple of alternative approaches to migration from JES3 to JES2. These alternatives include:

- Initially setting up a secondary JES2 under JES3. This can provide experience without the need to set up a new LPAR.
- Creating a new target LPAR with JES2 and migrating by some criteria such as application or application groups and migrating in a phased approach. This approach is less disruptive for application systems.
- There is also the “all at once” or “Big Bang” approach by which JES3 is quiesced having the new JES2 components, Spool, Init decks built. When the old spool is drained, the new JES2 system is started in its place. Of course, one would have already tested all the applications for the new target system in a JES2 test/dev environment first.

For the purposes of this book, we focus on the phased application system migration.

It is perfectly fine to run multiple JESPLEX within the same Parallel Sysplex. These can be run using two different JESPLEX (JES2 and JES3) or the same JESPLEX (JES2 and JES2) in a single Sysplex.

Choosing the correct mix of IBM and ISV products to replace like kind functionality will be dependent upon goals, resource personnel availability, and cost considerations. Some ideas are as follows:

- Base JES2 with minimal installation exits
- Highly customized JES2 with complex exits and user mods
- Additional IBM products
- Additional ISV products

6.2.1 Storage management

At some point in the planning for migration, you will want to engage the storage management group to verify a few items. At this point, you need to understand the number of jobs per day, the size of the input and output, and have defined an appropriate checkpoint and spool for the replacement configuration. Some items to discuss with the storage group include:

- SPOOL allocations and management.
- Any vendor-specific products that might be interfacing with JES and SMS.
- Deciding on how storage device allocations with be handled such as job tape setup, device sharing, and removing single points of failure.

6.2.2 Education

As part of the migration, it is imperative to provide some level of education for your system programmers, application developers, and operations personnel. IBM has an assortment of educational training available, both in class lectures and web-based content. There are classes on implementation and customizing JES2, including SDSF, and RACF considerations. See the following website for more information:

http://www-304.ibm.com/jct03001c/services/learning/ites.wss/us/en?pageType=page&contentID=a0000048

6.2.3 Vendor management

As part of a migration to JES2, contact the ISV vendors to check if there are any issues, technical or licensing, that need to be addressed. Since JES2 is a base-delivered JES with z/OS, this is normally not a problem. However, it is better to review it early rather than during implementation.

Check with the system automation and scheduler software personnel in your installation for any products that might be JES3-specific. There could be some potential cost savings. Also, check with any products that interface with storage, either tape or DASD.

Refer to Chapter 9, “Operational considerations” on page 119 for more ISV information.

6.3 Migration process

This section of the chapter describes the migration process and focuses on reaching the goal of a running workload in the new JES2 system. This section covers the following topics:

- Setting up JES2
- Functional testing
- System and integration testing
- Implementation

6.3.1 JES2 setup

The recommendation would be to move from JES3 to JES2 in as nondisruptive manner as possible. This entails setting up a new target JES2 system and migrating applications in a controlled manner.

The migration would be slightly easier if your applications support data sharing because that lets you move jobs from one LPAR to another on a job-by-job basis.

IBM does not support running JES2 and JES3 in the same LPAR (though it has worked with no issues). It *is* supported to run JES2 and JES3 in the same sysplex. To move spool files from JES3 to JES2 or vice versa, the only option is NJE; spool offload files are not transportable across JESs.

As mentioned previously, it depends on what your installation requirements are regarding how to install and test the new target JES2 environment. Regardless of the method, when the new JES2 image is initialized, the way to share workload or functions between the JES3 and JES2 images are by means of NJE. Obviously, you will want to do this in a System Programmer “sandbox” environment first.

Based on the output from the discovery analysis phase, map out and install JES2 exits where appropriate. In addition, you will want to review and set up the JES2 Initialization deck. You might want to also review the CBT tape for mods. Another thing to review in the JES2 setup are system management changes from JES3 to JES2. Table 6-1 provides a model table for you to track your review plan.

Table 6-1 Sample review plan

Functional areas for review	Resp. group	Time Alloc.	Completed
Job management changes and testing			
Configuration and definition			
System initialization			
Security changes and testing			
Reliability and serviceability			
Exits and mods (might attach separate table)			
Automation tools configuration and testing			
Initialization deck changes			

To ensure timely and consistent conversion of your JES3 jobs to ones that run cleanly on JES2, you need to develop a program using the output from the discovery analysis phase and map the functions and components that have been identified for migration.

A useful document would be *Introduction for JES2 System Programmers*. See this SHARE presentation:

http://www-03.ibm.com/support/techdocs/atsmastr.nsf/WebIndex/TD102498

Below is a list of pre-work items to complete during JES2 setup:

- Create new JES2 SPOOL packs.
- Stage JES2 configuration with mapped SPOOL, NJE, and OUTPUT classes.
- Stage SPOOL Management regarding initiators, example AM/PM initiators, if applicable.
- Rework all ISV products that have JES3 SPOOL offsets and JES3 specific parameters. Examples are CA-View, XPAF (Xerox), and Mobius. Also, SDSF if you are not (yet) on z/OS V2R1.
- Stage new WLM policies with only single character job classes (for z/OS releases earlier than V2R1).
- Stage RACF, security, rules for JES2 congruent with JES3.
- If your installation is not using an ISV product to assist in DJC, you will need to verify that the job mapping has been complete within your installation's scheduler such as TWS or CA-7/ESP.
  - Include NJE route conversion and other needed JECLs identified from the Discovery Analysis phase
- Review how external JCL/system cards/operation prod control will be handled.
- Communicate "Dataset not found" behavior change for jobs. This has to do with the difference between the way JES3 and JES2 handle data set availability. JES3 performs checking at job initiation time, whereas JES2 performs this on a STEP boundary.

Note: For installations that use an output archiving product such as CA-View, some thought around handling output and viewing will need to be addressed. For example, CA-View(SAR) parses for//*MAIN and then looks for USER= relative on the same line to output the job to a common SYSID, like a TSO user ID. Under JES2, that parsing will still occur but no action taken since there are no //*MAIN cards. This is another example of why to contact ISV vendors for special handling of existing functions and how they might work under JES2.

6.4 Functional testing

In bringing up the new target JES2 system, you might want to start things manually to gain insight to how the new system works. The next thing that you will most likely want to do is to set up automation of certain procedures and processes. When the new JES2 image is up and running, you will want to familiarize yourself as well as others in how jobs and subsystems interact with it.

While in this mode of testing, it would be prudent to check out how the base components are functioning. Some of the things that will need to be verified are listed here:

- Jobs running in the appropriate classes
  - Are WLM policies working as expected?
- Validate scheduler functionality
- Validate any output/archiving product functions

- Test tape handling and execution
- Verify automation is functioning correctly
- NJE validation
- Verify external feeds are working (might not be able to test until next level)
- Validate that GRS/MIM is functioning correctly
- Test TSO/E interface along with SDSF
- SPOOL off load
- If possible, making a dry run of this migration in a Disaster Recovery test environment would be beneficial

Table 6-2 shows a body of testing functional areas. Some of these might not be able to be tested fully until there are applications brought into the target system.

Table 6-2 Sample test plan

Functional testing areas	Resp. group	Time alloc.	Completed
Job management and tracking (schedulers)			
Validate resource management tools			
Accounting software			
Security functioning as expected (RACF, ACF2, and so on)			
Automation tool validation			
ISV products			
TSO/E usage			
SDSF functions			
Application group “A”			
Application group “B”			

6.5 System and integration testing

The next stage of migration is to move and create the similar JES2 system in the Test/Development environment. In this environment, you will then want to stage applications in for testing. This will be the application areas’ first opportunity to do their functional testing.

In this phase of the migration, there are some other job behavior and processing aspects that will need to be validated. These items include:

- Handling of GDGs
- Data set locking
- Data set enqueuing
- Job C/I setup and execution
- Using cataloged procedures
- Job Abend and back out processing
- Validate RACF or Security rules are working
- Test SPOOL off load including fall-back scenarios or make this a clean-up activity

In addition, do some performance/stress testing of the new JES2 environment. Table 6-3 shows some example items.

Table 6-3 Performance and stress test items

Task	Resp. group	Time Alloc.	Completed
Test Batch and TSO			
Stress Test Batch and TSO			
Online testing and Load test			
Test Output devices, RJE, NJE			
Weekend simulation cut over			
Review results from simulated cut over			

6.6 Implementation

In preparation for production implementation, there are some areas to keep in mind. This depends on the needs of your installation. Here are a few ideas:

- Make necessary configuration changes for the new system
- Migrate application systems or group of systems based on a predetermined schedule. Make a list of what has to stay together and what can be moved independently to the new JES2 subsystem:
  - Input devices
  - Execution environment
  - Output processing
  - Printing
  - TSO/E
  - RJE
  - NJE

Switching local devices, RJE workstations, and connections to NJE is fairly straightforward and can be accomplished without disruption to the existing JES3 system. However, note that for job execution, resources that were managed by JES3 will now have to be managed by z/OS for functions like GMS and MDS. Review how your installation will be using WLM regarding this. Table 6-4 shows migration cutover planning charts.

Table 6-4 Sample cutover plan

Production cutover	Resp. group	Time Alloc.	Completed
Idle current system for conversion and archive JES3 SPOOL			
Shut down JES3 system LPARs			
Bring up JES2 system LPARs			
Have WLM in place to handle 36 one-character classes or equivalent WLM scheme in place			

Production cutover	**Resp. group**	**Time Alloc.**	**Completed**
IVP Infrastructure			
Dry-run test of spool transfer and fail back			
Move over JES devices, TP lines, apps, users			
Last-minute items from system integration testing			
Notify users and operation of schedule for implementation			
Cut over (first run) and IVP			
Validate applications check out			
Make go/no-go decision			
Fail back if necessary			
Clean up activities			

Note: Placing //JOB cards in your scheduling package is beneficial since most of the JCL is done at that point. Any external JES3 JCL would be treated as having no job-specific class due to the JCL being treated as //* comment. Therefore, external jobs would then run as a default CLASS.

Each of the rest of the chapters contains details about migrating a specific component from JES3 to JES2.

7

JES procs and initialization decks

This chapter describes the differences between the JES3 and JES2 initialization statements. As you would expect, many of the concepts are similar. However, the way that they are defined to the respective JES is very likely to be different. Also, there are some functions or resources that exist in JES3 that do not exist in JES2. Studying your JES3 initialization statements and mapping them to their JES2 equivalent (where possible) provides you with a valuable insight to how many changes you will need to make if you want to migrate from JES3 to JES2.

While they are a lot simpler than the initialization statements, the started procedures for JES2 and JES3 are also described.

7.1 Introduction

A JES3 JESplex consists of a Global system and zero or more Local systems. The Global system is the first system to perform a cold start or warm start following a JESplex-wide IPL. During a cold start or warm start, the Global system reads the initialization deck. When initialization of the Global is complete, any other systems in the JESplex might start JES3 as locals. JES3 on a Local communicates with the Global through XCF. A Local system never accesses the JES3 initialization deck. It obtains the information that it needs about the configuration by reading the checkpoint data set. It uses the information in the checkpoint to initiate communication with the Global system.

The JES3 Global function can be moved to a Local during either a planned or unplanned outage by performing a Dynamic System Interchange (DSI). The initialization deck is not read during a DSI. Instead, JES3 uses its checkpoint data set to bring the JESplex back to normal function.

A JES2 JESplex consists of one or more systems. If there are multiple systems, they have a peer-to-peer relationship. All JES2 systems read the initialization deck during all starts. The amount of initialization deck processing that occurs during a JES2 start depends on the start type. See Table 7-1 for more information.

JES2 does not have an equivalent function to DSI because each system in the JESplex can perform all JES2-related functions, so there is no concept of a master system.

7.2 Types of start

Both JES2 and JES3 have several varieties of startup. While a cold start is basically the same for both, the various warm start and hot starts have differences in processing of the initialization deck. This information was provided briefly in 2.2.3, “JES startup processing” on page 36, but we provide it here in Table 7-1 in the context of discussing the initialization statement processing.

Table 7-1 JES start types

Type of start	JES3[a]	JES2[b]
Hot start	Performed: ▶ After orderly shutdown ▶ After a JES3 failure ▶ After an IPL The initialization deck is not read during a normal hot start.	JES2 is being restarted after a $PJES2,ABEND or if JES2 abended. Requires that the JES2 subsystem being started was already active on that system. If the system was IPLed, a warm start will be performed.
Hot start with analysis	Same as a hot start except that JES3 performs further job validation, allowing the operator to delete invalid jobs.	No JES2 equivalent.
Hot start with refresh	Reads initialization deck, but not all statements are processed[c]. A hot start with refresh lets you change many JES3 parameters without the disruption of a warm start.	Single member warm start.
Hot start with refresh and analysis	Combines the hot start with analysis and refresh.	No JES2 equivalent.

Type of start	JES3[a]	JES2[b]
Warm start	Reads initialization deck All statements are processed[d] A JES3 warm start requires that all systems in the JESplex are IPLed.	All-member warm start JESplex restart of JES2 required.
Warm start with analysis	Reads initialization deck JESplex IPL required Removes and logs jobs with invalid control information.	No JES2 equivalent.
Warm start to replace a spool data set	Reads initialization deck Allows replacement of a spool data set JESplex IPL required[e].	No JES2 equivalent JES2 supports dynamic addition and removal of Spool data sets.
Warm start with analysis to replace a spool data set	Reads initialization deck Allows replacement of a spool data set. JESplex IPL required Removes and logs jobs with invalid control information.	No JES2 equivalent.
Cold start	All job data is lost Reads initialization deck All statements are processed JESplex IPL required.	All job data is lost JESplex IPL required.
Local start	Performed only on JES3 locals.	Quick start.

a. See *z/OS JES3 Initialization and Tuning Reference*, SA22-7550 to determine which type of start is needed for each parameter that is changed.

b. See *z/OS JES2 Initialization and Tuning Reference*, SA22-7533 to determine which type of start is needed for each parameter that is changed.

c. See *z/OS JES3 Initialization and Tuning Guide*, SA22-7549 Table 3 to determine which statements can be changed during a hot start with refresh.

d. The following parameters cannot change during any type of warm start:
BUFSIZE on the BUFFER statement
SE on the OPTIONS statement
STT or STTL on the TRACK statement for existing spool data sets
The sequence of the MAINPROC statements

e. z/OS V1R13 added the ability to add a JES3 spool data set dynamically using the *MODIFY CONFIG,ADD= command. V2R1 added this for JES2.

7.3 Initialization statements

Table 7-2 lists all current JES3 initialization statements and their JES2 equivalents, if they exist. Not every JES3 statement has a JES2 equivalent, and some JES3 parameters might require updates to several JES2 statements.

Note: A sample JES2 HASPARM template is supplied in SYS1.SHASSAMP member HSAIPARM.

Table 7-2 JES3 to JES2 Initialization statement mapping

JES3 statement	JES2 statement
ACCOUNT	No JES2 equivalent; optional for JES3
BADTRACK	BADTRACK

JES3 statement	JES2 statement
BUFFER	BUFDEF; SPOOLDEF
CIPARM	JOBCLASS
CLASS	JOBCLASS
COMMDEFN	LOGON(n)
COMMENT (*)	/* followed by */
COMPACT	COMPACT
CONSOLE	No JES2 equivalent[a]
CONSTD	CONDEF
DEADLINE	No JES2 equivalent[b]
DESTDEF	No JES2 equivalent
DEVICE	MEMBER statements for processors; LINE, PRT, PUN,RDR, OFFLOAD, RMT statements for devices
DYNALDSN	No JES2 equivalent
DYNALLOC	PROCLIB statements for proclibs only
ENDINISH	No JES2 equivalent
ENDJSAM	No JES2 equivalent
FORMAT	SPOOLDEF
FSSDEF	FSS
GROUP	INIT
HWSNAME	No JES2 equivalent[c]
INCLUDE	INCLUDE
INTDEBUG	DEBUG
MAINPROC	MEMBER
MSGROUTE	No JES2 equivalent
NETSERV	NETSERV
NJECONS	No JES2 equivalent
NJERMT	NODE
OPTIONS	OUTDEF
OUTSERV	OUTDEF, PRINTDEF
RESCTLBK	No JES2 equivalent
RESDSN	No JES2 equivalent
RJPLINE	LINE
RJPTERM	RMT
RJPWS	RMT

JES3 statement	JES2 statement
SELECT	INIT; INITDEF
SETACC	No JES2 equivalent
SETNAME	No JES2 equivalent
SETPARAM	No JES2 equivalent
SETRES	SOCKET
SOCKTER	SOCKET
SPART	FENCE parameter for SPOOLDEF
STANDARDS	JOBCLASS, ESTBYTE, ESTIME, ESTLNCNT, ESTPAGE, ESTPUN
SYSID	No JES2 equivalent[d]
SYSOUT	OUTCLASS
TRACK	SPOOLDEF

a. JES3 supports consoles for NJE and RJP only
b. JES2 does not support deadline scheduling
c. JES3 uses HWSNAME, RESDSN, SETACC, SETNAMES, and SETPARAM only when Main device Scheduling (MDS) is in use
d. JES3 uses SYSID only to communicate with BDT when SNA/NJE is in use

We suggest that you sit down with your JES3 initialization deck and a copy of *z/OS JES3 Initialization and Tuning Reference*, SA22-7550 and *z/OS JES2 Initialization and Tuning Reference*, SA22-7533 and identify how you will replace each JES3 statement when you migrate to JES2. It is not necessary to go to the level of detail of identifying spool data set names at this point (for example). However, this is a good time to identify any JES3 functions that you might be using that do not have a JES2 equivalent. An obvious example would be the HWSNAME statement that you use to control High Water Mark processing.

7.3.1 Dynamic changes

There are six ways to change JES2 initialization parms. From the easiest to the most disruptive, they are:

1. Operator command
2. Hot start
3. Quick start
4. Single-member warm start
5. All-member warm start
6. Cold start

Tables 9 - 83 in *z/OS JES2 Initialization and Tuning Reference*, SA22-7533 list the minimum action needed to modify each parameter in the initialization deck. The values of certain parameters can be *increased* with an operator command but require a cold start to decrease. If a parameter is increased dynamically using a JES2 command, the system programmer must update the initialization deck to match the new value. Refer to Chapter 3 in *z/OS JES2 Initialization and Tuning Reference*, SA22-7533 for the actions needed to decrease the value of a parameter.

Table 3 (“Initialization Statement Summary A through L”) and Table 4 (Initialization Statement Summary M through Z) in *z/OS JES3 Initialization and Tuning Reference*, SA22-7550 provide a list of all the JES3 initialization deck statements and the type of JES3 start that is required to change the statement.

7.3.2 Verifying the JES initialization deck

JES3 has an initialization deck checker program IATUTIS that allows the system programmer to verify that the deck has no errors before a scheduled restart. Installations that still have disk or tape DEVICE statements might use option 2.4 in the HCD ISPF panels to create members that will then be pointed to by the STG1CODE DD statement in the checker job. The initialization deck checker will then verify that the DEVICE statements agree with the devices in the HCD. Example 7-1 shows sample JCL for initialization deck checking.

Example 7-1 Sample JCL for IATUTIS initialization deck checker

```
//INITCHK    JOB 'ACCTINFO','NAME',MSGLEVEL=(1,1),
//                MSGCLASS=R,...
//IATUTIS    EXEC PGM=IATUTIS,PARM='P=1F1R'
//STEPLIB    DD DSN=SYS1.SIATLIB,DISP=SHR
//JESABEND   DD DUMMY
//JES3IN     DD DSN=INIT.PARMLIB(JES3IN00),DISP=SHR
//JES3OUT    DD SYSOUT=*
//STG1CODE   DD DSN=INSTALL.JES3,DISP=SHR
//IATPLBST   DD DSN=SYS1.PROCLIB,DISP=SHR
//
```

JES2 does not provide an initialization deck checker. However, you can set up a secondary JES2 (see *z/OS JES2 Initialization and Tuning Guide*, SA22-7532) and use that to validate the JES2 initialization statements if wanted. Errors in the initialization deck cause a $HASP469 message to be issued. The operator can reply with the correct parm, bypass the error, or terminate JES2.

7.4 JES procedures

The JES3 proc points to a single member of a PDS, which can be overridden by replying M=xx to the IAT3012 message. INCLUDE statements can be used in the initialization deck to separate groups of statements into separate PDS members if wanted.

The JES2 proc can also point to a single PDS member, or several data sets or members can be concatenated if you want to break up your JES2 parms into different members. For example, you might have one member that is common across the entire Multi-Access SPOOL (MAS), and a set of members that contain information that is specific to each system. Configurations with many NJE nodes often have a member set aside for just the NODE and CONNECT statements.

It is a common practice to code variables in the JES2 proc to allow overrides if there is a problem with the initialization deck or a PROCLIB because all the PROCLIBs for JES2 were traditionally defined in the JES2 PROC. JES3, alternatively, tended to have simpler procs because its PROCLIB concatenations were defined using DYNALLOC statements in the JES3 initialization deck. Refer to 1.2.2, “JES2 PROC JCL” on page 9 for a description of the current options for defining its PROCLIB and other data sets in the JES2 proc.

7.4.1 The JES3 procedure

Example 7-2 shows a JES3 procedure with all statements hardcoded.

Example 7-2 Typical JES3 procedure

```
//IEFPROC EXEC PGM=IATINTK,TIME=1440,PERFORM=255
//STEPLIB DD   DISP=SHR,DSN=SYS1.SIATLIB
//CHKPNT  DD   DISP=SHR,DSN=SYS1.JES3.CHECKPT
//CHKPNT2 DD   DISP=SHR,DSN=SYS1.JES3.CHECKPT2
//JES3IN  DD   DISP=SHR,DSN=SYS1.JES3.PARMLIB(JES3IN00)
```

The JES3IN DD statement on the JES3 procedure points to a single member of a PDS. The operator can select a different member by replying M=xx to the IAT3012 message. JES3 supports INCLUDE statements so the system programmer can break up the initialization deck into multiple members. This can be used, for example, to isolate statements that change frequently, such as DEVICE statements for printers or NJERMT statements, from the more critical parts of the deck.

JES3 supports the use of system symbols in its initialization statement, as does JES2. However, in JES3, the only system that reads the initialization statements is the Global, compared to JES2 where every system reads the initialization members. As a result, the use of system symbols in the initialization deck is probably less likely in a JES3 environment than in a JES2 one.

Other DD statements that can be included in the JES3 procedure are JES3JCT, JES3OUT, JES3SNAP, JESABEND, IATPLBxx, and JES3DRDS. However, these are typically defined in DYNALLOC statements within the initialization deck rather than in the JES3 proc. Using DYNALLOC allows the system to bypass missing data sets. If a data set referenced in the JES3 proc cannot be opened JES3 will fail with a JCL error.

Example 7-3 shows a series of DYNALLOC statements for PROCLIBs in the JES3 initialization deck.

Example 7-3 Sample JES3 DYNALLOC statements to define PROCLIBs

```
* DYNALLOCS FOR PROCLIBS
* PROCLIBS ARE ACCESSED THROUGH THE CATALOG UNLESS UNIT AND VOLSER ARE CODED
* PROCLIB ST FOR STANDARD JOBS
DYNALLOC,DDN=IATPLBST,DSN=SYS1.SY1.PROCLIB
DYNALLOC,DDN=IATPLBST,DSN=SYS1.PROCLIB
DYNALLOC,DDN=IATPLBST,DSN=SYS1.IBM.PROCLIB
* PROCLIB 01 FOR TSO LOGONS
DYNALLOC,DDN=IATPLB01,DSN=SYS1.LOGON.PROCLIB
DYNALLOC,DDN=IATPLB01,DSN=SYS1.PROCLIB
DYNALLOC,DDN=IATPLB01,DSN=SYS1.IBM.PROCLIB
* PROCLIB 02 FOR STARTED TASKS
DYNALLOC,DDN=IATPLB02,DSN=SYS1.STARTED.PROCLIB
* PROCLIB FI FOR FINANCIAL JOBS
DYNALLOC,DDN=IATPLBFI,DSN=USER.FINANCE.PROCLIB,UNIT=3390,VOLSER=FIN001
```

The PROCLIB concatenations can be specified in the STANDARDS statement. INTPROC=ST specifies the standard PROCLIB concatenation for jobs entered using the internal reader, STCPROC=02 specifies the concatenation for started task jobs, and TSOPROC=01 specifies the concatenation for TSO logons. Using the definitions shown in

Example 7-3 on page 105, jobs submitted by the finance users can use their dedicated PROCLIB by specifying PROC=FI on the //*MAIN JECL statement in their jobs.

7.4.2 Typical JES2 procedure

Example 7-4 shows a JES2 proc with some PROCLIBs defined using system symbols and some hardcoded.

Example 7-4 Typical JES2 procedure

```
//JES2    PROC M=JES2IN00,M1=JES2IN&SYSCLONE,
//             PL=SYS1.JES2.PARMLIB,
//             PROC00='SYS1.PROCLIB',PROC01='SYS1.IBM.PROCLIB'
//IEFPROC EXEC PGM=HASJES20,TIME=1440,DPRTY=(15,15)
//HASPLIST  DD DDNAME=IEFRDER
//HASPPARM  DD DSN=&PL(&M),DISP=SHR
//          DD DSN=&PL(&M1),DISP=SHR
//PROC00    DD DSN=SYS1.&SYSNAME..PROCLIB,DISP=SHR
//          DD DSN=&PROC00,DISP=SHR
//          DD DSN=&PROC01,DISP=SHR
//PROC01    DD DSN=SYS1.LOGON.PROCLIB,DISP=SHR
//          DD DSN=&PROC00,DISP=SHR
//          DD DSN=&PROC01,DISP=SHR
//PROC02 DD DSN=SYS1.STARTED.PROCLIB,DISP=SHR
//PROC03 DD DSN=USER.FINANCE.PROCLIB,DISP=SHR,UNIT=3390,VOL=SER=FIN001
```

JES2 allows you to concatenate two or more members on the HASPPARM DD statement. You can also have INCLUDE statements in the initialization deck data sets. In Example 7-4, member JES2IN00 contains common statements and member JES2INxx contains member-specific statements. Member-specific statements typically include devices such as channel-attached printers that can only be physically attached to one system.

System programmers must ensure all the data sets referenced in the JES2 proc are available or JES2 will fail with a JCL error.

In Example 7-4, the default PROCLIB that a job will use is determined by its jobclass, based on the JOBCLASS statements in the JES2 initialization deck. There can be up to 99 PROCLIB concatenations. JOBCLASS(TSU) uses PROCLIBs concatenation 01 and JOBCLASS(STC) uses concatenation 02. Jobclasses A - Z and 0 - 9 would normally use concatenation 00. If a jobclass was dedicated to finance jobs, the corresponding JOBCLASS statement could specify PROCLIB=03, otherwise finance customers could specify PROCLIB=03 on the /*JOBPARM JECL statement in their jobs.

The recommended method is to dynamically define proclibs using PROCLIB statements in the initialization deck. Using dynamic proclibs allows JES2 to start even if a PROCLIB is missing or was mis-defined in the JES2 initialization statements. If a PROCLIB is not found during JES2 startup, a message can be issued giving the operator the opportunity to correct or bypass the error.

Example 7-5 shows PROCLIB definitions in the JES2 proc.

Example 7-5 Static JES2 PROCLIBs

```
//PROC00    DD DSN=SYS1.&SYSNAME..PROCLIB,DISP=SHR
//          DD DSN=SYS1.PROCLIB,DISP=SHR
//          DD DSN=SYS1.IBM.PROCLIB,DISP=SHR
```

```
//PROC01    DD DSN=SYS1.LOGON.PROCLIB,DISP=SHR
//          DD DSN=SYS1.PROCLIB,DISP=SHR
//          DD DSN=SYS1.IBM.PROCLIB,DISP=SHR
//PROC02    DD DSN=SYS1.STARTED.PROCLIB,DISP=SHR
//PROC03    DD DSN=USER.FIN.PROCLIB,DISP=SHR,UNIT=3390,VOL=SER=FIN001
```

Example 7-6 shows the same set of PROCLIBs being defined dynamic PROCLIB statements.

Example 7-6 Dynamic JES2 PROCLIBs

```
PROCLIB(PROC00) DD(1)=(DSN=SYS1.&SYSNAME..PROCLIB
                DD(2)=(DSN=SYS1.PROCLIB)
                DD(2)=(DSN=SYS1.IBM.PROCLIB)
PROCLIB(PROC01) DD(1)=(DSN=SYS1.LOGON.PROCLIB
                DD(2)=(DSN=SYS1.PROCLIB)
                DD(2)=(DSN=SYS1.IBM.PROCLIB)
PROCLIB(PROC02) DD(1)=(DSN=SYS1.STARTED.PROCLIB
PROCLIB(PROC04) DD(1)=(DSN=SYS1.FIN.PROCLIB
```

Dynamic proclibs can be added, modified, or removed using the $ADD PROCLIB, $T PROCLIB, or $DEL PROCLIB commands.

7.4.3 Other procedures

The following JES address spaces all start automatically at IPL time. No setup is needed. They shut down automatically when JES2 or JES3 terminates:

JESXCF	Common to both JESs
JES2MON	Started automatically at JES2 initialization
JES2AUX	Auxiliary address space for JES2
JES3AUX	Auxiliary address space for JES3
JES3DLOG	Hardcopy log for JES3

If a JESplex uses TCP/IP to drive NJE connections, one or more network server address spaces are started. A JES2 network server is named *jesx*Snnn where *jesx* is the name of the owning JES2 address space and nnn is the subscript on the NETSERV(nnn) statement. For example, the first network server on a subsystem names JES2 would be JES2S001. A JES3 network server can have any name although a common name is JES3S001. A network server must be defined to the security product as a started task. IBM recommends using a common name pattern for network servers so that only one security profile is needed.

In addition, JES3 can have one or more CIFSS address spaces defined to offload some of the converter/interpreter workload. JES2 can process CI on any member of the JESplex and does not move this processing to a separate address space. In z/OS V2R1, JES2 likewise contains this function.

Both JESs can have one or more writer functional subsystems (FSSs) defined. These work the same way under both JES3 and JES2. An FSS can drive multiple printers. When a writer is called (JES3) or started (JES2) the system checks to see if the appropriate FSS is running. If it is not, the system starts it automatically. An FSS cannot be started with an operator command. The FSS terminates automatically when the last printer it is driving is shut down.

7.5 Automation considerations

JES3 startup issues WTORs to determine the start type, and select the initialization deck for a hot start with refresh, warm starts, or cold starts. JES2 can be started without operator interaction by specifying PARM='WARM,NOREQ' on the JES2 START command. The NOREQ parm relieves the operator from having to input $S to start processing. This is the equivalent of coding PARM=NOREQ on the EXEC statement of the JES3 procedure to relieve the operator of having to input *S JSS.

Any automation routines that deal with JES3 startup, shutdown, failures, or restarts will need to be changed to address JES2 messages and commands.

Many automation routines key on JES3 initialization message IAT3100. These must be changed to JES2 message $HASP492.

8 User exits

This chapter contains an approach to spell out the similarities and the differences between the JES3 user exits you might be running and how they could be implemented into JES2 exits.

8.1 JES3 user exits

Table on page 113 contains a list of all the JES3-provided user exits and a brief description of the purpose of the exit.

The first step is to identify if you are using the exit or not. Some exits might be written by your installation, others might be provided by products that need to communicate with or monitor JES3: a batch job scheduler, for example.

All JES3 user exits are called *IATUXnn*. You do not have to do anything in JES3 to tell it that you want to use a given user exit. You simply assemble and link edit the user exit and place the module in a load library that is accessible to JES3. It is strongly recommended that you use SMP/E to manage your JES3 user exits because the macros that they use can be changed by JES3 service. If you *do* use SMP/E, SMP/E can be used to determine which exits you are using.

If you are *not* using SMP/E to manage the exits, use the **`*I_X_M=IATUXnn`** command to display information about every JES3 exit. The response to this command, as shown in Example 8-1, reports the load module size, the link edit date, and the number of times the load module has been used. This information helps you determine if the exit is in use or not. Note that the IBM provided exit stubs are not all the same size. Some are x'40' bytes, some are x'80' bytes, and some are other sizes. The IBM provided stubs reside in SYS1.SIATLIB and SYS1.SIATLPA.

Example 8-1 Obtaining information about JES3 user exit load modules

```
*I X M=IATUX03
IAT8476 IATUX03  - RES=00005 USE=00001 LOADS=00001 EP=1A7C1FA0
DELETE=Y
IAT8476 REL=HJS7780 DATE=04/04/11 TIME=14:34 APAR=NONE PTF=1.13.0
IAT8476 SIZE=00000040 AMODE=31 RMODE=ANY
```

Before you do any work to create a JES2 version of an existing JES3 exit, we strongly recommend reviewing the function of the exit to determine if you still require that function or not. Also, verify that the function provided by the exit would still be required following the migration to JES2 - might be JES2 or some other product provides that function without requiring an exit? Test and updating user exits is one of the most time-consuming activities during system upgrades, so the more exits you can eliminate, the better.

Using Table 8-1, identify the JES2 exit points that correlate to the exit functions that you will still require after the migration.

Table 8-1 JES3 user exits and corresponding JES2 exits

JES3 exit	Description	JES2 exit
IATUX03	Examine/Modify Converter/Interpreter Text Created from JCL	
IATUX04	Examine the Job Information	
IATUX05	Examine the Step Information	
IATUX06	Examine the DD Statement Information	

JES3 exit	Description	JES2 exit
IATUX07	Examine/Substitute Unit Type and Volume Serial Information	
IATUX08	Examine Setup Information	
IATUX09	Examine Final Job Status, JST, and JVT	
IATUX10	Generate a Message	
IATUX11	Inhibit Printing of the LOCATE Request/Response	
IATUX14	Validate Fields in Spool Control Blocks During a JES3 Restart	
IATUX15	Scan an Initialization Statement	
IATUX17	Define Set of Scheduler Elements	
IATUX18	Command Modification and Authority Validation	
IATUX19	Examine/Modify Temporary OSE	
IATUX20	Create and Write Job Headers for Job Output	
IATUX21	Create and Write Data Set Headers for Output Data Sets	
IATUX22	Examine/Alter the Forms Alignment	
IATUX23	Create and Write Job Trailers for Job Output	
IATUX24	Examine the Net Id and the Devices Requested	
IATUX25	Examine/Modify Volume Serial Number	
IATUX26	Examine MVS Scheduler Control Blocks	
IATUX27	Examine/Alter the JDAB, JCT, and JMR	
IATUX28	Examine the JOB JCL Statement	
IATUX29	Examine the Accounting Information	
IATUX30	Examine Authority Level for TSO/E Terminal Commands	

JES3 exit	Description	JES2 exit
IATUX32	Override the DYNALDSN Initialization Statement	
IATUX33	Modify JCL EXEC Statement and JES3 Control Statement	
IATUX34	Modify JCL DD Statement	
IATUX35	Validity Check Network Commands	
IATUX36	Collect Accounting Information	
IATUX37	Modify the JES3 Networking Data Set Header for Local Execution	
IATUX38	Change the SYSOUT Class and Destination for Networking Data Sets	
IATUX39	Modify the Data Set Header for a SYSOUT Data Set	
IATUX40	Modify Job Header for a Network Stream Containing a Job	
IATUX41	Determine the Disposition of a Job that Exceeds the Job JCL Limit	
IATUX43	Modify Job Header for a Network Stream Containing SYSOUT Data	
IATUX44	Modify JCL Statements	
IATUX45	Change Job Information for Data Sets Processed by an Output Writer FSS	
IATUX46	Select Processors Eligible for C/I Processing	
IATUX48	Override Operator Modification of Output Data Sets	
IATUX49	Override the Address Space Selected for C/I Processing	
IATUX50	JES3 Unknown BSID Modifier Exit	
IATUX57	Select a Single WTO Routing Code for JES3	
IATUX58	Modify Security Information Before JES3 Security Processing	

JES3 exit	Description	JES2 exit
IATUX59	Modify Security Information After JES3 Security Processing	
IATUX60	Determine Action to Take When a TSO User Is Unable to Receive a Data Set	
IATUX61	Cancel Jobs Going on the MDS Error Queue	
IATUX62	Verify a Mount Request	
IATUX63	Provide SSI Subsystem Installation String Information	
IATUX66	Determine Transmission Priority for a SNA/NJE Stream	
IATUX67	Determine Action When Remote Data Set Is Rejected by RACF	
IATUX68	Modify Local NJE Job Trailers	
IATUX69	Determine If a Message is to be Sent to the JES3 Global Address Space	
IATUX70	Perform Additional Message Processing	
IATUX71	Modify a Tape Request Setup Message	
IATUX72	Examine/Modify a Temporary OSE or an OSE Moved to Writer Queue	

If you are trying to map JES3 exits into JES2 exits, you must consider how JES2 is handling the input stream of JCL and where its exits are located. This is what Figure 8-1 on page 114 is depicting on a global scale.

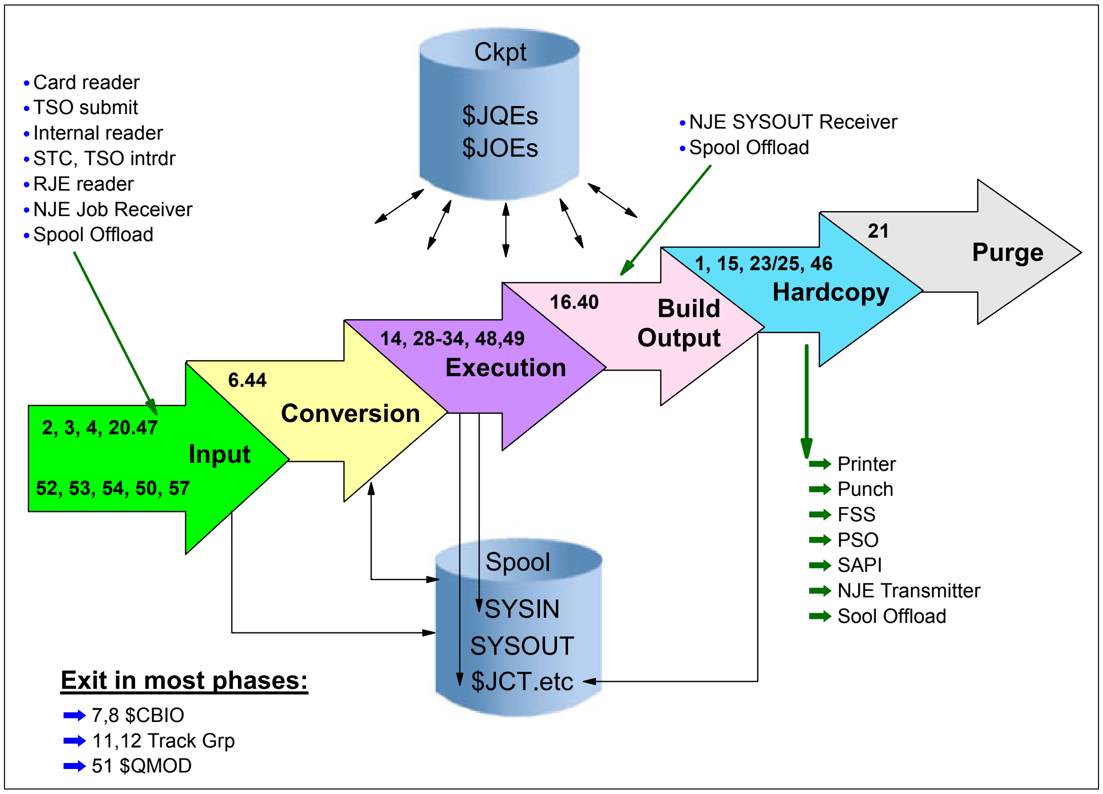


Figure 8-1 JES2 exits: an overall view

You can also list the existing JES2 exits. This is depicted in Table 8-2 on page 115.

Table 8-2 List of JES2 exits

Exit	Environment	Description
0	JES2	Initialization
1	JES2	Separator page
2	JES2	Job card
3	JES2	Accounting
4	JES2	JECL card
5	JES2	Command
6	SUBTASK	Converter
7	JES2	$CBIO
8	USER	$CBIO
9	USER	Excession
10	JES2	$WTO
11	JES2	$TRACK
12	USER	$STRAK
13	USER	Netmail (obsolete)
14	JES2	$QGET
15	JES2	DS Separator
16	JES2	Notify
17	JES2	BSC Signon
18	JES2	SNA Logon
19	JES2	Init statement
20	JES2	End of Input
21	JES2	SMF
22	JES2	Cancel/Status
23	FSS	JSPA Separator
24	JES2	Post-initialization
25	FSS	JCT I/O
26	JES2	Termination
27	JES2	PCE Attach
28	USER	Job Termination
29	USER	End of Memory
30	USER	OPEN
31	USER	Allocation
32	USER	Job Select

Table 8-3 Table 8-2 (continued)

Exit	Environment	Description
33	USER	CLOSE
34	USER	Unallocation
35	USER	End of Task
36	USER	Pre-SAF
37	USER	Post-SAF
38	JES2	TSO Receive SAF
39	JES2	NSR SAF
40	JES2	Modify Sysout
41	USER	Generic Grouping
42	USER	Notify SSI
43	USER	APPC
44	JES2	Converter
45	USER	Pre-SJF
46	JES2	NJE Hdr Xmit
47	JES2	NJE Hdr Recv
48	USER	Late Unallocation
49	JES2	QGOT
50	USER	End of Input
51	JES2	QMOD
52	USER	Job Card
53	USER	Accounting
54	USER	JECL Card
55	USER	NSR SAF
56	USER	NJE Hdw Xmit
57	USER	NJE Hdw Xmit

As you can easily figure out when comparing Table on page 113 and Table 8-2 on page 115, there are no obvious related exits between the two. You can approximate some similarities, which are exemplified in Figure 8-2 on page 117, but no systematic one-to-one can be made.

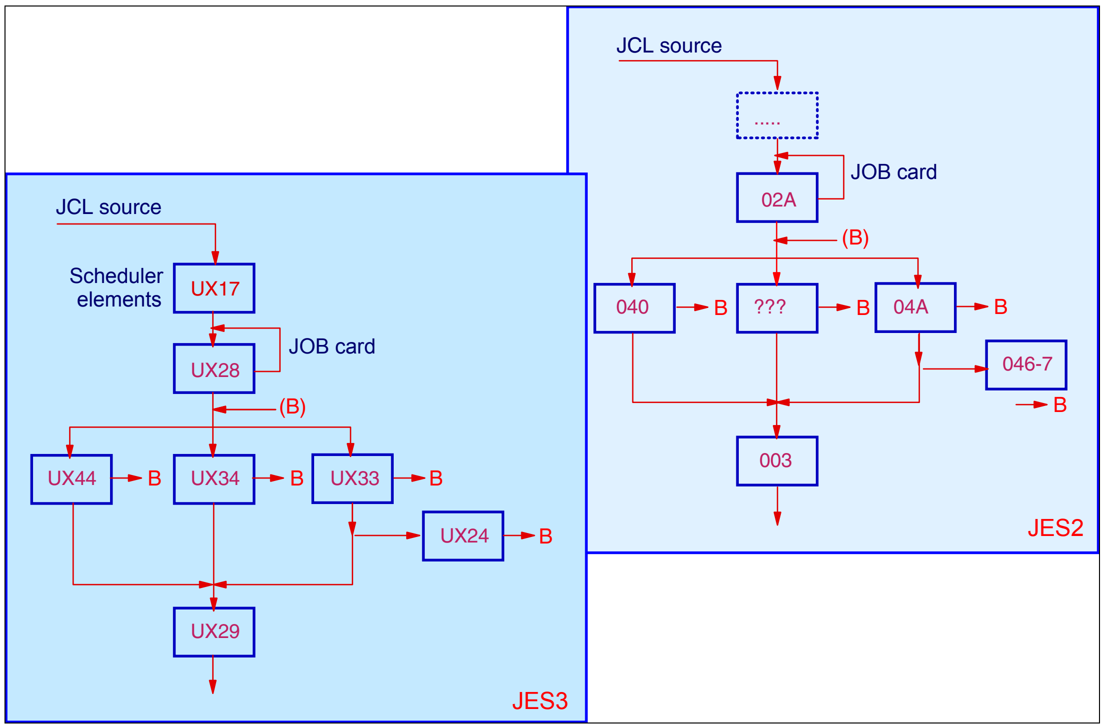


Figure 8-2 Some potential similarities between JES3 and JES2 exits

8.2 Source code changes (usermods)

Because the source code for JES3 is delivered with the product, it is possible that you have made changes to the source. In the past, a common example of such a change was to provide the ability to log on using the same TSO ID on more than one system in the JES3 complex. However, z/OS V2R1 provided that ability to JES3, and it was already available in JES2.

Take an inventory of any changes that you made to the JES3 source, and whether those changes would still be required following a migration to JES2.

You will probably need to change MPFLSTxx to cater for JES2 rather than JES3 messages.

9

Operational considerations

This chapter describes the operational differences between JES3 and JES2 and the changes that need to be considered from an operational perspective. The following topics are covered:

- Operational differences
- Single System Interface
- Consoles and DLOG
- System Display and Search Facility
- Starting JES
- Shutdown considerations
- Dynamic changes
- DFSMSdfp ACS routines &SYSNAME and &SYSPLEX
- DFSMShsm
- Miscellaneous other differences

9.1 Operational differences

The behavior of JES2 and JES3 reflect their history and the fact that JES3 was designed from the beginning to work in a configuration with multiple systems. JES2, alternatively, was designed to work in a single-system environment, then subsequently enhanced to work in an environment with multiple systems. Additionally, the early development, when the fundamentals of both products were established, was carried out by different parts of IBM. As a result, the operating environment in JES3 is quite different from that in JES2. For example, JES3 presents the operator with a single system image, while the systems in a JES2 environment appear as independent systems. Also, partially because of the centralized design of JES3 and partially because of the independent development, the command set used by the two products is also completely different. This chapter describes these, and other, operational considerations.

9.2 Single System Interface

All JES-related work in a JES3 complex is controlled by the Global. The pre-execution setup work is done by the Global, and the decision about which main each job will run on is also controlled by the Global. In fact, if you refer to Table 3-1 on page 54, you will see that, apart from the actual execution of the job, nearly all other activity from the point the job is initially submitted through to when it is purged is controlled by the Global. From an operator's perspective, this gives a JES3 complex very much of a single system image. Even though the work is being run on all systems in the complex, all operator interactions are with the Global.

By contrast, in a JES2 environment, every system is independent. The only aspect of JES2 operations that has a single system image feel to it, is anything to do with management of the checkpoint (because all systems in the MAS use the same checkpoint) and management of files in the spool (because all systems typically share the same set of spool data sets). All other aspects of controlling the environment are managed separately on each system.

For example, a printer could be attached to any member of the MAS and must be controlled *from that member*. And if you want to change the number of available initiators on a system, you would do that from that system.

Technically, this is not very difficult. However, it does require a change in mindset, and might take some time for the operators to get used to. Try to configure your test systems to look like the production systems to give the operators a chance to get used to the different way of managing the environment before migrating the production systems to JES2.

9.3 Operator commands

Perhaps the most obvious difference between JES2 and JES3 from an operator's perspective is that the commands used to control the two environments are completely different. Many of the concepts are common across both JESs (the idea of a spool, print queue, execution queue, starting and stopping initiators, starting printers, managing the NJE network, and so on), but the commands used to control the environments look very different.

Fortunately, most installations use additional products, such as System Display and Search Facility (SDSF), to manage their JES environment. For those products that support both JES2 and JES3, the operators can continue to use a tool they are familiar with.

Nevertheless, it is not possible to completely shield the operators from the underlying commands, and there will be some actions that require issuing JES2 commands directly, rather than using a tool like SDSF. Therefore, it is important that the operators and system programmers are familiar with the JES2 command set. They will also need to understand that some concepts that they are familiar with in JES3 will not carry over to the JES2 environment.

The following sections contain a brief description of the differences. A summary list of the more useful and common commands and their differences can be found in Appendix B, “Comparison of JES3 and JES2 commands” on page 233. There is also a valuable comparison of JES3 to JES2 commands on the following website:

http://www.yves-colliard-software.homepage.t-online.de/YCOS%20-%20JES2-JES3%20Compare%20Commands.pdf

9.3.1 Commands

JES3 commands entered by the operator on any system in a JES3 complex are generally directed to the Global processor regardless of which system in the complex they are issued on. JES3 commands usually start with **`*`** for Global processing, or **`8`** for Local processing. The JES3 Global processes any commands that are directed to it and returns the response to the operator on whichever system entered the command.

In a JES2 environment, the commands are always processed on the system they are entered on, unless you use the MVS **`ROUTE`** command to route them to another system. JES2 commands are prefixed by the console character that is defined by the CONCHAR= parameter on the CONDEF statement in the initialization deck. The default, ‘**`$`**’, is used in most installations. Unlike JES3 commands, spaces are ignored within the command when entering JES2 commands.

You can also stack multiple commands by entering a ‘**`;`**’ between the commands, for example:

```
$DJ(XY*);$DT(ZZ*)
```

9.3.2 Command filtering

Like JES3, JES2 uses filtering in its commands and does keyword limiting. For example, the command **`$T O JOBQ,/FORMS=123,FCB=456`** would modify all output with **`FORMS=123`** and change it to **`FCB 456`**. Filtering is required on some commands and parameters so that any commands that would result in massive changes are verified before execution. For example, if an operator missed the ‘`/`’ in the above command, the response would indicate that a filter is required.

In JES3, this is achieved by having defaults limiting the output for the command and using N=ALL to override.

9.3.3 Syntax checking of JES commands

There is a subtle difference in the JESs in how syntax checking is performed on JES commands.

In JES3, the command is syntax checked *before* executing the command, while in JES2 the command is syntax checked *during* its execution.

For example, an operator enters the following command on a JES3 system. See Example 9-1.

Example 9-1 JES3 command with syntax error

```
*I U J=XXXX,Q=HOLD,NQ=WTR
IAT8126 INVALID KEYWORD NQ=
```

On a JES2 system, the syntax is checked during execution and can be stacked using the ‘;’ command as shown in Example 9-2.

Example 9-2 ‘Stacked’ JES2 command, second has a syntax error

```
$DJ(IMS11*);DJOBCLASS($),LIST;DA
 $HASP890 JOB(IMS11MPP) 810
 $HASP890 JOB(IMS11MPP)  STATUS=(EXECUTING/SC47),CLASS=A,
 $HASP890                PRIORITY=10,SYSAFF=(SC47),
 $HASP890                HOLD=(NONE)
$HASP003 RC=(52),D 813
 $HASP003 RC=(52),D JOBCLASS($)  - NO SELECTABLE ENTRIES
 $HASP003           FOUND MATCHING SPECIFICATION
 $HASP890 JOB(IMS11MPP) 814
 $HASP890 JOB(IMS11MPP)  STATUS=(EXECUTING/SC47),CLASS=A,
 $HASP890                PRIORITY=10,SYSAFF=(SC47),
 $HASP890                HOLD=(NONE)
```

Note: The command prefix ‘$’ is only required in the first occurrence of all stacked JES2 commands.

In this example, you can see that even though the second of the three commands contained a syntax error (it should have contained **LONG** instead of **LIST**), it is ignored and the parser continues with the third command in the stacked string.

9.4 Consoles and DLOG

MVS originally had just 16 route codes for console messages. You can use route codes to control which messages are sent to a physical console. However, it was felt that 16 route codes did not provide sufficient granularity, so the number of route codes was increased to 128, with route codes 43 - 128 being used by JES3. Through the JES3 inish deck, you can cause messages pertaining to the various JES3 functional areas, systems (mains), and devices to be assigned to particular route codes, allowing the messages to be easily partitioned to specific operator consoles. No such capability exists within z/OS or JES2. JES3 console support and JES3 exploiting MCS console support is far better at subsetting responsibilities among operators than z/OS and JES2. Of course, back in the 1970's there were far more processes (mounting tapes and controlling printers) that required manual intervention. Also, the use of automation was only in its infancy then. While this capability still exists in JES3, most installations probably no longer require it, and the use of automation consoles is probably a more effective means to inform operators about any manual intervention that might be required.

OPERLOG is similar to JES3 DLOG in that it provides a consolidated log for all systems in a sysplex. Because DLOG does not exist in JES2, and OPERLOG is common to JES2 and JES3, start the migration to OPERLOG now to get the operators familiar with the new format in advance of the migration to JES2.

Another thing that you need to consider is any processing that you do that involves SYSLOG or console messages in general. Message Processing Facility (MPF) is cognizant of whether it is operating in a JES3 complex. If MPF on a JES3 Local processes a message, a bit gets set in the message and the message will then be ignored by MPF on the Global. If the message is not looked at by MPF on the Local, it will be looked at by MPF on the Global. The reason for this behavior is that the MPF that is the closest to the message issuer is best equipped to make a decision about the message, but some messages might be best handled across the complex (because there are complex-wide ramifications) by MPF on the Global. There is no comparable hierarchy provided in a JES2 environment. IBM recommends that IBM NetView® in a JES3 environment be set up similarly, with decisions that can be made at the local level being handled on the Local, and decisions that require a wider purview be handled by NetView on the Global. Tivoli System Automation is organized like JES3 with a "master" system even when running in a solely JES2 environment. In that respect, it provides a "single system image" that is more akin to how JES3 works.

9.4.1 Automation considerations

Automation support and operations will need to be involved in all aspects of a migration from JES3 to JES2. Review each section in this chapter for potential changes to any current automation routines you have. This includes startup, shutdown, and recovery of any JES-related tasks. Begin by examining the initialization messages for JES3 (IAT3011 for start type) and JES2 ($HASP492 start has completed) to the more complex DSI and checkpoint reconfiguration dialog. Most automation products have a separate built-in base configuration and setup for each of the JESs, but if not, you might also need to refer to Appendix B, "Comparison of JES3 and JES2 commands" on page 233.

9.5 System Display and Search Facility

System Display and Search Facility (SDSF) is a z/OS product that now supports both JES2 and JES3. SDSF originally supported only JES2. The initial JES3 support in SDSF was limited to a subset of JES3 functions, but as of z/OS 1.13, nearly all SDSF functions are available to both JESs.

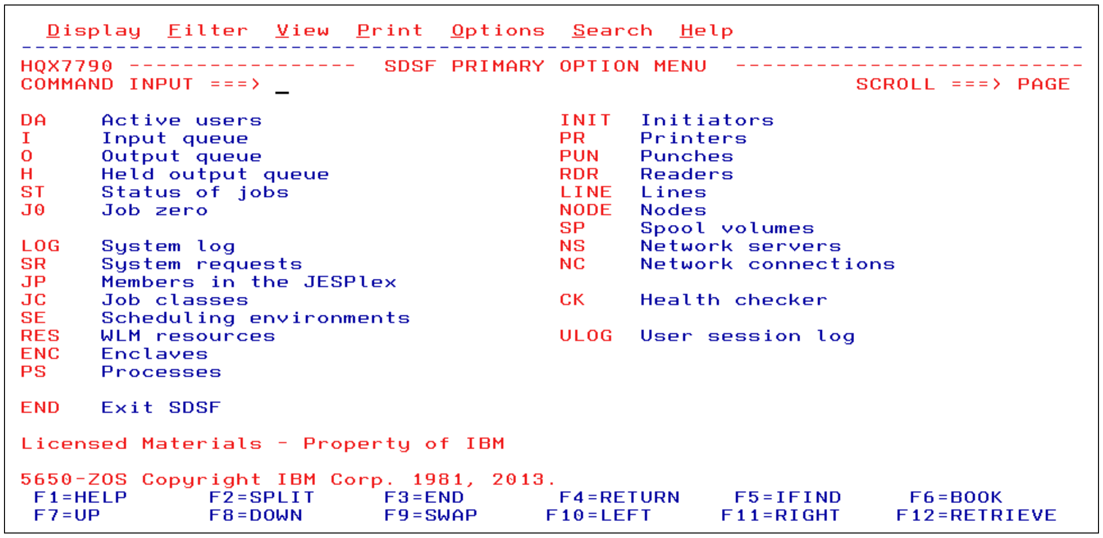
To extend the product to work with JES3, significant enhancements have been provided such as a JES3 Spool Data Set Browse (SDSB) application programming interface.

Indeed, SDSF (Program Number 5694-A01), a feature of IBM mainframes running z/OS, enables users and administrators to view and control various aspects of mainframes' operation. These include jobs in execution, job output, status of UNIX System Services processes, system information, workload scheduling, and log files.

SDSF displays data on panels. Commands and actions that you enter on the panels let you monitor and control jobs and system resources. The SDSF Primary Option Menu lists the panels that you are authorized to use. The objects, displayed on the SDSF panels, are initiators, printers and punches, jobs, SYSIN/SYSOUT data sets, and so on. Information for the objects is extracted using formal JES3 or MVS programming interfaces, for example SubSystem Interface calls. Actions against objects are also invoked through formal programming interfaces or operator commands. Most actions generate MVS or JES commands. In a JES3 environment, the MVS system authorization facility (SAF) is required for SDSF security. When a request is made to access a resource, and the profile that protects the resource is not defined, or the associated class is not active, SDSF fails the request. All SAF profiles must be defined and activated in all of the classes that are used for SDSF security.

Most SDSF panels display information in a tabular format. You can scroll the information up, down, right, and left.

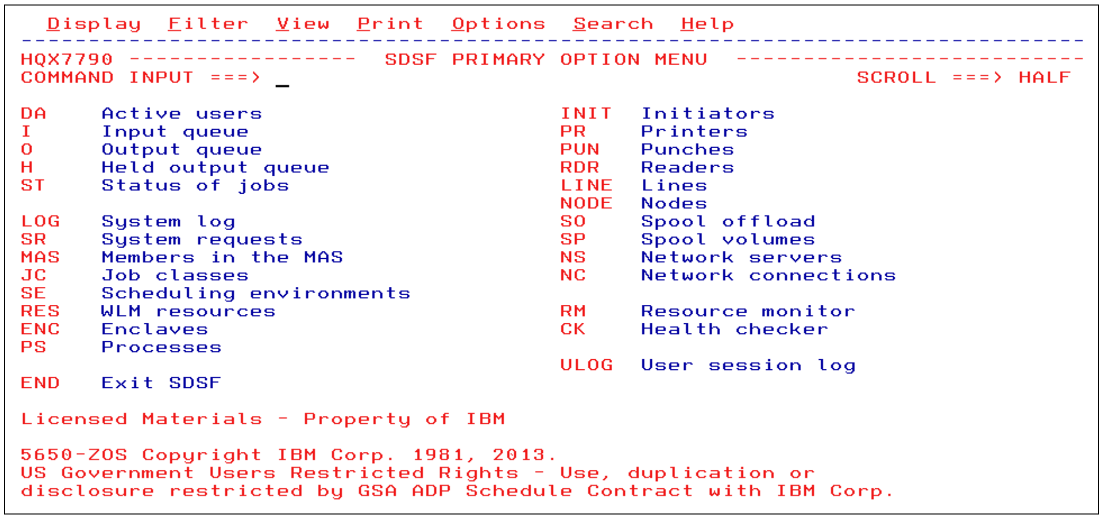
SDSF tries to find a balance between providing a common look and feel across the two JESs and supporting the functions that are unique to a particular JES. For example, the SDSF main menu for a JES3 system is shown in Figure 9-1.



```
  Display  Filter  View  Print  Options  Search  Help
-------------------------------------------------------------------------------
HQX7790 -----------------  SDSF PRIMARY OPTION MENU  --------------------------
COMMAND INPUT ===> _                                             SCROLL ===> PAGE

DA     Active users                     INIT   Initiators
I      Input queue                      PR     Printers
O      Output queue                     PUN    Punches
H      Held output queue                RDR    Readers
ST     Status of jobs                   LINE   Lines
J0     Job zero                         NODE   Nodes
                                        SP     Spool volumes
LOG    System log                       NS     Network servers
SR     System requests                  NC     Network connections
JP     Members in the JESPlex
JC     Job classes                      CK     Health checker
SE     Scheduling environments
RES    WLM resources                    ULOG   User session log
ENC    Enclaves
PS     Processes

END    Exit SDSF

Licensed Materials - Property of IBM

5650-ZOS Copyright IBM Corp. 1981, 2013.
 F1=HELP       F2=SPLIT      F3=END        F4=RETURN     F5=IFIND      F6=BOOK
 F7=UP         F8=DOWN       F9=SWAP      F10=LEFT      F11=RIGHT     F12=RETRIEVE
```

Figure 9-1   SDSF main menu for JES3

And the corresponding main menu panel for SDSF on a JES2 system is shown in Figure 9-2.



```
  Display  Filter  View  Print  Options  Search  Help
-------------------------------------------------------------------------------
HQX7790 -----------------  SDSF PRIMARY OPTION MENU  --------------------------
COMMAND INPUT ===> _                                             SCROLL ===> HALF

DA     Active users                     INIT   Initiators
I      Input queue                      PR     Printers
O      Output queue                     PUN    Punches
H      Held output queue                RDR    Readers
ST     Status of jobs                   LINE   Lines
                                        NODE   Nodes
LOG    System log                       SO     Spool offload
SR     System requests                  SP     Spool volumes
MAS    Members in the MAS               NS     Network servers
JC     Job classes                      NC     Network connections
SE     Scheduling environments
RES    WLM resources                    RM     Resource monitor
ENC    Enclaves                         CK     Health checker
PS     Processes
                                        ULOG   User session log
END    Exit SDSF

Licensed Materials - Property of IBM

5650-ZOS Copyright IBM Corp. 1981, 2013.
US Government Users Restricted Rights - Use, duplication or
disclosure restricted by GSA ADP Schedule Contract with IBM Corp.
```

Figure 9-2   SDSF main menu for JES2

You can see that there are small differences between the two, reflecting the differences between JES2 and JES3. For example, the JES2 menu does not have an option for JOB Zero, and JES3 does not have an option for Spool Offload or Resource Monitor, two functions that do not exist on JES3. And where a JES2 system has a MAS option, JES3 has an equivalent JP option to show the systems in the JESPlex.

Obviously, as you drill down within each option, the information that is presented, and the commands that can be used, will be different, reflecting the unique attributes of each JES.

One positive change that you will notice in the JES2 version of SDSF is that far more information is presented about the NJE configuration. Because JES3 uses BDT for SNA/NJE, and BDT does not provide interfaces for SDSF to retrieve the information, very little information is available on the JES3 NJE panels if you are using SNA/NJE. JES2 does not need BDT for SNA/NJE, so SDSF is able to provide a lot more information in its NJE displays.

9.5.1 Starting writers on printers

There are two types of writers in JES3:

- Hot writers

  A hot writer is invoked by the operator using an *X command. The command can either be issued by the operator or by JES3, depending on how the DEVICE associated with the writer is defined to JES3. If the DEVICE specifies NO for the DYNAMIC keyword in the JES3 initialization deck, the writer is controlled exclusively by the operator. If the DYNAMIC keyword specifies YES and a nonzero value for the timeout, the writer is eligible for an automatic *X by JES3. The writer notifies you when it is waiting for work and remains available for processing, as shown below. The writer waits for as long as the timeout period defined for the device. If no new work becomes available, it will then terminate. You control the writer using the *X, *S, *R, and *C commands. The operator can use the **`*F,W`** command to restrict a device to be started only as a hot writer or to change the associated timeout value. If the device is defined with the DYNAMIC keyword set to YES and a timeout value of zero, the device is eligible for a dynamic writer.

  Note: With JES3 terminology, any operator command with *X is referred to as a callable DSP. The DSP remains in the system forever or until the operator cancels it. When it is no longer active, it issues the following message:

  ```
  *IAT7005 WRITER PRT1 (00E) WAITING FOR WORK
  ```

- Dynamic writers

  A dynamic writer is a writer that JES3 output service starts and its associated device is based on the availability of an output device and the current output data set requirements. After JES3 initialization, the operator must use the *S command the first time you use a device associated with a dynamic writer. After that, printing begins automatically for properly prepared devices that are in the ready state. The operator can use the *S, *R, and *C commands to control dynamic writers while they are active. The dynamic writer will stop immediately after no suitable output is available for processing.

9.6 Starting JES

Operations support needs to be aware of the differences in the JES configurations and the implications of monitoring and starting a JES subsystem on each of the respective systems.

9.6.1 In a JES3 environment

In a JES3 environment, operators must know where the JES3 global resides or will be residing and that there is only one system that is designated global at any one time. The initial start of a designated JES3 global system after an IPL will be either a Hot Start or Hot Start Refresh depending on any JES3 initialization stream changes. However, if the start type of the JES3 global is a WARM or COLD, all other systems in the JES3plex must be shut down and IPLed as well. For any of the JES3 local systems, the only option is a local start.

Notes:

A **`*X DSI`** command can be issued to move the global function to a local system. That system then becomes the global and the previous global is not longer active. It could then be started as a JES3 local.

The JES3 global address space contains information that in a JES2 system is in the JES2 checkpoint data set. If you shut down the global system for any reason and you can do this with the **`*RETURN`** command, you can restart the global on the same LPAR by issuing the **`S JES3`** command.

9.6.2 In a JES2 environment

In a JES2 environment, all systems are independent of each other and would not be affected during IPL and start up, even if any other system is having problems *unless* the affected system also holds the checkpoint lock. This could be either via a hardware reserve or a flag in the lock that denotes the system that is holding the lock. Operations would need to be aware of the process to recover from this situation. A sample procedure for checkpoint lock recovery can be found in Chapter 4, “Checkpoint Dataset Definition and Configuration” in *z/OS JES2 Initialization and Tuning Guide*, SA22-7532.

From an operator’s perspective, there are only two types of JES2 starts: COLD and WARM. A COLD start is the initial start to format spool and would rarely, if ever, be performed on any subsequent starts. After that, all starts of JES2 are WARM. The reference to other JES2 start types, for example quick or hot, describe if the checkpoint data set is being read and how JES2 was shut down previously, but all are responded to using the same **`PARM=’WARM,NOREQ’`** command.

There might be some instances of errors in the JES2 initialization parameters that result in a WTOR being issued and the operator having to decide how to proceed. Generally speaking, the operator can continue by replying to the WTOR with the corrected parameter statement. An example of this is shown in Figure 9-3.

```
$HASP466 PARMLIB    STMT     1  CONNDEF CONCHAR=$,BUFNUM=200,A
 $HASP466 PARMLIB    STMT     1  UTOCMD=25
 $HASP003 RC=(03),CONNDEF  - INVALID PARAMETER STATEMENT
 REPLY 63,END
*063 $HASP469 REPLY PARAMETER STATEMENT, CANCEL, OR END
 IEE600I REPLY TO 063 IS;END
 IEF196I IEF285I   SYS1.PARMLIB                                 KEPT
 IEF196I IEF285I   VOL SER NOS= #@$#M1.
*$HASP451 ERROR ON JES2 PARAMETER LIBRARY
 REPLY 64,Y
*064 $HASP441 REPLY 'Y' TO CONTINUE INITIALIZATION OR 'N' TO TERMINATE
 IEE600I REPLY TO 064 IS;Y
```

Figure 9-3 Syntax error messages for a JES2 init parm statement during startup

Automation setup needs to be included as part of the migration checklist. See 10.14, “Automation” on page 141.

9.6.3 JES2 Checkpoint Reconfiguration Dialog

The JES2 Checkpoint Reconfiguration Dialog is a procedure that is unique to JES2 due to the criticality and dependency of the checkpoint data sets. It is an interaction with the operator that consists of a series of displays and operator responses. The purpose of the Checkpoint Reconfiguration Dialog is to let the operator move the checkpoint to a different location.

The dialog is typically initiated either by the system programmers when they need to move the checkpoint for some reason, by JES2 itself if it detects a problem with the checkpoint data sets.

Before invoking the dialog, operations would need to be familiar with the setup and configuration of the checkpoint data sets.

A sample JES2 command **$D CKPTDEF** can be used to display the setup of the checkpoint data sets in the configuration and the sample output of the display can be found in Figure 9-4 for data sets on DASD and Figure 9-5 for structures in the coupling facility (CF).

```
$HASP829 CKPTDEF
$HASP829 CKPTDEF  CKPT1=(DSNAME=SYS1.JES2.CKPT1,
$HASP829          VOLSER=volsr1,INUSE=YES,VOLATILE=NO),
$HASP829          CKPT2=(DSNAME=SYS1.JES2.CKPT2,
$HASP829          VOLSER=volsr2,INUSE=YES,VOLATILE=NO),
$HASP829          NEWCKPT1=(DSNAME=SYS1.JES2.CKPT3,
$HASP829          VOLSER=volsr3),
$HASP829          NEWCKPT2=(DSNAME=SYS1.JES2.CKPT4,
$HASP829          VOLSER=volsr4),MODE=DUPLEX,DUPLEX=ON,
$HASP829          LOGSIZE=8,
$HASP829          VERSIONS=(STATUS=ACTIVE,NUMBER=50,
$HASP829          WARN=80,MAXFAIL=0,NUMFAIL=0,
$HASP829          VERSFREE=50,MAXUSED=1),RECONFIG=NO,
$HASP829          VOLATILE=(ONECKPT=DIALOG,
$HASP829          ALLCKPT=DIALOG),OPVERIFY=NO
```

Figure 9-4 Sample output from $D CKPTDEF command with data set on DASD

```
$HASP829 CKPTDEF  CKPT1=(STRNAME=JES2CKPT_1,INUSE=YES,
$HASP829          VOLATILE=YES),CKPT2=(DSNAME=SYS1.JES2.CKPT3,
$HASP829          VOLSER=volsr2,INUSE=YES,VOLATILE=NO),
$HASP829          NEWCKPT1=(DSNAME=SYS1.JES2.CKPT1,
$HASP829          VOLSER=volsr3),NEWCKPT2=(DSNAME=,VOLSER=),
$HASP829          MODE=DUPLEX,DUPLEX=ON,LOGSIZE=7,
$HASP829          VERSIONS=(STATUS=ACTIVE,NUMBER=2,WARN=80,
$HASP829          MAXFAIL=0,NUMFAIL=0,VERSFREE=2,MAXUSED=1),
$HASP829          RECONFIG=NO,VOLATILE=(ONECKPT=WTOR,
$HASP829          ALLCKPT=WTOR),OPVERIFY=YES
```

Figure 9-5 Sample output from $D CKPTDEF command with structure in CF

As shown, it is recommended for an installation to include a primary and alternate checkpoint data set for any errors that occur in the primary and have two NEW Checkpoints for maintenance. The primary checkpoint might also be a structure in the CF and its backup on DASD.

For example, to invoke the checkpoint reconfiguration dialog, the command **$T CKPTDEF,RECONFIG=YES** is entered as shown in Figure 9-6 with a sample response.

```
-$T CKPTDEF,RECONFIG=YES
*$HASP285 JES2 CHECKPOINT RECONFIGURATION STARTING
*$HASP233 REASON FOR JES2 CHECKPOINT RECONFIGURATION IS OPERATOR
          REQUEST
*$HASP285 JES2 CHECKPOINT RECONFIGURATION STARTED - DRIVEN BY
          MEMBER #@$A
*$HASP271 CHECKPOINT RECONFIGURATION OPTIONS
*
          VALID RESPONSES ARE:
*
          '1' - FORWARD CKPT1 TO NEWCKPT1
          '2' - FORWARD CKPT2 TO NEWCKPT2
          '5' - SUSPEND THE USE OF CKPT1
          '6' - SUSPEND THE USE OF CKPT2
          'CANCEL' - EXIT FROM RECONFIGURATION
          CKPTDEF (NO OPERANDS)   - DISPLAY MODIFIABLE
                                    SPECIFICATIONS
          CKPTDEF (WITH OPERANDS) - ALTER MODIFIABLE
                                    SPECIFICATIONS
*050 $HASP272 ENTER RESPONSE (ISSUE D R,MSG=$HASP271 FOR RELATED MSG)
```

Figure 9-6 JES2 Checkpoint Reconfig Dialog

The applicable options to reply are shown as valid responses and the reply would be based on the action required.

A recommendation is to automate the recovery for the checkpoint data sets. The init parm as shown in Example 9-3 can be used.

Example 9-3 Sample init parm to automate checkpoint data set recovery

```
CKPTDEF NEWCPTn,OPVERIFY=NO
```

No matter how the JES2 configuration is designed, operations and system programmers need to become familiar with this critical procedure. For further reference in a typical configuration, the procedure is referenced in detail in *z/OS JES2 Initialization and Tuning Guide*, SA22-7532.

Planned outage

This procedure would be used in the case of moving the primary and alternate checkpoints to different volumes for maintenance and would be initiated with the command shown in the previous section.

Unplanned outage

This would occur if there is an I/O error on one of the primary checkpoint data sets. It would be assumed that responses to the WTORs would require assistance from the system programming team to respond.

9.6.4 Starting NJE lines and nodes

In JES3, NJE nodes are started automatically when networking is started to another node or when the operator command **`*NJE NAME=nodename`** is entered.

In JES2, NJE lines and nodes are *not* automatically started, so a **`$S N`** command might need to be entered (via automation) or put in the automatic command processing or included in JES2PARM. JES2 in z/OS 1.10 introduced a function called Network Monitor that performs an automatic restart of NJE connections at a designated time interval, with a default of every 10 minutes.

9.7 Shutdown considerations

A clean, or normal, shutdown of each JES subsystem is preferred and is described.

JES3 Global shutdown

Two types of procedures might be needed for JES3 global shutdowns based on the requirements and setup of the installation: Normal and Dynamic System Interchange (DSI).

Normal JES3 Global shutdown

On the JES3 Global, shutting down the JES3 address space would be done with the **`*RETURN`** command. It shuts down JES3 and related address spaces and quiesces JES processing for all systems in the JESplex.

Dynamic System Interchange (DSI)

If the installation cannot tolerate a JES3 Global outage for longer than the time of a system shutdown and IPL, a Dynamic System Interchange (DSI) procedure needs to be considered or invoked. This procedure can be found in detail in the JES3 commands manual. Each installation must be aware of any applications or configurations that are dependent on the usage of the Global system when this is performed. For example, the JES3 Global system might be the host for the primary AOMGR system or IBM Tivoli Workload Scheduler (TWS) system and operations need to make some changes when this is done. Practicing this procedure, and the reverse procedure to DSI back, is recommended as it might be required in emergency situations to avoid any extended outage.

JES3 local shutdown

A shutdown of JES3 on any local system only affects JES processing on that system. All JES3 local address spaces are recycled after a Hot start with refresh in order to pick up any changes from the inish deck.

JES2 shutdown

A normal or orderly shutdown of JES2 implies that you will also be performing a shutdown of the system since a majority of tasks that were submitted under the JES2 subsystem will still be executing on the system. The normal shutdown procedure would include an initial **`$P JES2`** command to start draining the queues. The operator would then proceed to shut down or stop all remaining address spaces. To determine what is remaining, the operator would enter a **`$D JES2`** command as shown in Example 9-4.

Example 9-4 Sample display from $D JES2 command

```
$HASP608 $D JES2 865
$HASP608 ACTIVE ADDRESS SPACES
$HASP608 ASID     JOBNAME  JOBID
```

```
$HASP608 -------- -------- --------
$HASP608 001E     RMF      STC28871
$HASP608 001F     BPXAS    STC29133
$HASP608 0028     SDSF     STC28872
$HASP608 0029     TCPIP    STC29138
$HASP608 002C     CANSCN   STC28868
$HASP608 012A     TSO      STC29129
$HASP608 012C     BPXAS    STC30092
$HASP608 012E     HZSPROC  STC29140
$HASP608 012F     PFA      STC29139
$HASP608 0130     RMFGAT   STC29143
$HASP608 0131     TN3270   STC29142
$HASP608 0133     HIS      STC29145
$HASP608 ACTIVE NETWORKING DEVICES
$HASP608 NAME                           STATUS
$HASP608 ------------------------------ -----------------
$HASP608 NETSRV1                        ACTIVE
```

The operator would proceed to shut down all the address spaces remaining and then enter the **$P JES2** command again.

If there are still tasks remaining, and the operator or automation cannot determine what remains, JES2 can be shut down using the **$P JES2,ABEND** command.

9.8 Dynamic changes

Dynamic changes can be made to most of the parameters in either JES configuration. The main difference currently is that there is limited functions to add and deleted JES3 spool volumes, which can be done in a JES2 configuration.

Also, dynamic changes made on JES2 systems might or might not remain across JES2 restarts.

9.9 DFSMSdfp ACS routines &SYSNAME and &SYSPLEX

In a JES3 environment, CDS displays the &SYSNAME of all the system names in the sysplex since the &SYSPLEX variable is not supported. When migrating to JES2, the &SYSPLEX variable can then be used.

9.10 DFSMShsm

A recommended positioning move is to convert your DASD configuration to SMS-managed volumes. If this is not done, you will notice that the behavior of DFSMShsm in a JES2plex or mixed JESplex is not quite the same and would need to take this into consideration as well to avoid any unexpected anomalies.

9.10.1 Difference for non-SMS managed volumes and data sets

For installations that have not migrated to SMS-managed volumes, the following needs to be reviewed.

DFSMShsm general pool

In JES2, there are two types of pools: Data set and volume pools, also known as *user-defined pools*.

In JES3, there is a third type of pool, other than the user-defined pools, known as the *general pool*. The general pool is composed of primary volumes that are online to the system at HSM startup time for all systems that reference the HOSTMODE=MAIN and the automatic recall attribute. When the primary HSM starts, no further primary volumes can be added using the **ADDVOL** command or removed using the **DELVOL** command since these volumes might contain data sets that are being used by jobs by any system in the JESplex.

9.10.2 Difference for SMS-managed volumes and data sets

Even if all volumes and data sets in the installation are SMS-managed, the following still needs to be reviewed for migration purposes.

Deletion of SMS-managed temporary data sets

For any SMS-managed temporary data sets that are not deleted at the end of the job, for example, as a result of a system outage, DFSMShsm attempts to delete them during its backup processing. However, in a JES3 installation, they must be at least two days old before it can be deleted. If migrating to JES2, this might cause some data sets to be deleted earlier than expected when the installation was in a JES3 configuration.

9.11 Miscellaneous other differences

9.11.1 Duplicate job names and multiple system signons

One of the inherent differences in the JESs has always been that duplicate job names and TSO logons are not allowed in JES3. Now in z/OS V2R1 JES3 allows duplicate TSO logons. In JES2, these can be allowed by changing to the parameter DUPL_JOB=NODELAY. If this enabled when the migration is done, there are possible side effects that might or might not be expected in your installation.

Note: In the *z/OS MVS Initialization and Tuning Reference,* SA23-1380 under the TEMPDSFORMAT(INCLUDELABEL) parameter of the ALLOCxx statement, there is this warning note:

When this parameter is specified and the Job Entry Subsystem (JES) allows multiple jobs with the same job name to execute at the same time, jobs with the same name, executing simultaneously, might fail with a duplicate data set name error.

9.11.2 Display of job numbers and job IDs

Another difference between the two JESs is the location of the job number and job names in message displays. Example 9-5 on page 132 shows the output from a JES3 command to display information about a job. As you can see, in JES3, the job number (JOB62993) is displayed as part of the message text.

Example 9-5 Sample display from JES3 command to display job/userid TSOUSR

```
*I J=TSOUSR
IAT8674 JOB TSOUSR (JOB62993) P=15 CL=A       MAIN(EXECUTING-SC43)
IAT8699 INQUIRY ON JOB STATUS COMPLETE,      5 JOBS DISPLAYED
```

Example 9-6 shows the output of an equivalent command in JES2. In JES2, the job number (TSU20886) is shown in the prefix of the message rather than in the HASP890 message text. Operators would need to be aware of where to find this information, and any automation routines or installation-written programs that process syslog would need to be aware of the change.

Example 9-6 Sample display from JES2 command to display job/userid TSOUSR

```
TSOUSR 00000290  $D T'TSOUSR'
TSU20886 00000090  $HASP890 JOB(TSOUSR) 449
     449 00000090  $HASP890 JOB(TSOUSR)   STATUS=(EXECUTING/SC42),
     449 00000090  $HASP890               CLASS=TSU,PRIORITY=15,
     449 00000090  $HASP890               SYSAFF=(SC42),HOLD=(NONE)
```

9.11.3 System symbols

When JES3 starts, each JES3 passes the symbols on that system, and the value of those symbols, to the other JES3s. This is because in JES3, the JCL for a started task could get interpreted on one system, but be executed on another. So JES3 needs to know the symbols on all systems so that the correct values can be substituted.

If you update a system symbol using IEASYMUP or a similar tool, the updated value does *not* get propagated to the other systems. So, in a JES3 complex, JCL that gets interpreted on other systems would continue to contain the previous value of the symbol, rather than the current, correct value as updated by IEASYMUP.[1]

On a JES2 system, the conversion and interpretation for started task JCL always happens on the system that the started task will run on, so there are no concerns about out-of-date system symbol values being used.

9.11.4 MVS checkpoint/restart

Checkpoint/restart is a function in DFSMSdfp that allows a job to be check pointed at certain intervals so that if the system fails, it can be restarted from the point of the most recently processed checkpoint and avoid a rerun of the entire job.

The difference in the JESs for checkpoint/restart processing is where it is referenced. In JES3 it is specified by the FAILURE= parm on the //*MAIN statement, and in JES2 RESTART=YES on the /*JOBPARM statement.

[1] It is a known requirement to add support in a future release such that for in-stream data, JES3 uses the common JES service, which gets the current symbol values.

Job journal for checkpoint/restart

One of the features for checkpoint processing is a job journal. A job journal consists of a sequential data set on spool with related control blocks for processing if the job is eligible for automatic restart.

In JES2 and JES3, the journal can be specified on the CLASS statements or referenced with RESTART on the RD parameter on the JOB or EXEC statement. However, in JES3, there is another way of specifying JOURNAL=YES by placing it on the MAIN JCL statement, which can override the CLASS statement entry. When the system is restarted, the operator is sent message IEF225D asking if the job should be restarted.

Therefore, during migration, review any JCL that uses FAILURE= or JOURNAL=YES on the //*MAIN STATEMENT and ensure that the class it will use on JES2 has this defined and that the proper RESTART is placed on the /*JOBPARM statement.

To summarize, either set up job classes that require journaling in JES2 or use the RD= parameter of JCL.

Notes:

Automatic Restart Manager takes precedence in all cases and behaves the same way in either JES.

See *z/OS DFSMSdfp Checkpoint/Restart,* SC26-7401 for a list of restrictions related to JES3.

9.11.5 Network Resource Monitor

There are parameters and commands in JES2 to assist in automating startup of NJE network components including LINES, LOGONs, and NETSERVs. These include using START=YES, RESTART=YES, CONNECT=YES parameters and **`$E NET`**, **`$Z NET`**, and **`$S NET`** commands.

9.11.6 Printer naming restrictions

In a JES3 environment, the names for printers are defined on the JNAME= parameter on the DEVICE statement. They can be any 1 - 8 character name with no leading slash (/) and can be grouped with other printers using a Device Group statement (DGROUP).

When converting to JES2, the printer names are defined consecutively as PRT(nnnn) for local printers or Rnnnn.PRn for RJE printers. So for example, the name of the first printer defined is PRT1 or PRT(1) and it can be modified using the **`$T PRT1`** command.

10

Related products

This chapter describes the considerations for products that might be impacted by the migration from JES3 to JES2. As one can imagine, there are many products that interact with JES (including products that run on platforms other than z/OS). It would be impossible to create an all-inclusive list of every product. However, to help you identify products that might require your attention, we have listed categories of products, which are based on the type of function they provide.

10.1 JES-neutral interface

In this chapter, we discuss various types of JES-related products. Over the years, new interfaces have been provided that allow programs to interface generically with JES, without regard to whether the underlying JES is JES2 or JES3. This type of interface is known as “JES-neutral”. Programs and products that use these interfaces have the advantage that they can be easily moved from one JES to another.

Subsystem Interface 80 is a good example of such a JES-neutral interface. The extended status function call (SSI function code 80) allows a program to obtain detailed status information about jobs and SYSOUT in the JES queue.

Products that use these interfaces have the advantage that you can transparently move them back and forth between one JES and the other. If you have any home-grown programs that interface with JES, encourage the use of a JES-neutral interface wherever possible.

10.2 Products that interact with JES

There are many products, both from IBM and from other vendors, that interact with JES. This chapter presents the different categories that these products fall into. It would not be possible to provide a complete list of all these products, but we hope that listing the different *types* of products will help you identify all the products in *your* environment that might be affected.

As you identify each product, consider whether it would still be required after the migration is complete. It might be that the function is no longer required, or, in fact, the product could be removed even if you do not proceed with the migration. Or, it might be that the function provided by the product is provided by some other product in a JES2 environment, an example would be BDT. If the product is no longer required, you can add any cost savings that might result from removing the product to the financial part of your migration plan. From a technical perspective, removing the product means that you do not need to worry about migrating it. It might also make your software management processes a little simpler if there is one less product to manage.

If you determine that the product *will* still be required, you need to investigate whether any changes will be required to the product in order for it to work with a different JES. Potentially, there might be a separate JES2 version of the product. In that case, you need to determine the financial considerations, and what, if any, migration actions are required. You also need to determine if there will be any changes to the user interface as a result of changing the underlying JES.

You also need to generate a list of any JES exits that might be required by any of these products. Hopefully you are using a standard version of the exit, and the product provides both a JES3 and a JES2 version of the exit. Or, even better, perhaps it does not require an exit at all when running in a JES2 environment.

Take this opportunity to ensure that all products are on supported releases. Enabling new functions in a product (for JES2 support) might uncover problems that you have not encountered before, so it is important that you get support from the vendor. You also need to check to ensure that no new license keys are required as a result of changing the JES.

The remainder of this chapter describes the different categories of products that you are likely to have.

10.3 Print output and archive products

Printing and output management is probably the category that has the largest number of products that interact closely with JES. The good news is that because there are more JES2 than JES3 installations, it is likely that any product you have will work with both JES2 and JES3.

If you use products that exploit JES3 functions such as //*FORMAT PR or PU, work with the vendor to determine what the considerations are for using that product with JES2. Table 4-10 on page 66 provides information about replacing //*FORMAT statements with // OUTPUT statements. However, you need to look at exactly how the //*FORMAT statements are being used and understand any specific requirements of the product in order to get identical results when it is used with JES2.

Another area to review is JES3 printers. A //*FORMAT DEST = NODE.REMOTE will be treated in JES2 as NODE.USERID. JES2 makes no distinction and will be in the same network SYSOUT data set header.

In other terms, JES3 JECL allows a //*FORMAT statement to specify JES3 processing instructions for SYSOUT data sets that are printed. These instructions permit special processing of SYSOUT data sets, such as:

- Multiple destinations
- Multiple copies of output with different attributes
- Force single or double space control
- Printer overflow checking

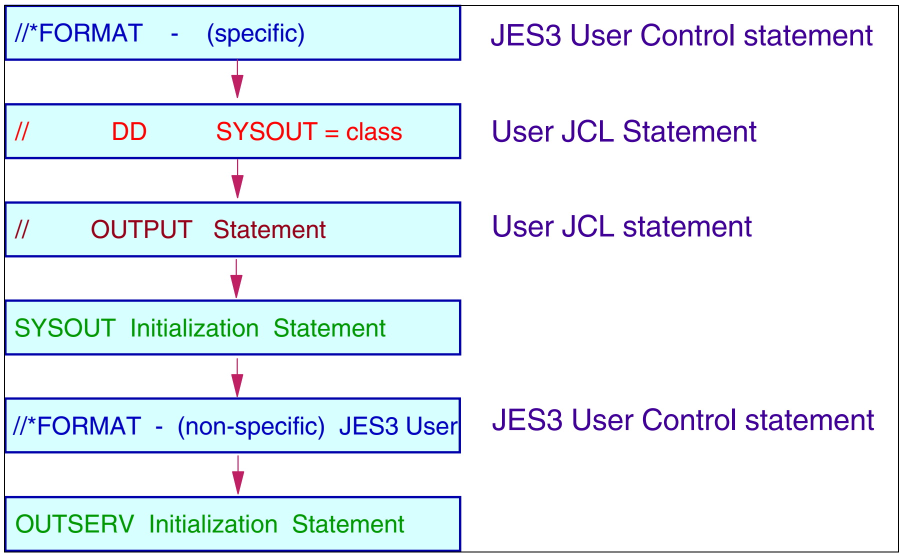

//*FORMAT - (specific)
JES3 User Control statement
// DD SYSOUT = class
User JCL Statement
// OUTPUT Statement
User JCL statement
SYSOUT Initialization Statement
//*FORMAT - (non-specific) JES3 User
JES3 User Control statement
OUTSERV Initialization Statement

Figure 10-1 JES3 SYSOUT order of services

Thanks to the SYSOUT order of overrides, Figure 10-1, a specific FORMAT statement overrides all the other statements, so a specific destination can be expressed for a given DD of a given job step.

On the contrary, JES2 allows only NODE.USERID to be specified as a destination for the entire set of DDs of a specific job step.

Most other end-user externals will not have to change.

If using an output archiving product, be careful of keywords on a JES3 //*MAIN card such as ORG=, that you might be using as an alternative archiving qualifier.

Some products allow one to control spooled output processing, enabling users to view and manage all types of output including batch jobs, started tasks, TSO users, APPC jobs, active tasks, held and non-held output across the input, execution, and output queues. Users can control job output through comprehensive browse, cut and paste, sort, find, print, copy to disk data set, hold, requeue, and delete capabilities. They generally provide functions to both JES2 and JES3 in a comprehensive and equivalent manner. As mentioned in 9.5, “System Display and Search Facility” on page 123, such is now the case also for SDSF.

10.4 Batch schedulers

Batch schedulers such as IBM Tivoli Workload Scheduler (TWS) are likely to have a close relationship with JES.

For one thing, it is probable that they will exploit some JES exits to enable them to monitor job-related activity in JES. Review the product installation guide to determine which exits the product requires, both for JES3 and for JES2. It is possible, although unlikely, that you are not using the function in the scheduler that requires the exits. Determine if you are using the product’s JES3 exits today; if you are, have you modified them or just used the standard exits as provided by the product? This will help you determine what you will need to do in the JES2 environment.

If you are still using Dependent Job Control or Deadline Scheduling under JES3, you will need to find some alternative way to provide that functionality under JES2. The most obvious tool to deliver that functionality is your batch scheduler. Ideally, you will start migrating from those JES3 functions over to the batch scheduler in advance of the migration. Functions such as Deadline Scheduling might require enabling specific features in the batch scheduler, such as the Workload Service Assurance feature in TWS.

Another reason that the scheduler requires your attention is that it is probably the repository for all your production JCL. Scan the production jobs to see if they use any of the capabilities documented in Chapter 4, “JECL and JCL differences” on page 55. If they do, create a plan to migrate those jobs to a mechanism that works in both JES2 and JES3.

Some batch schedulers have a JCL checker integrated into the product. Such JCL checkers typically provide the ability to specify local standards. You might be able to use that function to identify JCL that will not function in the same way in a JES2 environment. At a minimum, you need to tell the JCL checker if it is operating in a JES3 or JES2 environment so that it can treat any JECL it encounters accordingly.

Finally, do not forget to check for any remote scheduling that might be in use. Determine if jobs are being submitted via RJP or NJE. If they are, you will need to check if the jobs being submitted will work in your target JES2 environment.

10.5 JES-managed printers

All IBM mainframe printers support both JES2 and JES3. Refer to Table 4-11 on page 68 for Forms Control Buffer (FCB) considerations.

If you have non IBM JES3-managed printers, consult your vendor to determine if there are any considerations for using the printer with JES2 instead of JES3.

10.6 NJE and RJP devices

Any non System z product that emulates an NJE or RJP device to communicate with JES must be reviewed for use with running in JES2 versus JES3.

10.7 JCL generators

Any product that builds JCL "under the covers" and submits it to your system must be checked. It is unlikely that such products are generating JES3 JECL unless you customized them to do so, so hopefully it will be easy to address any products that are doing this type of thing.

The challenge could be in identifying those products. If the product submits the job through the internal reader, it is difficult to differentiate that job from any other job on the system. You might look at the security ID associated with any started tasks, and then look for jobs submitted under those IDs. Any package that runs under CICS or IMS might submit jobs using the CICS or IMS security ID, so you could look for that as well.

Do not forget to investigate products that run on other platforms that generate JCL for use on z/OS. For example, some software products are client-based and might run on Windows but they interface with z/OS or other vendor software.

10.8 JES operator, users, and administrator tools

SDSF is IBM's product for viewing and controlling the contents of the spool (see 9.5, "System Display and Search Facility" on page 123). It is also used for starting and stopping JES-managed devices, the NJE network, initiators, and so on. In order to do these things, the product needs to communicate with JES. For many years, SDSF only worked with JES2; now it also works with JES3.

There are similar products from other vendors as well that would also be expected to have an equally close relationship with JES. Such is the case with output viewing software, which equally runs in a JES2 or JES3 environment.

10.9 SMF analysis

Any product or installation-written program that reads SMF records containing information that is provided by JES2 or JES3 will need to be changed. These are described in more detail in Chapter 16, "Accounting and chargeback considerations" on page 207, but we include them in Table 10-1 for the sake of completeness.

Table 10-1 JES-related SMF record types

Record type	Description	JES2 or JES3?
6	Output writer	Both[a]
24	Spool offload	JES2
25	Device allocation	JES3
26	Job purge	Both[a]

Record type	Description	JES2 or JES3?
43	JES start	Both[a]
45	Withdraw/Stop	Both[a]
47	NJE SIGNON (BSC)	Both[a]
48	NJE SIGNOFF (BSC)	Both[a]
49	Integrity (BSC)	Both[a]
52	NJE LOGON	JES2
53	NJE LOGOFF	JES2
54	Integrity (SNA)	JES2
55	NJE SIGNON	JES2
56	Network Integrity	JES2
57	Network SYSOUT Transmission	Both[a]
58	NJE SIGNOFF	JES2
59	BDT File Transmission	JES3/BDT
84	JES3 Monitoring Facility	JES3

a. Both JES2 and JES3 create this type of SMF record, but the formats are different.

A good starting point would be to look in your SMFPRMxx member to see which of these record types are being collected today. Obviously, if a record type is not being collected, you do not need to worry about any migration efforts for that record type. However, even if you are collecting a given record type, that does not necessarily mean that it is being processed by any programs. Speak to the group that is responsible for processing SMF data to see if they use any of the record types shown in Table 10-1 on page 139. If they do, they need to take actions to process the JES2 version of the records rather than the JES3 version.

10.10 Users of JES exits

JES3 exits, and the JES2 equivalents (if one exists) are described in Chapter 8, “User exits” on page 109. In this chapter, we only mention JES3 exits because any product that uses JES3 exits will need to at least be reviewed, and possibly changed, as part of the migration to JES2. Also, scanning the list of used JES3 user exits might help you spot a product that you might otherwise have missed.

For information about how to determine if a given user exit is being used, refer to 8.1, “JES3 user exits” on page 110.

10.11 JCL checkers

There are a number of products that provide a JCL checking function. Most products are compatible with either JES2 or JES3, and most of them support customization to let you specify installation standards. These could be used to help flag jobs that use JES3 JECL statements that would need to be addressed before a migration to JES2. The product might also require recustomization at the time of the cutover to tell it that it is now operating in a JES2 environment.

10.12 Application software control

It is unlikely that an application software control product (such as SCLM by IBM) will have a close relationship to the type of JES that is being used. If there are any issues, they are most likely to be related to the use of JECL statements to route the output to a SYSOUT archiving product. Take a few minutes to check the JCL that gets submitted by your application software control product to ensure that it does not use any JCL or JECL statements that would have to be changed before migrating to JES2. Examples would include makefiles, build scripts, and any scripts, where JCL is dynamically built and submitted.

10.13 Other products

There might be other products that need to know if they are running in a JES2 or JES3 environment.

One example is DFSMShsm. In the ARCCMDxx member, if you are using JES3, you must include a SETSYS JES3 statement (if SETSYS JES3 is not specified, HSM defaults to JES2). HSM uses the information about which JESx is being used to determine which pool configuration rules to use.

If you have products that provide similar functions to DFSMShsm, they might have a similar requirement to know which job entry subsystem they will work with.

Another case is RMF. RMF uses information about which job entry subsystem is being used in order to collect subsystem delay information. Specify the subsystem type (JES3 or JES2) on the RESOURCE statement in the ERBRMF04 member.

Finally, the SMFPRMxx member contains a **SUBSYS** parameter that lets you tailor SMF processing for address spaces associated with that subsystem. If you use this capability, your SMFPRMxx member will contain a statement something like:

```
SUBSYS(JES3,EXITS(IEFUSI,IEFACTRT,IEFUJI,IEFACTRT),
       INTERVAL(SMF,SYNC),NODETAIL,
       NOTYPE(4,5,20,34,35,40,80,99))
```

Assuming that you want SMF to continue to work in the same way after the migration to JES2, all you need to do is replicate those statements and change “JES3” to “JES2”.

10.14 Automation

In addition to formal automation products, automation could include such elements as System Rexx execs, JES commands that are issued from jobs, Tivoli NetView for z/OS, and so on. System automation software products typically include scripting languages or REXX execs to assist in managing day-to-day operations. Many of these products are used on JES3 and JES2 systems. The JCL and code scripts they execute will need to be reviewed for possible changes when doing a migration.

10.15 Home-grown ISPF-based tools

Finally, do not forget to check any installation-written ISPF tools that you might have. Examples would be jobs to run program compiles, print files, generate RACF or performance reports, and so on. Check the standard ISPF data set concatenations, especially the ISPF table and skeleton data sets for JES3 JECL statements that would need to be converted.

10.16 Migration tools

When considering a migration from JES3 to JES2, one of the significant costs will be in identifying and modifying JCL/JECL that will not produce the wanted results in a JES2 environment, and in replacing JES3 functions that do not exist in JES2.

Of course, one way to address this is to create your own set of tools to assist with the migration effort. An alternative is to purchase products that will do this for you. Whichever path you choose, the things that your tools will need to address include the following:

- JCL compatibility/migration
- Dependent job control (DJC)
- Deadline scheduling (MDS)
- Dynamic support programs
- JES exit replacement/migration

10.16.1 Migration assistance products

There are products in the marketplace that can assist in making the migration from JES3 to JES2 easier and less time consuming. In addition, these products typically provide a rich set of system management functions. Two vendors that are mature in this space are:

- Thruput Manager AE from MVS Solutions
- zOSEM from Trident Services Inc.

These products take different approaches to migration from JES3 to JES2. A key note to make is these products are designed to run in only JES2 environments. They typically will read in and manage current JES3 JCL/JECL statements and translate them to the appropriate JES2 constructs along with other parameters at execution time to be run in a JES2 environment.

The zOSEM product, which stands for z Operation System Environment Manager, is ISPF panel driven to build criteria on how JES3 JECL will be managed running unchanged JES3 JCL in a JES2 environment. In addition, zOSEM allows for a way to manage JES exits and has a feature to better manage HSM.

Thruput Manager AE works on a proprietary script language similar to REXX to manage the execution of JES3 JECL conversion at run time. This product also helps with HSM management performance as well as managing peaks for a 4-hour rolling average.

The product that best fits your requirements will vary with your environment and which functions your shop is using today and in the future.

Remember to create a list of potential products and exits that your installation might be using today. This list will then be reviewed for functionality differences for JES2.

11

Security considerations

This chapter describes the security-related aspects of replacing JES3 with JES2. The focus will be on protecting JES-managed resources, rather than on giving JES access to the resources it needs.

z/OS JES2 Initialization and Tuning Guide, SA22-7532 and *z/OS JES3 Initialization and Tuning Guide*, SA22-7549 provide detailed information about securing the JES environment using both JES and RACF facilities.

We discuss any RACF entries required during a conversion from JES3 to JES2 and it assumes that the same RACF database will be used in the current and new environments. We also cover current RACF FACILITY classes, data set profiles, initialization parms affecting security, JES exits affecting security, and any other differences in the overall setup.

11.1 Security considerations

This section assumes that you are familiar with RACF and have RACF authority to list the current classes, profiles, and access lists in your test and production environments. It is not unusual to have more stringent security in production environments than in test or development ones, so it is important that you investigate the current setup in *every* environment.

RACF provides many options for securing resources. It is possible that you do not use all the available options. We assume that you want to maintain the same level of security in the JES2 environment that you currently have in JES3. In particular, we assume that you do not want to increase the level of security as part of the migration. It is possible that you will discover additional security features that you would like to exploit, however enabling those features is a separate project from the JES migration.

In this chapter, we cover:

- What does JES itself need access to in order to start and run?
- What security is controlled by JES parms?
- What JES resources are controlled by SAF?

Tip: Throughout this section, we assume that you are using the RACFVARS profile &RACLNDE to represent the NJE NODE ID of your JES3 complex in your RACF profiles and that you will use that same NODE ID when you move to JES2.

11.1.1 JES Started Task

Both JES products need access to various data sets. The initialization and tuning guides for JES2 and JES3 state that the JES started task must be defined to RACF with the TRUSTED attribute. This ensures that JES will be allowed to access whatever data sets you include in its started task JCL or in the initialization parameters.

11.1.2 JES data sets

Even though the TRUSTED attribute ensures that JES can use its resources, you also need to protect them to prevent unauthorized access by other users or jobs. The resources to protect include:

- SPOOL data sets
- Checkpoint data sets
- Initialization data set[1]
- Procedure libraries
- Load libraries
- SPOOL offload data sets

While the specific data sets might be different for JES2 and JES3, apply the same logic and the same access rules to the JES2 equivalent of your existing JES3 data sets.

[1] In JES2, this can be any data set, and is commonly SYS1.PARMLIB.

11.2 Initialization parameters that affect security

Both JES3 and JES2 provide some security features directly. The features are at the system level rather than the user level. For example, you can control whether commands can be included in JCL, but that control applies to *all* jobs, regardless of who submitted the job.

Table 11-1 shows the JES resource, the associated parameter for JES3, and its equivalent in JES2.

Table 11-1 Initialization parameters affecting security

JES resource	JES3 parm	JES2 equivalent
Bypass Tape Label Processing	CIPARM (14th character)	JOBCLASS BLP=
Ability to start an IBM VTAM® interface for RJP (or RJE and NJE for JES2)	COMMDEFN P=	LOGON Password=
RJP Console command authority	CONSOLE[a] LEVEL=	RMT(nnn) CONS= and PASSWORD=
Issue commands from a job	CIPARM AUTH and COMMAND keywords	INTRDR statement AUTH keyword JOBCLASS COMMAND=IGNORE
Ability to issue commands from NJE devices	NJERMT EXPWD=, PWCNTL, SECSIGNON, TLS	NODE AUTH=
Protect ability to sign on to RJP lines	RJPLINE P=	LINE Password=
Protect ability to establish an RJP session (BSC)	RJPTERM P=	RMT(nnnn) Password=
Protect ability to establish an RJP session	RJPWS P=	RMT(nnnn) Password=
Print security classification on output sent to AFP printers	OUTSERV FLASH= SYSOUT=FLASH	PRINTDEF NIFLASH=

a. Keyword ignored when OPERCMDS class is active and RJP workstation is defined to RACF.

Some of the capabilities provided by JES (the ability to use Bypass Label Processing, for example) can also be protected using RACF. If both RACF and JES definitions are in place to protect a resource, the RACF protection will override any definitions that you might have made in JES.

In addition to the standard security functions, JES provides a number of user exits that can be used to provide additional security checking. Table 11-2 displays the security-related JES3 user exits and the equivalent JES2 exit if one exists.

Table 11-2 JES Security-related user exits

JES resource	JES3 exit	JES2 equivalent
Local JES3 Commands	IATUX18	Exit 5
JES3 Commands from NJE/BDT	IATUX35 IATUX56	Exit 5

JES resource	JES3 exit	JES2 equivalent
Print Security Overlay	IATUX19	Exit 1, 15, or 34
TSO CANCEL,STATUS	IATUX30	Exit 22
TSO OUTPUT	IATUX30	none
IATUX58	Modify security information before JES3 security processing	Exit 36
IATUX59	Modify security information before JES3 security processing	Exit 37

11.3 JES-related SAF classes

In general, the SAF profiles to protect JES-related resources are used in the same way. In fact, if you look in *z/OS Security Server RACF Security Administrator's Guide*, SA22-7683, you will see that most RACF references to JES are generic. And in any cases where there are differences between JES2 and JES3, the book will explicitly state JES2 or JES3.

The following sections review the differences in RACF classes between the two JESs. It only describes the classes that are different, all other classes can be used for either JES environment.

11.3.1 JES3 consoles

You cannot use RACF to control access to locally attached JES3 consoles. In JES2, you can use RACF to control access to all consoles. If you use RACF to control access to consoles today, and you have JES3 locally attached consoles, the procedures for using those consoles will change when you move to JES2.

11.3.2 FACILITY class

For RACF to check logons and /*SIGNON passwords, there must be a profile for each workstation in both the FACILITY and USER classes.

In JES3, the name of the workstation is derived from the RJPWS initialization statement for an SNA workstation and from the RJPTERM initialization for BSC.

In JES2, alternatively, the workstation name is of the form RMT*nnnn*, where *nnnn* is the remote workstation number.

Note: Check your list of FACILITY class entries for any profiles called `RJE.*` or `**`. If these profiles exist, you need to create a user ID RMTxxx for each remote to allow signons.

11.3.3 DEVICES class

In JES3, this class protects devices that are defined by the DEVICE statements in the inish deck. The profiles would look similar to the following:

```
DEVICES sysname.dev-class.modelno.ddd
```

sysname	The name of the system
dev-class	The type of device
modelno	The model number of the printer
ddd	The device number associated with the printer

11.3.4 NODES class

The NODES class is used for security of inbound work for NJE nodes.

It is of the format *nodeid.keyword.name* where nodeid is the incoming known nodename, keyword is the type of work, and name is the name of the incoming work.

11.3.5 WRITER class

As NODES was used for inbound work for NJE nodes, the WRITER class is used for outbound work.

In JES3, the WRITER profiles start with JES3.* A corresponding JES2.* could be entered in the transition stage. These would need to be changed to the corresponding JES2.* where applicable.

Also, any WRITER profiles that start with JES3.RJP.* would need to be changed to JES2.RJE.*

SPOOL Offload

In JES2, there is a SPOOL offload function, similar to JES3 Dump Job that needs to be protected in the WRITER class entry

- jes2.LOCAL.OFF1.ST

For reloading, the JESINPUT class is used as referenced in 11.3.6, “JESINPUT class” on page 147.

11.3.6 JESINPUT class

The NODES class is used for inbound work.

JES3 uses the device name as JNAME.

NJERDR is used for JES3 SNA NJE (BDT) receivers and lastnodename is used for JES3 BSC NJE and JES2 NJE receivers.

Reload data after spool offload.

To reload data that was offloaded, the profiles in the JESINPUT class are used:

- OFFn.JR for jobs
- OFFn.SR for SYSOUT

DUMPJOB is used for dump job output in JES3 and OFFn.SR for JES2 spool offload sysout receivers and OFFn.JR for Spool offload Job receivers

11.3.7 JESSPOOL class

Most JESSPOOL classes can be used for either JES3 or JES2, assuming SDSF has been implemented. However, there are a few special cases as noted.

JESNEWS

JESNEWS is what is printed after the output of each job and the entry is dependant of the JES you are running. Therefore, look for the following entries in JESSPOOL:

- JESSPOOL node.JES3.JOB00000.D0000000.JNEWSLCL
- JESSPOOL node.JES3.JOB00000.D0000000.JNEWSRJP
- JESSPOOL node.JES3.JOB00000.D0000000.JNEWSTSO

In a JES2 system, it would be the entry:

- JESSPOOL node.userid.$JESNEWS.STCtaskid.Dnewslvl.JESNEWS

It is recommended that these classes are also put in the global access checking table (GAT), so it would also need to be reviewed and added or removed as appropriate.

Trace Data Sets

For JES2 only, trace data sets need to be protected in the JESSPOOL profile:

- JESSPOOL NODE1.JES.*.*.*.JESTRACE

SDSF references localnode.userid.jobname.jobid for job display

SDSF references localnide.xuserid.jobname.jobid.GROUP.grpname for data set groups

11.3.8 JES SPOOL data sets

Data sets on spool use the JESSPOOL entries of the form:

SYSIN and SYSOUT	*localnodeid.userid.jobname.jobid.dsnumber.name*
JESNEWS	*localnodeid.jesid*.$JESNEWS.STC*taskid*.D*newslvl*.JESNEWS
Trace data sets	*localnodeid.jesname*.$TRCLOG.*taskid.dsnumber*.JESTRACE
SYSLOG	*localnodeid*.+MASTER+.SYSLOG.*jobid.dsnumber*.?

If you use RACFVARS profile &RACLNDE for *localnodeid* and are using wildcards (*.*) for any the other parms, no changes are required.

11.3.9 OPERCMDS class

The syntax for entries in the OPERCMDS profile is of the form:JES3.*command.qualifier* for JES3 commands and JES2.*command.qualifier* for JES2 commands. However, it is not a one-to-one correlation and will require a review from the security administrator starting with the list found in Appendix B, “Comparison of JES3 and JES2 commands” on page 233 using the column denoted OPERCMDS.

As a recommended starting point, make sure that there is an entry for JES2.UNKNOWN with UACC(NONE) so all commands are initially protected.

These options are listed in *z/OS Security Server RACF Security Administrator's Guide*, SA22-7683:

- SETROPTS JES(BATCHALLRACF): Forcing batch users to identify themselves to RACF
- SETROPTS JES(XBMALLRACF): Support for Execution Batch Monitor (XBM) (JES2 only)
- SETROPTS JES(EARLYVERIFY): JES user ID early verification
- SETROPTS JES(NJEUSER): Understanding default user IDs
- SETROPTS JES(UNDEFINEDUSER): Understanding default user IDs

12

Workload Manager considerations

This chapter stresses Workload Manager (WLM) enhancements that complement or interact with JES2 or JES3 classical batch jobs handling. Among other things, it also speaks that from OS/390 V2R4 on, initiator can now be managed by WLM and no longer by the job entry subsystem (whether JES2 or JES3).

Indeed, WLM introduced in 1994, for handling workloads in parallel sysplex environments, was enhanced in 1997 (OS/390 V2R4) for managing resources “a la JES3”, at the sysplex scale.

Indeed, through this enhancement, WLM started to take the role traditionally assigned to the global system in a JES3 configuration.

12.1 JES2 address spaces

Do you need to add a classification rule to WLM to assign the correct WLM service class to all the JES2 address spaces? JES2 itself is in the PPT with a systask set so it will not need attention, other than perhaps having a rule to classify it as a high importance STC. The other spaces, such as JESXCF, FSS, CI, and network must be looked at for your system.

12.2 WLM-managed initiators

WLM manages initiators based on the goals specified in their service class period. In z/OSV1R8, IBM further enhanced this functionality by extending the scope of batch management to be able to optimize the distribution of the batch workload across a sysplex. This functionality is called improved batch initiator balancing.

JES2 Initiator Management

JES2 attempts to use approximately the same percentage of active WLM-managed initiators in each service class on each system. JES2 does not try to use the same number of initiators on each system; the number used is in proportion to the number started on each system. If WLM has started 20 initiators on system A and 10 on system B, and 15 jobs are submitted, 10 will run on system A and 5 on system B.

This balancing only comes into effect when there are idle initiators; most commonly when other jobs have finished but the initiator is still active. If there are no free initiators, jobs run wherever another job finishes, or WLM starts new initiators.

Initiator balancing does not necessarily keep the number of initiators the same, or keep all systems equally busy. It does stop one system from always selecting the work when there are free initiators elsewhere, which gives WLM a better picture of the workload when it is deciding whether to start or stop initiators.

JES3 Initiator Management

GROUP initialization statements combined with MAINPROC, SELECT, and CLASS statements are used to determine the algorithms used for job selection. The more common scheduling functions are discussed here. See *z/OS JES3 Initialization and Tuning Guide*, SA22-7549 for more information.

The GROUP statement defines attributes that JES3 uses for initiator management. Each job class group is defined as containing either WLM-managed or JES3-managed initiators. For JES3-managed initiators, the GROUP initialization statement determines how many initiators to create and when (they can be started at IPL time, manually, or dynamically.). For groups that use WLM-managed initiators, that function is controlled by WLM.

The installation can have JES3 or Workload Manager (WLM) or both manage initiators for batch jobs. JES3 or WLM control of initiators is at the job class group level. To specify whether JES3 or WLM manages initiators for a job class group, the installation uses the MODE parameter on the GROUP initialization statement. If MODE=JES is specified or defaulted, the initiators are managed by JES3. If MODE=WLM is specified, the initiators are managed by WLM. The MODE parameter can also be changed dynamically using the *MODIFY,G command. A group must be either WLM managed or JES3 managed; it cannot be WLM managed on one system and JES3 managed on another.

There are a number of reasons why you might choose WLM initiator management over JES3 initiator management: Fewer and simpler externals exist.

Less externals are needed in JES3 to control WLM-managed initiators and to perform workload balancing. When the service administrator defines the performance goals and classification rules in the WLM policy, the system takes over the job of starting and stopping initiators.

Externals reflect customer expectations.

With JES3 initiator management, it is the installation's responsibility to determine the number of initiators to be started on each system, the correct mix of jobs, and so on. The externals do not reflect an actual performance goal, such as a one-hour turn around time for jobs in class X. How do you translate a one-hour turn around time into initialization statements?

With WLM initiator management, initiators are managed by WLM according to the service classes and performance goals specified in the WLM policy. The performance goals are typically expressed in terms that are found in service level agreements (for example, a one-hour turn around time).

Dynamic, goal-oriented initiator management exists

The system adapts to changing conditions and how well the work is meeting its performance goals. The current JES3 initiator management puts the responsibility on the system programmer/operator to manage the work. Workload balancing algorithms used by JES3 are difficult to define and static in nature; they require an operator command and sometimes a JES3 hot start to change. But even if they can be changed easily, why should human intervention be required? An operating system can better perform these tasks.

Workload balancing across a SYSPLEX is automatic

WLM decides when to start initiators and how many to start based on performance goals and the importance of batch work regarding other work.

Note: WLM management of initiators does not necessarily imply that there will be an equal number of initiators on each system.

12.3 WLM scheduling environments

A WLM scheduling environment is a grouping of resource elements, each element having a specified required state, into a single named entity to be used in making scheduling decisions. The scheduling environment is either available (that is, all the resource elements are in the required state) or unavailable (that is, one or more of the resource elements are not in the required state) on an MVS image. When a scheduling environment is associated with a unit of work, it signifies the requirement for an execution environment providing ALL of the services/conditions needed for successful execution.

WLM scheduling environment is a kind of generalization of JES3 setup. In a JES3 complex, jobs using a WLM scheduling environment (SCHENV = on the job card) only run on systems where that scheduling environment is available.

The same is true for a MAS JES2 configuration for which it has been also a totally new function made available since 1997, starting with OS/390 V2R4.

12.4 WLM classification rules

If the job class of a batch job is used to assign its WLM service class, changes in the job class name that might result from the migration from JES3 to JES2 will need to be taken into account in the classification rules.

12.4.1 Defining z/OS WLM classification rules for performance policies

For virtual servers that run z/OS, you can map service classes from a Unified Resource Manager performance policy to z/OS WLM service classes to achieve end-to-end goal-based performance management for multitier applications. Through classification rules for the IBM defined EWLM subsystem type, z/OS WLM administrators can assign work requests that originate from virtual servers in an ensemble workload to service classes and report classes on z/OS.

The EWLM subsystem work requests include DB2 distributed data facility (DDF) requests that originate from an ensemble, through virtual servers (or hosts) that are classified within a Unified Resource Manager performance policy. Unified Resource Manager performance policies have service classes that specify either discretionary or velocity performance goals that the Unified Resource Manager uses to manage virtual servers in the ensemble workload. When you correlate these IBM zEnterprise® service classes to WLM service classes, you enable z/OS WLM to manage the work differently than it does for local z/OS work:

- z/OS WLM assigns the incoming work request to the appropriate WLM service class immediately, instead of going through the z/OS classification process.
- z/OS WLM can manage the work processed by a z/OS subsystem, such as DDF, according to different performance goals than those used for local z/OS work.
- z/OS WLM also can assign the work requests originating from virtual servers in the ensemble to different report classes than those used for local z/OS work.

The configuration tasks necessary for mapping zEnterprise service classes to WLM service classes are described in *z/OS JES3 Initialization and Tuning Guide*, SA22-7549. Another relevant reference is *z/OS MVS planning: Workload Management, SA22-7602*.

To address WLM, your group can include ensemble workload administrators, performance management administrators, and z/OS WLM administrators.

13

Performance and throughput considerations

Performance, like beauty, is in the eye of the beholder. It can be very subjective but at the same time, can be precisely measured. This chapter discusses the performance attributes of JES2 and JES3. Though it will not be able to answer the question of how JES2 will perform in your environment, it will improve your understanding of the performance differences.

13.1 Performance of JES2 compared to JES3

Measuring performance involves looking at factors such as CPU usage, I/O rates, and memory usage. Comparing one version of a product to another generally involves keeping the workload constant, updating the products being compared, rerunning the workload, and comparing the measurements. But when comparing JES2 to JES3, the problem is complicated by the differences in where each JES does its processing.

In JES3, there is a JES3 address space on each z/OS image. One of those address spaces is designated the JES3 global. The other JES3 address spaces are designated JES3 locals. The JES3 global maintains the job queues, selects work to be processed on each z/OS image in the JESplex, and processes many of the requests made to JES3. The JES3 locals normally do relatively little processing. The local's primary purpose is to act as a hot standby in case the global goes down, and to process requests that must be run on that particular z/OS image. For example, data set locate processing using a catalog or volume that is only accessible on that image, and commands that need processing on the local's image.

In JES2, similar to JES3, each z/OS image has a JES2 address space. However in JES2, all instances of JES2 are the same. There is no centralized global processor responsible for distributing work to each member of the JESplex. Instead, each JES2 address space is responsible for providing work to that z/OS system and for servicing requests from that system. Each address space maintains a copy of the work queues, selects work to process on the system, and handles all requests made of JES on that member.

There is no straightforward comparison that can be made of a JES3 address space and a JES2 address space. Perhaps in a single member JESplex, some comparisons can be made. But there are few of these environments in the real world. You could aggregate the consumption of all the JES3 address space and all the JES2 address spaces, but that would miss another important component, communications used by JES to accomplish the needed tasks. JES3 uses XCF (through a component called JESXCF) to perform the required JESplex communications. JES2 also uses XCF (through JESXCF) but to a lesser degree. However, JES2 uses the checkpoint to share the work queues among the member.

Processing for the XCF messaging done by JES is part of the JESXCF and XCF address spaces. Since JES3 does more of its communications using JESXCF, that also has to be considered when trying to measure performance. JES2 checkpoint processing occurs in the JES2 address space. If the JES2 checkpoint is in a Coupling Facility structure, there is the cost of XES processing and the Coupling Facility overhead that is also part of the performance picture.

The comparison is further complicated by the differences in where each JES performs different parts of their processing. Depending on the function (and sometimes the options), processing might be done in the JES address space or in another address space. Here are some examples of processing differences:

C/I Conversion and interpretation processing converts the JCL that was submitted into an internal representation that can be used by z/OS to run a job. In JES3, C/I processing can be done on the JES3 global or it can be offloaded to a JES3 C/I FSS. In JES2 V2R1, it works the same way. Before that release, conversion was always done in the JES2 address space and interpretation was done when work was selected for execution. That processing might occur on the system that the job was submitted on, but it does not necessarily have to occur there.

Input
Input processing creates the basic structure for a new job that is being submitted. During the input phase, a first pass through the submitted JCL is performed, processing any JECL cards and building the required instream data sets. In JES3, input processing is performed in the JES3 address space on the global. In JES2, internal reader and NJE over TCP/IP processing is primarily done in the submitting (application or NETSERV) address space. For other input sources, processing is done in the JES2 address space.

Extended Status
Extended status is an interface into JES to query the status of jobs and output in the system. JES3 does most of the processing in the JES3 address space on the global. All the results are sent from the global to the requesting address space via JESXCF. In JES2, all the processing for extended status is performed in the requesting address space.
Processing for the JES Property SSI and the JES Device SSI also is done by the global address space in JES3 and in the requesting address space in JES2.
These interfaces are used by products like SDSF, FTP, z/OSMF, and others that display information about jobs and devices associated with JES.

SNA NJE
NJE processing involves CPU intensive processing to build the data records required to transmit data across NJE. Since JES3 uses BDT to perform SNA NJE processing, that CPU intensive processing occurs in the BDT address space. JES2 does all the SNA processing in the JES2 address space.

Application SPOOL I/O
When applications write SYSOUT to SPOOL, they use a JES-specific access method under the covers to perform the SPOOL I/O. In JES3, SPOOL write I/Os from all address spaces on a system are queued to a single write queue (per SPOOL extent) to be written. The actual STARTIO request will be issued in whatever address space is in control when the current I/O ends or when the first element is added to the queue. JES3 maintains one active I/O per SPOOL extent per system.
In JES2, each address space performs its own SPOOL I/O. Each address space can start up to 2 I/Os per SPOOL extent. Each I/O is recorded (charged) to the address space that issued the I/O. I/O queuing priority also applies to SPOOL I/Os. JES2 can also take advantage of multiple paths (PAV) to the SPOOL volumes.

Monitor
Each JES has a real-time monitor that looks at what the JES is doing and reports anomalous situations. These monitors tend to poll JES looking for processes that are looping or waiting for extended periods of time. By their nature, they tend to use noticeable amount of CPU. In JES3 the monitor runs in the JES3 address space. In JES2, there is a separate address space (JES2MON) that performs the monitoring.

There are other more subtle examples of these differences. Because of the complexity of where processing is done and trying to identify what processing is JES-related versus application-related, we cannot make simple statements about how much CPU one JES uses compared to the other. What we can do is look at factors like overall system overhead to understand the requirements of each JES. We can also look at wall clock time to complete specific activities as a way to compare the performance of the two JESs.

13.1.1 Steady state processing

When trying to compare the performance of JES3 and JES2, one easy environment to measure is steady state processing. In an idle system how much CPU is being used by the JESs? In JES3's case, there is nothing for it to do when there are no requests to process and thus the global is mostly idle. It is awakened by requests to perform processing as they occur. However, each JES2 acts independently and primarily discovers work by reading the JES2 checkpoint data set. In effect, it is polling the checkpoint on a periodic basis to see whether any other member has added work that needs to be processed.

As a result, even in an idle system, JES2 is constantly reading and writing the JES2 checkpoint data set and consuming resources. How often this is done can be controlled by JES2 tuning parameters on the JES2 MASDEF statement. For more information about tuning JES2, see *z/OS JES2 Initialization and Tuning Guide*, SA22-7532.

Another thing that occurs on an idle JES2 system is the maintenance of data areas used by various interfaces. Even though JES2 processes extended status requests in the user address space, the data that is used to satisfy the request comes from a copy of the JES2 work queues that is maintained in a data space. The data space copy is kept up to date every time the JES2 checkpoint is written. Similarly, device information is stored in 64-bit common storage for use by the JES device information requests.

13.2 Throughput considerations

Where performance is often looked at as how much resource is needed perform a task, throughput is the amount of work that can get done in a given time. In some cases, throwing more resources at a problem can improve throughput. However, in some cases, throwing more resources at a problem will not get it done any faster.

Often the solution to a throughput problem is not more resources but a better way of utilizing the existing resources to get more stuff done in a unit of time.

13.2.1 MAS considerations

Both JES3 and JES2 have a single main task that is running in the JES address space. This task services the requests that are made of each JES.

In JES3 though, each JES3 address space has a main task, and most of the work occurs in the main task on the JES3 global. This task must handle most of the requests made of JES3 throughout the JESplex.

In JES2, the main task in each JES2 address space handles JES2 requests that originate on that member (they are not sent to other members to process). However, the JES2 main task must access the JES2 checkpoint to respond to many requests. Since only one member of a MAS can access the checkpoint at a time, there can be only one main task actively processing these requests in the JES2 MAS. However, not all processing done by the JES2 main task requires access to the checkpoint, meaning that some progress to be made on requests when the checkpoint is not owned.

If JES2 is running as a single member MAS, it can constantly own the checkpoint and there are no delays introduced by not owning the checkpoint. However, when a second member joins a MAS, the checkpoint access introduces a delay processing many requests (this is known as a MAS penalty).

Tuning of the JES2 checkpoint can greatly reduce the impact of checkpoint serialization. This can make the checkpoint available in a more timely manner. But there is only so much tuning that can be done for the checkpoint. Eventually, you need to start thinking about parallelizing processes to improve throughput.

Submitting jobs

Submitting jobs to JES involves allocating an internal reader, writing a JCL stream to the internal reader, then closing and freeing the internal reader. As part of this process, JES must examine every card that is submitted, process cards that have information needed at JES input processing time (JECL, JOB card information, and so on), and separate instream data from JCL cards.

In JES3, this processing is done by writing to SPOOL all the cards passed to the internal reader for the job. Once finished, the application must close the internal reader or issue an **ENDREQ** macro to get the SPOOL data set queued to the JES3 global. The JES3 local does not get involved in the submitting of jobs. So when the address space completes writing the job to SPOOL, a request is queued to the global to process the job. The task in the submitting address space will wait for input processing to complete before returning from the **ENDREQ** or **CLOSE** that was issued. To perform input processing for the job, the global reads the cards written to the internal reader data set and process the JCL stream. Each stream (internal reader submission) is processed sequentially, however, multiple streams can be processed by separate threads (DSPs) in the JES3 main task. In this scheme, adding more DSPs that can process internal reader data streams (via the **OPTIONS INTRDR=** keyword) can improve throughput but only to the point that the single main task TCB can interleave I/Os that read and write various data. Parallelizing internal reader submission (using 20 internal readers each submitting one job versus one internal reader submitting 20 jobs) only helps if there are enough DSPs to process the work.

JES2 performs input processing in a completely different way. In JES2, the bulk of the input processing work occurs in the address space that allocated the internal reader. However, the JES2 main task must place the job in the job queue at the start of the job submission (when the JOB card is detected) and move the job to its next queue at the end the job. These processes must access the checkpoint and are subject to checkpoint delays. Due to the nature of the main task processing, a single checkpoint update (delay) can process one or 100 or more jobs. So by submitting jobs using multiple internal readers, you can greatly increase the number of jobs per minute that can be submitted. One hundred internal readers can submit 100 jobs (one job per internal reader) in the same time one internal reader takes to submit one job. In JES2's case, there are no resources that can make input processing for a single internal reader go faster, but by parallelize the processing (using multiple internal readers), you can get a significant throughput improvement.

Many job scheduler products have options to increase the number of internal readers they use to submit jobs. Though these options might not have been needed in JES3, use them when you run JES2.

13.2.2 Setting limits and counts

Limits are a way to manage resources in a system. Limits can be used to manage a constrained resource like memory, or they can be used to limit the effect of runaway processes. Counts control the number of processors available to accomplish a function. These processes are consumers of resources. Some processes can be enhanced just by adding resources. Other processes can be enhanced by adding more processors. Still others need both more resources and more processors.

There are a number of ways to manage constrained resources. Applications could choose to not have any external limit and react when a resource is no longer available. Alternatively, an application could set a hard limit that causes processing to wait or fail when the limit is reached. Or the application could include code that cleverly manages limits.

Understanding what limits exist and how to properly manage those limits can improve throughput and prevent outages.

JES2 processing generally either has no limits or a hard limit. No limit is something that is only limited by the architecture (such as a data space can only be 2 gigabytes in size). A hard limit is something specified via a JES2 parameter (such as the number of data buffers in JES2). Hard limits can be changed but require someone to take an action to do so.

As we will see, most limits in JES2 can be displayed using commands and products such as SDSF. However, no matter how many ways an application notifies users when limits are being reached, it is no good if no one is looking or they are looking in the wrong place. JES2 has a number of ways to assist installations in manage resource limits. Understanding this will ensure that JES2 continues to operate smoothly.

Monitoring JES2 resources

The resource limits in JES2 are monitored by a process in JES2 called the *resource monitor*. The monitor is looking for resources whose use has reached an installation-specified limit. Every resource has an installation-specifiable warning level used by the resource monitor to determine at what point a HASP050 message will be issued. The warning level is specified as a percentage on **WARN=** keywords on various JES2 statements. The default is always **WARN=80** (start issuing warning messages when the resource usage reaches 80%).

When a resource usage reaches the warning level, a highlighted HASP050 message is issued. The HASP050 message contains the current utilization percentage. For example, a shortage of SPOOL space (Track GroupS – TGS) message would look like this:

```
$HASP050 JES2 RESOURCE SHORTAGE OF TGS - 93% UTILIZATION REACHED
```

Every 30 seconds the current utilization of any resource that whose usage has exceeded the threshold is examined. If the utilization has dropped below the warning level, the message is deleted (DOMed). If the percentage has changed but is still above the threshold, the old message is deleted (DOMed) and a new message issued with the new percentage. If the utilization is nearly 100%, the old message is deleted and a new message will be issued every 30 seconds. This ensures that the installation is reminded that the resource has been exhausted.

If this is a MAS-scope resource (such as SPOOL space), the first member that discovers the shortage issues the message. Other members defer messaging to that member. This prevents a flood of HASP050 message from each member. However, if resource utilization reaches near 100%, the HASP050 message is issued on every member.

The HASP050 message for some resources includes additional information to help determine a more appropriate resource limit. This occurs when a process requires a resource that has been exhausted. In this case, a longer form of the HASP050 message is used as shown in Example 13-1.

Example 13-1 Example of a long form HASP050 message

```
*$HASP050 JES2 RESOURCE SHORTAGE OF BUFX - 100% UTILIZATION REACHED
A TOTAL OF 292 BUFX ARE CURRENTLY DEFINED, OF WHICH:
   292 (100%) ARE IN USE
   3 (1%) ARE BEING WAITED FOR
   0 PROCESSORS REQUESTED BUFX BUT DID NOT WAIT
   THE LARGEST UNFULFILLED REQUEST WAS FOR 0 BUFX
A MINIMUM OF 295 BUFX IS REQUIRED TO SATISFY CURRENT DEMAND
```

In this example, the BUFX resource (specified on **BUFDEF EXTBUF**) has been completely exhausted and at least 295 buffers are needed to satisfy current demand. Use an operator command to increase the limit (to 295 plus at least another 20% so that you will not be above the warning threshold again). The long form of the message is used primarily for buffers used by multiple processes in JES2 (such as data, control block, VTAM, BSC, and header buffers).

Many installations automate on the HASP050 message and based on the resource, trigger a text message or send an email to someone responsible for the system. However, in many installations, the HASP050 message has become so frequent that it is ignored. This typically happens when the warning level for that resource is set too low and the system is normally operating above the warning level for some resources. As a result, the operations staff learns to ignore that HASP050 message because of the frequency of it, and end up ignoring *all* HASP050 messages.

Because of this, you must scan SYSLOGs for instances of HASP050 messages and take the appropriate action to address the cause of the message. You can either raise the resource limit, raise the warning level, or remove monitoring of the resource (by setting the warning level to 0). This ensures that you only see the HASP050 message when there is a problem and operations can be told to take action whenever they see the HASP050 message.

JES2 resources usage history

One of the functions of the JES2 health monitor is to maintain a history of the utilization of the major JES2 resources. The history is maintained in memory and encompasses the life of this JES2 subsystem. There are two ways to retrieve this information. The JES2 command **$JDHISTORY** will display this on the console as shown in Example 13-2.

Example 13-2 Display of the history of JES2 resource usage

```
$HASP9130 D HISTORY 866
$HASP9131 JES2 BERT     USAGE HISTORY
DATE     TIME         LIMIT     USAGE       LOW      HIGH   AVERAGE
-------- -------- --------- --------- --------- --------- ---------
2012.250 11:00:01       650       539       389       539       435
2012.250 10:27:14       650       389       129       490       310
$HASP9131 JES2 BSCB     USAGE HISTORY
DATE     TIME         LIMIT     USAGE       LOW      HIGH   AVERAGE
-------- -------- --------- --------- --------- --------- ---------
2012.250 11:00:01        42         0         0         0         0
2012.250 10:27:14        42         0         0         0         0
$HASP9131 JES2 BUFX     USAGE HISTORY
DATE     TIME         LIMIT     USAGE       LOW      HIGH   AVERAGE
-------- -------- --------- --------- --------- --------- ---------
2012.250 11:00:01       300       290         0       298       268
2012.250 10:27:14       292         0         0        22         2
$HASP9131 JES2 CKVR     USAGE HISTORY
DATE     TIME         LIMIT     USAGE       LOW      HIGH   AVERAGE
-------- -------- --------- --------- --------- --------- ---------
2012.250 11:00:01         2         0         0         0         0
2012.250 10:27:14         2         0         0         0         0
$HASP9131 JES2 CMBS     USAGE HISTORY
DATE     TIME         LIMIT     USAGE       LOW      HIGH   AVERAGE
```

```
-------- -------- --------- --------- --------- --------- ---------
2012.250 11:00:01       201         0         0         0         0
2012.250 10:27:14       201         0         0         0         0
$HASP9131 JES2 CMDS     USAGE HISTORY
DATE     TIME         LIMIT     USAGE       LOW      HIGH   AVERAGE
-------- -------- --------- --------- --------- --------- ---------
2012.250 11:00:01       200         0         0         0         0
2012.250 10:27:14       200         0         0         0         0
$HASP9131 JES2 ICES     USAGE HISTORY
DATE     TIME         LIMIT     USAGE       LOW      HIGH   AVERAGE
-------- -------- --------- --------- --------- --------- ---------
2012.250 11:00:01       100         0         0         0         0
2012.250 10:27:14       100         0         0         0         0
$HASP9131 JES2 JNUM     USAGE HISTORY
DATE     TIME         LIMIT     USAGE       LOW      HIGH   AVERAGE
-------- -------- --------- --------- --------- --------- ---------
2012.250 11:00:01      9999       212       111       212       142
2012.250 10:27:14      9999       111         0       211       104
$HASP9131 JES2 JOES     USAGE HISTORY
DATE     TIME         LIMIT     USAGE       LOW      HIGH   AVERAGE
-------- -------- --------- --------- --------- --------- ---------
2012.250 11:00:01       200        57         6        57        21
2012.250 10:27:14       200         6         0       104        46
$HASP9131 JES2 JQES     USAGE HISTORY
DATE     TIME         LIMIT     USAGE       LOW      HIGH   AVERAGE
-------- -------- --------- --------- --------- --------- ---------
2012.250 11:00:01       500       212       111       212       142
2012.250 10:27:14       500       111         0       211       104
$HASP9131 JES2 LBUF     USAGE HISTORY
DATE     TIME         LIMIT     USAGE       LOW      HIGH   AVERAGE
-------- -------- --------- --------- --------- --------- ---------
2012.250 11:00:01       166         0         0         1         1
2012.250 10:27:14       166         0         0         1         0
$HASP9131 JES2 NHBS     USAGE HISTORY
DATE     TIME         LIMIT     USAGE       LOW      HIGH   AVERAGE
-------- -------- --------- --------- --------- --------- ---------
2012.250 11:00:01        23         0         0         0         0
2012.250 10:27:14        23         0         0         0         0
$HASP9131 JES2 SMFB     USAGE HISTORY
DATE     TIME         LIMIT     USAGE       LOW      HIGH   AVERAGE
-------- -------- --------- --------- --------- --------- ---------
2012.250 11:00:01        52         0         0         0         0
2012.250 10:27:14        52         0         0         0         0
$HASP9131 JES2 TBUF     USAGE HISTORY
DATE     TIME         LIMIT     USAGE       LOW      HIGH   AVERAGE
-------- -------- --------- --------- --------- --------- ---------
2012.250 11:00:01       104         0         0         0         0
2012.250 10:27:14       104         0         0         0         0
$HASP9131 JES2 TGS      USAGE HISTORY
DATE     TIME         LIMIT     USAGE       LOW      HIGH   AVERAGE
-------- -------- --------- --------- --------- --------- ---------
2012.250 11:00:01       525       484       230       484       309
2012.250 10:27:14       525       230        21       525       254
$HASP9131 JES2 TTAB     USAGE HISTORY
DATE     TIME         LIMIT     USAGE       LOW      HIGH   AVERAGE
```

```
-------- -------- --------- --------- --------- --------- ---------
2012.250 11:00:01         3         0         0         0         0
2012.250 10:27:14         3         0         0         0         0
$HASP9131 JES2 VTMB     USAGE HISTORY
DATE     TIME         LIMIT     USAGE       LOW      HIGH   AVERAGE
-------- -------- --------- --------- --------- --------- ---------
2012.250 11:00:01        24         0         0         0         0
2012.250 10:27:14        24         0         0         0         0
$HASP9132 MAIN TASK SAMPLING PERCENT HISTORY
DATE     TIME    COUNT ACTIVE W-DISP   IDLE   WAIT L-LOCK N-DISP PAGING
-------- -----  ------ ------ ------ ------ ------ ------ ------ ------
2012.250 11:00   46885   0.06   0.00  99.93   0.00   0.00   0.00   0.00
2012.250 10:27   36608   1.84   0.00  97.91   0.24   0.00   0.00   0.00
$HASP9133 JES2 ERROR HISTORY
DATE     TIME         MAIN  DISTERR     CBIO  SUBTASK   NODUMP    OTHER
-------- -------- -------- -------- -------- -------- -------- --------
2012.250 11:00:01        0        0        0        0        0        0
2012.250 10:27:14        0        0        0        0        0        0
$HASP9134 JES2 <16M USER   STORAGE USE HISTORY (PAGES)
DATE     TIME      REGION   USAGE     LOW    HIGH AVERAGE
-------- -------- ------- ------- ------- ------- -------
2012.250 11:00:01   1,274     357     341     357     356
2012.250 10:27:14   1,274     341     299     341     340
$HASP9134 JES2 <16M SYSTEM STORAGE USE HISTORY (PAGES)
DATE     TIME      REGION   USAGE     LOW    HIGH AVERAGE
-------- -------- ------- ------- ------- ------- -------
2012.250 11:00:01   1,274      88      88      90      88
2012.250 10:27:14   1,274      88      61      90      86
$HASP9134 JES2 >16M USER   STORAGE USE HISTORY (PAGES)
DATE     TIME      REGION   USAGE     LOW    HIGH AVERAGE
-------- -------- ------- ------- ------- ------- -------
2012.250 11:00:01 476,928 163,550 163,450 163,551 163,541
2012.250 10:27:14 476,928 163,450 162,984 163,452 163,432
$HASP9134 JES2 >16M SYSTEM STORAGE USE HISTORY (PAGES)
DATE     TIME      REGION   USAGE     LOW    HIGH AVERAGE
-------- -------- ------- ------- ------- ------- -------
2012.250 11:00:01 476,928   3,422   3,422   3,464   3,422
2012.250 10:27:14 476,928   3,422   3,339   3,474   3,426
```

Entries in this display include one line for every time interval. Intervals start at the top of every hour and when the monitor is started (this system was IPLed at about 10:20, thus the first entry). The **$JDHISTORY** command by default displays all resources and up to 24 hours of history. The operands on the command can alter what is displayed. See *z/OS JES2 Commands*, SA22-7526 for more information about the command syntax.

The response includes information about other resources that JES2 uses. The sampling history contains information about what the main task is doing at each sample (active, idle, waiting, and so on) and there is also a history of the memory usage in the JES2 address space.

Another way to view the resource usage history is to use the SDSF RM display. By default, it displays only the current interval, but additional history is available. One advantage to the SDSF display is that it allows you to set a new limit for a resource or adjust the warning level via an overtype on the window. Example 13-3 on page 164 contains a sample display.

Example 13-3 Sample SDSF RM display

```
Display  Filter  View  Print  Options  Search  Help
-------------------------------------------------------------------------------------------
SDSF RESOURCE MONITOR DISPLAY  SY1                     LINE 1-17 (17)
COMMAND INPUT ===>                                            SCROLL ===> CSR
PREFIX=*  DEST=(ALL)  OWNER=*  SYSNAME=SY1
NP   RESOURCE SysId Status     Limit  InUse  InUse% Warn% IntAvg IntHigh IntLow OverWarn%
     BERT     IBM1  WARNING      650    541  83.23    80    464     541    389     49.22
     BSCB     IBM1                42      0   0.00    80      0       0      0      0.00
     BUFX     IBM1  WARNING      300    290  96.66    90    274     298      0     93.46
     CKVR     IBM1                 2      0   0.00    80      0       1      0      0.00
     CMBS     IBM1               201      0   0.00    80      0       0      0      0.00
     CMDS     IBM1               200      0   0.00    80      0       0      0      0.00
     ICES     IBM1               100      0   0.00    80      0       0      0      0.00
     JNUM     IBM1              9999    213   2.13    80    162     213    111      0.00
     JOES     IBM1               200     57  28.50    80     31      57      6      0.00
     JQES     IBM1               500    213  42.60    80    162     213    111      0.00
     LBUF     IBM1               166      0   0.00    80      0       1      0      0.00
     NHBS     IBM1                23      0   0.00    80      0       0      0      0.00
     SMFB     IBM1                52      0   0.00    80      0       0      0      0.00
     TBUF     IBM1               104      0   0.00     0      0       0      0      0.00
     TGS      IBM1  WARNING      525    487  92.76    80    358     487    230     49.28
     TTAB     IBM1                 3      0   0.00    80      0       0      0      0.00
     VTMB     IBM1                24      0   0.00    80      0       0      0      0.00
```

The information displayed contains some additional columns that the **$JDHISTORY** does not include. The **Status** column tells you if the current utilization is above the warning level and the **OverWarn%** column tells you the percent of time in the interval the resource was over the warning level. This percentage is very useful to determine if the shortage is due to a short-term spike in usage or indicative of a longer term trend.

Note: The list of resource short names and the commands that control them can be found in the description of the $HASP050 message in *z/OS JES2 Messages*, **SA22-7537**.

JES2 resource limits

Limits can be grouped into checkpoint limits (MAS scope limits) and local limits (single-member-scope limits). The key difference is the impact of a shortage and the effort needed to increase or manage the limit.

Checkpoint limits

The JES2 checkpoint, whether stored on a DASD data set or in a CF structure is limited in size. This limit is imposed by the media that the checkpoint is stored in and can only be altered by installation actions. As a result, data structures placed in the checkpoint are also limited in number. These structures are also limited by the internal structure of the data.

Because these resources exist in the checkpoint, any shortage impacts all members of the MAS. To prevent a MAS-wide impact, these resources must be monitored and appropriate notifications made when a shortage exists.

The following is the list of checkpoint-related limits:

JOBDEF JOBNUM	This defines the number of jobs that JES2 can store in the checkpoint at any one time. This is similar to the *joblim* value on the **OPTIONS JOBNO** JES3 initialization statement. When this value is reached, no new jobs (started tasks, time sharing users, or batch jobs) might enter

the system. New jobs will wait in their appropriate input processing routine for existing jobs to be purged.

JOBDEF RANGE

This defines the valid range of job numbers that can be assigned on this system. This is similar to the *lowest* and *highest* values on the **OPTIONS JOBNO** JES3 initialization statement. Jobs entering from NJE (including SPOOL reload processing) can be assigned numbers outside this range if **JOBDEF RASIGN** is set to YES. If all the job numbers in the specified range are in use, new jobs entering the system that require a job number within the range (non-NJE jobs for example) will wait in their appropriate input processing routine for an appropriate job number to be freed. The number of jobs in the range must be at least as large as the number of jobs defined on **JOBDEF JOBNUM**.

OUTDEF JOENUM

Unlike JES3, JES2 groups SYSOUT data sets for processing when a spin data set is created or when a job goes through output processing. When this is done, the group of SYSOUT data sets that will be processed together is said to be placed into a Job Output Element (JOE). For JES2 to process a group of SYSOUT data sets it must be grouped into a JOE. You need to define enough JOEs to hold all the spin SYSOUT data sets and non-spin output data sets that will be ready to process on SPOOL. If there are no JOEs available, spin data sets will be placed in an unspun state and processed when JOEs become available. Jobs that complete will wait, queued for output phase processing. However, jobs will continue to run even though their output cannot be processed. In addition to output not being processed, SPOOL utilization will increase since data sets and jobs are not being purged. A general rule of thumb is to define **JOENUM** to be at least 2.5 times **JOBNUM**.

CKPTSPACE BERTNUM

Another JES2 unique concept is BERTs. BERTs address the need to both extend the contents of data structures and add new data structures to the JES2 checkpoint. BERTs are generic 64-byte blocks of data that are stored in the JES2 checkpoint. They are used to extend the job and output group control blocks with optional data and they also store information on job class properties, duplicate job names, and WLM service class queues. The number of BERTs associated with an object depends on the data being stored in them. So their usage can change based on the property of the jobs and SYSOUT groups represented in the checkpoint. Because of their broad usage, a shortage of BERTs can create extreme problems for a JES2 system. JES2 takes a number of actions as the number of free BERTs falls including stopping SYSOUT creation (similar to a JOE shortage), stopping batch job input processing, and eventually stopping all job input processing. However, running out of BERTs has caused systems to unexpectedly fail in ways that forced a JES2 cold start.

JES2 will automatically increase the limit for BERTs if **JOBNUM** or **JOENUM** is increased and **BERTNUM** is outside what is considered normal. The checkpoint data set must be large enough for the increased number of elements being requested and the potential increase in the number of BERTs.

Warning: Do not allow your MAS to run out of BERTs. If you do encounter a severe BERT shortage, do not shut down JES2 or IPL the system. You must resolve the BERT shortage via operator commands such as increasing the number of BERTs or purging jobs that use BERTs before restarting a JES2 member. Failure to do so might force a JES2 cold start.
To increase the number of BERTs, you might need to decrease some other resource in the checkpoint to free up space in the checkpoint before increasing the number of BERTs.

For more information about BERTs, see "BERT shortage special processing" on page 169.

`SPOOLDEF TGSPACE` JES2 keeps track of what SPOOL space is available using a bitmap in the JES2 checkpoint. Each bit represents a track group of SPOOL space. The size of the track group bitmap is controlled by the **`MAX`** parameter on **`SPOOLDEF TGSPACE`**. This is not the amount of total SPOOL space that is available for use by JES2. Rather it is the amount that can be defined to JES2. If this value is too low, JES2 will not be able to start new SPOOL volumes. However, if the amount of defined SPOOL space is exhausted, jobs will stop processing data. Also, since job termination uses a small amount of SPOOL space, jobs often cannot come out of execution if they have run out of SPOOL space and no additional space is available. Unlike JES3, JES2 does not reserve SPOOL space for use by a certain class of jobs.

Non-checkpointed limits

There are a number of limits that deal with how storage is used. These limits prevent a process from consuming all available storage, thus ensuring that some storage will be available for general short term use. If the limit of any of these resources is reached, the process requesting the storage normally waits for elements to be freed. However, in some cases, the limit will cause additional processing that needs the resource to fail.

The following is the list of local (non-checkpoint-related) limits:

`TPDEF BSCBUF` These are buffers used in BSC communication. They are used for NJE (including NJE over CTCs) and RJE. These buffers are in 24-bit storage (below the 16-megabyte line). If you do not have BSC lines (**`LINE`*`xxxx`* `TYPE=`*`unit_addr`***), set the maximum for these to 0. If you *do* have BSC lines, JES2 calculates a default based on the number of lines defined. The default must be used for this parameter unless buffer shortages are reported by the HASP050 message. A shortage of these buffers will prevent BSC connections from being established. Because these buffers are in 24-bit virtual and real storage, it is recommended if you avoid using BSC for NJE or RJE, and that BSC lines are deleted and the number of BSC buffers defaulted (or be set to) 0.
If you dynamically add BSC lines using an operator command, this value must also be increased (by command) to prevent any shortages.

`TPDEF SNABUF` These are buffers used in SNA communication. They are used for both NJE and RJE. These buffers are in 31-bit storage (above the 16-megabyte line). If you do not have SNA lines (**`LINE`*`xxxx`* `TYPE=SNA`**), you must set the maximum for these at 0. If you do have SNA lines, JES2 will calculate a default based on the number of lines defined. The default must be used for this parameter unless buffer shortages are reported by the HASP050 message. A shortage of these buffers will prevent SNA connections from being established. If you are not using SNA for NJE or RJE, you can delete the SNA lines and set the number of SNA buffers to zero (or allow it to default to zero). If you dynamically add SNA lines using an operator command, this value must also be increased (by command) to prevent any shortages.

`BUFDEF BELOWBUF`	These are buffers in the JES2 address space used to perform I/O to JES2 data blocks (SYSOUT data sets). There are a number of older functions in JES2 (in particular JES2-managed printers, including RJE) that use 24-bit buffers (below the 16-megabyte line). JES2 will calculate the default number of buffers based on a number of initialization parameters. This is normally sufficient for most installations. However, some installations can benefit from an increased number of these buffers. The HASP050 message provides extended information on what to set this value to if a shortage is encountered. A shortage of these buffers will delay processing of SYSOUT. Over-specifying this parameter can result in storage shortages for other processes, especially those that require 24-bit storage.
`BUFDEF EXTBUF`	These are buffers in the JES2 address space used to perform I/O to various control blocks. Most processes in the JES2 address space use these buffers for I/O. They are in 31-bit virtual and real storage. They also have a small associated data area in 24-bit storage that is used for I/O. JES2 will calculate the default number of buffers based on a number of initialization parameters. This is normally sufficient for most installations. However, some installation can benefit from an increased number of these buffers. The HASP050 message provides extended information on what to set this value to if a shortage is encountered. A shortage of these buffers will delay processing of jobs and SYSOUT. Over-specifying this parameter can result in storage shortages for other processes (especially with the 24-bit area that is associated with these buffers).

Note: JES2 uses both 24-bit (`BELOWBUF`) and 31-bit (`EXTBUF`) buffers to perform I/O. With any new releases, the distribution of buffers used can change greatly as functions are reworked to use more 31-bit buffers and as work is shifted out of the JES2 main task. If a non-default number of buffers is specified in the JES2 initialization deck, when a new release of JES2 is being implemented, a test must be done to determine the new default JES2 is using, and the buffer counts adjusted accordingly.

`CONDEF BUFNUM`	When JES2 issues messages, it does not issue a WTO out of the JES2 main task since a WTO could do an MVS WAIT. To prevent the wait, WTOs are issued out of a subtask in the JES2 address space. This is similar to what JES3 does with its WTOs. The WTOs are queued to the subtask using the CMB data area. The CMBs are in 31-bit virtual storage. To prevent a runaway process from flooding the system with messages, the number of CMBs is limited by this parameter. JES2 defaults this value to 100. If there is a shortage of CMBs, the requester of the WTO will be suspended ($WAITed) until a CMB is available. This can slow command response processing. Over specifying this value would impact available 31-bit storage. If the HASP050 message reports a shortage, increase this value as needed. There is the concept of a single emergency CMB for use by the

	command processor when the pool of CMBs is exhausted. This will allow commands to be processed, though very slowly.
`CONDEF CMDNUM`	When a JES2 command is issued by the operator or received over NJE, a command buffer in ECSA is used to hold the command (31-bit common storage). These commands are queued to the JES2 command processor for processing. Unlike JES3, JES2 has one main processor of all commands (they are not distributed to other processes within JES2 for processing). The exception is JES2 monitor commands, which are processed by the monitor in the monitor address space. There are two instances of the JES2 command processor, but most commands are processed by the main command processor. If the command processor becomes backed up, new commands can consume command buffers until they are exhausted. At that point, no new JES2 commands can be entered. Increasing the number of command buffers will use additional ECSA. If the HASP050 message indicates a shortage of command buffers, investigate the source and nature of the command to determine why they are not being processed in a timely manner. If necessary, increase the number of command buffers.
`CKPTDEF VERSIONS`	Checkpoint versions allow an address space to access checkpoint data (job and output queues) without having to send a request to the JES2 address space. They provide a stable copy of the checkpoint that the application (either directly or through some other API) can use to satisfy query requests. JES maintains these versions as a background process that runs whether there are requester for versions or not. There is always an available current version and one that is ready to be updated. When an application is assigned a version, JES2 is locked out of updating that particular version (note that one version is assigned to multiple applications if it was current when they requested a version). If the application takes a long time to process the version (and thus keeps it locked), JES2 will create an additional version to satisfy the requirement of a current version and one waiting to be updated. The more applications that request and hold versions, the more concurrent versions are needed. `CKPTDEF VERSIONS` puts an upper limit on the number of versions that can exist at any time. Allowing more versions uses additional real storage to manage the copies of the checkpoint. The default of 2 is low for most systems that have processes that use tools to query or manage the JES2 work queues. A value of 10 is reasonable for most systems. If there is an HASP050 message warning of a shortage of versions, investigate the users of versions to determine why the applications using them are holding the version so long.
`TPDEF SESSIONS`	Every JES2 SNA session (NJE or RJE) that is established uses a data area called an ICE to represent the LU-LU session. These are in 31-bit storage in the JES2 address space. If there are no data areas available, JES2 will be unable to complete any additional SNA session. In general, you must define at least as many sessions as you have SNA lines (this is the default). However, certain RJE connections can have multiple LUs for a single RJE (a single line). If that is the case in your environment, you must define additional sessions for those RJEs.
`NJEDEF HDRBUF`	SNA and BSC NJE need to process NJE data headers that can be up to 32 KB. Header buffers are used to hold these buffers. They are also

	used in some cases outside of NJE to hold NJE headers. The header buffers reside in 31 bit-storage in the JES2 address space. JES2 calculates a default for the number of header buffers based on various initialization statements. Unless the HASP050 message indicates a shortage, the default value is normally sufficient.
`SMFDEF BUFNUM`	Like many things in JES2, SMF records cannot be written in the JES2 main task because of concerns for MVS waits. SMF records created by the main task are placed in SMF buffers and queued to a subtask for processing. These buffers are used to queue the SMF records to the subtask. The buffers are in 31-bit storage in the JES2 address space. If the number of buffers is exhausted, processes needing buffers will end up being suspended until there are free buffers. The default number of SMF buffers is a fixed value. Depending on your workload and the ability of the SMF process to record the SMF records, you might need to increase the number of SMF records you have defined. Monitor the HASP050 message to see if the number you have defined is sufficient. *Note this is one of the few buffers whose limit cannot be increased via operator command.*
`TRACEDEF TABLES`	Trace tables are used when JES2 tracing is active (using the `TRACEDEF` statement). Tracing can be done in memory only or they can be records to a SYSOUT data set. The more tables that you define, the more data that can be buffered waiting for tables to be written. In general, more smaller tables are better than fewer large tables. If there are no trace tables available when a trace record needs to be written, the trace record is discarded. Generally, IBM service recommends a trace table size and number for traces they request. Trace tables are in 31-bit common storage (ECSA). The size of a table and the number are controlled by the `TRACEDEF` statement.

BERT shortage special processing

The block extension reuse table (BERT) resource, because of the nature of how it is used, requires some special processing and careful monitoring. BERTs are extensions to multiple control blocks. They are variable in size and have the property that if the extension is not needed for a particular control block, no storage (that is BERT) is needed to back it. But at any time if the data associated with a control block changes and needs to be placed in a BERT, the number of BERTs required increases. BERTs also provide a serialization mechanism known as the BERT lock.

The difficulty occurs when a process needs to acquire a BERT either for additional data or for a lock and there are no BERTs available. That process must wait for a BERT to become available. Unfortunately, this can occur with processes that do not normally wait and often results in major processing becoming unavailable. For example, jobs often cannot be purged if there is a BERT shortage.

A further complication can occur at warm start time. There are processes that require the BERT lock or additional BERTs to complete warm start processing. If they are not available, some aspect of warm start processing might have to be delayed. However, there have been unfortunate cases where the shortage of BERTs at warm start has resulted in warm start failing. Usually these problems have been addressed, but there might still be situations where a warm start cannot complete. Because of problems in the past, customers have had to cold start to resolve cases where a BERT shortage prevented a warm start.

The best practice is to avoid BERT shortages. This can be done by paying particular attention to BERT usage at your installation and monitoring for HASP050 messages that indicate

BERT shortages. JES2 also has added two messages specifically for when BERTs are about to run out. The first message, shown in Example 13-4, indicates that the number of BERTs is getting critically low.

Example 13-4 Critical BERT shortage message

```
$HASP052 JES2 BERT resource shortage is critical --
IMMEDIATE action required

DO NOT TERMINATE JES2 OR IPL.  Doing so might result in a COLD
start.

CURRENT BERTNUM=650, Free BERTs=10

Correct BERT shortage by --
  - $T CKPTSPACE,BERTNUM=nnnn (increase BERTs)
  - $P Jnnnn (purge pre-execution jobs)
```

Depending on the number of BERTs remaining, certain processes will be suspended to prevent all BERTs being used. The following are additional lines that can be added to the message to indicate what processes have been suspended:

- No batch jobs can be submitted
- JOEs requiring BERTs cannot be added
- No batch jobs, STCs, or TSUs can be submitted

If the number of BERTs drops to the point where certain processes were failed because of the inability to obtain the needed BERTs, the message shown in Example 13-5 is issued.

Example 13-5 Extreme BERT shortage message

```
$HASP051 EXTREME BERT SHORTAGE DETECTED.
JES2 PROCESSING IS IN A NON-STABLE STATE.  RESTART JES2 AS
SOON AS MORE BERTS HAVE BEEN MADE AVAILABLE.
```

In this case, there might be errors in the data structures on this member or errors in the one of the JES2 work queues.

In the event of a critical BERT shortage, an action must be taken immediately to relieve the shortage. Increasing the number of BERTs via the **$T CKPTSPACE** command is the simplest action (assuming the checkpoint is large enough to contain additional BERTs). If the checkpoints are not large enough, you can either move to larger checkpoints using the checkpoint reconfiguration dialog or you can reduce the limit on some other checkpointed resource to make room in the checkpoint for additional BERTs.

Another way to address a critical shortage is to reduce the number of BERTs in use. You can use the **$D CKPTSPACE,BERTUSE** command to determine what processes are using the most BERTs. Example 13-6 shows a sample of the output from that command.

Example 13-6 Output of the $D CKPTSPACE,BERTUSE command

```
$HASP852 CKPTSPACE
$HASP852 CKPTSPACE  CURRENT BERT UTILIZATION
$HASP852            TYPE        COUNT  CB COUNT
$HASP852            -------- --------- ---------
$HASP852            INTERNAL       14         1
$HASP852            JQE         1,067       827
$HASP852            CAT           114        38
```

```
$HASP852          WSCQ          12       3
$HASP852          DJBQ           6       3
$HASP852          JOE          151     151
$HASP852          DAS            0       0
$HASP852          FREE      62,736       0
```

The command output helps you understand what data areas are using BERTs and if you need to purge jobs (JQEs) or output (JOEs) to free up BERTs. This also can help you understand if there is a runaway process that is consuming BERTs and what it is that it is using BERTs for.

An extreme BERT shortage must be dealt with in the same way. BERTs must be freed to allow the system to return to normal processing. But in the case of an extreme shortage, any member that is issuing the HASP051 message must be hot-started to clean up any residual affect of the shortage.

JES2 processor counts

Just as JES3 has DSPs and FCTs whose counts can be specified, JES2 has PCEs whose counts can be adjusted. Adjusting some of these counts alters the amount of work a particular JES2 address space can perform. However, unlike JES3, which does all its work on the global, JES2 performs work in any JES2 address space that has available processors to do the work. Decide if JES2 work will be spread across all members of the MAS or if some members will do less JES2 work than others (a purely symmetrical arrangement or an asymmetrical arrangement). When you have determined that, setting processor counts can influence what JES2 processing a member does.

The PCEs that an installation can influence the counts of are specified on the **`PCEDEF`** initialization statement. They can be set at JES2 initialization and cannot be altered by an operator command. The following counts can be specified on **`PCEDEF`**:

`CNVTNUM`

JES3 specifies the number of converters for demand select and batch when defining converter DSPs on the global or in a CI FSS. JES2 does not make any such distinction. Any JES2 converters can process any job type. Converters in JES2 run on all members of a MAS. Which system a job gets converted on is controlled by the system affinity (**`SYSAFF`**) of a job. Demand select jobs always have an affinity for the member they were started on or logged in to. Therefore, all JES2 members must have at least one converter.

JES2 has code to deal with waits that might be introduced by JCLLIB data sets. If a data set specified in a JCLLIB cannot be allocated due to ENQ contention (data set is allocated exclusively and cannot be allocated shared by conversion) or if a JCLLIB data set is archived, the converter could potentially wait for the data set. If a number of jobs that need archived or ENQed data sets are submitted, they could tie up most if not all of the JES2 converters. Since JES2 does not distinguish between demand select jobs and batch jobs, if the converters were all tied up, no one could log on to that member and no started tasks could be started.

To prevent this, JES2 does make a distinction between converters. The odd-numbered converters are defined as ones that can wait and the even-numbered ones cannot. This needs to be considered when defining the number of converters on a member. In general, define at least two converters, even if the member is not doing a significant amount of JES2 work. This prevents a single job waiting in converter

from preventing any starts or logons on the member.
The main resource used by converters is buffers (EXTBUFs) and TCBs (that consume below the line storage). The default number of converters is 10 and the maximum is 25. Each converter uses at least three buffers (more for jobs with more instream data). If a member is to do minimal JES2 work, define the number of converters down to two. If the member will be processing many jobs, increase the value to the maximum and potentially adjust the number of EXTBUFs up.

PURGENUM

JES2 purge processing handles not only the purging of jobs but also the purging of spin data set and output groups that require ENF 58 processing. All these operations tend to be I/O-intensive because JES2 used record 0 at the start of each track group to track ownership of the track group. When a track group is freed, record 0 must be updated to indicate the track is free. Because purge processing is so I/O intensive, it can benefit from additional PCEs. It primarily uses its own buffers to read and write the record 0 buffers but it does use a couple of EXTBUFs to hold various control blocks (two or three per purge processor).

Purge processing is not affected by any affinity rules and can be performed on any member. It is generally looked at as a background task. The exception is when there is a resource shortage, where purge processing becomes the bottleneck to freeing many checkpoint resources. The default number of purge processors is 10 and the maximum is 25. A JES2 member that does little JES2 work can have fewer purge processors. Other members must have the default of more purge processors defined. If jobs or output elements (JOEs) are not being purged in a timely manner, the number of purge processors must be increased.

PSONUM

The PSO interface is an older interface used to process requests from the TSO OUTPUT and RECEIVE command and requests from processes that use the SYSOUT external writer interface. These are lightly used interfaces in today's world where the SYSOUT API (SAPI) is one of the primary ways to access SYSOUT. PSO processors handle requests made from the local member (not from other MAS members).

The default number of purge processors is two and the maximum is 10. Unless you have a particularly old application that uses the PSO interface, the default on all members is generally sufficient. If you do have a large PSO user, increase this number on the member that the application uses and any alternate member the application could use. Additional PSO processors use additional EXTBUFs.

OUTNUM

Much like the output services phase in JES3, the JES2 output phase makes SYSOUT created by a job available for processing by printers and application interfaces such as SAPI. It is during the output phase that SYSOUT data sets are grouped into output groups (JOEs). All non-spin output is processed during the output phase. Output processing can occur on any member of the MAS. When a job is added to the output queue, its affinity is set to any member (thus system affinity does not apply to the output phase).

Output processing can be a heavy user of EXTBUFs. All the jobs non-spin output control blocks (PDDB IOTs) must be read into buffers before output processing starts processing a job. When a job completes output processing, the buffers are all written back out.

The default number of output processors is 10 and the maximum is 25. A JES2 member that does little JES2 work can have less output processors defined. Other members must have the default of more output processors defined. If jobs are spending a long time awaiting output, consider raising the number of output processors. However, first ensure that there are no shortages of EXTBUFs that are slowing down output processing. Remember when increasing the number of output processors, you need to watch the EXTBUF usage to ensure that you do not create a shortage.

`STACNUM`

Another, older, interface to access JES data is the TSO **`STATUS`** and **`CANCEL`** commands. These send requests to JES to inquire about the status of a job or to cancel (and potentially purge) a job. Though the **`CANCEL`** interface is still used today by some applications that want to cancel jobs without resorting to an operator command, the **`STATUS`** interface has been largely replaced by the extended status interface (except for the actual TSO **`STATUS`** command).

These commands both get processed in the JES2 address space by the STAC processors (STAtus/Cancel processors). These processes use very little resources but must make a SAF call to authorize any cancel request. They process all requests for the status and cancel interface issued on the local member. The default number of STAC processors is two and the maximum is 10. Unless you have an old application that uses the status or cancel interface, the default on all members is generally sufficient. If you do have a large user of the status or cancel interface, increase this number on the member that the application uses and any alternate member the application could use.

`SPINNUM`

There are two basic ways SYSOUT data sets are created. The traditional type of data set is processed when a job or step completes execution and is created through output processing. However, applications can choose to create output that is available for processing immediately upon deallocating the SYSOUT data set. These data sets are called *spin data sets* by JES.

JES2 queues these data sets to the JES2 address space by using an ECSA buffer. The spin process in the JES2 address space will build the appropriate output element (JOE) for the data set and queue it for processing. This process does a number of I/Os for each data set as part of the process of building the output element. All processing is done in the ECSA buffer that was obtained.

Additionally there is processing done by this process when there is an ECSA shortage, a JOE shortage, when SPIN data sets arrive via NJE, or when a system restarts. This is called *unspun processing*. It is an I/O intensive process that reads in the SPOOL control blocks for spin data sets and looks for data sets that could not be processed. Certain spin processors can do the unspun processing, while others are reserved to just handle ECSA requests.

Spin processors handle ECSA requests from the local member and unspun requests from any member. In general, ECSA requests can be processed by one or two spin processors. Unspun requests can benefit from additional spin processors. The default number of spin processors is three and the maximum is 10. Increasing the number of spin processors can help unspun processing, but that is not a common process. Because of this, consider increasing the number of spin processors on the members of the MAS that do most of the JES2 intensive work and use the default on members that do little other JES2 work.

JES2 subtask counts

Like JES3, JES2 has general-purpose subtasks that can be used to perform functions that must MVS wait. Originally these were developed to deal with SAF security requests, but now they are also used by other processes in JES2. In JES3, the number of these subtasks is managed by JES3. However, JES2 has a parameter to control the number defined. The JES2 **`SUBTDEF GSUBNUM`** parameter controls the number of subtasks JES2 creates during initialization. The default is 10 with a maximum of 50. Each subtask creates a TCB in the JES2 address space.

The number of TCBs that can be defined in an address space is limited by the amount of 24-bit storage that is available for z/OS control blocks. This is also influenced by the number of 24-bit buffers that you define. The default is acceptable for most installations. However, if you are a heavy user of security profiles that could cause delays in processing SAF requests, you might need to increase the number of subtasks defined. There is an internal command that tracks the usage of general-purpose subtasks, **`$D PERFDATA(SUBTSTAT)`** that, among other things, displays the queue time of general subtask requests (**`AVG_QUEUE_TIME=`**). This is displayed in seconds. If the time is more than 0.0001 seconds for the most common requests (for example, ENFISSUE or $RACROUT), consider increasing the number of subtasks.

13.2.3 Configuring work queues

To maximize throughput, one area you need to understand is how work is queued within JES and how JES accesses those work queues. One thing to avoid is long queues that must be run looking for work to process. This causes delays both because of the CPU needed to access the queue and the cache misses if the queue is particularly long. There are two queues that are of most concern, the job queue and the SYSOUT (or JOE) queue.

Job queue considerations

The job queue is divided by phase of processing that the job is in. Installations can only influence the execution queue. All other queues are managed totally by JES2.

The execution queue is processed by initiators selecting jobs to run. There are two types of initiators: JES-managed and WLM-managed. Jobs are placed on the WLM work queue if the jobclass they are in is set to **`MODE=WLM`**. The WLM service class queues are ordered by arrival time, with the oldest jobs earlier in the queue. Since this is the intended order of selection, there is little consideration needed when setting up the WLM service queue. Jobs on a WLM work queue are also located on a JES2 jobclass queue. But this jobclass queue only exists to organize the jobs and is not used for selection purposes.

Jobs that are not in WLM-managed job classes are selected by JES-mode initiators. These initiators have a list of job classes that they select from. This is different from JES3 where job classes are associated with job class groups and initiators are then assigned to a job class group. A JES-mode initiator in JES3 will only select jobs from the job class group that it is assigned to.

Since in JES2 initiators can be associated with one or more job classes, one question is what classes will be assigned to the initiator? The list of job classes is an ordered list and selection is done from the first class in the list down to the last looking for eligible work. The more classes that are associated with a job initiator, the more classes it must search looking for work. So is it better to have 10 initiators with 10 classes each, or to have the 10 initiators with one class each?

It depends on how you want to spend your resources. Idle initiators consume an address space and associated storage and must be examined whenever work is added to the awaiting-execution queue. When work becomes available on any execution queue, all idle initiators search for work that might have been added. There is no communication of what queue the work became available on. This means that all idle initiators will scan for pending work. Many idle initiators result in all their related job queues being scanned for available work.

There is some performance code for this case. If a class is found to not have any work for one initiator, it is assumed to not have work for any initiator on this member. So only one initiator needs to scan a class looking for work each time initiators are scanned. This implies that one or 100 idle class A initiators will consume similar amounts of CPU when looking for work.

Any exit 14 or 49 code that makes a distinction between which initiators can select a job based on some property other than class negates this performance benefit of this feature. For example, if exit 49 were to prevent odd-numbered jobs from running on even-numbered initiators (for whatever reason), a job that cannot be selected on one class A initiator might be selectable on another. This requires each initiator to scan every job queue associated with it every time.

The advantage of having just one class per initiator is that though there are fewer per class, they are reserved and waiting for that class.

Alternatively, if all 10 classes are associated with all 10 initiators, the only time an initiator will go idle is when all 10 classes have no work. But every time an initiator finishes its work, it needs to scan all the associated class queues looking for another job. This scan does not optimize the processing since the scan occurs when the initiator is idle and it cannot affect other initiators. However, in this case, there will be fewer idle initiators and less scans when work is added to the execution work queue. It also might be possible to get away with fewer initiators.

In general, even with the performance benefits that occur when idle initiators use the same classes, the processing needed to scan all the class queues when any job is added to an execution queue is generally greater than the effect of scanning all classes when an initiator goes idle. So loading initiators with job classes will perform better than having idle initiators waiting for a particular job class. There is a risk with this that an initiator will not be available when new work starts arriving for a job class.

But assuming one of the criteria you use to assign job classes is by expected execution time, you can minimize the wait time by limiting the number of long running job class jobs that can be active at one time. This can be set at the job class level with the **XEQCOUNT** and **XEQMEMBER** keywords. These work similar to the **TDEPTH** and **MDEPTH** parameter on the **CLASS** statement in JES3.

SYSOUT queue considerations

Unlike job queues, it is much more important to configure your SYSOUT queues properly as opposed to your job classes. This is because there are generally many different types of processes and devices selected during SYSOUT and far more characteristics to choose from. Also, with jobs, most jobs in an installation are awaiting hardcopy (output in JES3) processing; there are relatively few jobs in the execution queues. However, all SYSOUT is eligible to be processed on one queue or another.

In JES2, there are 36 SYSOUT classes (A-Z and 0-9). Each SYSOUT class has three queue heads that represent the type of destination that the SYSOUT is routed to (these are described in "SYSOUT destination types" on page 176). There is also a separate queue for held SYSOUT and one for SYSOUT destined for another NJE node.

Within each of the class queues (A-Z and 0-9), output elements (JOEs) are queued by route code (destination) and within destination, by priority. JOEs with the same destination and priority are queued first-in first-out (FIFO). The network queue is arranged in priority order with JOEs that have the same priority arranged FIFO. The held JOE queue is simply a last-in first-out (LIFO) queue (stack) of JOEs that are held.

Remember in JES2, the held queue is used for all output groups that are "not ready" to be processed. Output groups on the hold queue can be associated with TSO or external writers as is the case for JES3.

Properly balancing the use of these queues can have a large impact on SYSOUT selection performance. Additionally, using selection criteria that line up with the queuing methods can also lead to better throughput.

SYSOUT destination types

JES2 has three types of non-NJE SYSOUT destination: local, remote, and user. These represent the three queue heads associated with each SYSOUT class. JES2 stores the destination (or route code) of a SYSOUT data set as a 2-byte node number, a 2-byte binary destination, and an 8-byte user route code.

Remote destinations traditionally routed output to an RJE workstation. Internally they are stored as the 2-byte local node number, a 2-byte remote node number, and a blank user route code. These destinations are normally set to R*nnnnn* where *nnnnn* is the remote number.

There are two approaches of local destinations. There is the standard LOCAL or ANYLOCAL destination where the 2-byte node number is set to the local node, the binary destination is 0, and the 8-character user route code is blank. The second approach of local destination is called special local. This is where the 2-byte node number is set to 0, the 2-byte binary destination is set to a special local route code, and the user portion is blank. This is generally specified as U*nnnnn* where *nnnnn* is an arbitrary number assigned to the destination.

The user destination is any output that is routed to an 8-character destination that is not represented by a local or remote destination. It has the 2-byte local node number, a zero binary destination, and an 8-character user destination.

Any printer or writer can be set up to select from any of these destination types. A local printer can be set to select output with a remote destination. SAPI can select output routed to a special local route code.

By understanding how JES2 separates output on queues, you can spread SYSOUT across multiple queues and avoid performance problems associated with long SYSOUT work queues.

Using DESTIDs to route output

DESTIDs in JES2 are a way to assign a name to a destination. They are in effect an aliasing function that allows installations to rearrange how SYSOUT is managed without having to alter JCL.

Using DESTIDs, JCL and application writers do not need to know what the underlying route code is when directing output. For example, assume you want a printer to select route code U12 (special local 12) output. Applications can code DEST=U12. There are two problems with this approach. The first is that applications need to be aware of the U12 and associate it with a particular printer. Second, if you decide to consolidate the printers selecting U11 and U12, you need to set up the new printer to select multiple route codes. This can be done but is less efficient. Also, a printer can only select up to four route codes.

Using DESTIDs, you can assign a name to a route code. Assume you want to assign the name FRANK to route code U12. This is for the printer in Frank's office. You would then define a DESTID of:

```
DESTID(FRANK) DEST=U12
```

This statement works in two directions. Any JCL or operator command (or other external) that references FRANK will internally be converted to U12. It will also change all displays of U12 to instead display FRANK. In effect, the U12 internal route code has been replaced with the destination named FRANK.

Now assume that after looking at how output is being processed, you decide to move output for Frank's printer to a remote queue instead of the local queue (to improve some throughput issues). You can alter the DESTID for FRANK to a remote destination:

```
DESTID(FRANK) DEST=R12
```

This defines the FRANK DESTID to now be a remote route code and places it on the remote queue instead of the local queue. Note that no JCL needs to be changed; all commands that used FRANK are not affected, printer applications are not impacted, and so on. If there is existing output that might have been routed to the old U12 destination, you would need to issue the command:

```
$TOJ(*),/DEST=U12,DEST=FRANK
```

This might need to be issued a couple of times in case there are any jobs actively creating output for the old designation for FRANK.

You can also have two DESTIDs resolve to the same route code. So if Frank shares an office with John, you might want to add a DESTID for JOHN that goes to the same route code as the updated FRANK:

```
DESTID(JOHN) DEST=R12
```

But this introduces a problem. If you display something with a route code of R12, should it display FRANK or JOHN? The default processing is to display neither and instead display R12. This is not ideal. To address this issue, there is another keyword on DESTID to indicate which is the primary destination (and therefore, which are the alternate). Assuming we want the printer to display as FRANK even though it can also be referred to as JOHN you would specify:

```
DESTID(FRANK) DEST=R12,PRIMARY=YES
```

Now anything that references a destination of JOHN or FRANK or R12 will display as FRANK.

DESTIDS can also be used with the 8-character user route code. So you can set a device to select a route code of POK that has no DESTID. If you want NY to also be routed to POK, you can define a DESTID:

```
DESTID(NY) DEST=POK
```

In this case, assuming there is no DESTID for POK, all output with a route code of POK will display as NY (since the DESTID takes precedence for displays). If you want the destination to display as POK, define the DESTID:

```
DESTID(POK) DEST=POK,PRIMARY=YES
```

Effectively, this just changes how output routed to the 8-character user route code is displayed.

There are many other features and capabilities using DESTIDs including the ability to alter processing that occurs with NJE routed output and output received via NJE. There is also a `DESTDEF` statement that allows a more global control over how destinations are handled. A complete description of how DESTIDs work can be found in "Output Routing" section of the *z/OS JES2 Initialization and Tuning Guide*, SA22-7532.

Configuring SYSOUT queues for performance

Now that we understand what SYSOUT queues exist (class queues, hold queue and network queue), and how we can get output placed on various queues, we can begin to look at how to configure the MAS to get the best performance.

The first step is to understand the producers and consumers of SYSOUT. Is the consumer a series of real printers? How many devices do we have? Can they all be thought of in the same way but in separate locations? Do you have an archive product that manages SYSOUT groups? How does it select what to archive? Is most output sent over NJE to other nodes for processing?

On the producer side, does output get created in bursts or is it a steady pace? Does it produce output faster than the consumers can process it and you catch up during a quiet time? What kind of controls are there to alter the SYSOUT class or destination created by applications (what is possible to change easily)? Are the producers associated with some property of the SYSOUT class when they create their output, or is the class just there to help manage the output?

Looking at the consumers, let us look at how they select output. Both JESes have a rich set of criteria to select output. There are logical factors like class, external writer name, PRMODE, limits, and jobname. There are physical factors like destination (where is it printing), forms, and burst. But when defining how your system manages SYSOUT, you need to consider how the output is organized.

Let us look at a simple example. Assume that you have a printer that will process all output associated with an external writer. In JES2, output routed to an external writer can be processed by any printer, by SAPI, or by an actual external writer. To simplify the example, assume this is a printer selecting the output (since we can then look at the printer parameters being used). So in setting this up, the simplest thing to do would be to place only the external writer specification in the work selection list and set the writer value to what you want to process. Here is how you would specify that:

```
PRT1 WS=(W),WRITER=TEST
```

From a configuration standpoint, this is easy to do. But remember how JES2 arranges output. Based on this definition, you are not selecting on a class or a destination. So to find an element to process, the printer needs to look at all the class queues and the destination queues off the class queues. The result is that this simple selection of criteria will scan all the output elements on the system looking for something to print (except output elements queued for NJE or on the hold queue).

But it is worse than that. JES2 has code that can do a quick scan of output queues looking for where on the existing chains an output group might be found. But these are also dependent on the device selecting on class and destination. Since our simple setup did not select on class or destination, there was no optimization possible. So this simple setup is about as bad as it can get from an overhead perspective.

The HELD queue is not scanned in this case since real printers can only select ready output. But had this been a SAPI printer, there would be the capability of scanning the held queue too.

So let us try to improve this a bit. We probably do not want to select output routed to a specific destination. So we can restrict this to locally destined output. So our improved printer specification is:

```
PRT1 WS=(W,R),WRITER=TEST,ROUTECDE=LOCAL
```

Now you have reduced the number of output groups scanned significantly. JES2 will only examine the local queues of each output class and stop scanning that queue when it finds an output group that is not LOCAL. This simple but logical change can significantly reduce the overhead of scanning for work.

However, this points out a potential conflict that can occur when creating output. If you create a SYSOUT data set that has a writer of TEST specified and a routecode of R15 (remote 15), where should it be processed? Printers that print route code R15 output generally specify that the writer value be blank. So output with conflicting routing information might never be picked up for processing.

We can further improve this processing if we decide that output for writers will be sent to a specific class. We could then add the class to the selection criteria and further reduce the number of output groups scanned. Your printer definitions would then look something like this:

```
PRT1 WS=(W,Q,R),WRITER=TEST,Q=T,ROUTECDE=LOCAL
```

Now the printer will only look at class T output routed to destination LOCAL for output groups. Since class and destination are how JES2 arranges SYSOUT, this greatly reduces the overhead in searching for work for a printer. Since JES3 SYSOUT destined for an external writer already has to go to an external writer SYSOUT class, using a specific class for this purpose will be simple when moving from JES3 to JES2.

Routing using writer name is one example, but let us look at sending output to a printer using destination. In JES3, printers have names associated with them and SYSOUT is sent to the printer using the printer name. JES2 is similar in that output is sent to a destination, but the destination is not a printer name. It is one of the three types of destinations described in "SYSOUT destination types" on page 176. Generally, applications will not use the U*xxxxx* or R*xxxxx* formats of a destination to route output. Instead, the installation will define a DESTID that maps a meaningful name to a U*xxxxx* or R*xxxxx* route code.

The simplest destination type to use is the 8-character user route code. This is an arbitrary 8-character value that the creator and consumer of the output agree to use to identify the output to process. For example, an application can be created that runs a series of jobs and then runs a final job that uses SAPI to collect all the output. This could use the application name as the destination for the output and the selection criteria for the SAPI application.

This is fine for a simple application, but not for a major print processing application. For this type of application, we need to understand the nature of the output being created and the capability of the application to control what is selected.

If the application can only access a single SYSOUT class, assign it to a dedicated class (do not use that class for anything other than that application). If the application can use multiple classes, spread the SYSOUT over a logical mix of classes.

If the application is selecting on route codes, be sure to spread the route codes across the three queues that can be used for a class. If there are route codes that traditionally build large queues (for example, they never actually get processed but age off the system), assign those to higher-collating route codes (U9999 for example). Place the smaller, more transient route codes in the lower route codes. This avoids having to step over a large queue of output to get to frequently used output.

One mistake that often occurs is an application that by default creates the majority of its output using CLASS A with a destination of LOCAL. Then, the output never prints because there are no local printers left at the installation. Eventually the output gets deleted off the system because of age. By itself this is not a problem. But it can become a problem when installations decide that the convenience printers will also use CLASS A and a special local route code (using the form U*xxxx*). As the class A local work starts to back up, the overhead to select the output for the convenience printers starts to increase.

To prevent this, address the convenience printers by using R*xxxx* route codes or send them to other classes (always using DESTIDs to do the translation). Changing how the JCL DEST is handled is easy with DESTIDs, so periodically look at that.

Another mistake to be avoided is setting up applications that specify held output (OUTDISPs of HOLD or LEAVE). These output groups are placed on the hold queue in JES2 with no particular order to them. They were intended to be accessed through the job they are associated with (such as when using the TSO OUTPUT command). The SYSOUT API (SAPI) does support selecting held output. This is functionally compatible with what the PSO interface can do. However it has a much higher overhead than selecting ready output groups.

The period when you are migrating from JES3 to JES2 is a good time to set up new rules. This is when you can easily get work spread across multiple SYSOUT classes. The goal in all this is to use as many of the 108 SYSOUT queues JES2 provides as practical.

14 Output processing

This chapter discusses differences between JES3 and JES2 output processing, printer setup, and Remote Job Processing support.

Both JES3 and JES2 can process output in several different ways. Output can be:

- Held on spool for TSO
- Processed by an external writer
  - IBM external writer
  - ISV external writer
- Printed locally
- Printed remotely
- Archived

This chapter addresses each type of output showing the differences between JES3 and JES2 processing.

14.1 Held on spool for TSO

The JES3 initialization deck normally has at least one SYSOUT class defined as held for TSO. The class would be defined with statements similar to the following:

```
SYSOUT,CLASS=H,HOLD=(TSO),TYPE=(RSVD,PRINT)
```

These parameters allow TSO users to view their output using the TSO OUTPUT command or ISPF 3.8. By using a product such as SDSF or (E)JES, all output is generally available for viewing.

JES2 does not have the concept of “held for TSO”. Output classes can be defined as WRITE, HOLD, or PURGE:

```
OUTCLASS(A) OUTPUT=PRINT,OUTDISP=(WRITE,WRITE)
OUTCLASS(T) OUTPUT=PRINT,OUTDISP=(HOLD,HOLD)
OUTCLASS(Z) OUTPUT=DUMMY,OUTDISP=(PURGE,PURGE)
```

The TSO OUTPUT command or ISPF 3.8 will only allow viewing of held output, which can be requeued to the write queue if wanted. Products such as SDSF or (E)JES allow viewing of output on both the hold and writer queues. JES2 has conditional output dispositioning. The first parameter is for jobs that complete normally, the second is for jobs that abend.

With both JES3 and JES2, customers can requeue held output to the print queue if wanted.

14.2 Processed by an external writer

The process SYSOUT (PSO) interface (subsystem interface code 1) is used to obtain output from the JES spool. It is used for both the TSO OUTPUT and RECEIVE commands and for the external writer. JES3 and JES2 both process external writer output similarly.

A common practice is to define one or more held sysout classes for external writers. This will prevent work destined for the external writer from being accidentally printed. Conversely, the external writer can process data intended for printing if it is started for a print sysout class.

14.2.1 IBM supplied external writer

Both JES3 and JES2 can use the external writer (IASXWR00) to write data from the spool to a data set or special device. The external writer is supplied as part of the base z/OS operating system and is JES-independent. The external writer is a started task that selects jobs from a JES spool according to one or more criteria:

- Output class
- Job ID
- Forms specification
- Destination
- The writer name

A common use of the external writer is archiving SYSLOG data as shown in Example 14-1. While JES3 keeps its syslog for the entire JESplex in a single stream, JES2 creates a separate syslog for each LPAR in the JESplex. To keep the archives separate, a sysout class can be defined for each LPAR.

Example 14-1 Sample external writer procedure

```
//SYSLOG PROC
//SYSLOG EXEC PGM=IASXWR00,REGION=4M,PARM='P&LOGCLS'
//IEFRDER DD DSN=SYSLOG.&SYSNAME.(+1),DISP=(NEW,CATLG,CATLG),
// UNIT=3390,SPACE=(CYL,9100,30),RLSE),
// DCB=(LRECL=133,BLKSIZE=12901,DSORG=PS,RECFM=FBM)
```

In this example, symbol &LOGCLS is defined in parmlib member IEASYM00 for each LPAR. Only the sysout class can be coded in the PARM field; other criteria such as form name or destination must be specified by the operator using the z/OS MODIFY command against the started task.

14.2.2 Vendor-supplied external writers

Both IBM and independent software vendors (ISVs) provided specialized external writers with products designed to archive and print reports. An IBM example is Report Management and Distribution System (RMDS). These products use the PSO interface to obtain output from the spool and perform the required processing. Other vendors use external writers to send output to specialized devices such as plotters. Refer to the vendor's documentation to note if there are any differences in how the product works between JES3 and JES2.

14.3 Local printing

Both JES3 and JES2 can have a vast number of printers defined. JES3 uses the JNAME parm to name the printer, while JES2 numbers its printers 1 - 9999. There are many parameters that can specify print criteria such as form name, destination, SYSOUT class, and so on.

If you have non IBM printers, check with your vendor for any differences in setup or function between JES3 and JES2.

14.3.1 Printer naming conventions

JES3 allows installations to name printers by using the JNAME parameter on the DEVICE statement. The name limit is 8 characters. Additionally, printers can be grouped by using the DGROUP parameter. This allows several printers to be pooled at one location and allow jobs to be printed on any printer that has the correct form name or other setup criteria. Jobs can be routed to a certain printer by using the name or group as the destination.

For example, an installation has four high-speed printers named HQPRT1, HQPRT2, HQPRT3, and HQPRT4 in DGROUP HQPRINT. A job routed to HQPRINT with FORM=STD can print on any printers set up with that form. If that printer is down for service, the operator can set up a different printer with that form. Jobs would then print on the alternate printer without intervention.

JES2 printers must be named PRT(nnnn) for local printers or R(nnnn).PRn for RJE printers. The JES2 definition for a printer has a series of one to four routecodes. A DESTID statement can be used to define an eight-character name for a printer. The SDSF Printer display indicates which printer is associated with which DESTID under the SDESTn columns.

If more than one printer shares the same routecode, a job with that destination can print on any printer with the correct setup.

14.3.2 Advanced Function Printers (Infoprint)

AFP is a JES-independent method of printing also known as all-points-addressable printing. It has the ability to place text or images at any point on a sheet of paper. It is commonly used for printing to high-speed production printers such as the Infoprint 4100.

The product used to drive the printers is called IBM Print Services Facility™ (PSF). PSF accepts jobs from either JES and provides identical output. A started task called a functional subsystem (FSS) is started automatically when the operator calls (JES3) or starts (JES2) the writer. The FSS takes data from the JES spool and sends it to the printer as an intelligent print data stream (IPDS). A control block called the job separator page area (JSPA) is used to provide job and data set information to the FSS. JES3 uses exit 45 (IATUX45) and JES2 uses exit 23 (HASX23A) to modify the JSPA. PSF exits can access the JSPA to create separator pages or do other processing as required. Both JES3 and JES2 have other exits that affect print processing in general but exits 45/23 are the only ones that affect PSF directly.

See Chapter 10, "Related products" on page 135 for more information on AFP and Infoprint.

14.3.3 JES printers

Older channel-attached printers such as the IBM 3211 are controlled by JES directly instead of through an FSS. Card punches (if any still exist) are controlled by writers similar to printers.

14.4 Remote printing

JES3 and JES2 support remote printers through binary synchronous communications (BSCs) and Systems Network Architecture (SNA). JES3 calls its support Remote Job Processing (RJP) while JES2 calls it Remote Job Entry (RJE). RJP or RJE workstations can be dumb workstations, intelligent workstations, or other computers. Support is provided for card readers and punches, but often this support is merely used for transferring input and output files from and to the workstations rather than for actual card processing.

14.4.1 BSC remote workstations

JES3 defines remote lines using RJPLINE statements in its initialization deck. RJP workstations are defined using RJPTERM statements. Printers, punches, and card readers are defined with DEVICE statements. Console support is provided through a CONSOLE statement.

JES2 defines remote lines with LINE(nnnn) UNIT=*dev* statements and RJE workstations with RMT(nnnn) statements.

14.4.2 SNA remote workstations

JES3 uses a COMMDEFN statement to associate JES3 with a VTAM applied. RJP workstations are defined using RJPWS statements. Printers, punches, and card readers are defined using DEVICE statements. Console support is provided through the CONSOLE statement.

JES2 uses a LOGON(n) statement to associate JES2 with a VTAM applied. SNA lines are defined using LINE(nnnn) UNIT=SNA statements. RJE workstations are defined using RMT(nnnn) statements. Printers, punches, and card readers are defined using R(nnnn).PR(n), R(nnnn).PU(n), and R(nnnn).RD(n) statements.

14.5 Archiving output

Both JES3 and JES provide facilities to offload spool data. They can be used to archive data, provide additional spool space during times of heavy workload, or assist in migrating between major software releases.

14.5.1 JES3 dump job (DJ)

JES3 can offload spool data to tape using the JES3 dump job (DJ) utility. Jobs can be purged from the spool or kept as required. All control blocks are dumped with the jobs. The DJ utility can also be run in server mode. In this case, it runs in a separate address space rather than as a JES3 task, and uses standard data management rather than devices defined in the JES3 initialization deck.

14.5.2 JES2 OFFLOAD

JES2 provides an OFFLOAD facility that can offload data to either tape or DASD. From one to eight offload tasks can be defined. JES2 treats the offload transmitters and receivers like printers and uses similar commands. Some installations use offloaders to back up retained spool data on a scheduled basis.

14.5.3 Limitation between JES3 dump job and JES2 spool offload

Neither DJ nor OFFLOAD is meant for use by the end user. They are not compatible with each other; for example, an installation cannot dump JES3 spool data to tape using DJ and then restore it to a JES2 system using OFFLOAD. However, an installation can exchange data between JES3 and JES2 systems using NJE.

14.6 Spin processing

Are there differences between JES3 and JES2 in relation to freeing up spool files so they can be printed, deleted, or whatever?

14.7 Printer processing

Printers for both JESs can be managed by an operator issuing commands through a console or automation session, or by an Interactive System Productivity Facility (ISPF) interface such as SDSF or (E)JES. With both JESs, the operator can set up a printer to select jobs by a number of criteria, such as destination, form name, carriage tape or FCB, sysout class, or special printer features. The operator can also allow the printer to process one job at a time or run continuously, backspace the printer to reprint pages, or cancel jobs.

Operator commands are beyond the scope of this book. Command syntax is different between the two JESs and print operator training is crucial to a successful migration.

14.8 SYSOUT class redirection

In JES3, you can specify that a certain class of SYSOUT is redirected to another environment. In JES2, the use of Exit 40 is required for this purpose.

15

NJE considerations

This chapter describes the differences between JES3 and JES2 network job entry (NJE) processing.

JES3 customers can issue the *MODIFY CONFIG operator command to add SNA RJP workstation and non-channel-attached printers to the JES3 configuration without a restart. This command can also be used to add spool data sets to JES3 as of z/OS V1R13.0. In JES2, various $ADD commands can be used to add local and remote printers or TCPIP/NJE components.

15.1 Networking protocols

Both JESs can support NJE using three protocols:

- Binary synchronous communication (BSC)
- Systems Network Architecture (SNA)
- Transmission Control Protocol/Internet Protocol (TCP/IP)

If an installation is considering converting from JES3 to JES2, a pre-positioning move would be to switch from SNA protocol to TCP/IP. This would eliminate the need to use (and pay for) Bulk Data Transfer (BDT) and eliminate the need for BDT operations skills.

15.2 Product considerations

In JES3, SNA/NJE support is provided through BDT. BDT is a separate product and must be licensed on the global processor. JES2 and RSCS have native SNA/NJE support and does not need BDT installed.

BDT operates by taking jobs off the BDT queue in the JES3 spool, bringing them into the BDT spool, then transmitting them to the connected node. The connected system can be another JES3 sysplex running BDT or a non-JES3 sysplex running native SNA/NJE. Jobs received by BDT are sent to the JES3 spool by the NJERDR DSP. One or more NJERDRs must be running on the JES3 system or jobs will remain in the BDT queue. If a JES3 cold start is performed, the operator must issue the *X NJERDR command to call a read. The NJERDR is checkpointed so subsequent non-cold starts do not require this command.

15.3 Operating system considerations

NJE is defined to provide any-to-any connectivity for all operating systems, including z/OS with JES2 or JES3, z/VM with RSCS, zVSE/POWER, and IBM AS/400®. Data transmitted from one node will always appear identically in the receiving node, even if they or an intermediate node has a different operating system. These protocols also allow data from even a highly downlevel OS to transmit to a node with a current OS.

15.4 Pathing differences

JES3 nodes (except the home node) are defined as either directly connected, by including the TYPE keyword on the NJERMT statement, or indirectly connected, by including the PATH keyword. Indirectly connected nodes use store-and-forward logic to route the transmission to its final destination. Each node in the path must have the destination node defined.

JES2 nodes can determine their own path when all the nodes in the path are JES2. If a JES3, VM, or POWER node is in the path, store-and-forward is used. Paths can be explicitly defined by using CONNECT statements. A group of remote nodes can use the SUBNET parameter on their NODE statements to simplify routing.

15.5 JES2 subnets

A subnet is a group of nodes that might share a common characteristic such as location. One of the nodes will act as the gateway. If the originating system has a static path to the gateway defined through a CONNECT statement, all nodes in that subnet have an implicit path.

JES3 does not use subnets. Transmissions going to an indirectly connected node go only to the next node as defined by the PATH parameter in the NJERMT statement. JES3 has no knowledge of the path that a transmission will take beyond the adjacent node.

15.6 Performance considerations

NJE transmissions within an entirely JES2 path can complete faster using path manager. This because JES2 will forward packets of data directly to the next node in the path as long as it is a JES2 node. If path manager is not used, or a non-JES2 node lies in the path, the entire transmission must be received at every intermediate node, then sent to the next node in the path.

The JES2 command $DPATH(*nodename*) can be used to determine the path a transmission takes to a particular node. By examining SMF type 57 records, the system programmer can determine which nodes if any have high activity and take steps to reduce the path length. For example, if a sysplex A transmits many jobs to node B but they are routed through node C, it might be beneficial to define a direct connection between nodes A and B.

15.7 Initialization statement comparison

Table 15-1 relates JES3 NJE statements to those for JES2.

Table 15-1 NJE statements: not all parameters are shown

JES3 statement	JES3 parameter	JES2 statement	JES2 parameter
NJERMT (home node)	HOME=YES	NJEDEF	OWNNODE=nnnn
	NAME	NODE	NAME
	ALIAS[a]	no JES2 equivalent	
	BDTID	LOGON(x)	APPLID
	NJEPR	NJEDEF	STNUM
	NJEPU	NJEDEF	JTNUM

JES3 statement	JES3 parameter	JES2 statement	JES2 parameter
NJERMT (remote node)	NAME	NODE	NAME
	AUTO (BSC only)	LINE	CONNECT
	RDLY	LINE	RESTART
	BFSIZ (BSC only)	TPDEF	BSCBUF
	CTC (BSC only)	LINE	UNIT
	LINE (BSC only)	NODE	LINE
	MAXLINE	no JES2 equivalent	
	PATH	CONNECT	NODEA
	TLS	SOCKET	SECURE
	TYPE	LINE	UNIT
NJECONS		No JES2 equivalent	
NETSERV	NAME	NETSERV	NETSERV(n)
	HOSTNAME	NETSERV	SOCKET
	PORT	SOCKET	PORT
	SYSTEM	No JES2 equivalent	
	STACK	NETSERV	STACK
	ITRACE	NETSERV	TRACEIO=YES
	JTRACE	NETSERV	JES=YES
	VTRACE	NETSERV	VERBOSE=YES
SOCKET	NAME	SOCKET	SOCKET(xxxxxxxx)
	HOSTNAME	SOCKET	IPADDR
	PORT	SOCKET	PORT
	NETSERV	SOCKET	NETSRV
	NODE	SOCKET	NODE
SYSID (SNA only)	NAME	Not needed as BDT is not required on JES2	

a. ALIAS specifies an alternate name for the home node. For example, if two nodes are merged, the nodename of the merged system could be specified to avoid the need for the customer to modify their jobs.

15.8 NJE examples

The following initialization deck fragments illustrate the differences between JES3 and JES2. The USA nodes are JES3 in the first examples and JES2 in the second. The CAN* nodes are JES3, and the MEX* nodes are a mix of VM and JES2. Node USAMVS1 has two channel-to-channel connections with node USAMVS2, a SNA connection with node CANMVS1, and a TCP/IP connection with node MEXVM1. Other nodes are indirectly connected.

15.8.1 JES3 example

JES3 parms: Only NJE parms are shown in this fragment:

```
COMMDEFN,APPLID=JES3SNA
SYSID,NAME=USABDT
NJECONS,CLASS=S12
NETSERV,NAME=JES3S001,HOSTNAME=USAMVS1.USBANK.COM,PORT=175,SYSTEM=SY1
DEVICE,DTYPE=NJELINE,JNAME=LNUSA2A,JUNIT=(220E,SY1)
DEVICE,DTYPE=NJELINE,JNAME=LNUSA2B,JUNIT=(220F,SY1)
SOCKET,PORT=175,NETSERV=JES3S001,NAME=MEXVM1,HOSTNAME=MEXVM1.MEXICOBANK.MX,
NODE=MEXVM1
* HOME NODE
NJERMT,NAME=USAMVS1,BDTID=USABDT1,HOME=YES,NJEPR=5,NJEPU=5
* DIRECTLY-ATTACHED NODES - CTC
NJERMT,NAME=USAMVS2,LINE=LNUSA2A,BFSIZ=1992,AUTO=YES,CTC=YES,MAXLINE=2
* DIRECTLY-ATTACHED NODES - SNA
NJERMT,NAME=CANMVS1,TYPE=SNA
NJERMT,NAME=CANMVS2,TYPE=SNA
* DIRECTLY-ATTACHED NODES - TCP/IP
NJERMT,NAME=MEXVM1,TYPE=TCPIP
* INDIRECTLY-ATTACHED NODES
NJERMT,NAME=USAMVS3,PATH=USAMVS2
NJERMT,NAME=USAMVS4,PATH=USAMVS2
NJERMT,NAME=MEXMVS1,PATH=MEXVM1
NJERMT,NAME=MEXMVS21,PATH=MEXVM1
NJERMT,NAME=MEXVM2,PATH=MEXVM1
*
```

BDT parms: Not all parms are shown in this fragment

```
* BDT SYSTEM ID
SYSID,NAME=USABDT,APPLID=JES3SNA1,NJEAPPL=JES3SNA1,NJENAME=USAMVS1
* BDT HOME NODE
BDTNODE,LU=5,BUFSZ=1024,BUFNO=10,N=USABDT
* BDT REMOTE NODES
BDTNODE,LU=9,BUFSZ=3072,BUFNO=7,TYPE=NJE,APPL=CANAPPL1,A=YES,N=CANMVS1
BDTNODE,LU=9,BUFSZ=3072,BUFNO=7,TYPE=NJE,APPL=CANAPPL2,A=YES,N=CANMVS2
*
```

15.8.2 JES2 example

```
NJEDEF JRNUM=2,JTNUM=2,SRNUM=2,STNUM=2,LINENUM=5,CONNECT=(YES,5),
MAILMSG=YES,MAXHOP=25,NODENUM=10,OWNNODE=1
NODE(1) PATHMGR=YES,AUTH=(DEVICE=NO,JOB=NO),REST=100,NAME=USAMVS1
NODE(2) PATHMGR=YES,AUTH=(DEVICE=NO,JOB=NO),REST=100,NAME=USAMVS2
NODE(3) PATHMGR=YES,AUTH=(DEVICE=NO,JOB=NO),REST=100,NAME=USAMVS3
NODE(4) PATHMGR=YES,AUTH=(DEVICE=NO,JOB=NO),REST=100,NAME=USAMVS4
NODE(5) PATHMGR=NO,AUTH=(DEVICE=NO,JOB=NO),REST=100,NAME=CANMVS1
NODE(6) PATHMGR=NO,AUTH=(DEVICE=NO,JOB=NO),REST=100,NAME=CANMVS2
NODE(7) PATHMGR=NO,AUTH=(DEVICE=NO,JOB=NO),REST=100,NAME=MEXMVS1,SUBNET=MEXICO
NODE(8) PATHMGR=NO,AUTH=(DEVICE=NO,JOB=NO),REST=100,NAME=MEXMVS2,SUBNET=MEXICO
NODE(9) PATHMGR=NO,AUTH=(DEVICE=NO,JOB=NO),REST=100,NAME=MEXVM1,SUBNET=MEXICO
NODE(10) PATHMGR=NO,AUTH=(DEVICE=NO,JOB=NO),REST=100,NAME=MEXVM2,SUBNET=MEXICO
LINE1 UNIT=220E
LINE2 UNIT=220F
```

```
LINE3-10 UNIT=SNA
LINE11-15 UNIT=TCP
LOGON(1) APPLID=JES2SNA
APPL(CANAPPL1) NODE=5
APPL(CANAPPL2) NODE=6
NETSRV(1) RESTART=(YES,15),SOCKET=LOCAL,START=YES,TR=NO
SOCKET(MEXVM1),NODE=7,IPADDR=MEXVM1.MEXICOBANK.MX
```

15.9 NJE exits

Both JES3 and JES2 provide exit points to modify network data. Table 15-2 lists the JES2 equivalent of the JES3 exits.

Table 15-2 NJE exits

JES3 exit	Function	JES2 equivalent
IATUX35	Validity check network commands	Exit 5
IATUX36	Extract accounting information from the job header for the JMR	Exits 47 and 57
IATUX37	Modify the incoming data set header for local SYSOUT processing	Exits 39 and 55
IATUX38	Modify the job data set (JDS) entry created from the data set header for local SYSOUT processing	Exits 47 and 55
IATUX39	Modify the data set header for an outbound SYSOUT stream	Exits 46 and 56
IATUX40	Modify the job header for an outbound job stream	Exits 46 and 56
IATUX42	TSO screening and notification	Exits 38 and 40
IATUX43	Modify the job header for an outbound SYSOUT stream	Exits 46 and 56
IATUX67	Determine action when remote data set is rejected by SAF	Exit 39
IATUX68	Modify the local NJE job trailers for a network stream containing SYSOUT	Exits 47 and 57

BDT provides multiple exits. They are described in *z/OS V1R7.0 BDT Installation* GA22-7511. Since BDT is based on JES3, the exits are just placed in the BDT load library. If they are there the systems will load them, if not they are ignored.

JES3 complexes that use SNA/NJE also require a separate initialization deck for BDT. This chapter contains a discussion about the differences between JES3 and BDT NJE definition statements and their JES2 equivalents. JES2 does not require BDT for SNA/NJE, so the SNA/NJE definitions for JES2 are part of the JES2 initialization statements.

15.10 Security considerations

RACF NODES (for inbound work) and WRITER profiles (for outbound work) are generally similar between JES3 and JES2. However, modifications are needed as follows:

- RACF WRITER profiles must be modified to begin with the new JESname, for example, JES2 rather than JES3. Also, JES3.RJP.*devicename* profiles must change to JES2.RJE.*devicename*.
- RACF NODES profiles on other systems in the NJE network must be modified to include the new JES2 nodename if changed.

15.11 SMF considerations

JES3 writes SMF type 57 records for transmissions using BSC or TCP/IP protocols. BDT/SNA transmissions write type 59 records. JES2 writes type 57 records for all NJE protocols.

The SMF type 57 records differ between JES3 and JES2. Jobs that analyze SMF data will need to change. Refer to *z/OS MVS System Management Facilities (SMF)*, SA22-7630 for the different record mappings. JES2 also provides records type 55 and 58 for network SIGNON and SIGNOFF.

15.12 Entering commands from a remote node

JES3 provides user exit 35 (IATUX35) which allows installations to determine which if any commands can be entered from another node. Exits 58 and 59 (IATUX58 and IATUX59) can perform pre-processing and post-processing of the SAF call.

JES2 has exit 5 available for command authentication and manipulation, and exits 36 and 37 are available for the pre-processing and post-processing of the SAF call.

15.13 NJE incompatibilities

Here are some areas of incompatibility between JES3 and JES2 that need your attention.

15.13.1 JES3 NJE functions not supported in JES2

The following JES3 parameters on the NJERMT initialization statement have no equivalent in JES2, but do not represent any loss of function when going to JES2:

- AUTO=YES|NO JES2 does not automatically restart the line
- RDLY=mm JES2 does not automatically restart the line
- RETRYCT=nnn JES2 retries the line 10 times
- EXSIG=recpwd2 Line passwords not supported for NJE signon
- SIG=sendpwd2 Line passwords not supported for NJE signon
- PRTDEF JES2 doesn't change the SYSOUT class

- PRTXWTR sets a default sysout class for external writer output received by NJE. JES3 allows a sysout class to have a DEST parm to route all output for that class to an NJE node. This allows that destination to set a class for that output as it might not have used the same class as the originating system.
- PUNDEF (JES2 ignores the NDHGF2PU/F2PR flags)

15.13.2 Unnecessary JES3 NJE parameters

The following JES3 parameters on the NJERMT statement have no equivalent in JES2, but do not represent any loss of function when going to JES2:

- TYPE=BSC|SNA JES2 handles dynamically
- CTC=YES|NO JES2 handles dynamically
- BDTID=sysid BDT not used for NJE
- MAXLINE=n JES2 handles dynamically
- PRTTSO JES2 uses the NDHGF1HD "hold" flag for TSO output (I think this might relate to the default sysout class set in the TSO segment of a RACF user ID.)
- NJEPR=n JES2 does not use these internal devices
- NJEPU=n JES2 does not use these internal devices (these are dummy devices JES3 uses to process jobs and have no JES2 equivalent)

15.13.3 Additional JES2 NJE parameters

The following JES2 parameters on the NJEDEF statement have no equivalent in JES3. Those marked with an asterisk "*" must be specified when going to JES2:

- COMPACT= Data compaction table to be used for this node
- LINENUM=nnnn * Number of logical lines used for NJE
- NODENUM=nnnn * Number of nodes in the network
- PATH=n Number of paths to a nonadjacent node
- RESTxxx= Resistance settings for the path manager (let them all default to begin with)

The following JES2 parameters on the NODE statement have no equivalent in JES3, but can be specified when going to JES2:

- AUTH= Authority level allowed for commands from this node
- HOLD= Hold jobs coming from this node
- RECEIVE= Receive jobs, SYSOUT or both from this node (NORCV parameter on the *CALL NJE command is not specified)
- TRANSMIT= Send jobs, SYSOUT or both to this node (equivalent to the NORCV JES3 parameter on the *CALL NJE command)
- REST= Resistance settings for this node (let them all default to begin with)
- PATHMGR=YES|NO Connections to JES2 nodes only can use path manager (this can speed up transmission as the job does not need to complete on an intermediate node before being sent along to its final destination)

15.13.4 BDT parameters

We also need to show how to convert the relevant NJE parameters in BDT to JES2 parameters, and any other VTAM stuff that will be changed.

Even though JES2 does not use BDT for SNA NJE, a current JES3 user who is using BDT for both SNA NJE and File-to-File functions can still use BDT for the same File-to-File function after converting to JES2. The following are the most noticeable considerations:

- JES initialization statements:
  - JES2 does not require BDT to be defined in JES2 Initialization stream.
  - JES3 requires BDT to be defined in a JES3 Initialization stream using three statements:
    - CONSOLE, SYSID, and NJERMT (see *z/OS JES3 Initialization and Tuning Reference*, SA22-7550 for detailed explanations for these statements.)
- ACF/VTAM:
  - BDT has to be defined as a File-to-File application to VTAM, using APPL definition statement (see MVS/BDT Installation, chapter 4.)
  - BDT has to be defined as a Cross-Domain Resource to VTAM, using CDRSC definition statement (see MVS/BDT Installation, chapter 4.)
  - BDT has to be defined as a File-to-File session in the VTAM Logon Mode Table, using MODEENT statement (see MVS/BDT Installation, chapter 4.)
- MVS:
  - BDT has to be defined as a secondary subsystem in SYS1.PARMLIB member IEFSSNxx (see MVS/BDT Installation, chapter 3.)
  - The SYSNAME parameter of SYS1.PARMLIB member IEASYSxx will be used in FORMAT control statement for initializing message data sets (see MVS/BDT Installation, chapter 3.)
  - BDT has to be specified in Global Resource Serialization (GRS) parameter, if GRS is in use.
- TQI:
  - The Transaction Queuing Integrity (TQI) facility is an optional service, used to ensure that commands and file-to-file transactions that users submit reach the BDT work queue. It is also used to route BDT messages to the users at a different processor.
  - Even though running TQI is optional and might even affect system performance, JES2 users must consider the following disadvantages without TQI:
    - JES2 users can submit commands and file-to-file transactions only if they are at the same processor with the BDT address space. JES3 users do not encounter this restriction.
    - Commands and file-to-file transactions can be submitted when BDT is active. If BDT is not active, the requests are lost because they cannot be held on a checkpoint data set.
- Miscellaneous:
  - Giving APF authority to BDT library SYS1.BDTLIB.
  - Allocate BDT and TQI data sets (BDT Initialization Stream, BDT Work Queue, System GMJD Library, ISPF, TQI Checkpoint, TQI Bit-Map, Message.) See MVS/BDT Installation, chapter 5.
  - BDT Installation exits.

15.14 Using networking with TCP/IP

Every IBM networking partner now supplies a native TCP/IP implementation. As the importance of BSC and SNA networks decrease, the importance of an NJE over TCP/IP for MVS increases. It was a major requirement for customers to have this function available and it is now incorporated in both JES2 and JES3.

NJE over TCP/IP was first implemented by VM (RSCS) and has since been implemented by AS/400 and VSE/POWER (the other major NJE partners). Since the VM implementation, a number of developments in TCP/IP required some updates to the existing protocol.

The TCP/IP NJE support in JES2 and JES3 now implements the protocol first introduced by RSCS and it extends that protocol to include the following new support for:

- IPv6
- Enhanced security (SSL/TLS)
- Changes to the basic sign-on protocols
- SYSIN data streams that have an LRECL of up to 32 K

The goal of the redesign is to improve RAS and to get better performance (both for JES2 and for the NJE data transfers). JES3 is implementing TCP/IP NJE in parallel with the JES2 support. Some of the new code is part of a common component (common to JES2 and JES3).

A network can consist of any combination of SNA, BSC, and TCP/IP connections. Each of these has its advantages. Your choice depends on the available hardware and software at your installation.

A JES2 or JES3 complex can use BSC, SNA, or TCP/IP protocol, or all. A user submitting an NJE job is not aware of whether JES2 is using BSC, SNA, or TCP/IP. To define the type of protocol that JES2 uses, code the TYPE parameter on the NJERMT initialization statement.

NJE over TCP/IP compatibility

The TCP/IP NJE protocol is an addition to the SNA and BSC protocols that JES already supported. There is no plan to alter the level of support for SNA or BSC (or BDT). Of course, in order to use the TCP/IP NJE support, both sides of the connection must support TCP/IP NJE. The implication is that z/OS releases will be capable of using TCP/IP, SNA, or BSC to connect to other JES2 or JES3 systems with RSCS, AS/400, and VSE/POWER.

15.15 JES3 TCP/IP networking

BSC and SNA nodes are dependent on hardware equipment such as the IBM ESCON CTC adapters or the IBM 3705 communications controller. TCP/IP uses a layered stack structure and therefore has the advantage of being hardware independent. This allows the installation to implement an NJE networking environment over existing Internet Protocol networks using hardware protocols such as Ethernet and token ring. NJE over TCP/IP also supports IPv6 and TLS.

NJE over TCP/IP can be thought of as a hybrid between BSC and SNA. NJE over TCP/IP uses the same record structure and data streams as BSC NJE. But like SNA NJE, the NJE communication is driven through a separate NETSERV address space that is analogous, although not functionally identical, to JES3 BDT.

15.15.1 The NETSERV address space

NETSERV is a common JES component used by both JES2 and JES3. The role of the NETSERV address space is to make all the communication interface calls to TCP/IP on behalf of JES3. NETSERV listens for connections, incoming and outgoing data, and cooperates with JES3 to read data from the spool or write data to the spool.

The NETSERV address space can run either on the global or a local. There is no limit to the number of NETSERV address spaces that can run in the JES3 complex at a single time. You specify the system on which a NETSERV is to run when you define it to JES3.

A NETSERV is a started task that requires authority to communicate with JES in order to spool and de-spool jobs, and data sets. NETSERV also uses the common NJE component, IAZNJTCP, which uses TCP/IP services, and then it must operate in the z/OS UNIX System Migration considerations Services environment. Therefore, NETSERV must have the proper authority to function. The required authority consists of:

- A definition in the STARTED class to associate the started task with a user ID. It is suggested that you associate NETSERV started tasks with the same user ID that you associate with JES3. Although you can pick a different ID with no impact on security, it will be simpler for you to use the JES3 user ID.

  ```
  RDEFINE STARTED JES3*.* STDATA( USER(JES3ID) GROUP(SYS1) TRUSTED(YES))
  ```

- An OMVS segment in the security definition for the user ID associated with the started task.

  ```
  ALTUSER JES3ID OMVS(UID(0) HOME('/') PROGRAM('/'))
  ```

Figure 15-1 shows the relationship between JES3, NETSERV, and TCP/IP.

Note: The NETSERV address space must be defined to the security product STARTED class. To minimize the number of STARTED profiles, it is suggested that you define all of your NETSERVs with a common name pattern so that you can cover them all with one generic profile.

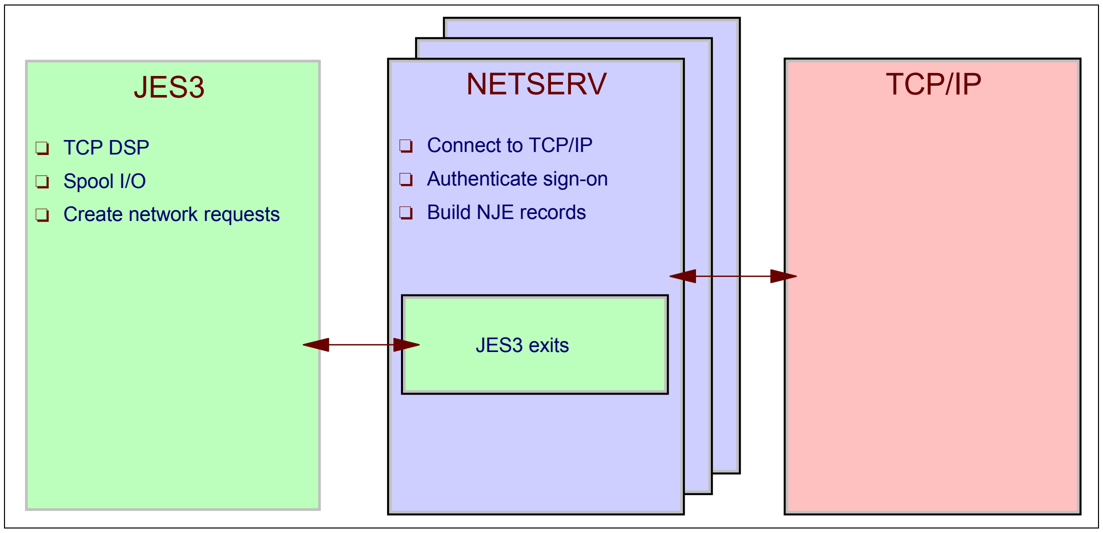


Figure 15-1 The relationship between JES3, NETSERV, and TCP/IP

The name of the NETSERV address space can be any address space name allowed in z/OS. It is your responsibility to make sure that you have the proper RACF STARTED class definitions in place for this address space name to run in the system.

Sockets

Under TCP/IP, information is communicated under a logical entity known as a *socket*. To JES3, a socket describes a task under NETSERV that communicates with either another NETSERV (if the remote node is JES2 or JES3) or a VM (RSCS) or VSE socket. The socket is described by the NETSERV it runs under, the node it connects to, and the host name and port that it communicates with.

In order for two nodes to communicate using the TCP/IP protocol, a socket must be started on one node. This node is referred to as a *client*. The corresponding node is referred to as a *server*. If JES3 is a server node, a server socket will be defined automatically. If JES3 is a client node, a socket must be defined using the SOCKET initialization statement or JES3 commands.

Note: The NETSERV address space must be defined to the security product STARTED class. To minimize the number of STARTED profiles, it is suggested that you define all of your NETSERVs with a common name pattern so that you can cover them all with one generic profile.

15.15.2 Defining TCP/IP NJE definitions with commands

Figure 15-2 on page 199 shows how to use JES3 commands to dynamically define an NJE over TCP/IP connection between a JES3 node called WTSC75J3 and a JES2 node called WTSC75. The JES3 node runs on system SC75 and its IP address is 10.1.40.4. We begin with defining a NETSERV address space called JES3TCP, and then define a socket named J2SC75 under it.

```
*F NETSERV,ADD=JES3TCP
IAT8162 ADD COMPLETE FOR NETSERV JES3TCP
*F NETSERV=JES3TCP,SYSTEM=SC75
IAT8162 MODIFY COMPLETE FOR NETSERV JES3TCP
*F NETSERV=JES3TCP,HOSTNAME=10.1.40.4
IAT8162 MODIFY COMPLETE FOR NETSERV JES3TCP
*F NETSERV=JES3TCP,PORT=175
IAT8162 MODIFY COMPLETE FOR NETSERV JES3TCP
*F NETSERV=JES3TCP,STACK=TCPIP
IAT8162 MODIFY COMPLETE FOR NETSERV JES3TCP
*I NETSERV=JES3TCP
IAT8707 NETSERV INQUIRY RESPONSE
INFORMATION FOR NETSERV JES3TCP
SYSTEM=SC75, HOST=10.1.40.4, PORT= 175, STACK=TCPIP, JTRACE=NO,
VTRACE=NO, ITRACE=NO, ACTIVE=NO
SOCKETS DEFINED IN THIS NETSERV
NONE
END OF NETSERV INQUIRY RESPONSE
*F SOCKET,ADD=J2SC75
IAT8160 ADD COMPLETE FOR SOCKET J2SC75
*F SOCKET=J2SC75,NETSERV=JES3TCP
IAT8160 MODIFY COMPLETE FOR SOCKET J2SC75
*F SOCKET=J2SC75,HOSTNAME=10.1.40.4
IAT8160 MODIFY COMPLETE FOR SOCKET J2SC75
*F SOCKET=J2SC75,PORT=2345
IAT8160 MODIFY COMPLETE FOR SOCKET J2SC75
*F SOCKET=J2SC75,NODE=WTSC75
IAT8160 MODIFY COMPLETE FOR SOCKET J2SC75
*I SOCKET=J2SC75
IAT8709 SOCKET INQUIRY RESPONSE
INFORMATION FOR SOCKET J2SC75
NETSERV=JES3TCP, HOST=10.1.40.4, PORT= 2345, NODE=WTSC75,
JTRACE=NO, VTRACE=NO, ITRACE=NO, ACTIVE=NO, SERVER=NO
END OF SOCKET INQUIRY RESPONSE
```

Figure 15-2 Commands to dynamically define an NJE over TCP/IP node

15.15.3 NJE security enhancements

With TCP/IP, there is a greater potential for a hacker to pretend to be an NJE over TCP/IP node than with BSC or SNA nodes. It is much harder for someone to hack into BSC or SNA and pretend to be an NJE node. Generally, this can happen if hardware equipment is connected the wrong way. To cope with this potential risk, JES3 provides new mechanisms to secure the NJE over TCP/IP communications. These are secure signon and the transport layer secure (TLS) protocol.

Secure signon

Secure signon is selected by using the SECSIGNON=YES/NO parameter on the NJERMT statement. It uses profiles in the RACF APPCLU class to provide an encryption key. Name the RACF profile `NJE.homenode.remotenode`.

A node participating in secure signon supplies a random string to its NJE partner. The other node encrypts the string and sends the result back to the first node. At the same time, it supplies its own random string. The first node must then encrypt that string and send the result back. If both nodes have the same encryption key defined to the RACF APPCLU class, the encryptions will yield the same result and the nodes are allowed to sign on. Otherwise, the signon fails. This is a means for two nodes to authenticate one another. Figure 15-3 illustrates the process of secure signon.

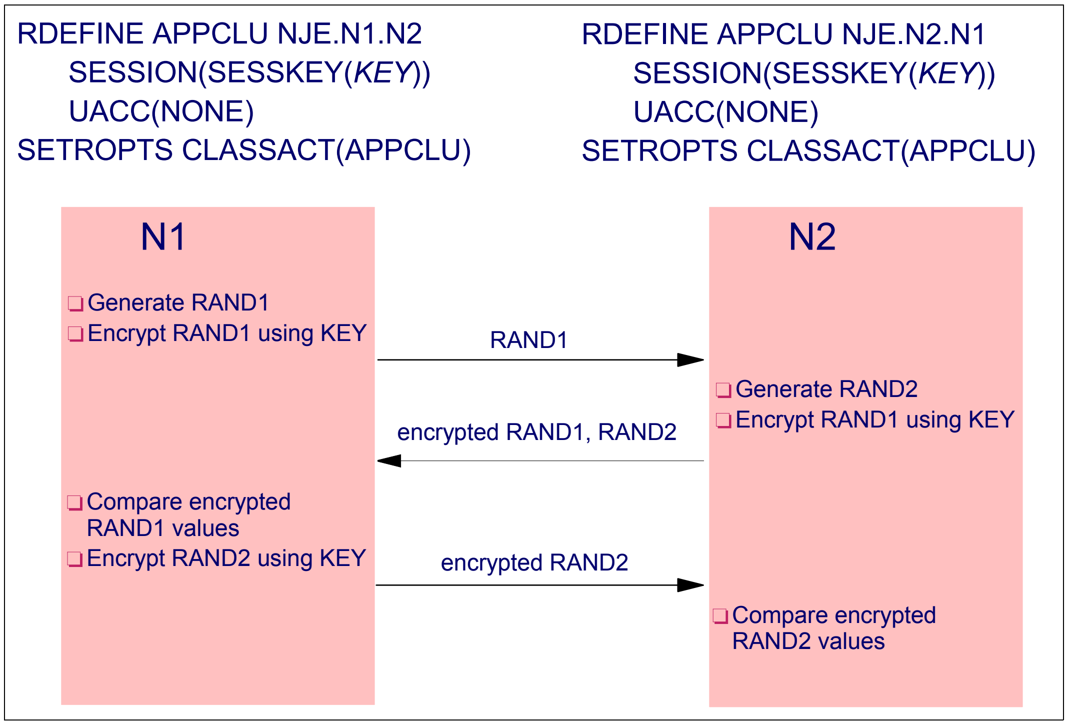


Figure 15-3 The process of secure signon

Transport layer secure (TLS)

The TLS protocol provides communications security over TCP/IP. The protocol uses encryption to allow two applications to communicate safely. To indicate TLS is desired, use the TLS=YES parameter of the NJERMT statement or the *MODIFY,NJE command.

TLS support is transparent to JES3 and is provided by z/OS Communications Server. For further information about TLS, see *z/OS Communications Server: IP Configuration Guide*, SC31-8775.

15.15.4 JES3 commands to start TCP/IP NJE

Now that the node NETSERV and socket are defined to JES3, we need to start them. The JES3 node is connecting to a JES2 node. This is shown in Figure 15-4 on page 201.

```
*X TCP,NETSERV=JES3TCP
IAT6306 JOB (JOB04055) IS TCP , CALLED BY PELEG
IRR812I PROFILE JES3*.* (G) IN THE STARTED CLASS WAS USED 328
TO START JES3TCP WITH JOBNAME JES3TCP.
IAT6100 ( DEMSEL ) JOB IEESYSAS (JOB04056), PRTY=15, ID=JES2
ICH70001I JES2 LAST ACCESS AT 10:19:53 ON WEDNESDAY, might 10, 2006
IAT9301 TCP START SUCCESSFUL FOR SERVER JES3TCP
IAZ0542I JES3TCP IAZNJTCP for HBB7730 compiled Mar 26 2006 23:31:05
*IAZ0537I JES3TCP NJETCP SERVER WAITING FOR WORK
*S TCP,SOCKET=J2SC75
*IAZ0537I JES3TCP NJETCP SERVER WAITING FOR WORK
IAZ0543I JES3TCP TCP/IP connection with IP Addr: 10.1.40.4 Port: 2345
Initiated
*IAZ0537I NETSRV1 NJETCP SERVER WAITING FOR WORK
IAZ0543I JES3TCP TCP/IP connection with IP Addr: 10.1.40.4 Port: 2345
Successful
IEF196I IGD103I SMS ALLOCATED TO DDNAME SYS00025
IEF196I IGD104I HFS FILE WAS RETAINED, DDNAME IS (SYS00025)
IEF196I FILENAME IS (/etc/resolv.conf)
IEF196I IGD103I SMS ALLOCATED TO DDNAME SYS00028
IEF196I IGD104I HFS FILE WAS RETAINED, DDNAME IS (SYS00028)
IEF196I FILENAME IS (/etc/resolv.conf)
IEF196I IGD103I SMS ALLOCATED TO DDNAME SYS00029
IEF196I IGD104I HFS FILE WAS RETAINED, DDNAME IS (SYS00029)
IEF196I FILENAME IS (/etc/hosts)
*IAZ0537I NETSRV1 NJETCP SERVER WAITING FOR WORK
IAZ0543I NETSRV1 TCP/IP connection with IP Addr: wtsc75.itsot2.ibm.com Port:
1036 Successful
IAZ0544I NETSRV1 LINE5 NJE connection with IP Addr: wtsc75.itsot2.ibm.com
Port: 1036 Successful
$HASP200 WTSC75J3 STARTED ON LNE5 NODE WTSC75 BFSZ=32768
IAT9305 NODE WTSC75 SIGNED ON NETSERV JES3TCP SOCKET J2SC75
IAZ0544I JES3TCP J2SC75 NJE connection with IP Addr: wtsc75.itsot2.ibm.com
Port: 2345 Successful
*I NETSERV=JES3TCP
IAT8707 NETSERV INQUIRY RESPONSE 733
INFORMATION FOR NETSERV JES3TCP
SYSTEM=SC75, HOST=10.1.40.4, PORT= 175, STACK=TCPIP, JTRACE=NO,
VTRACE=NO, ITRACE=NO, ACTIVE=YES
SOCKETS DEFINED IN THIS NETSERV
SOCKET ACTIVE NODE SERVER
J2SC75 YES WTSC75 NO
END OF NETSERV INQUIRY RESPONSE
```

Figure 15-4 Commands to start the NETSERV address space and socket

15.16 JES2 supported protocol for TCP/IP networking

The protocol used is based on the same one used by VM (RSCS). Originally described as BITNET II, this protocol has been assigned:

- Port 175 for non-secure sessions
- Port 2252 for secure sessions

The protocol is essentially the BSC CTCA protocol with an IP wrapper and some additional sign-on protocols. The original protocol was designed for IPv4 and uses hardcoded IP addresses in some of the data records. This original protocol is tolerated; however, a newer protocol that uses IPv6 addresses and does not send IP addresses in the header is the preferred protocol. The formal documentation of this protocol, with upgrades to this release, is in publication:

z/OS Network Job Entry (NJE) Formats and Protocols, SA22-7539

It can be found on the Internet at the following site:

http://publibfp.boulder.ibm.com/cgi-bin/bookmgr/BOOKS/iea1m503/6.5

15.16.1 Secure sign-on protocol

One of the enhancements being made is secure sign-on protocol using SAF/RACF. This new protocol applies to TCP/IP NJE as well as BSC and SNA sign-ons, and has the following characteristics:

- Uses SAF/RACF APPCLU class
- Is controlled by NODEnnn SIGNON=SECURE|COMPAT

TCP/IP NJE supports all the NJE constructs (ENDNODE, SUBNET, Store and forward, and so on) that the owning JES supports.

15.16.2 TCP/IP NJE address space

One of the primary concerns customers have had with JES2 address space outages is that when the JES2 address space ABENDs, all NJE connections are lost. Even though the JES2 address space can be restarted, the NJE connections must also be reinitialized. Many customers have automation to do this, but only when the system is IPLed. So, the JES2 address space outage turns into a system outage.

In order to address this, the TCP/IP NJE support is moved to a separate address space, where the connection can remain active when the JES2 address space is unavailable. This approach has multiple benefits:

- Availability

  Outages of the JES2 address space do not affect the availability of the NJE connection. Conversely, problems in the NJE address space do not result in a JES2 outage as they would have in the past.

- Performance

  Most of the work associated with the NJE connection (I/O, building headers and trailers, and so forth) is being done outside the JES2 address space, rather than under the JES2 main task. This frees up cycles in the already overtaxed JES2 main task to do other things.

The actual communication with TCP/IP is being done via a new common JES2 and JES3 component, IAZNJTCP.

The $S NETSRVx command is used to create the NETSERV address space. The name of the address space is based on the owning JES2 subsystem name and the associated NETSERV number. The PGM= for the address space is IAZNJTCP. The ASCRE macro is used with the IEESYSAS PROC. Figure 15-5 on page 203 shows the new address space name.

```
jesxSnnn
Where: jesx – subsystem name
nnn – NETSRV number
```

Figure 15-5 JES2 NETSRV address space name

JES2 definitions

The owning JES is responsible for:

- Initialization statements
- The command interface
- Network topology
- Selecting jobs or SYSOUT to transmit
- The creation of new job or output structures for inbound data

TCP/IP NJE address space functions

The TCP/IP NJE address space main functions are:

- Communicates with TCP/IP
- Creates or processes NJE data streams
- Reads and writes data from the JES spool

There can be multiple TCP/IP NJE address spaces associated with each JES.The TCP/IP NJE address space can be associated with one of the following:

- A specific IP address and TCP/IP stack
- All IP addresses supported by a stack
- All IP addresses known to this system

However, each TCP/IP NJE address space must have a unique port number assigned to it for incoming connections. TCP/IP NJE supports all the NJE constructs (ENDNODE, SUBNET, store and forward, and so forth) that the owning JES supports.

NETSRV devices

Each TCP/IP NJE address space is associated with a NETSRVn device in JES2, as shown in Figure 15-6 on page 204. The name of the new TCP/IP NJE address space has been chosen to aid operations in associating the address space with the corresponding JES2 NETSRV device. Since there can be multiple address spaces per JES2 and multiple JES2s per MVS image, the name of this new address space must include the owning JES2 name as well as the internal JES2 structure identifier.

The format that was chosen is jesxSnnn where jesx is the owning JES name (for example, JES2) and nnn is the associated JES2 network server number. For example, JES2S001 is the address space name associated with the JES2 subsystem, network server NETSRV1.

Code that runs in the new address space is common code between JES2 and JES3 (IAZ component) that calls JES-specific processing routines. It is the responsibility of the JES to start the address space, and in the address space initialization routine, initialize an interface control block.

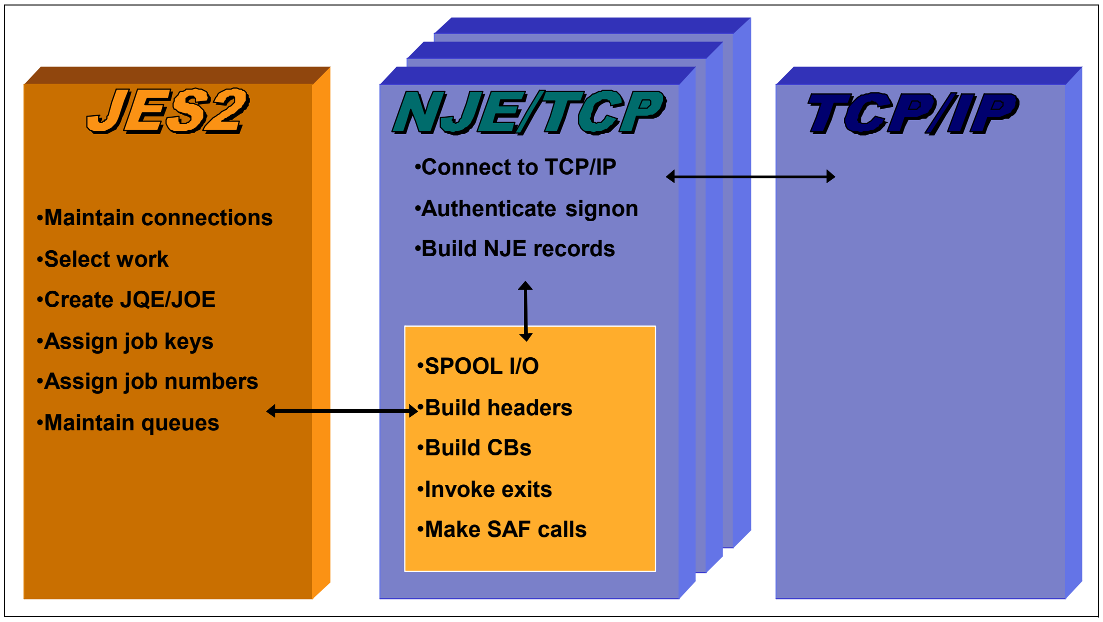


Figure 15-6 Networking address spaces with TCP/IP

Losing NJE connections

In addition to the new support for TCP/NJE, a number of external changes are being driven by the nature of how TCP/NJE operates. SNA/BSC NJE connections are lost when the JES2 address space terminates. As a result there is no need to preserve any state information over a JES2 hot start. However, TCP/NJE connections, by design, persist over a JES2 hot start. This implies that state information must be preserved over a hot start and that warm start (hot start) processing cannot just discard jobs that are on JOB and SYSOUT receivers.

The major state information that must persist over a hot start is connection information. This is the information used by the network path manager (NPM) to determine the network topology (how to get from node A to node B). Previously, it was assumed that when a node went down, any connection records ($NATs) could be marked inactive. Now, for TCP/IP NJE connections, they must be maintained unless the MVS also goes down. Furthermore, the node definitions cannot change over a hot start, since connections are based on node statements and changing node definitions can have unpredictable results.

15.16.3 JES2 NODE definitions

To preserve node definitions (and to address a number of user requirements), portions of the NODE initialization statements are saved in a new checkpoint CTENT (NAME=, SUBNET=, PATHMGR=, PRIVATE=, ENDNODE=). The result is that the values now have a JESPLEX (MAS) scope and will persist over a hot start. These operands on a $TNODE commands are propagated to all members of the MAS. Unlike other MAS scope settings, the values in the initialization deck are used on all but a hot start. This is because most customers do not use operator commands to update NJE settings, but rather a restart of JES2.

Because of the new CTENT and other changes for TCP/IP NJE, the following new support is included in this support:

- You can increase the number of defined nodes via the following operator command:
  - $TNJEDEF,NODENUM=num
- Support to decrease the value was not implemented.

  – Change the local node name via the following operator command:

- $TNODE(n),NAME=name
  - This can affect the XCF Group name if it defaults to the local node.
- Change the name of another node via the following operator command even if there are existing connections to the node:
  - $TNODE(n),NAME=name

  There are two cases for this. $TNODE(num),NAME=name implies a new node and any existing connection records for the node are deleted. $TNODE(oldname),NAME=newname implies the node is being renamed and any connection records for the node are retained (and updated with the new name).

Warm start processing

Warm start processing is updated to properly handle new scenarios of a JES2 hot start with TCP/IP NJE address spaces active. JES2 must be able to reconnect to the existing TCP/IP NJE address space and properly handle jobs that were active on TCP/IP NJE. This includes jobs that were being transmitted as well as jobs that were being received. The code must also handle the case that while JES2 was down, the TCP/IP NJE address space terminated. Code must also deal with the case where the LINE or NETSRV associated with the TCP/IP NJE address was not specified in the initialization deck.

JES2 control blocks

The basic data structure at the LINE level in the JES2 address space has not changed. There is a DCT at the NETSRV level and the LINE level. Then each subdevice also has a DCT associated with it. The PCEs that represent the subdevices still exist.

15.17 Migration considerations

A JES3 to JES2 migration can take place in several ways.

15.17.1 Single event

If a JES3 to JES2 migration is performed as a single event, that is, the JESplex running JES3 is shut down and re-IPLed with JES2, it is possible to retain the original NJE node name. Connected systems still need to be made aware of the change, as they might need to make changes as noted in the next section.

15.17.2 New JESplex

An alternate approach is bringing up a separate JES2 JESplex either within the original sysplex or as a new sysplex. In this case, a new nodename will be needed to allow both JESs to communicate to external connections. Connected systems will need to add definitions for the new node. This could amount to a considerable amount of work depending on how many external systems are affected.

Areas affected include:

- NJE definitions including BDT for JES3/SNA connections
- RACF definitions if the NODES class is used (or the equivalent ACF2 or Top Secret rules)
- VTAM definitions if SNA/NJE is used

- JES2 or JES3 NJE exits if external connections do special processing
- Customer communications so people are aware of the new node
- Jobs submitted through NJE from a JES3 node must have an NJB statement acceptable to the executing system. Submitting a job containing an NJB statement on a JES2 system will cause a JCL error. These statements must be commented out or removed during cutover.

If a gradual cutover of applications from JES3 to JES2 is made rather than scheduling a single event, the connected systems might need to make adjustments on a per case basis depending on the application involved. This might be a drawn-out process if the conversion does not happen quickly.

Note: JES2 requires a cold start to change the OWNNODE parameter of the NJEDEF statement. In other words the node name cannot change without another cold start.

16

Accounting and chargeback considerations

This chapter describes accounting and chargeback considerations for JES3 and JES2 environments. This information might be valuable to Information Technology (IT) departments and IT management that account for and recover cost, for the use of mainframe resources. This chapter does not go into detail about managing SMF records, nor does it discuss IBM or ISV products that perform accounting or chargeback functions. Basic information is presented about what SMF record types are created by JES2 or JES3 and the differences between the two environments. Topics included in this chapter are:

- Overview of resource usage accounting within IT
- Chargeback overview
- SMF record types
- Job accounting information
- Processing differences between JES2 and JES3

16.1 Overview of resource usage accounting within IT

Technology continues to develop, grow and evolve, resulting in increased functionality and capability, which many times translates into more complex IT systems. This is especially true in today's business climate composed of mostly composite applications, where multitier architectures are common. Over the years, businesses have become very reliant on IT solutions to run their business entities.

IT solutions have greatly increased productivity over time contributing to a higher output of goods and services. This has resulted in larger IT costs due to increased complexity and volume of new integrated application systems. These costs include the operations, software, infrastructure, and labor, among others that contribute to increasing business costs.

Today's IT management has an increased focus on the total cost of ownership (TCO) of IT solutions. In managing cost structures, it becomes very important to use the most accurate data possible to make informed business decisions. One way to manage these costs is for an installation to institute a chargeback model. In this manner, appropriate charges for allocation of resources can be more equitably distributed back to those business areas.

In today's economic times, there is added focus on IT budgets. IT capacity and performance area roles have been or new departments created entirely within IT organizations to support and manage costs and budgets. A large area of focus within these new departments is on the accounting and chargeback process to recover costs.

16.2 Chargeback overview

The primary function of the chargeback process is to recover the costs in aggregate of the investments made in the overall IT infrastructure and services. The chargeback process typically attempts to fairly distribute the cost of providing the corporate IT service across users of that service based on CPU usage or some metric.

Some areas to be considered in relation to chargeback include:

- Identifying what metrics will be used in the chargeback formula for that business application.

  Examples are CPU, print, storage, and use of off-site backups. Whatever metrics you use must be consistent across technology changes and as simple as possible. For example, if you charge for each I/O, a simple change in blocksize for a busy data set might result in a significant change in the cost of the associated jobs. Therefore, try to select metrics that will not be easily affected by trivial changes.

- Understanding service levels required by the business application.

  For example, services that require a response time guarantee or higher priority for batch work might be charged a premium for that added level of service.

- Future planning including cost reduction measures by the business application. Examples include:
  - Combining workload for application efficiency.
  - Resource routing of workload for cost reduction to a smaller system. This is especially true of ISV products that use MIP-based model pricing.

Usage information is typically stored in SMF records. The particular records that you use will depend on which metrics you base your chargeback algorithms on.

There are a number of products available from IBM and other vendors that understand the layout of the most commonly used SMF records and that can form the basis of your chargeback application. Typically, installations use third-party products to format and report this type of data. For more information about accounting and chargeback details, see *Accounting and Chargeback with Tivoli Decision Support for OS/390*, SG24-6044.

16.3 SMF record types

Both JES systems use SMF records to collect and record activity on job-related functions. Most SMF record types are unaffected by which flavor of JES you run. However, there are some record types that are created by JES2 or JES3, and some record types where the type of JES that you are running can have an impact on specific fields. The SMF records that are sensitive to the JES you are running and that are likely to be of interest from a chargeback perspective are:

- SMF Type 6 - Print and SYSOUT
- SMF Type 24 - Spool offload
- SMF Type 25 - JES3 device allocation
- SMF Type 26 - Purging the job from the system
- SMF Type 30 - Common Address Space Work
- SMF Type 43/45 - Start/Stop
- SMF Type 59 - MVS/BDT

There are other SMF record types that are created by JES2 or JES3, mainly related to NJE. However, those records are not related to chargeback so we will not cover them here. If you are interested in those record types, search *z/OS MVS System Management Facilities (SMF)*, SA22-7630, for instances of JES2 or JES3.

It is also possible that you might have JES3 user exits that create SMF records. If that is the case, you need to determine how you will handle that as part of your migration plan.

16.3.1 SMF type 6: External writer

Both JES3 and JES2 create SMF Type 6 records for output writer processing. This record identifies the output by SYSOUT class, form number, and identifies the job according to job log identification, JES2 assigned job number, and user identification. However, there are slight differences in the record for JES2 compared to JES3.

If an external or user-supplied writer is used, SMF will produce an incomplete Type 6 record. This happens only when the external writer directs output to a printer or punch. Also, if the external writer directs output to tape or disk, SMF does not produce a Type 6 either. The incomplete Type 6 record differs the JES2 Type 6 record as follows:

- The number of logical records (offset 51)
- I/O status indicators (offset 55)
- Subsystem generating Identification (offset 62)
- Data set control indicators (offset 66)
- JES2 logical output device name (offset 72)

JES3

The JES3 Output Service creates an SMF Type 6 (JES3 Output Writer) record for each data set processed. One Type 6 record is written for each data set section within an output scheduler element (OSE). If a printer is running under the control of a functional subsystem

(FSS), a Type 6 record is written on the system containing the FSS address space that processed the data set.

JES2

The JES2 writer creates a Type 6 record when processing is completed for a job output element (JOE), or when there is a change in certain information describing SYSOUT data sets processed in the same JOE. JES2 also creates a Type 6 record for SYSOUT data sets. If a printer is running under the control of an FSS, a Type 6 record is created. This record is also created for spin data sets. This behavior can be adjusted on a job class basis using JES2 exit 21.

Both JESs create the following information in the Type 6 record:

- Number of logical records processed
- Number of data sets processed
- Output service start time and date
- I/O status indicators
- Data set control indicators
- JES3 logical output device name
- Output activity

For specific information about the difference between the JES3 Type 6 records and the JES2 Type 6 record, refer to *z/OS MVS System Management Facilities (SMF)*, SA22-7630.

16.3.2 SMF type 24: JES2 spool offload

This record type is written whenever a job or SYSOUT is written to or received from a JES2 offload data set. This record identifies the name, time, and date for each job that has been transmitted or received. It includes specific information about jobs in a record subtype. For jobs not yet run, it reports job-related information such as job class and system affinity in both the job selection criteria section and in the system affinity section. For jobs that have run, it reports information about SYSOUT data sets such as output group ID and forms name in a SYSOUT selection criteria section and the job selection criteria section.

There is no equivalent record for JES3.

16.3.3 SMF type 25: JES3 device allocation

A Type 25 record is written for each job that completed JES3 converter/interpreter (C/I) processing. One Type 25 record is written for each job, whether the job contains DD statements or not. A separate Type 25 record is written for each job that uses a private catalog or each Main Device Scheduling (MDS) dynamic unallocation request.

This record also contains allocation-related information such as the number of tape and disk volumes fetched and mounted, and the time of the JES3 device verification. The job log identification is composed of the job name, time, and date that the reader recognized the JOB card of the job. This does not have an equivalent in JES2.

16.3.4 SMF type 26: job purge

The SMF Type 26 record is created by both JES3 and JES2 and contains information about job purge processing. Note that the record mapping for the Type 26 record is slightly different for JES2 and JES3.

JES3

A Type 26 record is created by JES3 at job purge after all SYSOUT for the job has been processed. This record identifies the job by job log identification, JES3 assigned job number, and the programmer's name.

The JES3 Type 26 record contains the following information:

- Message class
- Job class
- JES3 job selection priority
- JES3 logical input device name
- Processing time
- Output lines
- Output punched cards
- Deadline schedule type (not in JES2)
- Deadline schedule time and date (not in JES2)
- Start and stop times for:
  - Reader
  - Converter
  - Execution processor
  - Output processor

JES2

After all SYSOUT processing is complete, JES2 calls SMF exit IEFUJP. This allows an installation to decide whether to write a Type 26 record at job purge. Similarly to JES3, this record identifies a job by job log identification, JES2 assigned job number, and programmer's name. JES2 can be configured to not create this record type.

Record type 26 in JES2 contains operating information such as:

- Message class
- Job class
- JES2 job selection priority
- JES2 logical input device name
- Processing time
- Output lines
- Output punched cards
- Start and stop times for"
  - Reader
  - Converter
  - Execution processor
  - Output processor

For specific information about the layout of the two SMF records, refer to *z/OS MVS System Management Facilities (SMF)*, SA22-7630.

16.3.5 SMF type 30: common address space work

The Type 30 SMF record is the SMF record type that is most commonly used by accounting and chargeback programs. This record consolidates data that is found in other type records. It consolidates data that is found in record types 4, 5, 20, 34, 35, and 40, and it provides additional information. The Type 30 is the recommended record for chargeback and accounting because the record types that it replaces are generally not being updated with new measurement data.

Most of the fields in the type 30 record do not come from JES3 or JES2 directly. Note that the type 30 record has many subtypes; it is not the intent to cover them all here. Also, information might differ by subtype. Examples of subtype data recorded can include:

- Job log identification
- Step name
- Number of steps within the job
- User Identification
- Program name
- Performance group number or service class name
- JES job number

There are a few Type 30 field differences between JES3 and JES2. They are as follows:

- SMF30JPT - JES input priority for JES itself
- SMF30CL8 - 8-character job class (left-justified, padded with blanks)
  - JES2 taken from the SFM30CLS field (if not specified)
  - JES3 taken from the CLASS= parameter on //*MAIN (if valid) or the JES3 default (JES3BATCH)
- SMF30RDR - Reader device class as defined in JESPARMS
- SMF30RDT - Reader device type as defined in JESPARMS

For more information about the SMF record types, refer to *z/OS MVS System Management Facilities (SMF)*, SA22-7630.

16.3.6 SMF type 43: JES start

These records are written when either JES is started. For JES3, a Type 43 record is written during JES3 and the converter/interpreter functional subsystem (C/I FSS) initialization. This contains an indicator for the type of JES3 Start, JES3 initialization deck origin type, and contents, and JES3 procedure name.

16.3.7 SMF type 45: stop

For JES3, a Type 45 record is written when JES3 and C/I FSS are terminated.

For JES2, a Type 45 record is written when a **$PJES2** command (to withdraw JES2 form the system) is issued.

16.3.8 SMF type 59: MVS/BDT

A Type 59 record is written when MVS/Bulk Data Transfer (MVS/BDT) completes a file-to-file transmission over an SNA connection. When the connection is BSC, BDT does not write the record: in that case, JES3 writes a Type 57 record instead. MVS/BDT writes Type 59 records from the global node where the transaction is queued and produces a record regardless of whether the transmission was successful or not.

This record type does not exist for JES2.

16.4 Job accounting information

Many installations use the accounting information on the job accounting string on the //JOB card. The JOB card can contain two positional parameters (see Table 16-1) as well as keyword parameters. The positional parameters are accounting and programmer name and must be coded before any keyword parameters. Within the positional parameters, the accounting information is the first one.

Table 16-1 Format of accounting and programmer name fields on the JOB card

Positional Parameters	Values	Purpose
([account-number] [,accounting-information]...)	account- number ,accounting-information: up to 143 characters	Specifies an account number and other accounting information, formatted as required by the installation. This parameter might be required by the installation.
Programmer's-name	programmer's-name: 1- 20 characters	Identifies the owner of the job. This parameter might be required by the installation.

Syntax

```
([account-number][,accounting-information]...)
```

Where account-number is installation-defined accounting information and accounting-information is more installation-defined accounting information. For example, department or room number.

There are nine different subtypes of the accounting-information parameter. They are as follows:

Subparameter syntax
(pano,room,time,lines,cards,forms,copies,log,linect)
Code a comma in place of each omitted subparameter when other subparameters follow

Here is an example of the positional nature of the accounting information.

Example 3

//JOB45 JOB (AB01,PL105,15,,,,3)

This statement shows a JES2 accounting information parameter where programmer's accounting number is AB01, room number is PL105, estimated job time is 15 minutes, and three copies are requested.

For more information, refer to *z/OS V1R13.0 MVS JCL Reference*, SA22-7597.

In addition, JOB accounting information can be overridden in JES2 by the /*NETACCT statement and in JES3 by the //*NETACCT. Both of which are used to specify an account number for a network job.

16.4.1 Scanning the JOB statement accounting field

Use the ACCTFLD= parameter on the JOBDEF statement to control whether JES2 is to scan the accounting field of the JOB statement. The accounting field is valid if its format matches the format specified for JES2. (For a description of the JES2 format for the accounting field, see the accounting field parameter described in *z/OS MVS JCL Reference*, SA22-7597.)

JOB statement accounting field scan exit

You can supply an Exit 3 or Exit 53 installation exit routine to scan and change the accounting field in the JOB statement. See *z/OS JES2 Installation Exits*, SA22-7534 for information about Exit 3 or Exit 53 and for the exit routine's relationship to the ACCTFLD= parameter on the JOBDEF initialization statement.

Network accounting

The JES2 NETACCT initialization statement determines the correspondence between the network account numbers and the local account numbers for a node. JES2 uses the parameters on the NETACCT statement to build the network account table used by input processing, the job and SYSOUT transmitters, and the job and SYSOUT receivers.

JES2 recognizes the following network accounting control statement:

```
/*NETACCT=vvvvvvvv
```

where vvvvvvvv is a 1-byte to 8-byte alphanumeric character network account identifier. When the input service processor encounters this statement, it inserts the network account identifier in the job's job header and looks up the number in the network account table built at JES2 initialization to find an associated local account number. If a local account number is found, JES2 will move the number into the job's job control table, overriding any account number present. For more information, see *z/OS JES2 Initialization and Tuning Guide*, SA22-7532.

16.5 Processing differences

Most of the accounting information for JES2 and JES3 are similar. One item that to be aware of in terms of accounting and chargeback is the slight differences in where some JES-related processing takes place. This is centered around the fact the two JESs, at an elementary level, function differently.

JES3 is made up of two components, the JES3 Global and Local. In JES3, all processing is serialized through the JES3 Global before any execution takes place. Whereas in JES2, this concept does not exist and set up is done differently in JES2.

The JES3 Global is where the bulk of the work is done, such as all the converter/interpreter (C/I) setup and might then run the job in JES3 Local on another system. Whereas in JES2, the converter function is completed in the JES2 address space and interpretation is completed in the Users address space when it is selected for execution. So now, some setup that used to be in the JES2 MAS and still is in the JES3 Global can cause a shift of CPU to be used in the Users address space. Overall, this is more efficient processing. Since most of the overhead set up in JES3 is done on the Global, the total CPU time associated with the job might only reflect what was in the Local, thus skewing the CPU results.

Refer to 13.1, "Performance of JES2 compared to JES3" on page 156 for more details about this.

The difference in accounting stems from the fact that most installations probably absorb the costs that are associated with setup processing in JES3 Global. This means, currently, the CPU cost isn't counted toward the business application area running the job. In JES2, since this does not exist, there might be a slightly higher cost associated with running the same job in JES2. In essence, the business unit is being fairly billed, since the costs that were taken in with JES3 Global processing should have been federated to the business units anyway. This is something to review and keep in mind as your installation migration effort reaches a point of testing where there is enough volume in the system to analyze the differences. As part of a migration plan, this type of information must be communicated to business areas so they can make adjustments.

Use information that you gathered at your installation that is based on the activity in this chapter to be stored for later use. In particular, review 5.2, "Discovery analysis process" on page 81 about using this information as part of that process.

17

Availability considerations

This chapter describes the considerations of availability in each of the JES environments and compares how each of them are maintained.

17.1 Standard practices for availability

It is assumed that for either JES configuration, IBM standard practices are followed with regards to availability. All configurations must maintain redundant processes that are documented and known to operations support staff. For example, neither JES configuration should require an all-systems sysplex IPL for normal routine maintenance. The concept of a rolling IPL can be the standard for change management.

Also, it is assumed that in either environment, there is security protection from the more destructive operator commands.

17.2 Tailoring JES for best availability

In normal situations, there are parameters in both JES3 and JES2 that help building a more reliable, hence available, environment.

17.2.1 JES3 global-local communication

Implementing XCF signaling through coupling facility list structures provides significant advantages in the areas of systems management and recovery, and thus, provides enhanced availability for sysplex systems, as global/local communication is concerned.

17.2.2 JES3 spool partitions

A spool partition is a logical grouping of spool data sets. You control five factors:

- The number of spool partitions used.
- The number of spool data sets that are in each spool partition.
- The work load distribution across spool partitions.
- The type of spool data to be included in each spool partition.
- The size of a track group for each partition.

These factors influence the reliability, availability, and serviceability (RAS) of spool data sets and the performance impact of accessing a spool data set.

If a spool data set fails, the failure affects only a subset of the jobs in the JES3 complex. That is, the failure affects only those jobs that have data in the spool partition including the failed spool data set, not jobs that have data in other spool partitions. (The failure might not affect all jobs in that spool partition, however. Some jobs might not have had any data on the failed data set.) Thus, spool partitioning improves spool RAS.

17.2.3 JES3 spool volume recovery

Spool recovery procedures and operator commands are an important part of maintaining the JES3 system. When spool volume errors occur, you must have procedures in place to provide recovery from any type of error.

The `*F Q` command allows you to control activity on a spool data set. For spool volume recovery, you can do the following:

You can stop JES3 from allocating additional space on a specific spool data set and then restart space allocation processing at a later time. This action does not affect the jobs that already have data on this data set; the jobs continue to run in the normal manner.

If necessary, you can place a spool data set and all jobs with spool data on the data set in hold status and release both the data set and the jobs at a later time.

Another parameter allows you to place the data set in hold status and cancel all jobs with spool data on the data set. You then can release the data set from hold status and resume allocating space on the data set.

All these changes remain in effect when you restart JES3, using a hot or warm start:

- Avoid cold starts due to spool failures
- Single track errors or large numbers
- Provide method for suspending volume use
- Prevent new allocations to volume
- Replace spool volumes
- No RAS facilities for JCT and Checkpoint

Other techniques could also be possible.

17.2.4 JES3 system select phase

The system select phase of MDS is performed when a job requires one or more resources managed by the storage management subsystem (SMS). If the job does not require any SMS-managed resources, the job proceeds directly to MDS allocation. JES3 is not aware of the availability or connectivity of SMS-managed resources. If a job requires SMS-managed resources, JES3 requests SMS to determine the availability of those resources and to determine which mains have access to those resources. If JES3 determines that one or more mains have access to all of the required resources, the job proceeds into the allocation phase.

JES3/DFSMS communication

JES3 SMS support provides complex-wide data set awareness for DFSMS-managed data sets through subsystem interface communication with DFSMS, main processor, and DFSMS resource availability are determined for scheduling jobs into execution using these interface calls.

JES3 and DFSMS communication requires to:

- Make sure that catalog locates are done on a processor with access to the required catalogs.
- Make sure that jobs requiring DFSMS resources execute on processors to which the resources are accessible and available.
- Provide complex-wide data set awareness for all DFSMS managed requests (even for new, non-specific requests).
- Remove JES3 awareness of units and volumes for DFSMS managed data sets. (One of the DFSMS objectives is to remove user awareness of the physical storage.)

17.2.5 JES2 checkpoint data set

We described checkpoint data sets in 1.2.5, “Checkpoint data set” on page 14. When you review the allocation of checkpoint data, your interests are not solely performance related. The prime objective is availability. You need to adopt either dual or duplex mode for your checkpoint. Duplex mode must always be established when using a coupling facility structure for your checkpoint.

Dual mode can only be established when coupling facilities are not being used. In dual mode, processor overhead is used to reduce the amount of data transferred to the device and thus shortening the total time for the I/O to complete. Dual mode can add an additional 10% in JES2 processor cycles.

Generally, with modern DASD, the amount of I/O delay reduced by adopting dual mode is often consumed by the extra time needed by the additional JES2 processor cycles. Therefore, dual mode does not provide a significant advantage.

When configuring checkpoint, we do not recommend placing both checkpoints on coupling facility structures or placing the primary checkpoint on DASD and the secondary checkpoint on a coupling facility structure.

If both checkpoints reside on coupling facilities that become volatile (a condition where, if power to the coupling facility device is lost, the data is lost), your data is more susceptible than when a checkpoint data set resides on DASD. If no other active MAS member exists, you can lose all checkpoint data and require a JES2 cold start.

Placing the primary checkpoint on DASD while the secondary checkpoint resides on a coupling facility provides no benefit to an installation.

17.2.6 Coupling facility structure duplexing

The introduction of coupling facility structure duplexing with CFLEVEL12 on System z processors with z/OS V1R4 and above gives recommended availability benefits. There is no significant performance overhead to JES2 itself, and it will avoid the operational use of the JES2 checkpoint reconfiguration dialog.

17.2.7 JES2 SPOOL partitioning

SPOOL partitioning (or fencing) allows track group allocation across explicit volumes. Standard JES2 processing allows the allocation of track groups across any available spool volume. By fencing volumes, you can improve JES2 performance. Frequently, run system jobs can be isolated on separate volumes leaving the remaining volumes to be used by user-defined work. This can realize performance improvements for sysout intensive batch jobs. However, in general terms, performance will be better with FENCE=NO. Jobs might access all volumes, so increasing pathing capability and reducing the impacts of IOSQ and PEND time.

The main benefit of fencing is availability. Without it, the loss of a spool volume can result in the loss of all jobs. With it, jobs can be limited to one volume and isolation of the failure. With the common usage of RAID DASD, spool fencing might seem less beneficial with data striped across multiple physical volumes. However, this only caters to the physical residence of the data. There are potential failure scenarios at UCB level that continue to qualify the availability benefits of fenced volumes.

Spool fencing is enabled by the `FENCE` parameter on the SPOOLDEF initialization statement. Exit11 and Exit12 facilitate masks to limit access to the volumes based on jobname or jobclass. Alternatively, fencing can be used in association with spool affinity (see section 17.2.8, “JES2 SPOOL affinity” on page 221) to control the member selection criteria for fenced volumes.

17.2.8 JES2 SPOOL affinity

Spool affinity is an alternative (or addition) to fencing and provides a mechanism to logically partition the spool. This gives availability benefits. It facilitates the split of a large DASD spool pool into smaller pools for critical systems. It also provides a mechanism to have spool space online and ready to use if spool volumes start to fill up.

17.3 Updating the configuration

The following section provides a brief overview of what is required to update the configuration of the respective JES systems.

17.3.1 In a JES3 environment

There are three types of configuration changes to consider: JES3 initialization parm changes, system changes, and JES3 Exit changes.

JES3 initialization changes

Changes to most JES3 parameters do not require JES3plex IPLs, and can be done dynamically by performing a JES3 hot start with refresh. This is a recycle of the JES3 address space on the Global system and then restarting it using a new inish deck with a certain parameter member.

Dynamically adding volumes with SPOOL ADD and SPOOL DELETE (using FORMAT, SPART, and TRACK) can be done in JES3. However, spool delete requires a rolling IPL because a “spool migrate” function does not exist.

The following changes also require a JES3plex IPL:

- BADTRACK (bypass defective tracks)
- CONSTD (console service standards)

System changes

In a JES3 environment, system changes via IPL can be implemented using a “Rolling IPL” method for all JES3 Local systems but then must be considered for implementing on the JES3 Global system. When implementing on the JES3 Global system, either you must tolerate an outage of the JES3 Global system and suspend processing on all the Locals at the time or perform a DSI to move the Global function to another system in the sysplex before IPLing.

JES3 exits implementation

To implement changes to JES3 exits, the dynamic exit facility can be invoked. However, ensuring that all systems in the JES3plex will pick up the change and test it before production implementation requires a separate JES3plex configuration. Be aware that the default exits have the capability of setting a flag to prevent any additional calls being made to them.

17.3.2 In a JES2 environment

A normal recycle of JES2 is more closely associated with a shutdown and IPL of a system since all jobs running on the system are submitted under the JES2 subsystem. This can be seen in response to a **`$D JES2`** command where all started tasks and jobs are shown. However, since most changes can be performed dynamically, the requirement to do this occurs less often. Most changes can be made dynamically and then added to the initparm to have it take affect permanently on subsequent IPLs.

Adding or replacing spool volumes

JES2 spool volumes in a MAS can be added by using the **`$S SPOOL`** command. Spool volumes can be removed by using the **`$Z SPOOL`** command to halt or the **`$P SPOOL`** command to drain. The procedure can be found in *z/OS JES2 Initialization and Tuning Guide*, SA22-7532.

Checkpoint reconfiguration dialog

Maintenance and changes to the checkpoint data sets are done through the checkpoint reconfiguration dialog. Operational procedures need to be created for this highly critical process. See 9.6.3, "JES2 Checkpoint Reconfiguration Dialog" on page 127 for a sample display of the dialog. The procedure is explained in further detail in *z/OS JES2 Initialization and Tuning Guide*, SA22-7532.

JES2 exits implementation

Since each system in a JES2 configuration is independent, each system might have its own set of JES2 exits. JES2 provides more dynamic capability than JES3. Implementation of changes to new exits either dynamically or with a system IPL can be implemented using a rolling IPL method.

For more information, see:

http://publib.boulder.ibm.com/infocenter/ieduasst/stgv1r0/topic/com.ibm.iea.zos/zos/1.7/JobEntrySS/JES2_Exits_Overview.pdf

17.3.3 Secondary JES

JES2 can be started as a secondary subsystem when the primary subsystem is JES2. This is also known as *Poly-JES* and is primarily used for testing new configurations, release or Initparms in JES2. It would not be recommended to remain in production for any length of time since there are known limitations from other system products and their support. For example, WLM does not support a secondary JES2 subsystem in the same MAS as the primary. WLM-managed initiators will only select jobs from the primary JES.

Note: JES2 can run as a secondary subsystem when the primary subsystem is JES3, but is not supported.

17.4 Unplanned outage

This section describes the impact of an unplanned outage in each of the JES configurations with a brief review of the recovery process.

17.4.1 JES3 unplanned outages

The following describes the two types of unplanned outages in a JES3 configuration, JES3 local outage and a JES3 global outage.

JES3 local outage

Outside of a normal shutdown of a JES3 local, using the `8RETURN` command, an outage of a JES3 local system only requires a restart of the local JES3 address space. If a JES3 local fails to restart, a possible IPL of that system or the JES3plex would be required and could be a symptom of a larger problem.

JES3 global outage

If the JES3 global system goes down or the JES3 address space on the current global system goes down, job processing is quiesced for all systems in the JES3plex when an element of work on the local wants a JES service. Thus IMS, DB2, and CICS (for example) might not be immediately impacted. Immediate recovery must be attempted depending on the recovery scenario. If the JES3 global address space cannot be restarted with a hot start on the currently defined JES3 global system, there are different options that can be considered:

- Restart the JES3 address space with HA start and remove any jobs that might be causing the problem.
- Re-IPL the global system and attempt to restart JES3.
- Perform a Dynamic System Interchange (DSI) and move the global function to another system in the JES3plex.

17.4.2 JES2 unplanned outage

The configuration of a JES2 JESplex is set up so that each system acts independently of each other and an outage in any one system does not affect any other JES2 system in the complex. A sample JES2 configuration is shown in Figure 17-1. Note in this configuration that member SYSA has AUTOEMEM=YES in the MASDEF of the init parm defined and SYSB and SYSD have RESTART=YES defined.

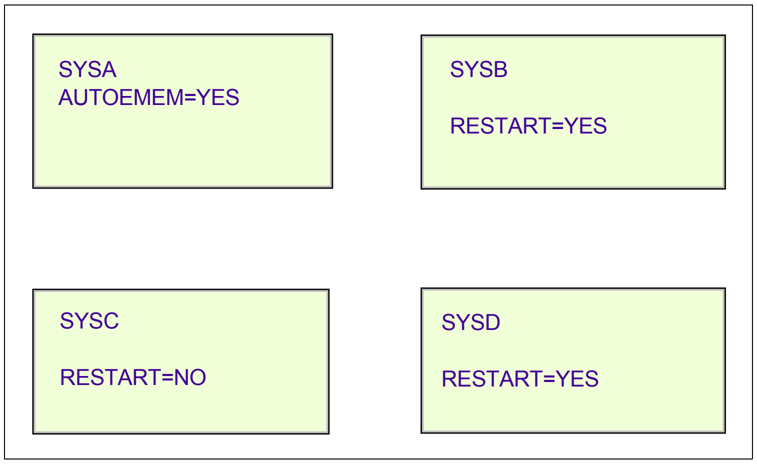


Figure 17-1 Four system JES2plex

If one of those members, system SYSA in the example, is removed from the JESplex, the work scheduled to that system can either wait for that system to be restarted into the JESplex or be automatically attempted to be restarted on any other eligible system in the JESplex as shown in Figure 17-2 on page 224.

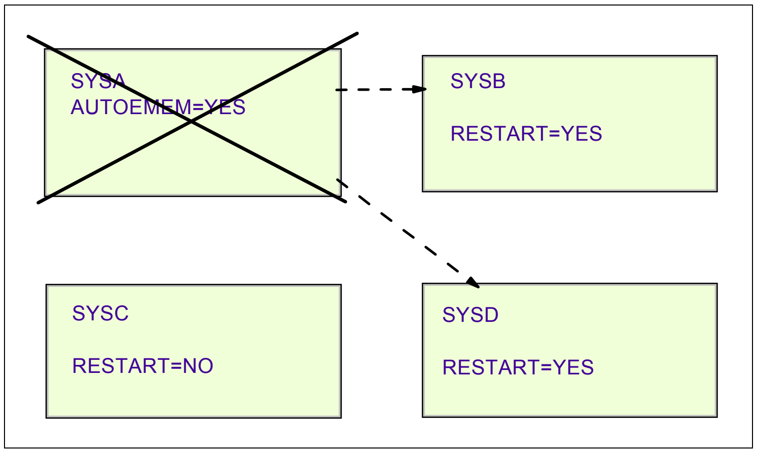


Figure 17-2 Four system JES2plex with one system removed

Any work scheduled to that system can either wait for that system to be restarted into the JESplex or be automatically attempted to be warm started on any other eligible member in the JESplex that has RESTART=YES.

Note: The usage of AUTOEMEM might not be what is intended for all installations. Consideration needs to be made for any applications with a system affinity to a particular system or set of systems. For example, some production DB2 applications are only available on SYSA and SYSC and cannot run on SYSB and SYSD, so they would not be candidates for this system switch if SYSA were to come down. In this case, it would be preferred to use automatic restart manager (ARM) or automation.

17.4.3 JES2 abend

If the JES2 address space itself abends or a system that contains a JES2 subsystem terminates abnormally, the behavior is the same as the system coming down and would depend on the recovery options in the configuration.

An operator initiated abend can be displayed as shown in Example 17-1.

Example 17-1 Sample response after $P JES2,ABEND is entered

```
*$HASP198 REPLY TO $HASP098 WITH ONE OF THE FOLLOWING: 101
   END            - STANDARD ABNORMAL END
   END,DUMP       - END JES2 WITH A DUMP (WITH AN OPTIONAL TITLE)
   END,NOHOTSTART - ABBREVIATED ABNORMAL END (HOT-START IS AT RISK)
   SNAP           - RE-DISPLAY $HASP088
   DUMP           - REQUEST SYSTEM DUMP (WITH AN OPTIONAL TITLE)
*062 $HASP098 ENTER TERMINATION OPTION
```

This can also be done when a system is removed from the MAS by using the **`$E MEM(`*`sysname`*`)`** command. This also restarts jobs on other eligible systems. Note: this command will be ignored on an active system.

17.4.4 JES2 checkpoint data set errors

It is recommended to have an alternate entry for the JES2 checkpoint data set so any errors that are encountered will be automatically detected and switched to the alternate or the checkpoint reconfiguration dialog is displayed.

JES2 checkpoint data set configuration

It is recommended that the JES2 configuration include a primary checkpoint data set that can be defined in the Coupling Facility as a structure or on DASD and an alternate on DASD.

Note: If you only have one Coupling Facility (CF), do not define both the CKPT1 and CKPT2 data sets as structures to that CF. Make sure that at least one checkpoint data set is defined on DASD. Otherwise, you might unintentionally lose your checkpoint data sets if access to your CF is disrupted, which will potentially require a JES2 cold start to recover.

Also, for checkpoint data sets that are defined on DASD, it is recommended to add the entry to the GRSRNLxx PARMLIB member to minimize contention on the data set.

A

Sample JES3 exit to analyze JECL usage

This appendix contains sample code for JES3 user exit 33 that helps you detect the use of JCL or JECL statements that would require reviewing and possibly replacing as part of the move to JES2.

Sample JES3 user exit 33

Example A-1 Sample code for detecting JCL and JECL

```
UX33     TITLE 'JES3 CONTROL STATEMENT USER EXIT'                       00010000
IATUX33  AMODE 31                                                       00011000
IATUX33  RMODE ANY                                                      00012000
         IATYASM                                                        00013000
*START OF SPECIFICATIONS*********************************************** 00020000
*                                                                     * 00021000
* MODULE-NAME = IATUX33                                               * 00022000
*                                                                     * 00023000
* $MOD(IATUX33)    PROD(JES3):                                        * 00030000
*                                                                     * 00040000
* DESCRIPTIVE NAME=                                                   * 00050000
*           JES3 CONTROL STATEMENT USER EXIT                          * 00060000
*                                                                     * 00070000
* *01* PROPRIETARY STATEMENT=                                         * 00080000
*  **PROPRIETARY_STATEMENT********************************************* 00090000
*                                                                     * 00091000
*                                                                     * 00100000
*   LICENSED MATERIALS - PROPERTY OF IBM                              * 00110000
*   THIS MODULE IS "RESTRICTED MATERIALS OF IBM"                      * 00120000
*   5694-A01 COPYRIGHT IBM CORP. 2013                                 * 00130000
*                                                                     * 00140000
*   STATUS= HJS7780                                                   * 00150000
*                                                                     * 00160000
*  **END_OF_PROPRIETARY_STATEMENT************************************** 00170000
*                                                                     * 00171000
* Input Registers =                                                   * 00190000
*           R0    Irrelevant                                          * 00200000
*           R1    Address of current JCL statement                    * 00210000
*           R2-R9 Irrelevant                                          * 00220000
*           R10   IATUX33 base register                               * 00230000
*           R11   IATYFCT address                                     * 00240000
*           R12   IATYTVT address                                     * 00250000
*           R13   IATYISD input service data area                     * 00260000
*           R14   Return address                                      * 00270000
*           R15   Entry point address                                 * 00280000
*                                                                     * 00290000
* Entry purpose =                                                     * 00300000
*           IATUX33 is entered for each logical record of JCL         * 00310000
*           EXEC statements and for JES3 control statements           * 00320000
*           except DATASET/ENDDATASET.                                * 00330000
*                                                                     * 00340000
*                                                                     * 00350000
* Input =   R1 points to the current JCL record                       * 00360000
*                                                                     * 00370000
* Exit  =   ARETURN=0                                                 * 00380000
*                                                                     * 00390000
*           *** R1 MUST NOT BE CHANGED ***                            * 00400000
*                                                                     * 00410000
* Output =  JES3 JECL statements have been tracked.                   * 00420000
*                                                                     * 00430000
*                                                                     * 00440000
*END OF SPECIFICATIONS************************************************* 01350000
*                                                             *         01370000
*    Turn tracking on:     SETCON TRACKING=ON                 *         01373000
*                                                             *         01374000
*    Turn tracking off:    SETCON TRACKING=OFF                *         01375000
*                                                              *
*    Display data:        DISPLAY OPDATA,TRACKING              *
*                                                              *
*    Clear tracked data by turning tracking off and then on.   *
*                                                              *
****************************************************************
         COPY  IATYGLOB
IATUX33  START
         TITLE 'JES3 General Equates'
         IATYEQU
         TITLE 'JES3 General Registers'
         IATYREG
         TITLE 'JES3 TVT'
         IATYTVT
         TITLE 'JES3 Job Control Table'
         IATYJCT
         TITLE 'JES3 Input Service Data Area'
         IATYISD
         TITLE 'Security Control Block'
         IATYSEC
         TITLE 'Tracking Facility Request Parameters'
         CNZTRPL ,
         TITLE 'Job Data Accounting Block'
         IATYJDA
         TITLE 'Local DSECT for info field'
U33CNZDE DSECT
DESC1    DC    CL3''               Eyecatcher = J3:
DESC2D   DC    CL3''               Input service day
DESC2H   DC    CL2''               Input service hour
DESC2M   DC    CL2''               Input service minute
DESC3    DC    CL2''               JECL card detected and space
DESC4    DC    CL8''               Job name
DESC5    DC    CL8''               User ID/POE
         TITLE 'JES3 Control Statement User Exit 33'
*--------------------------------------------------------------*
*        IATUX33 entry point                                   *
*--------------------------------------------------------------*
IATUX33  CSECT
         LR    R10,R15             Establish module base
         USING IATUX33,R10         Establish using for module
         USING IATISDT,R13         Input service work area
         IATYMOD BR=YES            Module entry point ID
         SPACE 1
*--------------------------------------------------------------*
*        Save statement address and zero the CNZ parm area.    *
*--------------------------------------------------------------*
         LR    R9,R1               Save JCL statement address
         LA    R8,UX33WA           Get work area address
         USING TRPL,R8             CNZTRKR parameter list
INFO     USING U33CNZDE,TRPL_Track_Info  CNZTRKR info field
         XC    TRPL(TRPL_LEN),TRPL  Clear the parm list
*--------------------------------------------------------------*
*        Is current statement //*MAIN or /*MAIN                *
*--------------------------------------------------------------*
         CLC   T33TMAN,0(R9)       Check for //*MAIN
         JE    UX33C005            If yes, handle it
         CLC   1+T33TMAN(L'T33TMAN-1),0(R9)  Check for /*MAIN
         JNE   UX33C220            If not, continue checking
UX33C005 DS    0H
         MVC   INFO.DESC3,T33TMAO  Indicate MAIN
         J     UX33T800            Go track stmt
*-------------------------------------------------------------*
*        Is current statement //*PROCESS                      *
*-------------------------------------------------------------*
UX33C220 DS    0H
         CLC   T33TPRC,0(R9)       Check for //*PROCESS
         JE    UX33C225            If yes, handle it
         CLC   1+T33TPRC(L'T33TPRC-1),0(R9)  Check for /*PROCESS
         JNE   UX33C240            If not, continue checking
UX33C225 DS    0H
         MVC   INFO.DESC3,T33TPRO  Indicate PROCESS
         J     UX33T800            Go track stmt
*-------------------------------------------------------------*
*        Is current statement //*ENDPROCESS                    *
*-------------------------------------------------------------*
UX33C240 DS    0H
         CLC   T33TEPR,0(R9)       Check for //*ENDPROCESS
         JE    UX33C245            If yes, handle it
         CLC   1+T33TEPR(L'T33TEPR-1),0(R9)  Check for /*ENDPROCESS
         JNE   UX33C260            If not, continue checking
UX33C245 DS    0H
         MVC   INFO.DESC3,T33TEPO  Indicate ENDPROCESS
         J     UX33T800            Go track stmt
*-------------------------------------------------------------*
*        Is current statement //*FORMAT                        *
*-------------------------------------------------------------*
UX33C260 DS    0H
         CLC   T33TFRM,0(R9)       Check for //*FORMAT
         JE    UX33C265            If yes, handle it
         CLC   1+T33TFRM(L'T33TFRM-1),0(R9)  Check for /*FORMAT
         JNE   UX33C280            If not, continue checking
UX33C265 DS    0H
         MVC   INFO.DESC3,T33TFRO  Indicate FORMAT
         J     UX33T800            Go track stmt
*-------------------------------------------------------------*
*        Is current statement //*NET                           *
*-------------------------------------------------------------*
UX33C280 DS    0H
         CLC   T33TNET,0(R9)       Check for //*NET
         JNE   UX33C300            If not, continue checking
         MVC   INFO.DESC3,T33TNEO  Indicate NET
         J     UX33T800            Go track stmt
*-------------------------------------------------------------*
*        Is current statement //*NETACCT                       *
*-------------------------------------------------------------*
UX33C300 DS    0H
         CLC   T33TNTA,0(R9)       Check for //*NETACCT
         JNE   UX33C320            If not, continue checking
         MVC   INFO.DESC3,T33TNTO  Indicate NETACCT
         J     UX33T800            Go track stmt
*-------------------------------------------------------------*
*        Is current statement //*ROUTE                         *
*-------------------------------------------------------------*
UX33C320 DS    0H
         CLC   T33TRTE,0(R9)       Check for //*ROUTE
         JNE   UX33C340            If not, continue checking
         MVC   INFO.DESC3,T33TRTO  Indicate ROUTE
         J     UX33T800            Go track stmt
*-------------------------------------------------------------*
*        Is current statement //*OPERATOR                      *
```

```
*--------------------------------------------------------------*        02380000
UX33C340 DS    0H                                                       02390000
         CLC   T33TOPR,0(R9)       Check for //*OPERATOR                02400000
         JE    UX33C345            If yes, handle it                    02410000
         CLC   1+T33TOPR(L'T33TOPR-1),0(R9)  Check for /*OPERATOR       02420000
         JNE   UX33T900            If not, done checking                02430000
UX33C345 DS    0H                                                       02440000
         MVC   INFO.DESC3,T33TOPO  Indicate OPERATOR                    02450000
         J     UX33T800            Go track stmt                        02460000
*--------------------------------------------------------------*        02470000
*        Track JES3 JECL statement usage                       *        02480000
*--------------------------------------------------------------*        02490000
UX33T800 DS    0H                                                       02500000
         MVC   TRPL_ACRO,=CL4'TRPL'  Set parm list eye catcher          02510000
         MVI   TRPL_VERSION,TRPL_K_JBB7727  Set parm list version       02520000
         ST    R10,TRPL_VIOLATORS_ADDR  Set event address               02530000
         MVC   INFO.DESC1,=CL3'J3:'  Indicate JES3 event                02540000
*--------------------------------------------------------------*        02550000
*        Include day and time the job went through input       *        02560000
*        service.  The format is DDDHHMM where:                *        02570000
*             DDD = day of the year                            *        02580000
*             HH  = hour of day                                *        02590000
*             MM  = minutes                                    *        02600000
*        This uses DESC4 as a work area.                       *        02610000
*--------------------------------------------------------------*        02620000
         L     R7,JDABADDR         Get JDAB                             02630000
         USING JDABSTRT,R7         JDABSTRT                             02640000
         UNPK  INFO.DESC2D,IRDATON  Set day                             02650000
         L     R15,IRTIMON         Get hundredths of seconds           +02660000
                                     since midnight                     02670000
         XR    R14,R14             Clear for divide                     02680000
         D     R14,=F'360000'      Get hours                            02690000
         CVD   R15,INFO.DESC4      Convert to packed dec                02700000
         OI    INFO.DESC4+7,X'0F'  Turn on sign bits                    02710000
         UNPK  INFO.DESC2H,INFO.DESC4+5(3)  Make printable              02720000
         LR    R15,R14             Move remainder to R15                02730000
         XR    R14,R14             Clear for divide                     02740000
         D     R14,=F'6000'        Get number of minutes                02750000
         CVD   R15,INFO.DESC4      Convert to packed dec                02760000
         OI    INFO.DESC4+7,X'0F'  Turn sign bits on                    02770000
         UNPK  INFO.DESC2M,INFO.DESC4+5(3)  Make printable              02780000
*--------------------------------------------------------------*        02790000
*        Add job name and user ID                              *        02800000
*--------------------------------------------------------------*        02810000
         MVC   INFO.DESC4,JDABJNAM  Indicate job name                   02820000
         MVC   INFO.DESC5,ISTUSID  Indicate user ID                     02830000
         DROP  R7                  JDABSTRT                             02840000
*--------------------------------------------------------------*        02850000
*        Track the event.                                      *        02860000
*--------------------------------------------------------------*        02870000
         CNZTRKR (R8)              Track the event                      02880000
*--------------------------------------------------------------*        02890000
*        Replace user ID with port of entry (POE).             *        02900000
*--------------------------------------------------------------*        02910000
         MVC   INFO.DESC5,ISDPOE   Set port of origin                   02920000
*--------------------------------------------------------------*        02930000
*        Track a second time, now with POE.                    *        02940000
*--------------------------------------------------------------*        02950000
         CNZTRKR (R8)              Track the event                      02960000
         DROP  R8                  CNZTRKR parms                        02970000
```

```
*-------------------------------------------------------------*
*        Setup for return                                     *
*-------------------------------------------------------------*
UX33T900 DS    0H
         LR    R1,R9               Restore JCL statement address
         LA    R15,0               Always use normal return
         ARETURN                   Return to caller
*-------------------------------------------------------------*
*        IATUX33 module work area                             *
*-------------------------------------------------------------*
UX33WA   DC    CL(TRPL_LEN)''
*-------------------------------------------------------------*
*        IATUX33 module constants                             *
*-------------------------------------------------------------*
T33TRTE  DC    CL9'//*ROUTE '
T33TRTO  DC    CL2'R'
T33TFRM  DC    CL10'//*FORMAT '
T33TFRO  DC    CL2'F'
T33TPRC  DC    CL11'//*PROCESS '
T33TPRO  DC    CL2'P'
T33TEPR  DC    CL14'//*ENDPROCESS '
T33TEPO  DC    CL2'E'
T33TMAN  DC    CL8'//*MAIN '
T33TMAO  DC    CL2'M'
T33TOPR  DC    CL12'//*OPERATOR '
T33TOPO  DC    CL2'O'
T33TNTA  DC    CL11'//*NETACCT '
T33TNTO  DC    CL2'A'
T33TNET  DC    CL7'//*NET '
T33TNEO  DC    CL2'N'
*-------------------------------------------------------------*
*        IATUX33 epilog                                       *
*-------------------------------------------------------------*
         IATXPTCH LT               Expand literals
APARNUM  DC    CL7'       '        APAR number
PTFNUM   DC    CL7'&J3REL  '       PTF number
         END   IATUX33
```

B

Comparison of JES3 and JES2 commands

This appendix contains a reference to the differences in the commands provided by JES3 and JES2. It assists a team who is considering the migration from JES3 to JES2. Changes to OPERCMDS profiles are referenced where applicable. It is not a complete list of all possible commands but it provides examples of each type of command.

List of commonly used JES3 and JES2 commands

Table B-1 contains a list of the JES commands that the operators are most likely to use frequently. The table shows the JES3 command and the JES2 equivalent. Also, if you use SAF to protect your operator commands, the table shows the OPERCMDS profile that protects the commands.

Table B-1 Commands and OPERCMDS profiles

Type of command	JES3	JES2	OPERCMDS profile

Shutdown	`*RETURN`
	`*DUMP`
		`$P JES2`
		`$P JES2,ABEND`
		`$P JES2,ABEND,FORCE`
			JES3.STOP.RETURN
			JES3.STOP.DUMP
			JES2.STOP.SYS

Printer devices	`*S PRT`	`$S PRTn`	JES3.START.DEV.dev JES2.START.DEV
Job queue	`*F Q H`	`$HA`	JES3.MODIFY.Q JES2.MODIFYHOLD.JOB

Initiator	`*F G main G inits`
	
	`*I G main G init`
		`$S INnn-nn`
		`$P INnn-nn`
		`$D INnn-nn`
			JES3.MODIFY.G
			JES2.START.INITIATOR
			JES2.STOP.INITIATOR

MVS	`*I D D=dddd`	`MVS D U,,,dddd,1`	JES3.DISPLAY.D MVS.DISPLAY.*
Device related	`*X CR,IN=RMT01RD1,K`	`$S R1.RD1`	JES3.CALL.dspname JES2.START.RMT
Remote console	`*I O`	`no equivalent`	JES3.DISPLAY.O
Remote printer	`*S RMT01PR1`	`$S R1.PR1`	JES3.START.name JES2.START.RMT

Spool related	`*I Q S`
	`*I J=`*`jobname`*
	`*I A`
		`$D Q`
		`$D'`*`jobname`*`'`
		`$D A`
			JES3.DISPLAY.Q
			JES3.DISPLAY.JOB
			JES3.DISPLAY.A
			JES2.DISPLAY.JOB

Restart	`*R J=nnnn`	`$E Jnnnn`	JES3.RESTART.name JES2.RESTART.BAT
Job modify	`*F J=nnnn,C`	`$C Jnnnn,P`	JES3.MODIFY.JOB JES2.CANCEL.BAT

Job output	`*I U J=nnnn`	`$L Jn`
		`$L Tn`
		`$L Sn`
			JES3.DISPLAY.U
			JES2.DISPLAY.BATOUT
			JES2.DISPLAY.TSUOUT
			JES2.DISPLAY.STCOUT

Reroute job	`*F U`
	`J=nnnn,ND=dest`
		`$R ALL,J=nnnn,R`	JES3.MODIFY.U
			JES2.ROUTE.JOBOUT

SPOOL		`$S SPOOL`	JES2.START.SPOOL
Unknown commands			JES3.UNKNOWN JES2.UNKNOWN

Related publications

The publications listed in this section are considered particularly suitable for a more detailed discussion of the topics covered in this book.

IBM Redbooks

The following IBM Redbooks publications provide additional information about the topic in this document. Note that some publications referenced in this list might be available in softcopy only.

- *ABCs of z/OS System Programming Volume 2*, SG24-6982
- *ABCs of z/OS System Programming Volume 13*, SG24-7717

You can search for, view, download, or order these documents and other Redbooks, Redpapers, Web Docs, draft and additional materials, at the following website:

ibm.com/redbooks

Other publications

These publications are also relevant as further information sources:

- *z/OS MVS JCL Reference*, SA22-7597
- *z/OS MVS JCL Users Guide*, SA22-7598
- *z/OS V1R13.0 MVS Planning: Operations*, SA23-1390
- For z/OS V2R1 JES2 and JES3 manuals, see the IBM Knowledge Center:
  http://pic.dhe.ibm.com/infocenter/zos/v2r1/index.jsp

Online resources

These websites are also relevant as further information sources:

- “JES2/JES3 JCL/JECL Differences” SHARE presentation

  https://share.confex.com/share/118/webprogram/Session10845.html
- “z/OS Basics: JES 201 - Differences Between JES2 and JES3” SHARE presentation

  http://www.share.org/p/do/sd/sid=4521&type=0

Help from IBM

IBM Support and downloads

ibm.com/support

IBM Global Services

ibm.com/services

IBM

Redbooks

JES3 to JES2 Migration Considerations

(0.5” spine)
0.475”<->0.873”
250 <-> 459 pages

JES3 to JES2 Migration Considerations

Differences between JES2 and JES3

Preparation and planning for a migration

Migration considerations

This book deals with the migration from JES3 to JES2. Part One describes this decision. Part Two describes the steps and considerations of this migration.

This IBM Redbooks publication provides information to help clients that have JES3 and would like to migrate to JES2. It provides a comprehensive list of the differences between the two job entry subsystems and provides information to help you determine the migration effort and actions.

The book is aimed at operations personnel, system programmers, and application developers.

SG24-8083-00 ISBN 0738440159