



ABCs of IBM z/OS System Programming Volume 2

Guillermo Cosimo

Lutz Kuehner



IBM Z



Redbooks



International Technical Support Organization

ABCs of IBM z/OS System Programming Volume 2

April 2018

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

SG24-6982-04 (April 2018)

This edition applies to IBM z/OS V2R3 (product number 5650-ZOS) and to all subsequent releases and modifications until otherwise indicated in new editions.

© Copyright International Business Machines Corporation 2003, 2018. All rights reserved.

Note to U.S. Government Users Restricted Rights -- Use, duplication or disclosure restricted by GSA ADP Schedule Contract with IBM Corp.

Contents

Notices	vii
Trademarks	viii
Preface	ix
Authors	x
Now you can become a published author, too!	x
Comments welcome	xi
Stay connected to IBM Redbooks	xi
Chapter 1. z/OS implementation and daily maintenance	1
1.1 Overview of z/OS implementation	2
1.2 Parmlib overview	3
1.2.1 Using system symbols in parmlib	4
1.2.2 Parmlib recommendations	6
1.3 z/OS system initialization overview	7
1.3.1 Types of IPL	8
1.3.2 The IPL process	9
1.4 Catalog overview	11
1.4.1 Catalog management	12
1.5 System administration	13
1.6 SMF records introduction	15
1.7 LOGREC data	16
1.8 SYSLOG data	17
Chapter 2. Subsystems and subsystem interface	19
2.1 Subsystem and SSI concepts	20
2.1.1 Subsystems	20
2.1.2 The subsystem interface	20
2.2 Subsystem initialization	21
2.2.1 Initializing a subsystem	21
2.2.2 Specifying an initialization routine	21
2.2.3 Using the START command	22
2.3 Subsystem requests	22
2.3.1 Requests under the hood: Control blocks and vector tables	22
Chapter 3. Job management	25
3.1 Understanding JCL	26
3.2 JCL control statements	26
3.2.1 Syntax rules	26
3.2.2 Required control statements	26
3.2.3 Optional JCL control statements	28
3.2.4 Optional JECL control statements	28
3.3 What the JES does	28
3.4 JES versions	30
3.4.1 JES2 architecture	30
3.4.2 JES3 architecture	31
3.5 JES2	32
3.5.1 JES2 functions	32
3.5.2 JES2 job flow	34

3.5.3 JES2 spool	36
3.5.4 JES2 configuration	55
3.5.5 JES2 exits	66
3.5.6 JES2 start	71
3.5.7 JES2 start options	74
3.5.8 JES2 stop	75
3.6 JES3	76
3.6.1 JES3 functions	79
3.6.2 JES3 job flow	80
3.6.3 JES3 checkpoint	82
3.6.4 JES3 spool	83
3.6.5 JES3 JCT	91
3.6.6 JES3 system configuration	94
3.6.7 JES3 exits	101
3.6.8 JES3 start	104
3.6.9 JES3 start types	106
3.6.10 JES3 Stop	108
Chapter 4. LPA, LNKLST, and authorized libraries	111
4.1 The creation of an executable program	112
4.1.1 Program load order	113
4.2 Link pack area	114
4.3 LPA subareas	116
4.4 Pageable link pack area (PLPA/extended PLPA)	117
4.5 LPA parmlib definitions	118
4.6 Coding a LPALSTxx parmlib member	119
4.7 Fixed link pack area	119
4.8 Coding the IEAFIXxx member	120
4.9 Specifying the IEAFIXxx member	121
4.10 Modified link pack area (MLPA)	122
4.11 Coding the IEALPAxx member	123
4.12 Specifying the IEALPAxx member	124
4.13 Dynamic LPA functions	124
4.14 The LNKLST	126
4.15 Dynamic LNKLST functions	128
4.16 Library lookaside	129
4.17 CSVLLAxx SYS1.PARMLIB member	130
4.18 Compressing LLA-managed libraries	132
4.19 Virtual lookaside facility	134
4.20 COFVLFx member	135
4.21 Authorized program facility (APF)	138
4.22 Authorizing libraries	139
4.23 Dynamic APF functions	141
Chapter 5. SMP/E for z/OS	145
5.1 Basic concepts	146
5.1.1 What is a SYSMOD?	146
5.1.2 SYSMOD package	146
5.1.3 Types of SYSMODs	147
5.1.4 SYSMOD prerequisites	147
5.2 Libraries, zones, and data set allocation	148
5.2.1 SMP/E libraries	148
5.2.2 Consolidated software inventory	149

5.2.3 SMP/e zones.....	149
5.3 SMP/E data set allocation.....	149
5.4 SYSMOD Processing and SMP/E Data Display	150
5.4.1 Displaying SMP/E data	151
5.4.2 SMP/E reports.....	152
Chapter 6. Language Environment	155
6.1 Language Environment.....	156
6.2 Assembler language and programs	156
6.3 High-level language and programs	159
6.4 Creating execution programs	159
6.5 IBM compiler products and Language Environment.....	160
6.6 Language Environment standards	161
6.7 Language Environment components	162
6.8 Language Environment common runtime environment	162
6.9 Language Environment runtime environment for AMODE 64.....	163
6.10 Object code and Language Environment	165
6.11 Callable services.....	165
6.12 Language Environment program management terms and terminology	166
6.13 Language Environment program management model	168
6.14 Language Environment program management full model	170
6.15 Language Environment condition handling model description and terminology	171
6.15.1 Language Environment condition handling steps	172
6.15.2 Language Environment condition handling signaling	172
6.16 Language Environment storage management model	173
6.17 Language Environment runtime options customization	174
6.17.1 Language Environment system configuration.....	174
Related publications	179
IBM Redbooks	179
Other publications	179
Help from IBM	180

Notices

This information was developed for products and services offered in the U.S.A.

IBM may not offer the products, services, or features discussed in this document in other countries. Consult your local IBM representative for information on the products and services currently available in your area. Any reference to an IBM product, program, or service is not intended to state or imply that only that IBM product, program, or service may be used. Any functionally equivalent product, program, or service that does not infringe any IBM intellectual property right may be used instead. However, it is the user's responsibility to evaluate and verify the operation of any non-IBM product, program, or service.

IBM may have patents or pending patent applications covering subject matter described in this document. The furnishing of this document does not give you any license to these patents. You can send license inquiries, in writing, to:

IBM Director of Licensing, IBM Corporation, North Castle Drive, Armonk, NY 10504-1785 U.S.A.

The following paragraph does not apply to the United Kingdom or any other country where such provisions are inconsistent with local law: INTERNATIONAL BUSINESS MACHINES CORPORATION PROVIDES THIS PUBLICATION "AS IS" WITHOUT WARRANTY OF ANY KIND, EITHER EXPRESS OR IMPLIED, INCLUDING, BUT NOT LIMITED TO, THE IMPLIED WARRANTIES OF NON-INFRINGEMENT, MERCHANTABILITY OR FITNESS FOR A PARTICULAR PURPOSE. Some states do not allow disclaimer of express or implied warranties in certain transactions, therefore, this statement may not apply to you.

This information could include technical inaccuracies or typographical errors. Changes are periodically made to the information herein; these changes will be incorporated in new editions of the publication. IBM may make improvements and/or changes in the product(s) and/or the program(s) described in this publication at any time without notice.

Any references in this information to non-IBM Web sites are provided for convenience only and do not in any manner serve as an endorsement of those Web sites. The materials at those Web sites are not part of the materials for this IBM product and use of those Web sites is at your own risk.

IBM may use or distribute any of the information you supply in any way it believes appropriate without incurring any obligation to you.

Information concerning non-IBM products was obtained from the suppliers of those products, their published announcements or other publicly available sources. IBM has not tested those products and cannot confirm the accuracy of performance, compatibility or any other claims related to non-IBM products. Questions on the capabilities of non-IBM products should be addressed to the suppliers of those products.

This information contains examples of data and reports used in daily business operations. To illustrate them as completely as possible, the examples include the names of individuals, companies, brands, and products. All of these names are fictitious and any similarity to the names and addresses used by an actual business enterprise is entirely coincidental.

COPYRIGHT LICENSE:

This information contains sample application programs in source language, which illustrate programming techniques on various operating platforms. You may copy, modify, and distribute these sample programs in any form without payment to IBM, for the purposes of developing, using, marketing or distributing application programs conforming to the application programming interface for the operating platform for which the sample programs are written. These examples have not been thoroughly tested under all conditions. IBM, therefore, cannot guarantee or imply reliability, serviceability, or function of these programs.

Trademarks

IBM, the IBM logo, and ibm.com are trademarks or registered trademarks of International Business Machines Corporation in the United States, other countries, or both. These and other IBM trademarked terms are marked on their first occurrence in this information with the appropriate symbol (® or ™), indicating US registered or common law trademarks owned by IBM at the time this information was published. Such trademarks may also be registered or common law trademarks in other countries. A current list of IBM trademarks is available on the Web at <http://www.ibm.com/legal/copytrade.shtml>

The following terms are trademarks of the International Business Machines Corporation in the United States, other countries, or both:

C/370™	Language Environment®	RMF™
CICS®	MVS™	VisualAge®
DB2®	OS/390®	VTAM®
GDPS®	Parallel Sysplex®	z/Architecture®
Geographically Dispersed Parallel Sysplex™	Print Services Facility™	z/OS®
IBM®	RACF®	z/VM®
IMS™	Redbooks®	
	Redbooks (logo) 	

The following terms are trademarks of other companies:

Java, and all Java-based trademarks and logos are trademarks or registered trademarks of Oracle and/or its affiliates.

UNIX is a registered trademark of The Open Group in the United States and other countries.

Other company, product, or service names may be trademarks or service marks of others.

Preface

The ABCs of IBM® z/OS® System Programming is a 13-volume collection that provides an introduction to the z/OS operating system and the hardware architecture. Whether you are a beginner or an experienced system programmer, the ABCs collection provides the information that you need to start your research into z/OS and related subjects. If you want to become more familiar with z/OS in your current environment or if you are evaluating platforms to consolidate your e-business applications, the ABCs collection can serve as a powerful technical tool.

This volume describes the basic system programming activities related to implementing and maintaining the z/OS installation and provides details about the modules that are used to manage jobs and data. It covers the following topics:

- ▶ Overview of the parmlib definitions and the IPL process. The parameters and system data sets necessary to IPL and run a z/OS operating system are described, along with the main daily tasks for maximizing performance of the z/OS system.
- ▶ Basic concepts related to subsystems and subsystem interface and how to use the subsystem services that are provided by IBM subsystems.
- ▶ Job management in the z/OS system using the JES2 and JES3 job entry subsystems. It provides a detailed discussion about how JES2 and JES3 are used to receive jobs into the operating system, schedule them for processing by z/OS, and control their output processing.
- ▶ The link pack area (LPA), LNKLST, authorized libraries, and the role of VLF and LLA components.
- ▶ An overview of SMP/E for z/OS.
- ▶ An overview of IBM Language Environment® architecture and descriptions of Language Environment's full program model, callable services, storage management model, and debug information.

Other volumes in this series include the following content:

- ▶ Volume 1: Introduction to z/OS and storage concepts, TSO/E, ISPF, JCL, SDSF, and z/OS delivery and installation
- ▶ Volume 3: Introduction to DFSMS, data set basics, storage management, hardware and software, catalogs, and DFSMSStvs
- ▶ Volume 4: Communication Server, TCP/IP, and IBM VTAM®
- ▶ Volume 5: Base and IBM Parallel Sysplex®, System Logger, Resource Recovery Services (RRS), global resource serialization (GRS), z/OS system operations, automatic restart management (ARM), IBM Geographically Dispersed Parallel Sysplex™ (IBM GDPS®)
- ▶ Volume 6: Introduction to security, IBM RACF®, Digital certificates and PKI, Kerberos, cryptography and z990 integrated cryptography, zSeries firewall technologies, LDAP, and Enterprise Identity Mapping (EIM)
- ▶ Volume 7: Printing in a z/OS environment, Infoprint Server, and Infoprint Central
- ▶ Volume 8: An introduction to z/OS problem diagnosis
- ▶ Volume 9: z/OS UNIX System Services
- ▶ Volume 10: Introduction to IBM z/Architecture®, the IBM Z platform and IBM Z connectivity, LPAR concepts, HCD, and the DS Storage Solution
- ▶ Volume 11: Capacity planning, performance management, WLM, IBM RMF™, and SMF

- ▶ Volume 12: WLM
- ▶ Volume 13: JES3, JES3 SDSF

Authors

This IBM Redbooks® publication was produced by a team of specialists from around the world working at the International Technical Support Organization, Poughkeepsie Center.

Guillermo Cosimo has almost 20 years in IT. He is an Ingénieur Systèmes who specializes in Mainframe technologies at Synchrone for BPCE-IT in France, where he moved in 2015. Before moving, Guillermo worked in Argentina as a z/OS System Programmer and previously as an IT Specialist for distributed environments. This double experience serves him well as an interface between both worlds. He has given support at different levels for z/OS, JES2, USS, file systems (HFS, ZFS, NFS), SMP/E, z/OS Communications Server (TCP/IP and SNA/VTAM), and WLM and Performance and Security (RACF, ACF2 and TSS).

Lutz Kuehner is a Senior System Engineer working for Credit Suisse AG in Switzerland. Previously, he worked for 10 Years as a Client Technical Specialist with the System Technologie Group sales organization at IBM Germany. Lutz has 31 years of experience in the field of Mainframe Technology, and his areas of expertise are z/OS topics. Lutz has written extensively about z/OS-related topics and contributed to the ITSO team in teaching classes around the world and in writing IBM Redbooks publications.

Thanks also to the following contributors to this edition:

- ▶ Lydia Parziale, Robert Haimowitz, William G. White
International Technical Support Organization, Poughkeepsie Center
- ▶ Manfred Schemmelmann, z/OS System Engineer
Helsana Insurance, Switzerland
- ▶ Rafael Lima
DXC Technology, Brazil

Thanks to the authors of the previous edition of this book.

- ▶ Paul Rogers, Alvaro Salla, Miriam Gelinski, Sergio Munchen, Joao Natalino Oliveira, Monica Mataruco, Julio Grecco Neto, and Antonio Orsi

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:

<http://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>



z/OS implementation and daily maintenance

A system programmer needs to understand the different aspects of a z/OS implementation. This chapter covers the basics from installation and customization to daily activities.

Installation means to create data sets in your DASD volumes and receive the z/OS data and programs. *Customization* refers to the adjustment of parameters in order to suit your business needs and technical requirements. It is also a portion of the skill set of the z/OS systems programmer to update, upgrade, and maintain the mainframe as part of the IT organization and in cohesion with the business.

This introductory chapter provides a basic approach to system implementation as well as the initial program load (IPL) process from the perspective of the system programmer. It describes the main parameters and system data sets and members that are necessary to initialize and run a z/OS operating system. It also discusses the periodic tasks that a system programmer performs to maximize the advantages that a well-implemented operating system can offer to your IT infrastructure.

It includes the following topics:

- ▶ Overview of z/OS implementation
- ▶ Parmlib overview
- ▶ z/OS system initialization overview
- ▶ Catalog overview
- ▶ System administration
- ▶ SMF records introduction
- ▶ LOGREC data
- ▶ SYSLOG data

1.1 Overview of z/OS implementation

To successfully build a z/OS system, the system programmer must understand the current needs of the installation and must analyze and plan for future requirements. Although this will sometimes mean a compromise to offer the most adaptive system, it also means finding ways to build a flexible system that is easy to install, easy to migrate, easy to extend, and most important, easy to manage. A well-planned structure leads to being able to anticipate changes and reduce unnecessary downtime as well as guaranteeing system robustness and availability.

The system programmer must be concerned with, and align with, the reliability, availability, and serviceability (RAS) philosophy that describes the robustness of the IBM Z.

A phased approach often proves most feasible and can begin to control the installation and migration workload in the least time. This approach provides benefits, starting with the next installation and migration cycle, while controlling the work involved in implementation. As systems evolve, system programmers must assure that the installation and migrations plans also evolve.

Figure 1-1 shows at a high level, the basic considerations that must be made in planning the system.

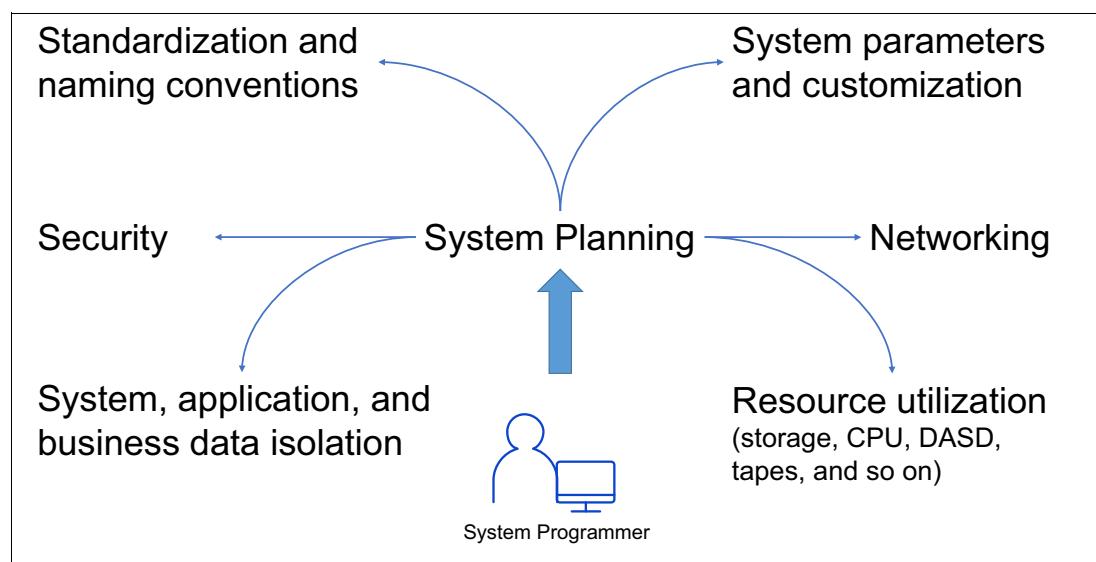


Figure 1-1 Basic system planning for z/OS implementation

If you apply a drill-down technique in each of these areas, you can gain granularity and find more detail about each of the basic functions. Some of the aspects that must be considered in adopting a structured approach to installation are described in the following sections of this chapter:

- ▶ Parmlib overview
- ▶ z/OS system initialization overview
- ▶ Catalog overview
- ▶ System administration
- ▶ SMF records introduction
- ▶ LOGREC data
- ▶ SYSLOG data

1.2 Parmlib overview

SYS1.PARMLIB is the required and default partitioned data set (PDS) that contains IBM-supplied members, with system and application parameters. The system programmer is in charge of adding new or modifying existing members in the parmlib. Because SYS1.PARMLIB can be concatenated with up to 16 other concatenated PDSs, the whole set of libraries, including the SYS1.PARMLIB (as the seventeenth data set) are referred to in this book as the *parmlib*.

The members of the parmlib (such as those shown in Figure 1-2) are read by the system at IPL, and later by many components and subsystems that use parmlib members, to implement dynamic changes in the system through the following operator commands:

- ▶ Workload Manager, Health Checker
- ▶ Generalized Trace Facility
- ▶ System Management Facilities (SMF)

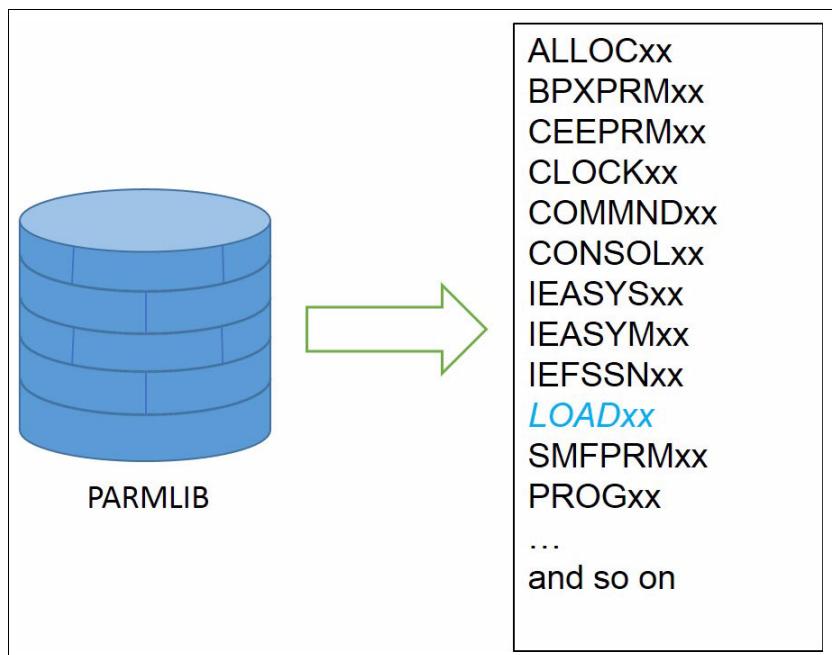


Figure 1-2 Parmlib members

The parmlib provides system parameters in a pre-specified form in a single concatenated data set to help minimize delays and unnecessary interactions that might lead to an unintended outage. The parmlib data sets can be blocked and can have multiple extents but must reside on a single volume, such as any PDS data set.

Hint: Use the **DISPLAY PARMLIB** command to obtain the detail of the current parmlib concatenation.

Being one of the most important data sets in the z/OS operating system and having a great impact on the system, the parmlib data sets must be correctly defined in terms of size and must be protected. Keeping track of modifications along with backups is strongly recommended.

You can find a list of all members of the SYS1.PARMLIB data set online in IBM Knowledge Center:

https://www.ibm.com/support/knowledgecenter/SSLTBW_2.3.0/com.ibm.zos.v2r3.ieae200/ieae20036.htm

The following important members are included with the parmlib:

LOADxx	This member, which might be found in another data set, contains information about the I/O definition file (IODF) data set. It describes the z/OS I/O configuration, which the master catalog uses and which the IEASYSxx and IEASYMxx members of parmlib need to use.
IEASYSxx	This member allows the specification of system parameters that will be used during the IPL process. Multiple system parameter lists are valid, and all the parameters in this member are used to customize the z/OS configuration and its subsystems.
IEASYMxx	This member specifies for one or more z/OS system in a multi-system environment the static IEASYSxx parmlib members that a specific z/OS system is to use.

1.2.1 Using system symbols in parmlib

Flexibility is as important in helping the system programmer in daily tasks as is the evolution of the system. Using system symbols that work similarly to system variables is a good way to increase that flexibility.

System symbols, among other things, allow you to share parmlibs definitions while retaining unique values in those definitions. In this way, the parmlib is used by different systems that can replace the content of the symbol while maintaining the definition. Thus, system symbols are not only focused in parmlib but can be used also in the job control language (JCL), both started tasks and batch jobs, and the Interactive Problem Control System (IPCS), which is the component that formats memory dumps.

The following terms describe the elements of system symbols:

<i>Symbol name</i>	The name that is assigned to a symbol. It begins with an ampersand (&) and optionally ends with a period (.).
<i>Substitution text</i>	The character string that the system substitutes for a symbol each time it appears. Substitution text can identify characteristics of resources, such as the system on which a resource is located or the date and time of processing.

Types of system symbols

The following terms describe the types of system symbols:

<i>Dynamic</i>	A system symbol whose substitution text can change at any point in an IPL. Dynamic system symbols represent values that can change often, such as dates and times. A set of dynamic system symbols is defined automatically by the z/OS system; your installation cannot provide additional dynamic system symbols.
<i>Static</i>	A symbol whose substitution text is defined at system initialization and remains fixed for the life of an IPL being unique in one z/OS system. One exception, &SYSPLEX, has a substitution text that can change at one point in an IPL. Static system symbols are used to represent fixed values, such as system names and sysplex names. You can have both <i>system-defined</i> and <i>installation-defined</i> static system symbols.

When you define additional system symbols in the IEASYMxx parmlib member, ensure that you do *not* specify the names that are reserved for system use. If you try to define a system symbol that is reserved for system use, the system might generate unpredictable results when performing symbolic substitution.

Note: Use the **DISPLAY SYMBOLS** operator command to display the static texts that are in effect for a system.

Examples using system symbols

The following examples assume that you have a sysplex with of two systems, named PRD1 and PRD2, for the &SYSNAME variable, and that &SYSCLOSE is defined in a way that it consists of the last two characters of the &SYSNAME, having as a result D1 and D2.

Data sets

Assume that each system in your sysplex requires one unique data set for SMF recording. If all systems in the sysplex use the same SMFPRMxx parmlib member, you can specify the following naming pattern to create different SMF recording data sets on each system:

```
SYS1.&SYSNAME..SMF.DATA
```

When you complete the IPL process for each system in the sysplex, the &SYSNAME system symbol resolves to the substitution text that is defined on the current system and produces the following data sets:

- ▶ SYS1.PRD1.SMF.DATA on system PRD1
- ▶ SYS2.PRD2.SMF.DATA on system PRD2

Parmlib members

You can apply the same logic to system images that require different parmlib members. For example, assume that system images PRD1 and PRD2 require different CLOCKxx parmlib members. If both systems share the same IEASYSxx parmlib member, you can specify &SYSCLOSE in the value on the CLOCK parameter as follows:

```
CLOCK=&SYSCLOSE;
```

When each system in the sysplex initializes with the same IEASYSxx member, &SYSCLOSE resolves to the substitution text that is defined on each system. Accepting the default value for &SYSCLOSE produces the following results:

- ▶ CLOCK=D1 (specifies member CLOCKD1 on system PRD1)
- ▶ CLOCK=D2 (specifies member CLOCKD2 on system PRD2)

JCL for started tasks and batch jobs

When you start a task in the COMMNDxx parmlib member, you can also specify system symbols. This member has console commands that execute automatically after the completion of the IPL and master scheduler initialization. Assume that PRD1 and PRD2 system images both need to start SDSF with different configurations. If both system images share the same COMMNDxx parmlib member, specify the &SDSFPRM system symbol on a **START** command in COMMNDxx to start SDSF with different parameters on each, as follows:

```
S SDSF,M=&SDSFPRM
```

When each system in the sysplex initializes with the same COMMNDxx member, &SDSFPRM resolves to the substitution text that is defined on each system. If &SDSFPRM is defined to P1 on PRD1 and P2 in PRD2, the systems start SDSF with the following commands:

```
S SDSF,M=P1 (ISFPRMP1 member on PRD1)  
S SDSF,M=P2 (ISFPRMP2 member on PRD2)
```

In the case of a batch job, use symbols while allocating a new file, as shown in Example 1-1.

Example 1-1 Using symbols while allocating a new file

```
//STEP1 EXEC PGM=IEFBR14
//FILE001 DD DSN=MYUSER.OUTPUT.N&SYSNAME,
//           DISP=(NEW,CATLG,DELETE),
//           SPACE=(TRK,(1),RLSE),
//           DCB=(RECFM=FB,LRECL=80)
```

After the symbol substitution, the data set name translates to MYUSER.OUTPUT.NPRD1 or MYUSER.OUTPUT.NPRD2, depending on where the job is submitted.

Verifying system symbols

The parmlib symbolic preprocessor tool allows you to test symbol definitions to validate any new implementation or change. This tool shows how a parmlib member displays after the system performs symbolic substitution.

You can find the IEASYMCK sample in the SYS1.SAMPLIB. Add this verification to your implementation and change procedures.

1.2.2 Parmlib recommendations

A parmlib concatenation allows you to have more flexibility in managing parmlib members and changes to parmlib members. To control parmlib and to ensure that it is manageable, consider the following recommendations:

- ▶ Up to 16 data sets are available for the parmlib concatenation. Use these data sets to simplify and isolate parameters when required.
- ▶ Keep unmodified IBM-supplied members in a different data set of the concatenation.
- ▶ Separate your parmlib environments (test, development, production, and so on).
- ▶ Delete unsupported parameters and members. Because most components treat unsupported parameters from previous releases as syntax errors, remove the old parameters or build parmlib from scratch. This action can minimize the need for operator responses during an IPL process. Then, save space by removing unsupported members.
- ▶ Use the parmlib members for the appropriate functions. For example, use COMMNDxx to contain commands that are useful at system initialization. Use IEACMDxx for IBM-supplied commands.
- ▶ Update parmlib with new or replacement members as you increase experience with new releases.
- ▶ Keep track of which parameters are included in particular parmlib members. This bookkeeping is necessary for the following reasons:
 - The system doesn't keep track of parmlib members and their parameters.
 - The default general IEASYS00 parameter list is always read by the IPL process and the master scheduler initialization.

The parameters in IEASYS00 can be overridden by the same parameters when they are specified in alternate general lists, such as IEASYS01 or IEASYS02. Then, certain parameters (such as FIX, APF, and MLPA) direct the system to particular specialized members (in this example, IEAFIXxx and IEALPAxx).

Keep records of which parameters, which values are in particular members, and which general members point to which particular specialized members (COMMNDxx, IEALPAxx,

and so forth). A grid or matrix for such bookkeeping can be helpful. Also, you can (and should) document the default values in each of the members. This documentation can save time-consuming searches when you want to revert to defaults.

- ▶ Allocate sufficient space for parmlib. One way to estimate space is to count the number of 80-character records in all members that are to be included in one parmlib data set and factor in the block size of the data set. Then add a suitable growth factor (for example, 100% to 300%) to allow for future growth of alternate members. To recapture space occupied by deleted members, use the compress function of IEBCOPY. However, if the data set runs out of space, you can copy the members to a larger data set, create a new LOADxx member in which you replace the PARMLIB statement for the full data set with a PARMLIB statement for the new larger data set, and then issue a **SETLOAD** command to switch to the concatenation with the new data set.
- ▶ Decide which volume or volumes and device or devices you want to hold the parmlib concatenation. The data set must be cataloged, unless it resides on SYSRES or its volume serial number is included on the PARMLIB statement in LOADxx. The data set can be placed on a slow or moderate speed device.
- ▶ Use a security product (such as RACF) to protect the data sets. The purpose is to preserve system integrity by protecting the appendage member (IEAAPP00) and the authorized program facility members (IEAAPFx and PROGxx) from user tampering.
- ▶ SYS1.PARMLIB is not be accessed by a running z/OS system. Be careful of starting or restarting started tasks, which access their control parameters from SYS1.PARMLIB.

1.3 z/OS system initialization overview

The IPL program means that an executable program is read by the hardware from a specific device into the central storage for the execution of that initial program. The IPL operation is handled with the Hardware Management Console (HMC), where the following entities are required for the procedure:

- ▶ The IPL device number (also called the *SYSRES volume*)
- ▶ The IODF device number
- ▶ LOADPARM options

This HMC console is connected to and managed by the processor controller, also commonly known as a *support element* (basically, a notebook attached to the machine), and it can be accessed remotely.

The IPL program loads the z/OS kernel and the nucleus initialization program (NIP) from the SYS1.NUCLEUS data set located in the IPL device. After that, the CPU control is passed to NIP. During the NIP processing, it prompts the operator to provide system parameters that control the operation of IBM MVST™. The system also issues informational messages that inform the operator about the stages of the initialization process.

The LOADxx member (in SYSn.IPLPARM or SYS1.PARMLIB) allows the installation to control the initialization process. For example, in LOADxx you specify the suffixes for IEASYSxx or IEASYMxx members that the system is to use; the system does not prompt the operator for system parameters that are specified in those members. Instead, it uses the values in those members.

Tip: A preferred practice is to develop well thought-out naming conventions regarding PARMLIB member names, location of system data sets, location for plex-shared data sets, and naming of suffixes in the parmlib.

1.3.1 Types of IPL

Before explaining the detail of the different types of IPL, keep in mind the following concepts:

- ▶ *Pageable link pack area* (PLPA) is a virtual storage area where IPL pre-loads key programs (load modules) that have a high probability of being executed when the system reaches the steady state. Go to 4.2, “Link pack area” on page 114 for more details about virtual storage areas.
- ▶ *Virtual I/O* is a z/OS technique to speed up, through virtualization, the processing of temporary files, keeping them in local page data sets.

The objective of this section is to condense the IPL types and process to give you an overview of the IPL. Depending on the level of customization, an IPL can bring many different configurations; however, only the following basic types of IPL are available:

- ▶ Cold start

Any IPL that loads or reloads the PLPA to the PLPA page data set and does not preserve virtual I/O (VIO) data sets. The first IPL after system installation is always a cold start because the PLPA must be initially loaded. At the first IPL after system installation, the PLPA is loaded automatically from the LPALST concatenation. The page data sets for this IPL are those specified in IEASYSxx, plus any additional ones specified by the operator. Subsequent IPLs only need to be cold starts if the PLPA must be reloaded either to alter its contents (a new program, for example) or to restore it (the PLPA page data set) when it has been destroyed.

The PLPA needs to be reloaded:

- At the first IPL after system initialization, when the system loads it automatically.
- After modifying one or more programs (load modules) in the LPALST concatenation.
- After the PLPA page data set has been damaged and, therefore, is unusable, so its contents must be restored. The PLPA can be reloaded by responding create link pack area (CLPA) to the SPECIFY SYSTEM PARAMETERS prompt.

- ▶ Quick start

Any IPL that does not reload the PLPA and does not preserve VIO data sets. The system re-creates page and segment tables to match the in-use PLPA in PLPA page data set. The PLPA from the previous IPL can be used without reloading it from the LPALST concatenation. A quick start can be initiated by replying clear VIO (CVIO) to the SPECIFY SYSTEM PARAMETERS prompt.

- ▶ Warm start

Any IPL that does not reload the PLPA and preserves journaled VIO data sets. The operator does not enter CLPA or CVIO at the SPECIFY SYSTEM PARAMETERS prompt. Any definitions of existing page data sets as non-VIO local page data sets are preserved.

Due to the high speed of DASD devices, a cold start does not provide much difference (in time) relevant to other types of starts.

Terminology note: The expressions *cold start* and *warm start* are also used for the JES (JES2 or JES3) initialization, referring to whether or not the JES spool data set contents are going to be kept.

1.3.2 The IPL process

To restart a system using the IPL process, a specific volume has to contain loadable code (IPL text); otherwise, any attempt to use the IPL process from a disk without the IPL text results in a wait state error condition. A disk that does include IPL text is generally referred to as an *IPLable* disk and, more specifically, as the *SYSRES volume*. These disks contain a bootstrap module at cylinder 0, track 0, and block 0. At the IPL, this bootstrap is loaded into storage at real address zero and then control is passed to it. The bootstrap then reads the IPL IEAIPLO0 control program (also known as the IPL text) and passes control to it. This process in turn starts the more complex task of loading the operating system and executing it.

Figure 1-3 illustrates this process flow.

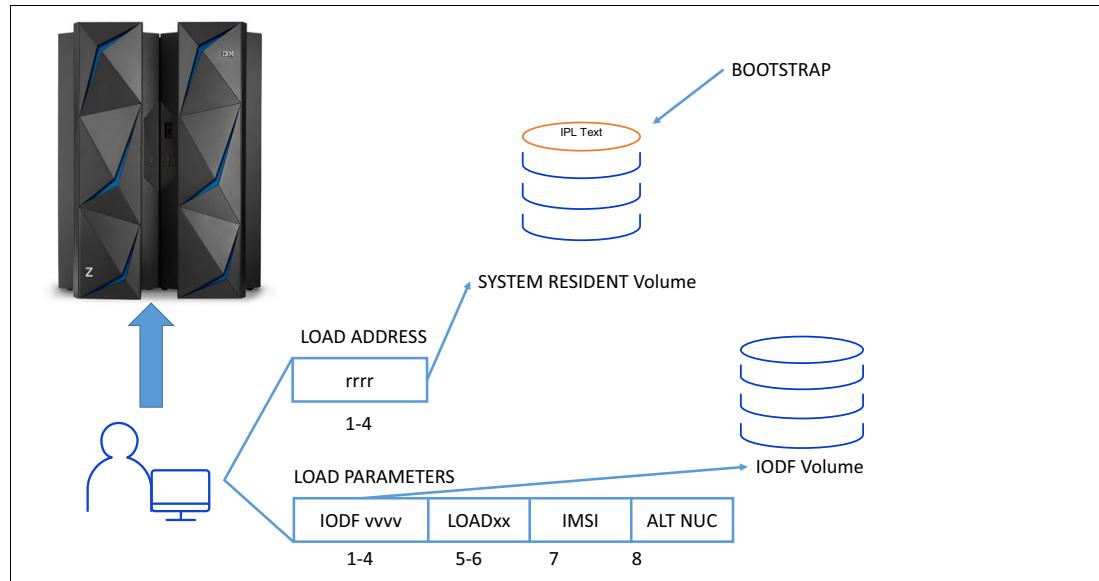


Figure 1-3 IPL load address and load parameters process

The IPL control program (IEAIPLO0)

The IEAIPLO0 control program prepares an environment that is suitable for starting the programs and modules that make up the z/OS system. First, it clears main storage to zeros before defining storage areas for the master scheduler address space. It then locates the IEANUC0x member in SYS1.NUCLEUS data set on the SYSRES volume and loads a series of programs from it known as *IPL resource initialization modules* (IRIMs). These IRIMs then start to construct the normal operating system environment of control blocks and subsystems that make up a z/OS system. The IRIMs complete the following more significant tasks:

- ▶ Read the LOADPARM information entered on the HMC at the time the IPL command was executed.
- ▶ Search the volume specified in the LOADPARM as containing the IODF data set for the LOADxx member (see Figure 1-4 on page 10); the value of xx is also taken from the LOADPARM. The IRIM first attempts to locate LOADxx in SYS0.IPLPARM. If this attempt is unsuccessful, it looks for SYS1.IPLPARM, up to and including SYS9.IPLPARM. If, at this point, the parameter still is not located, the search continues in SYS1.PARMLIB.

When a LOADxx member is successfully located, it is accessed looking for the following information:

- IEANUC0x suffix (unless overridden in LOADPARM)
- The master catalog name
- The suffix of the IEASYSxx member

- ▶ Load the z/OS nucleus.
 - ▶ Initialize virtual storage in the master scheduler address space for the following areas:
 - The system queue area (SQA)
 - The extended system queue area (ESQA)
 - The local system queue area (LSQA)
 - The prefixed save area (PSA)
- Also, the NIP process loads modules in some of these areas if required. At the end of the IPL sequence, the PSA replaces IEAIPL00 code at real storage location zero.
- ▶ Initialize real storage management, including the master scheduler segment table entries for common storage areas and the page frame table.

The last of the IRIMs then loads the first part of the NIP, which invokes the RIM, one of the earliest of which starts communications with the z/OS console. After the system component address spaces are running, such as the Master Scheduler, the IRIM continues with the subsystem's initialization, for example the primary subsystem JES2.

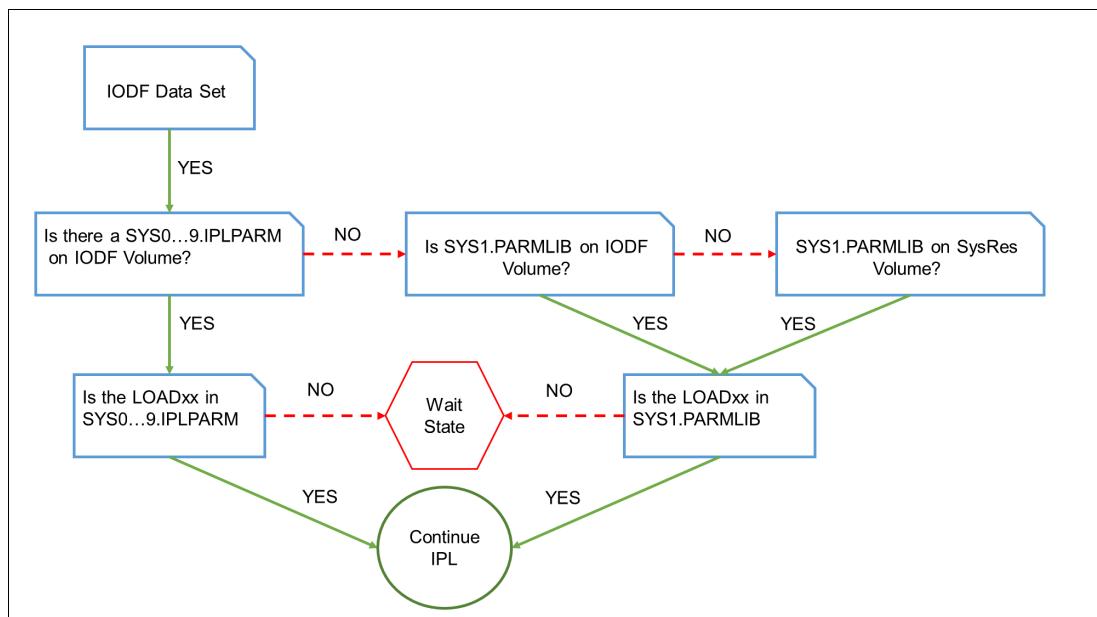


Figure 1-4 LOADxx member search

Important: To complete the IPL process for z/OS V2R3, a zEC12-capable machine is required. Upgrades in the z/Architecture are not available in previous hardware versions.

The load parameter (LOADPARM)

The most basic level of control is through the LOADPARM load parameter. This parameter is an eight-character value that is made up of four fields that the IPL program refers to during the IPL process, causing it to take the specified actions. The LOADPARM is made up of the following components:

- ▶ IODF vvvv

The IODF device number, where vvvv represents a device address. This device is also the device on which the search for the LOADxx member of SYSn.IPLPARM or SYS1.PARMLIB begins. The first four characters of the LOADPARM parameter specifies the hexadecimal device number for the device that contains the IODF Virtual Storage

Access Method (VSAM) data set. If you do not specify the device number, the hardware uses the device number of the system residence (SYSRES) volume.

► LOADxx

The xx suffix of the LOADxx. The LOADxx member contains information about the name of the IODF data set, which the master catalog uses and which the IEASYSxx and IEASYMxx members of parmlib use for concatenation. The default for the LOADxx suffix is zeroes. The IPL program reads the LOADxx and NUCLSTxx members from SYSn.IPLPARM or SYS1.PARMLIB on the IODF device that is specified in the LOADPARM (or the SYSRES volume, if no volume is specified). Later, the NIP opens the master catalog to locate the volumes of all data sets to be concatenated in the parmlib.

► IMSI

The prompt feature character specifies the prompting and message suppression characteristics that the system is to use at the IPL process. This character is commonly known as an *initialization message suppression indicator* (IMSI). Some IMSI characters suppress informational messages from the system console, which can speed up the initialization process and reduce message traffic to the console. It can also lead to missed critical messages, so always review the hardcopy log after initialization is complete. One of the reasons to select prompting suppression is to enable unattended operation.

► Alt NuZ

The last character specifies the alternate nucleus identifier (0–9). If you do not specify an alternate nucleus identifier, the system loads the standard (or primary) nucleus (IEANUC01), unless the NUCLEUS statement is specified in the LOADxx member.

1.4 Catalog overview

A *catalog* is a VSAM data set that holds information about other data sets, such as their name, location, the device type, and other attributes. A catalog allows users to find a data set by knowing only the data set name and without having to memorize location information for the data set.

Figure 1-5 shows an illustration of the hierarchy of the master and user catalogs.

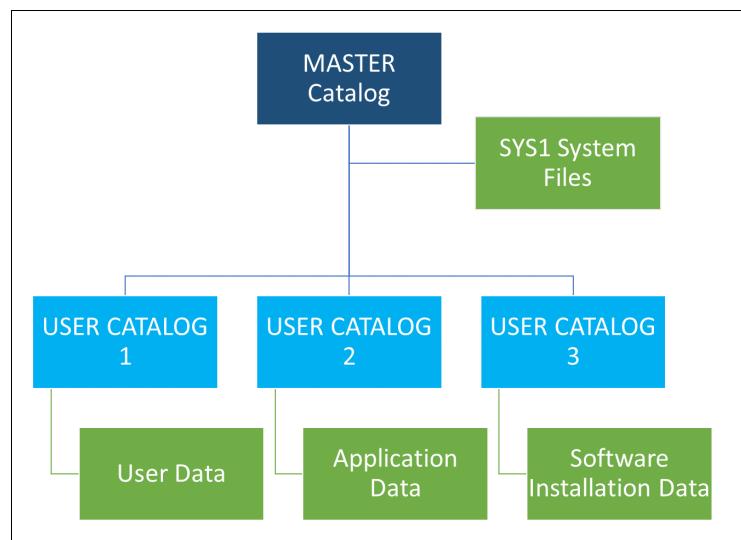


Figure 1-5 Master and user catalogs hierarchy

Cataloging data sets also simplifies backup and recovery procedures. Catalogs are the central information point for VSAM data sets; all VSAM data sets must be cataloged. In addition, all SMS-managed data sets (data sets whose performance and availability are managed by the SMS system component) must also be cataloged. As expected, catalogs also adds flexibility to the system, so it has to be maintained in order to keep the performance.

Functionally, the catalogs are divided into master catalog (*mastercat*) and user catalogs (*usercat*). The master catalog has z/OS system data set information and pointers to user catalogs that is based on the data set name high-level qualifier of the data set being located. User catalogs contain non-z/OS data set information. For each high-level qualifier that is associated with a specific user catalog, the installation defines one alias in the master catalog that points to the user catalog, as illustrated in Figure 1-5. There is no physical difference between a master catalog and user catalogs.

Recommendation: Define a backup for the master catalog. The system programmer specifies this alternate master catalog during system start. The backup master catalog should be in a separate volume in case the master catalog becomes unavailable.

Using indirect catalog entries

When sharing the master catalog, it might be desirable for an alias to reference a different data set or user catalog, depending on the z/OS system it is accessed from. To implement this type of facility, you can use the following functions:

- ▶ *Indirect* volume serial support allows the system to dynamically resolve volume and device type information for non-VSAM, non-SMS managed data sets that reside on the SYSRES volume when accessed through the catalog. This function allows you to change the volume serial number (VOLSER) or the device type of the SYSRES without also having to recatalog the non-VSAM data sets on that volume.
- ▶ *Extended indirect* volume serial support allows catalog entries to be resolved using system symbols that are defined in an IEASYMxx member of parmlib, so that indirect references can be made to one or more logical extensions to the system residence volume. Similar to indirect catalog support, this support lets you change the volume serial numbers or device types of system software target volumes without having to recatalog their non-VSAM data sets. You can define an alias pointing to a user catalog or an alias for a non-VSAM or VSAM data set.

Therefore, you can have multiple levels of z/OS data sets residing on multiple sets of volumes with different names and device types, and you can use them with the same master catalog.

Note: When the VOLSER information displays as a series of asterisks (*****), it refers to the current system resident of this z/OS system.

1.4.1 Catalog management

Ensuring that catalogs are effectively managed is a crucial aspect of the system and storage management. Typically, a system programmer uses the following common catalog management tasks:

- ▶ Defining and maintaining the master catalog. A general recommendation is that every z/OS system has its own master catalog connected to shared (between other z/OS systems) user catalogs. This task improves continuous availability by preventing the master catalog from being a single point of failure.

- ▶ Defining and maintaining the aliases and user catalogs, which distributes the load of the user catalogs.
- ▶ Protecting the catalogs with a security product (such as RACF).
- ▶ Cleaning up catalogs.
- ▶ Backing up catalogs and tuning performance.

Information and data isolation

When installing a z/OS system, one of the recommended tasks is to plan and separate user and application data from the z/OS software and operating system. With this procedure you eliminate many tasks and rework that would otherwise need to be performed each time you upgrade or replace z/OS software or programs. One effective way to achieve this process is to use dedicated pools of direct access storage device (DASD). You can use the following types DASD pools:

- ▶ Customization data, including most system control files, such as parmlib
- ▶ IBM non-z/OS base system products, for example IBM CICS® and IBM DB2®
- ▶ Non-IBM software
- ▶ User-defined exits
- ▶ User data
- ▶ Installation libraries
- ▶ File systems (local and network shared)
- ▶ Shared data across systems

1.5 System administration

This section introduces some of the tasks a system programmer handles (as illustrated in Figure 1-6). Depending on each structure, some of these tasks might be performed by different teams who are dedicated purely to that area. Clear examples of such tasks are security-related elements or storage administration (DASD and tape).

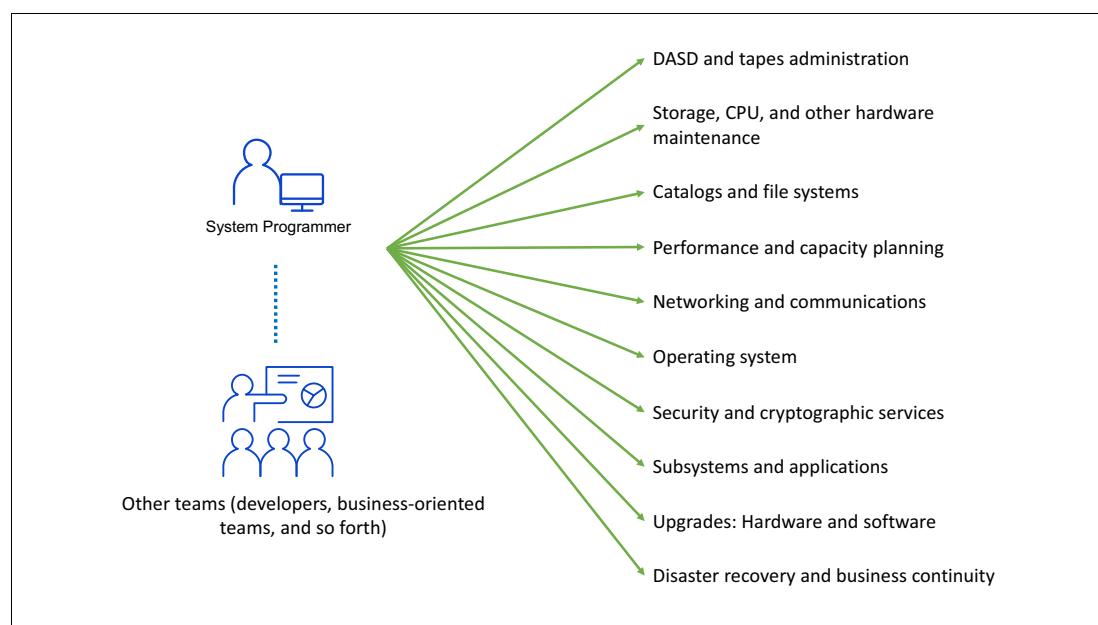


Figure 1-6 Example of system programmer tasks

When the system is up and running, most of the system administrator's time is spent maintaining the system (using for example the help of Health Checker and the z/OSMF) and providing technical support to the users. Thus, the system programmer needs to keep the system running and available but must also communicate and interact with other teams to coordinate changes to minimize any impact of an interruption. Communication and relationships with coworkers are key tools for the system programmer and are as critical as other tasks that are performed by the system programmer.

Storage administration

On a large installation, a person or team might be dedicated to storage administration. In that environment, you might not need to deal with storage directly, other than modifying IODF and IOCDs using HCD. However, your installation might be of a smaller scale that doesn't warrant having a dedicated team of storage administrators, so you might have to take on far more of these roles.

A system programmer who is dedicated to storage administration can be responsible for the hardware upgrades, logical definitions in the equipment, performance incidents, capacity planning, adding devices, implement tools for housekeeping and backups, catalogs, updating new technologies, and so forth.

Networking and communications

The tasks related to network, communication devices, protocols (SNA/VTAM and TCP/IP) and interconnectivity inside and outside the machine, logically and physically to the whole IT Infrastructure and external users or clients. Because the information is highly sensitive, these tasks are tightly related with security and cryptographic services.

Security and cryptographic services

In a similar case to the Storage Administration, a team might be dedicated to the daily tasks like Users and Profile creations, Password resetting, Groups administration, while other team handles tasks like Security configuration, Cryptographic Services, Security integration with Applications and Communications, Audits, and some others.

Performance and capacity planning

These tasks are mostly trusted and used with different tools and records that are saved by the system, but depending on customization. The following most common tasks are related to the system:

- ▶ **SMF records**

The SMF records contain a variety of account information that enables you to produce many types of analysis reports and summary reports so that you can evaluate overall performance, changes in configuration, changes in workload, or job scheduling procedures by studying the trends in the data.

You can also use SMF data to determine system resources wasted because of problems such as inefficient operational procedures or programming conventions, as well as billing data for resource consumption.

- ▶ **LOGREC**

The LOGREC data set contains statistical data about machine failures (processor failures, I/O device errors, channel errors). It also contains records for z/OS and subsystems program error, missing interrupt information, and dynamic device reconfiguration (DDR) actions.

- ▶ Syslog

This data set resides in the JES spool data set. It records all the communications made by z/OS, application programs, and operators through MVS consoles, and it contains all console messages and operator commands for audit and diagnosis purposes. OPERLOG is a data set that merges all of the z/OS SYSLOGs in a Parallel Sysplex. Implement OPERLOG to simplify tracking of an error within a sysplex.

A Syslog daemon is also available to collect UNIX System Services TCP/IP configuration. The messages captured by the Syslog daemon can be sent to SMF, to another system, or to a file for further analysis.

- ▶ Dumps

These are sequential data sets containing system dumps that record areas of virtual storage in case of z/OS program task failures. Refer to *z/OS MVS Diagnosis: Tools and Service Aids*, GA32-0905, for more information about the following types of dumps:

- SVC Dumps
- Transaction Dumps
- Abend Dumps
- SNAP Dumps
- Stand-Alone Dumps

1.6 SMF records introduction

SMF records are generated by hundreds of collecting routines spread all over the z/OS components and other software products. In general, SMF records describe the use of system resources by transactions. The variety of information in the SMF records enables installations to produce many types of analysis reports and summary reports, and the volume of data should be considered in order to have the necessary information when it is required, and avoid the abuse of resource utilization.

You can find more detailed information about performance, capacity planning, and SMF records in *ABCs of z/OS System Programming Volume 11*, SG24-6327.

Command: Use the **DISPLAY SMF** command to obtain information that is related to the SMF.

Collecting SMF records

You can collect the SMF records using one of the following methods:

- ▶ -VSAM data sets

The records are received by the facility and held in a buffer. After the buffer is full or a defined time interval is reached, the data is written in a data set. Eventually, the defined data set fills up, and SMF automatically switches to the next empty one. Most sites use one of these events to kick off a process to extract all data from the full data set and then clear that data set, making it available for use again.

- ▶ -Logstreams

The System Logger is a logging facility for subsystems and applications. System Logger exploiters use a system service to pass blocks of data to Logger. These blocks of data can contain one log record or many log records, as decided by the exploiter. The blocks of data, log blocks in System Logger terminology, are stored by Logger in a log stream.

When you define one log stream you have to define the SMF records type that will be written to this log stream. You can define a default log stream to write all the remaining SMF records types not defined to a specific log stream. Utilize System Logger to improve the write rate and increase the volume of data that can be recorded. System Logger utilizes modern technology, such as the Coupling Facility and media manager, to write more data at much higher rates than the current SMF data set allows.

Unlike the scheduled approach in SMF data set recording, this task is already started and ready to write. In addition, writes to System Logger are done at memory-to-memory speeds, with System Logger accumulating many, many records to write out, resulting in an improved access currently not possible with SMF data set recording.

The benefit in all this is that you can write more data faster, with more functionality. The System Logger was created to handle large volumes of data. With minimal SMF switch processing and no record numbering schemes to maintain, this eliminates the switch SMF command bottleneck. For the offload of this data, exists the possibility to allocate data sets, this is to avoid or significantly reduce offload delay from space issues.

Be aware that SMF records are basically used for recovery of catalogs, therefore be extra careful not to lose any records.

Access real-time SMF records

SMF records have a really important role for the system programmer and for the rest of the people that interacts with the system. It is important to be able to access this records, and in some cases the access must be in the moment that things are going on, to analyze and help predict behavior. For this reason, we can define special resources, called *In-Memory* (INMEM), that allow data to be accessed data as it is buffered when using LogStream.

There is also the possibility to use special exits to access the SMF records, but the main difference between the SMF Real-Time API and the IEFU8x exits is the protection of the records. The new API allows read access and SAF protected to predefined SMF records in the SMFPRMxx member of the parmlib, while the exits can edit all the records.

Note: You can find expanded details about IEFU8x Exits in *z/OS MVS Installation Exits*, SA23-1381.

Potential use cases include:

- ▶ Detecting security violations in real-time
- ▶ Real-time monitoring resource use
- ▶ Dynamic Job Scheduling based on current resource consumption

Tip: The parameter configuration of the SMF is made in the SMFPRMxx member of the parmlib.

1.7 LOGREC data

The information recorded by the z/OS in the LOGREC data set provides a history of all hardware failures, selected z/OS and subsystem software errors, and selected system conditions. Observe that software application abends and errors are not reported in SYS1.LOGREC.

You can use the Environment Record Editing and Printing (EREP) program to complete the following tasks:

- ▶ Print reports about the system records
- ▶ Determine the history of the system
- ▶ Learn about a particular error

Before the system can record all this information, you must first allocate the LOGREC data set and initialize it.

Figure 1-7 illustrates how you can use LOGREC and EREP to manage system data.

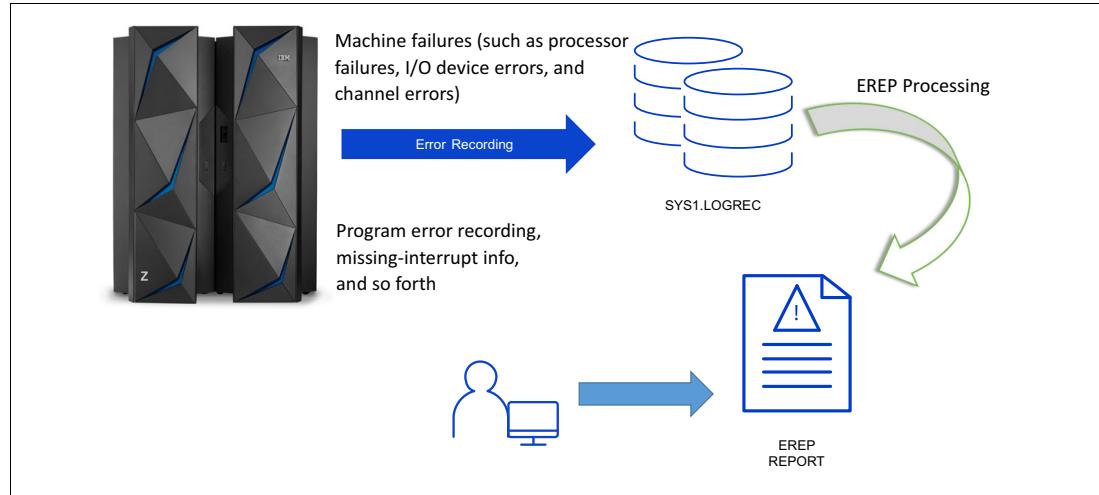


Figure 1-7 Managing system data using LOGREC and EREP

Note: For more information, refer to *Environmental Record Editing and Printing Program (EREP) User's Guide*, GC35-0151, which is available at:

<https://ibm.co/2pNtDIw>

1.8 SYSLOG data

An MVS console is a terminal device where human operators exchange information (data) with software. This data is called *messages* when coming from the software and *replies* or *commands* when coming from the operator (or automation software). The software that is exchanging data can be z/OS components, subsystem products (such as DB2 and CICS), or application programs. The communication task is the z/OS component that is in charge of the console I/O. The following communication task functions can be used by software to send messages to the console:

- ▶ WTO
- ▶ WTOR

For performance and availability reasons, it is preferred to have several consoles in a z/OS environment. Some of the interactions and operations with the console can be automatized using specific tools.

All the messages, replies, and commands in all the z/OS consoles are stored in a log file, either OPERLOG or SYSLOG, or both. This file is part of the SYSLOG data set that is stored in a JES (JES2 or JES3) spool data set as a SYSOUT data set. The SYSLOG data set is used primarily for problem determination and audit.

In addition to the hardcopy log, the SYSLOG data set consists of the following information:

- ▶ All messages issued through write to log (WTL) macros
- ▶ All messages entered by the **LOG** operator commands
- ▶ Any messages routed to the SYSLOG from any system component or program.

Note: The SYSLOG data set is a SYSOUT data set that is provided by the job entry subsystem (either JES2 or JES3).

SYSOUT data sets are 132 characters per line and are stored on direct access storage devices (DASD). An installation should print the SYSLOG periodically to check for problems. You can limit the maximum number of messages that are allowed for the SYSLOG at IPL time by using the LOGLMT parameter in the IEASYSxx parmlib member. The value is used by log processing to determine when the SYSLOG should be scheduled for sysout printing processing by JES. When this value is reached, the log processing completes the following tasks:

- ▶ Issues a simulated **WRITELOG** command to close and free the current SYSLOG.
- ▶ Releases the closed SYSLOG to the printer queue, whose output class is specified by the LOGCLS parameter of the IEASYSxx parmlib member.
- ▶ Allocates and opens a new SYSLOG data set.

Remember that the SYSLOG data set provides a permanent, consolidated record of all console messages and commands (that is, the event log of the system). Thus, you might want to review the SYSLOG at a later date when you are doing problem diagnosis. Instead of having the SYSLOG written to the output queue after a certain number of WTLs, set up a procedure to organize the SYSLOG data set, depending on your installation's requirement.

If you have a Parallel Sysplex, merge all the z/OS SYSLOGs in an OPERLOG data set. This OPERLOG is sorted by an internal time stamp (that is not always identical with the time stamp of the application that issues the WTO or WTOR) for all commands, replies, and messages issued in the Parallel Sysplex. With OPERLOG, you can speed up the problem determination processing in a Parallel Sysplex complex.



Subsystems and subsystem interface

This chapter provides a first approach to the subsystems, the subsystem interface (SSI), and the relation between them. This introduction guides you through the process of understanding the terms and components. You can choose to write your own subsystem or use IBM-provided (as well as third-party vendors) subsystems.

This chapter includes the following main topics:

- ▶ Subsystem and SSI concepts
- ▶ Subsystem initialization
- ▶ Subsystem requests

2.1 Subsystem and SSI concepts

This section provides an overview of subsystems as well as concepts associated with SSI.

2.1.1 Subsystems

A *subsystem* is a service provider that can execute one or many functions as long as it is requested, otherwise it rests on hold. We refer to subsystem to the Master Scheduler Subsystem (MSTR) or any other that is specifically defined to the MVS.

All subsystem callers (programs, users and the MVS) communicate with the subsystem through an API called the subsystem interface (SSI).

Types of subsystems

You can define the following basic types of subsystems to the MVS:

- ▶ Primary subsystems: JES2 or JES3, one of them must be defined
- ▶ Secondary subsystems: IBM Subsystems (such as RACF, SMS, RRS, DB2, IMS™, and so on) or non-IBM third-party vendors

Writing your own subsystem falls under this category.

Defining a subsystem

To define a subsystem, you can choose between the parmlib member definition (member IEFSSNxx), using the IEFSSI macro or finally, with the **SETSSI** command. After the subsystem is correctly defined, it can be activated, deactivated, or deleted.

To avoid unexpected behavior, do not define a subsystem more than once in a combination of IEFSSNxx members that can be used together in an IPL.

Note: Using the Display SSI command shows all the current defined subsystems.

Refer to *z/OS MVS Initialization and Tuning Reference*, SA23-1380 for more information about the subsystem definition using IEFSSNxx.

Tip: When declaring a Subsystem in the IEFSSNxx member, use the keyword format, which allows you to define and dynamically manage (activate and deactivate) subsystems.

2.1.2 The subsystem interface

The SSI is the interface used by routines (IBM, vendor, or installation-written) to request services of, or to pass information to subsystems using the IEFSSREQ macro. An installation can design its own subsystem and use the SSI to monitor subsystem requests, and it can also use the SSI to request services from IBM-supplied subsystems. The SSI acts only as a mechanism for transferring control and data between a requestor and the subsystem; it does not perform any subsystem functions itself.

Figure 2-1 provides an overview depicting the z/OS subsystem interface and how it relates to the overall environment.

Note: The Acronym SSI is also found in IBM z/VM® Systems, but it is *not* related.

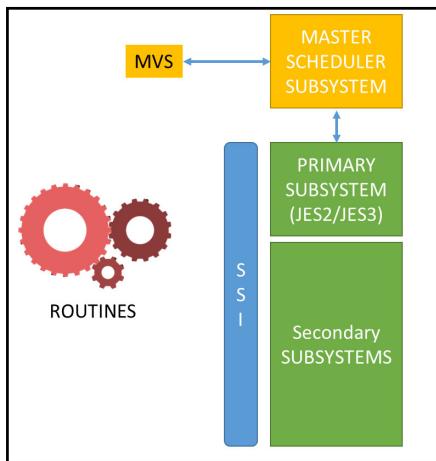


Figure 2-1 The subsystem interface (SSI)

2.2 Subsystem initialization

To process jobs and started task address spaces, z/OS requires that at least one subsystem be defined as a primary job entry subsystem (JES) to receive jobs into the system. You can select either JES2 or JES3, and if you do not specify an IEFSSNxx member in parmlib, z/OS attempts to use the system default member (IEFSSN00, supplied by IBM) which contains the definition for the default primary job entry subsystem, JES2.

If you attempt to complete the IPL process without specifying an IEFSSNxx member and if IEFSSN00 is not present or does not identify the primary subsystem, the IPL code issues a message and prompts the operator for the primary subsystem.

2.2.1 Initializing a subsystem

When you initialize a subsystem, you call specific routines that prepare the subsystem to receive SSI calls. You can initialize the subsystem using one of the following methods:

- ▶ Specifying an initialization routine at IEFSSNxx
- ▶ Using the **START** command

You can also combine these methods, doing part of the setup through an initialization routine, and then completing initialization by using a **START** command.

2.2.2 Specifying an initialization routine

You can optionally specify the name of the subsystem initialization routine when you define the subsystem. If the functions that the subsystem supplies might be needed during the IPL process, define your initialization routine in IEFSSNxx. In this case, the initialization routine handles all the preparation to ensure that the subsystem is active.

2.2.3 Using the START command

If the subsystem is not required at IPL time, you can use the **START** command to invoke the programs to initialize the subsystem in another moment. For more information about the **START** command, refer to *z/OS MVS System Commands*, SA38-0666, and *z/OS MVS JCL Reference*, SA23-1385.

2.3 Subsystem requests

The SSI handles the following types of subsystem caller requests, as shown in Figure 2-2:

- ▶ *Directed requests*, which can be defined by the installation, are guided to *one* named subsystem. For a directed request, the caller informs the named subsystem of an event or asks the named subsystem for information. For example, you can access JES SYSOUT data sets with a directed request.
- ▶ *Broadcast requests*, which can only be called by a z/OS routine, provide the ability for subsystems to be informed when certain events occur in the system. Broadcast requests differ from directed requests in that the system allows multiple subsystems to be informed when an event occurs. The SSI gives control to each subsystem that is active and that has expressed an interest in being informed of the event. For example, the subsystem can be informed when a write to operator with reply (WTOR) message is issued in order to automate a response to the WTOR.

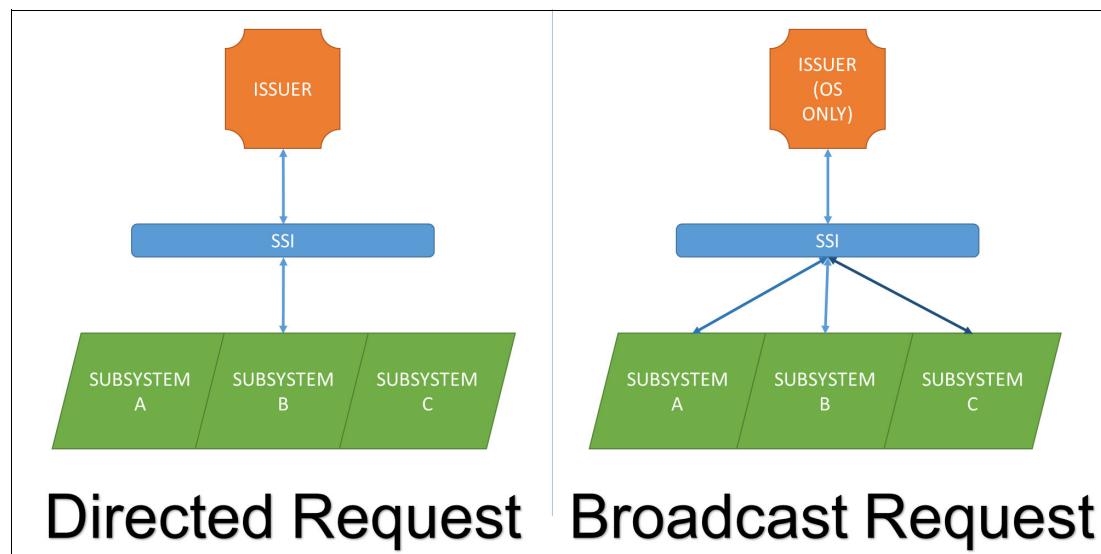


Figure 2-2 Subsystem requests

2.3.1 Requests under the hood: Control blocks and vector tables

The z/OS nucleus contains a *control block*, which is basically a block of memory with information in it, called the *JES control table* (JESCT). This block contains information that is used by the subsystem interface routines, such as a pointer to the primary subsystem communication vector table (SSCVT), which is located in common service area (CSA). The primary subsystem is the JES (JES2 or JES3), and there is at least one other subsystem, the master scheduler, which means at least another SSCVT.

The caller or issuer uses the IEFSSREQ macro to communicate with the subsystem. The caller sets the subsystem options block (SSOB), which identifies the function that the caller is requesting and provides the information that is required to perform the function. Attached to the SSOB, you might find a subsystem identification block (SSIB), which identifies the subsystem and, if required, a function dependant area, which holds additional information for the function.

Figure 2-3 shows an illustration of this request.

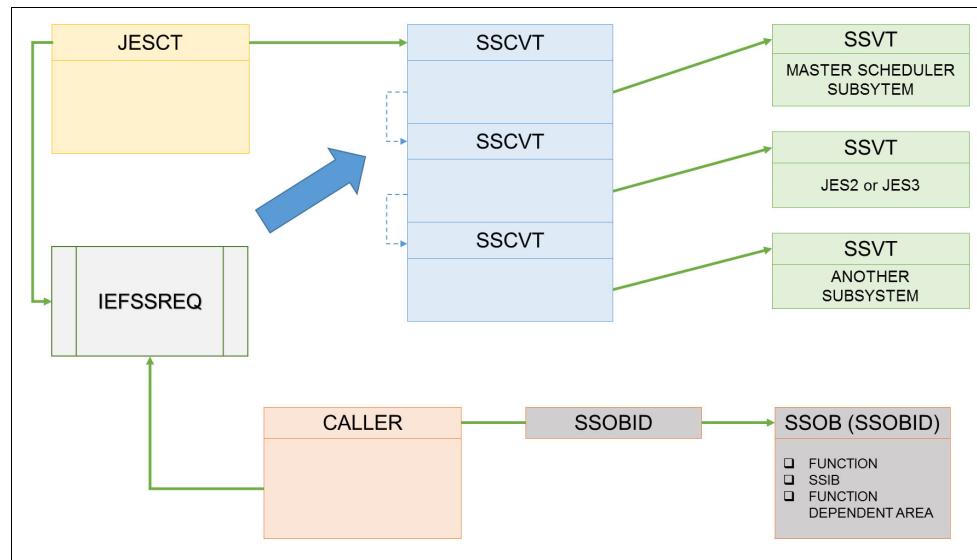


Figure 2-3 Request map

Note: You can find more detail about the SSI and the subsystems at the z/OS MVS Using the Subsystem Interface, SA38-0679.

Figure 2-3 shows other details, for example the connection between JESCT and the request (IEFSSREQ), because JESCT has a pointer to the location of the macro. Another thing you will notice, is that the SSCVTs are one after each other, the reason is because in each SSCVT for every subsystem, there is a pointer to the next, which is signaled by the blue dashed line.



Job management

z/OS job management in a z/OS system uses a Job Entry Subsystem (JES) to receive jobs into the operating system, schedule them for processing by z/OS, and control their output processing. The JES provides supplementary job management, data management, and task management functions, such as scheduling, control of job flow, and spooling. The JES is designed to provide efficient spooling, scheduling, and management facilities for the z/OS system.

For a program to execute on the computer and to perform the work it is designed to do, the program must be processed by the operating system. The operating system consists of a Base Control Program (BCP) with a JES (JES2 or JES3) installed with it. For the operating system to process a program, programmers must perform certain job control tasks. These tasks are performed through the job control statements (JCL). The job control tasks are performed by the JES and are as follows:

- ▶ Entering jobs
- ▶ Processing jobs
- ▶ Requesting resources

A major goal of an operating system is to process jobs while making the best use of system resources. To achieve that goal, the operating system does *resource management*, which consists of the following types of tasks:

- ▶ Before job processing, reserve input and output resources for jobs
- ▶ During job processing, manage resources, such as processors and storage
- ▶ After job processing, free all resources that are used by the completed jobs, which makes the resources available to other jobs

Important: The information provided in this chapter is based on z/OS V2R3.

3.1 Understanding JCL

To get your z/OS system to do work for you, you must describe to the system the work you want done and the resources you will need. You use JCL to provide this information to the z/OS system. At any instant, a number of jobs can be in various stages of preparation, processing, and post-processing activity. To use resources efficiently, the operating system distributes jobs into queues to wait for needed resources according to their stages, such as: conversion queue, waiting execution queue, processing queue, output queue, and so forth. The function of keeping track of which job is in which queue is called *workflow management*.

The z/OS system shares the management of jobs and resources with a JES. JES does job management and resource management before and after job execution, and the z/OS system does job management during job execution. The JES receives jobs into the z/OS system, schedules them for processing by the z/OS system, and controls their output processing.

3.2 JCL control statements

The JCL specifies how programs are executed in the mainframe. JCL functions are the interface between the programs and the operating system. Because JCL has the ability to define data set names, parameters and system output devices, the individual programs can be flexible in their use because these items are not hard coded in the programs. Without the need for re-compiling, the same program can be used to access different data sets and can behave differently based on parameters specified in the JCL.

For every job that you submit, you need to tell the z/OS system where to find the appropriate input, how to process that input (that is, what program or programs to run), and what to do with the resulting output. You use JCL to convey this information to z/OS through a set of statements known as job control statements. The JCL set of job control statements is quite large, enabling you to provide a great deal of information to z/OS.

3.2.1 Syntax rules

A JCL source member consists of a file of 80-byte, fixed-length records. The records (or JCL statements) are written using positions 1-71. Position 72 is reserved for continuation, a space character indicates no continuation and a non-space character indicates the next statement will be a continuation of the current statement.

A JCL statement starts with two slashes (//) in positions 1 and 2. Every JCL member must begin with a *job card* that specifies the job name and other information about how the job executes. Each step executes a program or procedure (PROC). You can add comments statements to the JCL using two slashes and an asterisk (//*) in positions 1 to 3. The JCL is completed by using two slashes (//) in positions 1 and 2 or when the last statement is processed.

3.2.2 Required control statements

Every job must contain a minimum of the following types of control statements:

- ▶ A JOB statement, to mark the beginning of a job and to assign a name to the job. The JOB statement also provides certain administrative information, including security, accounting, and identification information. Every job has one and only one JOB statement.

- An execute (EXEC) statement to mark the beginning of a job step, to assign a name to the step, and to identify the program or procedure to be executed in the step. You can add various parameters to the EXEC statement to customize the way the program executes. Every job has at least one EXEC statement.

In addition to the JOB and EXEC statements, most jobs usually also include one or more data definition (DD) statements that identify and describe the input and output data to be used in the step. You can use the DD statement to request a previously created data set, to define a new data set, to define a temporary data set, or to define and specify the characteristics of the output.

Attention: The names of DD statements are strongly dependent of the calling program. The calling program expects fixed and defined DD names.

Example 3-1 shows a basic JCL example that copies selected records from one data set to another using the z/OS standard **SORT** program.

Example 3-1 Basic JCL example

```
//LUTZSORT JOB 100,CLASS=A,MSGCLASS=(1,1),NOTIFY=&SYSUID      (1)
//*
//STEP010 EXEC PGM=SORT                                     (2)
//SORTIN   DD DSN=LUTZ.INPUT,DISP=SHR                      (3)
//SORTOUT  DD DSN=LUTZ.OUTPUT,                               (4)
//          DISP=(NEW,CATLG,CATLG),DATACLAS=IBMDFLT        (5)
//SYSOUT   DD SYSOUT=*                                      (6)
//SYSUDUMP  DD SYSOUT=M                                     (6)
//SYSPRINT  DD SYSOUT=*                                      (6)
//SYSIN    DD *                                           (6)
          SORT FIELDS=COPY
          INCLUDE COND=(12,4,CH,EQ,C'ITSO')
/*
                                                 (7)
```

The numbers shown in bold font in Example 3-1 apply to the following information:

1. The JOB statement specifies the information that is required for the spooling of the job, such as the job ID, the priority of execution, and the user ID to be notified upon completion of the job.
2. The ///* provides a comment statement and can be used anywhere.
3. The EXEC statement specifies the PROC or program to be executed. In this example, a **SORT** program is executed (that is, the program sorts the input data in a particular order).
4. The SORTIN DD statement specifies the type of input to be passed to the program mentioned in (3). In Example 3-1, a physical sequential (PS) file is passed as input in shared mode (DISP = SHR).
5. The SORTOUT DD statement specifies the type of output to be produced by the program upon execution. In this example, a PS file is created. If a statement extends beyond the 70th position in a line, it is continued in the next line, which should start with two slashes (//) followed by one or more spaces.
6. The information DD statement specifies other types of DD statements for additional information for the program. In this example, the **SORT** condition is specified in the SYSIN DD statement. In addition, it specifies the destination for an error or execution log (for example, SYSUDUMP/SYSPRINT). DD statements can be contained in a data set (mainframe

file) or in stream data (information that is hard coded within the JCL) as given in this example.

7. The /* marks the end of in-stream data.

All the JCL statements except in-stream data start with two slashes (//). At least one space must come before and after the JOB, EXEC, and DD keywords, and there should be no spaces in the remainder of the statement.

3.2.3 Optional JCL control statements

The following optional JCL statements allow the user to define additional runtime options during the job run time:

- ▶ Main storage allocations
- ▶ Timeout definitions
- ▶ System affinity definitions
- ▶ defining priority of the job
- ▶ conditional processing

3.2.4 Optional JECL control statements

For the operating system to process a program, programmers must perform certain job control tasks. These tasks are performed through the job control statements, which consist of:

- ▶ JCL statements for additional functions
- ▶ JES2 control statements
- ▶ JES3 control statements

In general, statements in the z/OS JCL and in the Job Entry Control Language (JECL) for JES2 and JES3 subsystems are used to define a job's execution steps and resource requirements. The z/OS system creates an internal format of the job's JCL statements before it can process it. The z/OS converter or interpreter transforms the external format job description (JCL) into the internal format, called the scheduler work area (SWA). Whatever the source of the input, JES is signalled that an input stream is to be read. This action begins a chain of events that includes deciding where the job executes and creates output data sets that later are processed by JES to the output devices.

3.3 What the JES does

The JES is a component of the z/OS system that receives jobs into the computer system, schedules those jobs for processing based on their priority, and controls their output processing. It is designed to provide efficient spooling, scheduling, and management facilities for the z/OS system. By separating job processing into a number of tasks, the z/OS system operates more efficiently. At any point in time, the computer system resources are busy processing the tasks for individual jobs, and other tasks are waiting for those resources to become available.

In its most simple view, the z/OS system divides the management of jobs and resources between the JES and the Base Control Program (BCP) of the z/OS system. In this manner, the JES manages jobs before executing during the reading and scheduling phases. After the completion of running the program, the JES manages the produced output in printing, punching, and purging the jobs. The BCP manages them during processing, usually without any specific knowledge of the JES.

The z/OS system includes a master JES and a JES2 and JES3. The *master* subsystem is used during system initialization and for starting system tasks that must run outside of the control of the primary JES. In particular, the master JES is used to start the primary JES.

Figure 3-1 shows the job flow using JCL through a z/OS system.

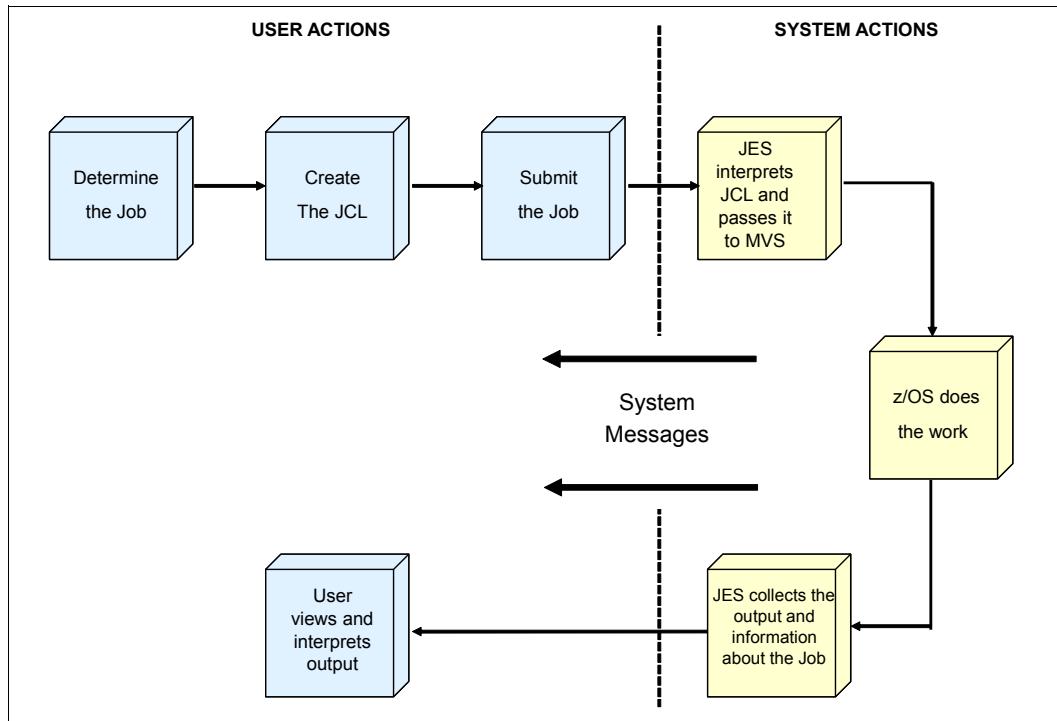


Figure 3-1 JCL flow through the system

The JES completes the following actions to execute a given job using JCL:

Job submission	Submitting the JCL to the JES.
Job conversion	The JCL along with the PROC is converted into an interpreted text to be understood by the JES and is stored into a data set, which is called the <i>spool</i> .
Job queuing	The JES decides the priority of the job based on the CLASS and PRTY parameters in the JOB statement (which is explained in 3.2, “JCL control statements” on page 26). The JCL errors are checked, and the job is scheduled into the job queue if there are no errors.
Job execution	When the job reaches its highest priority, it is taken up for execution from the job queue. The JCL is read from the spool, the program is executed, and the output is redirected to the corresponding output destination as specified in the JCL.
Purging	When the job is complete, the allocated resources and the JES spool space is released. To store the job log, you need to copy the job log to another data set before it is released from the spool.

3.4 JES versions

The two JES version have little in common from an architecture point of view; however, the z/OS batch jobs using JCL usually require little to no changes to be able to run on both JES versions. This design makes it easier to migrate from one JES version to another. In many ways, JES2 provides many of the same functions as JES3 but generally provides these functions in a completely different way. These differences arose over the years, partly because of different functional requirements but also because of fundamental differences in philosophy.

JES3's basic philosophy is one of centralized control where a global system controls some number of local systems, freeing up the local systems 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 that is requesting a function or by the corresponding JES2 address space.

3.4.1 JES2 architecture

In JES2, the work queue resides in a construct called the *checkpoint*. The checkpoint can be placed in a direct access storage device (DASD) data set or in a coupling facility structure. A JES2 member accesses this checkpoint and adds work to, or takes work from, one of the work queues. JES2 can have up to 32 members accessing the checkpoint. All members must also have access to a set of data sets that contain the spool data. Therefore, a multiple-access spool (MAS), also known as *JESplex*, is 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.

Figure 3-2 illustrates at a high-level the JES2 architecture.

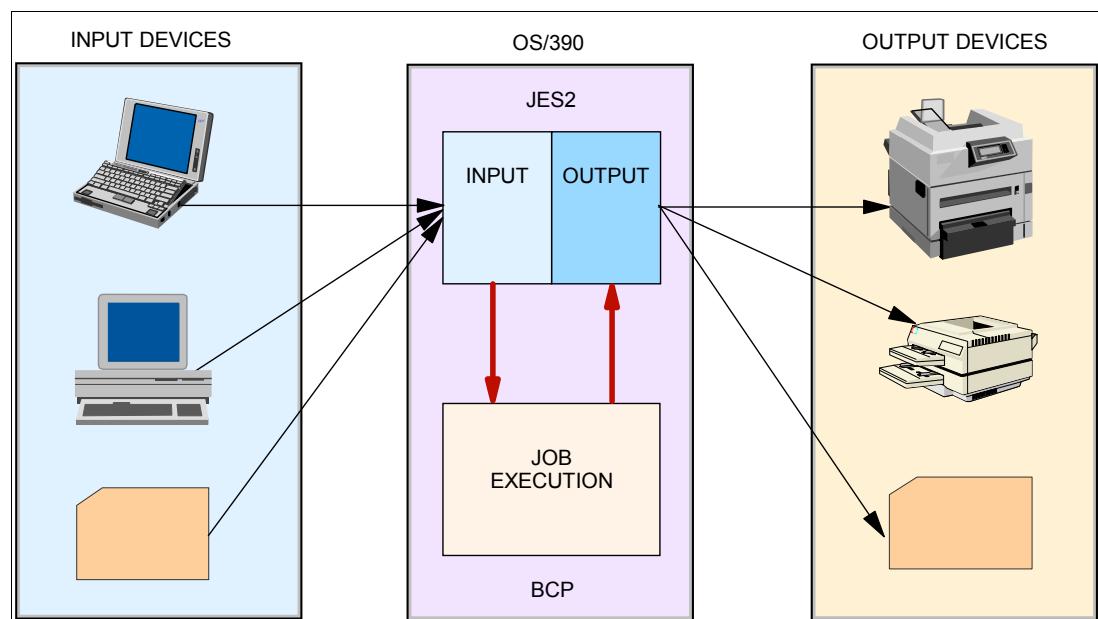


Figure 3-2 JES2 architecture

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 other functions are performed only 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 such as remote job entry (RJE) and network job entry (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. NJE can also be done between JES2 and JES3 or any other operating system that supports the NJE protocol.

3.4.2 JES3 architecture

JES3 runs on either one processor or up to 32 processors in a sysplex. In a sysplex, your installation must designate one processor as the focal point for the entry and distribution of jobs and for the control of resources needed by the jobs. That processor, called the *global processor*, distributes work to the processors, called *local processors*. JES3 manages jobs and resources for the entire complex from the global processor, matching jobs with available resources. JES3 manages processors, I/O devices, volumes, and data. To avoid delays that result when these resources are not available, JES3 ensures that those resources are available before selecting the job for processing. JES3 keeps track of I/O resources and manages workflow in conjunction with the workload management component of z/OS by scheduling jobs for processing on the processors where the jobs can run most efficiently.

Figure 3-3 shows the JES3 architecture at a high-level.

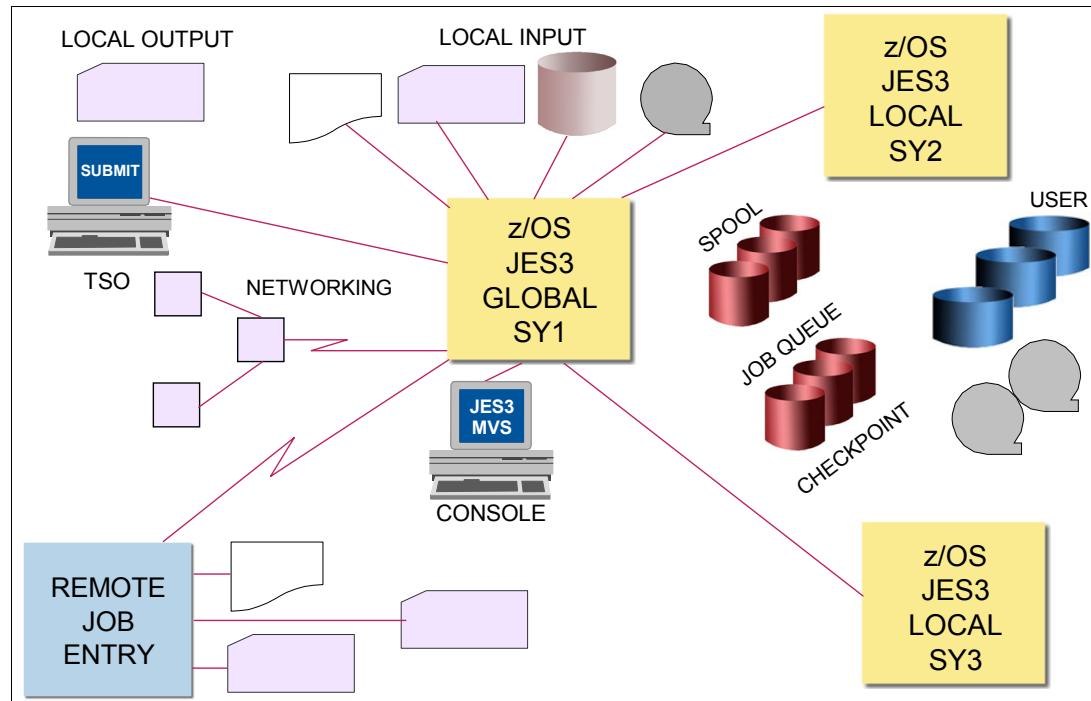


Figure 3-3 JES3 architecture

JES3 maintains data integrity, which means that it will not schedule two jobs to run simultaneously anywhere in the complex if they are going to update the same data. JES3 can be operated from any console that is attached to any system in the sysplex. From any console, an operator can direct a command to any system and receive the response to that command. In addition, any console can be set up to receive messages from all systems or a subset of the systems in the sysplex. Thus, there is no need to station operators at consoles attached to each processor in the sysplex. If you want to share I/O devices among processors, JES3 manages the sharing. Operators do not have to manually switch devices to keep up with changing processor needs for the devices. The JES3 architecture of a global processor, centralized resource and workflow management, and centralized operator control is meant to convey a single-system image, rather than one of separate and independently-operated computers.

3.5 JES2

Simply stated, JES2 is that component of the z/OS system that provides the necessary functions to get jobs into, and output out of, the z/OS system. It is designed to provide efficient spooling, scheduling, and management facilities for the z/OS operating system.

But, none of this explains why z/OS needs a JES. Basically, by separating job processing into a number of tasks, the z/OS system operates more efficiently. At any point in time, the computer system resources are busy processing the tasks for individual jobs, and other tasks are waiting for those resources to become available. In its most simple view, the z/OS system divides the management of jobs and resources between the JES and the BCP of the z/OS system.

Important: To ensure that you can use the provided information in your installation, the JES2 checkpoint must be at the Z22 level. That level also requires the use of SP00LDEF CYL_MANAGED=ALLOWED. See the output of the \$DACTIVE command shown in Example 3-2.

Example 3-2 The \$DACTIVE command output

```
$HASP895 $DACTIVE
$HASP895 JES2 CHECKPOINT MODE IS CURRENTLY Z22
$HASP895 THE CURRENT CHECKPOINT:
$HASP895 -- CONTAINS 6100 BERTS AND BERT UTILIZATION IS 7
$HASP895     PERCENT.
$HASP895 -- CONTAINS 700 4K RECORDS.
$HASP895 z11 CHECKPOINT MODE ACTIVATION WILL:
$HASP895 -- REDUCE CHECKPOINT SIZE TO 630 4K RECORDS.
```

3.5.1 JES2 functions

To manage job I/O, JES2 controls the following functional areas, all of which you can customize according to your installation's need:

- ▶ Getting work into and out of z/OS (I/O control)

JES2 controls output and input devices, such as local and remote printers, punches, and card readers. You define each device to JES2 using JES2 initialization statements. All are directly under JES2's control, with the exception of those printers that operate under the IBM Print Services Facility™ (PSF). Printed and punched output can be routed to a variety of devices in multiple locations. The control JES2 exercises over its printers ranges from

the job output classes and job names from which the printer can select work, to specifications such as the forms on which the output is printed. This control allows the system programmer to establish the job output environment most efficiently without causing unnecessary printer backlog or operator intervention.

Through JES2, the installation defines the job input classes, reader specifications, and output device specifications. As a result, JES2 is the central point of control over both the job entry and job exit phases of data processing. You can also enter jobs to JES2 using Time Sharing Option Extensions (TSO/E) or any other product that can read jobs from a data set and pass them to JES2 via internal reader.

► Maximizing efficiency through job selection and scheduling

JES2 allows the installation to define work selection criteria, and that criteria can be specific for each output device (local and remote printers and punches and offload devices). The work selection criteria is defined in the device initialization statements and can be changed dynamically using JES2 commands. You can define the following information:

- Specification of job and output characteristics that JES2 considers when selecting work for an output device
- Priority of the selection characteristics with or without Workload Manager (WLM)
- Characteristics of the printer and job that must match exactly

A job's output is grouped based on the data set's output requirements. The requirements can be defined in the job's JCL or by JES2-supplied defaults.

When selecting work to be processed on a device, JES2 compares the device characteristics with the output requirements. If they match, JES2 sends the output to the device. If they do not match, JES2 does not select the work for output until the operator changes the device characteristics or the output requirements.changes the device characteristics or the output requirements.

► Offloading work and backing up system workload

JES2 gives the installation the capability to off load data from the spool and later reload data to the spool. This function is useful if you need to:

- Preserve jobs and SYSOUT across a cold start.
- Migrate the installation to another release of JES2.
- Convert to another DASD type for the spool.
- Archive system jobs and SYSOUT.
- Relieve a full-spool condition during high-use periods.
- Provide a backup for spool data sets.
- Back up network connections.

JES2 offers many selection criteria to limit the spool offload operation. These selection criteria can be changed by operator command, according to an initial specification in the JES2 initialization statements.

► System security

JES2 provides a basic level of security for resources through initialization statements. That control can be broadened by implementing several JES2 exits that are available for this purpose. A more complete security policy can be implemented with a System Authorization Facility (SAF) and a security product, such as RACF.

► Supporting advanced function printing (AFP)

JES2 is responsible for all phases of job processing and monitors the processing phase. The BCP and JES2 share responsibility in the z/OS system. JES2 is responsible for job entry (input), the BCP for device allocation and actual job running, and finally JES2 for job exit (output).

3.5.2 JES2 job flow

During the life of a job, both JES2 and the base control program of MVS control different phases of the overall processing. Figure 3-4 presents an overview of the typical job phases in a JES2 job flow.

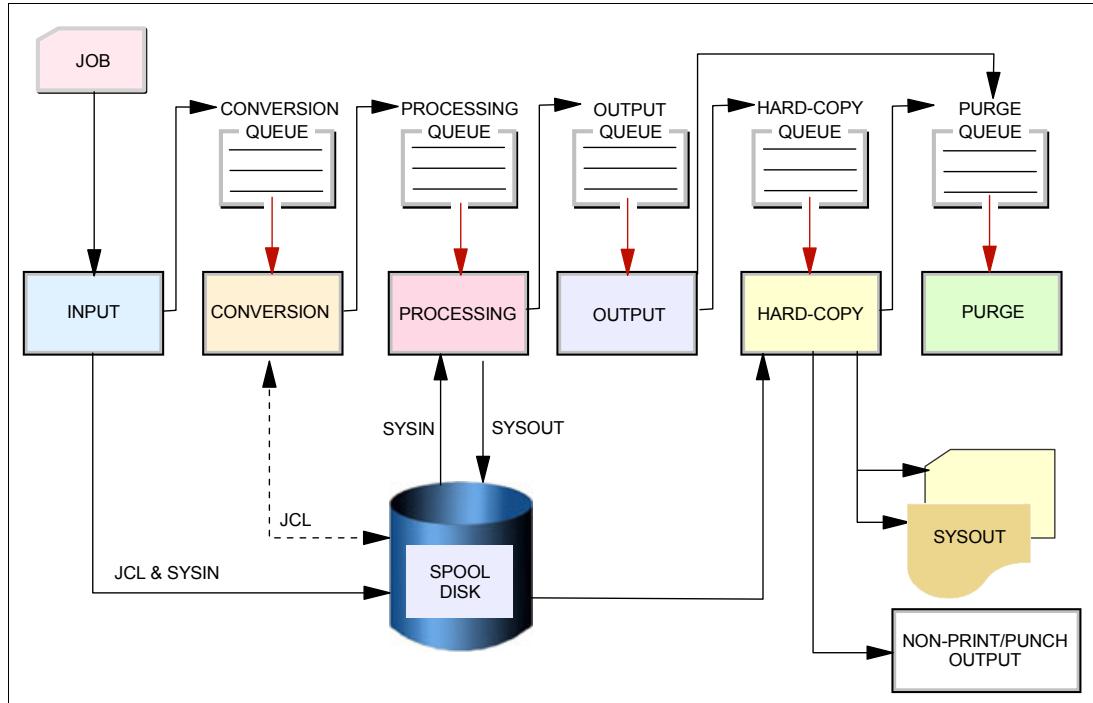


Figure 3-4 The JES2 job flow

The JES2 job flow job processing includes the following phases:

1. Input phase

JES2 accepts jobs, in the form of an input stream, from input devices, internal readers, other nodes in a job entry network, and from other programs. The *internal reader* is a program that other programs can use to submit jobs, control statements, and commands to JES2. Any job running in the z/OS system can use an internal reader to pass an input stream to JES2. JES2 can receive multiple jobs simultaneously through multiple internal readers. The z/OS system uses internal readers, allocated during system initialization, to pass to the JES2 the JCL for started tasks, START and MOUNT commands, and TSO LOGON requests.

The system programmer defines internal readers to be used to process all batch jobs other than STCs and TSO requests. These internal readers are defined in JES2 initialization statements, and JES2 allocates them during JES2 initialization processing. The internal readers for batch jobs can be used for STCs and TSO requests, if not processing jobs. JES2 reads the input stream and assigns a job identifier to each job JCL statement. JES2 places the job's JCL, optional JES2 control statements, and SYSIN data onto DASD data sets called spool data sets. JES2 then selects jobs from the spool data sets for processing and subsequent running.

2. Conversion phase

JES2 uses a converter program to analyze each job's JCL statements. The converter takes the job's JCL and merges it with JCL from a procedure library. The procedure library can be defined in the JCLLIB JCL statement, or system or user procedure libraries can be

defined in the PROCxx DD statement of the JES2 startup procedure. Then, JES2 converts the composite JCL into converter or interpreter text that both JES2 and the job scheduler functions of the z/OS system can recognize. Next, JES2 stores the converter or interpreter text on the spool data set. If JES2 detects any JCL errors, it issues messages, and the job is queued for output processing rather than run. If there are no errors, JES2 queues the job for execution.

3. Processing phase

In the processing phase, JES2 responds to requests for jobs from the z/OS initiators. An *initiator* is a system program that is controlled by JES or by WLM (used in goal mode with WLM Batch Initiator Management). JES2 initiators are defined to JES2 through JES2 initialization statements. JES2 initiators are started by the operator or by JES2 automatically when the system initializes. The installation associates each initiator with one or more job classes to obtain an efficient use of available system resources. Initiators select jobs whose classes match the initiator assigned class, obeying the priority of the queued jobs.

WLM initiators are started by the system automatically based on the performance goals, relative importance of the batch workload, and the capacity of the system to do more work. The initiators select jobs based on their service class and in the order they were made available for execution. Jobs are routed to WLM initiators via a JOBCLASS JES2 initialization statement. In goal mode, with WLM Batch Initiator Management, a system can have WLM initiators and JES2 initiators.

After a job is selected, the initiator invokes the interpreter to build control blocks from the converter or interpreter text that the converter created for the job. The initiator then allocates the resources specified in the JCL for the first step of the job. This allocation ensures that the devices are available before the job step starts running. The initiator then starts the program requested in the JCL EXEC statement.

JES2 and the z/OS BCP communicate constantly to control system processing. The communication mechanism, known as the *subsystem interface*, allows the z/OS system to request services of JES2. For example, a requestor can ask JES2 to find a job, message, or command processing or to open (access) a SYSIN or SYSOUT data set. Further, the BCP notifies JES2 of events, such as messages, operator commands, the end of a job, or the end of a task. By recognizing the current processing phase of all jobs on the job queue, JES2 can manage the flow of jobs through the system.

4. Output phase

JES2 controls all SYSOUT processing. SYSOUT is a data set in a printer device format, that is, it is ready to be printed. Intermediately, a SYSOUT file is stored in the spool data set. There are many advantages in spooling a SYSOUT to JES, such as faster processing (DASD is faster than a printer), optimization of the printer devices, and spool backup and archive. The z/OS system directs to SYSOUT system messages that are related to job execution.

After a job finishes, JES2 analyzes the characteristics of the job's output in terms of its output class and device setup requirements. Then JES2 groups data sets with similar characteristics. JES2 queues the output for print processing.

5. Hardcopy phase

JES2 selects output for processing from the output queues by output class, route code, priority, and other criteria. The output queue can have output to be processed locally or output to be processed at a remote location (either an RJE workstation or another node). After processing all the output for a particular job, JES2 puts the job on the purge queue.

6. Purge phase

When all processing for a job completes, JES2 releases the spool space that is assigned to the job, making the space available for allocation to subsequent jobs. JES2 then issues a message to the operator indicating that the job has been purged from the system.

3.5.3 JES2 spool

The *spool* is the bulk data repository in JES2. It primarily contains JES-managed data sets that contain job input (in-stream data) and output (SYSOUT), which includes job-oriented data sets such as 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 have only a single extent (because of the way the data set is managed). JES2 can access only one spool data set per volume because JES2 uses the volume serial number (VOLSER) that the spool data set resides on to manage the spool data set. Because of this, the terms *spool volume* and *spool data set* can be used interchangeably when referring to a spool extent.

The spool configuration is established when JES2 is cold started but can be altered via operator commands. To show the current spool configuration, use the **\$DSPL** command to show all defined spool volumes, as shown in Example 3-3.

Attention: 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.

Example 3-3 JES2 \$DSPL command output

```
-$DSPL
$HASP893 VOLUME(BH8SP1) STATUS=ACTIVE,PERCENT=23
$HASP646 23.8195 PERCENT SPOOL UTILIZATION
```

For more details about the current spool definitions, use the **\$DSP00DEF** command, as shown in Example 3-4.

Example 3-4 JES2 \$DSP00DEF command

```
-$DSP00DEF
$HASP844 SP00LDEF
$HASP844 SP00LDEF BUFSIZE=3856,DSNAME=SYS1.HASPACE,DSNMASK=,
$HASP844 FENCE=(ACTIVE=NO,VOLUMES=1),GCRATE=NORMAL,
$HASP844 LASTSVAL=(2007.274,21:41:14),LARGEDS=ALLOWED,
$HASP844 SPOOLNUM=32,TGSIZE=60,TGSPACE=(MAX=97728,
$HASP844 DEFINED=30030,ACTIVE=30030,PERCENT=23.8228,
$HASP844 FREE=22876,WARN=80),TRKCELL=3,VOLUME=BH8SP
```

Simultaneous peripheral operations online (spool) have several meanings as used in this book and throughout JES2 documentation. *Spooling* is a process by which the system manipulates its work and includes the following types of activities:

- ▶ Using storage on DASDs as a buffer storage to reduce processing delays when transferring data between peripheral equipment and a program to be run.
- ▶ Reading and writing I/O streams on an intermediate device for later processing or output.
- ▶ Performing an operation, such as printing, while the computer is busy with other work.

Spooling is critical to maintain performance in the z/OS JES2 environment. JES2 attempts to maximize the performance of spooling operations, which ultimately benefits the throughput of work through the JES2 installation.

As noted previously, spooling provides simultaneous processing and a temporary storage area for work that is not yet completed. After JES2 reads a job into the system, JES2 writes the job, its JCL, its control statements, and its data to a spool data set until further processing can occur. For these reasons, spooling is critical to maintain performance in the z/OS JES2 environment.

Spool allocations and management

Spool volumes are identified to JES2 by a volume serial number, the first four or five characters of which are specified by the VOLUME= parameter on the SP0OLDEF statement, as shown in Example 3-5 during JES2 initialization. The fifth and sixth characters can be assigned to designate individual spool volumes and can be any characters that are valid in a volume serial number.

Example 3-5 SPOOLDEF parmlib example

```
SPOOLDEF VOLUME=SYA2
/* SPOOL HAS TO BE ON "SPECIAL" VOLUMES - PREFIXES          */
/*-----*/                                                 */
SPOOLDEF DSNMASK=JES2#A.&SYSNODE..&JESENV.O.SP0OL*           */
/* MASK NAME FOR ALL SPOOL DATASETS (MUST CONTAIN GENERIC CHARS!) */
/*-----*/                                                 */
SPOOL(SYA280) DSNAME=JES2#A.&SYSNODE..&JESENV.O.SP0OL80
SPOOL(SYA281) DSNAME=JES2#A.&SYSNODE..&JESENV.O.SP0OL81
SPOOL(SYA282) DSNAME=JES2#A.&SYSNODE..&JESENV.O.SP0OL82
SPOOL(SYA283) DSNAME=JES2#A.&SYSNODE..&JESENV.O.SP0OL83
SPOOL(SYA284) DSNAME=JES2#A.&SYSNODE..&JESENV.O.SP0OL84
SPOOL(SYA285) DSNAME=JES2#A.&SYSNODE..&JESENV.O.SP0OL85
SPOOL(SYA286) DSNAME=JES2#A.&SYSNODE..&JESENV.O.SP0OL86
SPOOL(SYA287) DSNAME=JES2#A.&SYSNODE..&JESENV.O.SP0OL87
```

Any volume for which the first four to five characters of the volume serial are a value matching the VOLUME= parameter value on the SP0OLDEF statement is assumed to be a spool volume and needs an allocated SYS1.HASPACE data set. If, however, a SYS1.HASPACE data set is not on the volume, JES2 issues the \$HASP414 message, and the volume is not used as a spool volume.

Important: The data set name must be SYS1.HASPACE (the default) unless the DSNAME= parameter on the SP0OLDEF statement specifies a different data set name.

Figure 3-5 shows an example of spool tracks, track cells, and track groups.

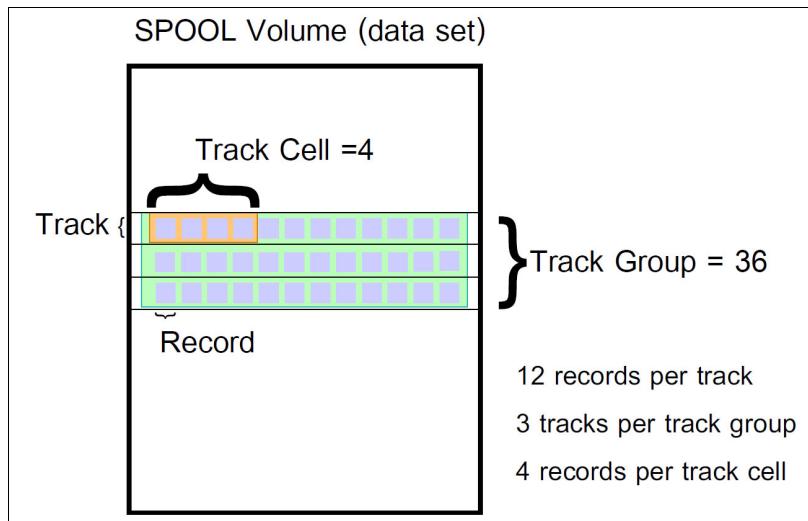


Figure 3-5 Spool allocations

Allocate spool data sets as single-extent data sets. If you allocate additional extents, JES2 uses only the first extent for spool space. Each spool volume must contain a data set that is named SYS1.HASPACE (unless the DSNAME= parameter on the SP00LDEF statement specifies another name) to be used as a spool volume.

Note: JES2 supports using a data set more than 64 K tracks in size up to 1,048,575 tracks.

You can allocate spool space by using any valid space specification, but keep the following considerations in mind:

- ▶ To minimize unused DASD space, allocate spool space contiguously, because JES2 only uses the first extent of the spool data set.
- ▶ The spool allocation must be equal to or greater than the number of tracks in a track group.
- ▶ Track (TRK) allocations waste less DASD space than cylinder (CYL) allocations, because you can specify the allocation as an integral multiple of a track group.

Table 3-1 provides device utilization percentages for 3390 DASD based on several JES2 initialization parameter BUFSIZE specifications. The best DASD utilization rate on 3390 spool volumes can be achieved with a buffer size of 3992.

Table 3-1 Spool BUFSIZE comparison

DASD	Bytes/Track	BUFSIZE	Buffers/Track	Storage utilization
3390	56664	3992	12	97
		3856	12	94
		3768	13	92
		1944	22	95

Spool operations

Spool volumes can be added after a cold start by using the **\$S SP00L** 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. Thus, 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 SP00L** 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 purge can take a number of IPL processes plus waiting for the jobs to age off the system. In many installations that do not complete the IPL process frequently, this purging might take weeks or months. Alternatively, *spool migration* can move data from one volume to another by using the **\$M SP00L** command 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 SP00L** command). It is not required that all spool data sets have the same data set name.

Terminology note: The *initialization deck* refers to the statements that are used by JES during its initialization to obtain the resources it will work with and the parameters that control its processing.

Table 3-2 lists all the possible statuses of a single SPOOL volume. The status can be gathered by using the **\$DSP00L** command that is shown in Example 3-3 on page 36.

Table 3-2 Status of a single spool volume

Status of spool volume	Description
ACTIVE	<ul style="list-style-type: none">▶ Normal state of a volume▶ Selectable and allocatable
STARTING	<ul style="list-style-type: none">▶ Volume transitioning to ACTIVE▶ Could be initial use of a volume▶ Not selectable or allocatable▶ Could be transitioning from a state below▶ Selectable and allocatable based on old state
HALTING	<ul style="list-style-type: none">▶ Transitioning to INACTIVE▶ Not selectable and not allocatable▶ Waiting for active address processes to stop
INACTIVE	<ul style="list-style-type: none">▶ Not z/OS allocated but still defined▶ Not selectable and not allocatable
DRAINING	<ul style="list-style-type: none">▶ Transitioning to deleted (not exist)▶ Selectable but not allocatable▶ Waiting for all jobs with space on the volume to go away
Reserved	<ul style="list-style-type: none">▶ Property of an ACTIVE volume▶ Not allocatable but selectable▶ Not waiting or transitioning to a new state▶ Set or reset by the \$T or \$\$ command

Status of spool volume	Description
Migrating	<ul style="list-style-type: none"> ▶ Active migration moving data ▶ Not selectable and not allocatable ▶ Will transition to MAPPED status when migration completes, and switches to DRAINING status if migration fails
Mapped	<ul style="list-style-type: none"> ▶ Physically exists on a target volume ▶ Not allocatable, selectable inherited from target volume ▶ Waiting for all jobs with space on volume to go away ▶ Will transition to does not exist (like DRAINING) ▶ No commands allowed against volume (cannot change state)
Extending	<ul style="list-style-type: none"> ▶ Size of volume being increased ▶ Selectable and allocatable ▶ Returns to ACTIVE when process completes

To operate with spool volumes, JES2 provides several commands, which can be found in *z/OS JES2 Commands*, SA32-0990.

EAV considerations

To expand the capacity of DASD storage volumes beyond 65,520 cylinders, the z/OS system had to extend the track address format. Thus, the name extended address volume (EAV) is used for a volume of more than 65,520 cylinders, as shown in Figure 3-6.

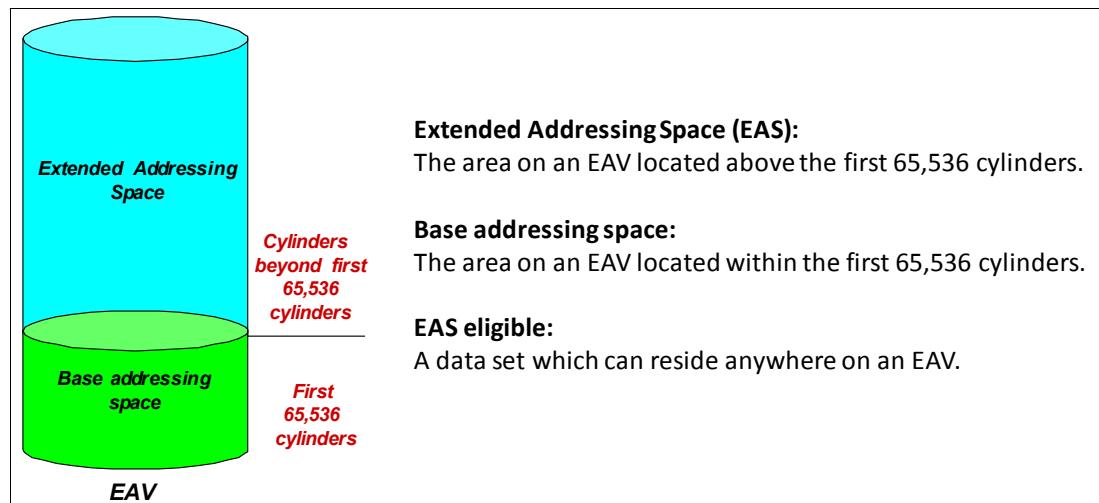


Figure 3-6 JES2 EAV

EAVs provide increased z/OS addressable disk storage. EAVs help to relieve storage constraints and simplify storage management by providing the ability to manage fewer, large volumes as opposed to many small volumes.

DFSMS added support for base and large format sequential data sets that now can be used for JES2 spool and checkpoint data sets. EAS eligible data sets are defined to be those that can be allocated in the EAS and have extended attributes. EAS is sometimes referred to as cylinder-managed space. Below EAS is sometimes referred to as *track-managed space*. A new data set attribute, EATTR, allows a user to control whether a data set can have extended attribute DSCBs and, thus, control whether it can be allocated in EAS.

Attention: To enable JES2 using EAV managed data sets you have to enable CYL_MANAGED=ALLOWED in the JES2 SPOOLDEF statement as shown in Figure 3-7.

```
$TSPPOOLDEF,CYL_MANAGED=ALLOWED
$HASP844 SPOOLDEF BUFSIZE=3992,DSNAME=SYS1.HASPACE,
$HASP844           FENCE=(ACTIVE=NO,VOLUMES=1),GCRATE=NORMAL,
$HASP844           LASTSVAL=(2010.058,22:48:01),LARGEDS=ALLOWED,
$HASP844           SPOOLNUM=32,CYL_MANAGED=ALLOWED,TGSIZE=6,
$HASP844           TGSPACE=(MAX=16288,DEFINED=525,ACTIVE=525,
$HASP844           PERCENT=6.2857,FREE=492,WARN=80),TRKCELL=3,
$HASP844           VOLUME=SPOOL
```

Figure 3-7 JES2 activate CYL_MANAGED=ALLOWED

JES2 use of the EAV support allows you to define EAS eligible data sets for use as SPOOL extents and checkpoint data sets. The data sets can be placed anywhere on a EAV volume, but they cannot be larger than the 1,048,575 track JES2 architectural limit in size.

Spool migration

A JES2 spool migration moves an existing JES2 spool volume (an extent or data set) to a new spool volume or merges an existing volume with another existing spool volume. Migrating a spool volume to a new spool volume is called a *move migration*. Migrating a spool volume to an existing spool volume is called a *merge migration*.

As described here, the **\$MSPL(volsr,)** command queues a request to move one or more existing spool volumes to a new spool volume or to merge them with an existing spool volume:

- ▶ For move requests, the target spool volume specified as a VOLSER is not recognized by JES2. When the move completes, the source spool volumes are no longer recognized by JES2.
- ▶ For merge requests, the specified target must be a spool volume that is recognized by JES2. When the merge completes, the source spool volumes remain in the MAPPED state until all users of the source spool volumes are purged.

Spool migration types

Both types of spool migration enhance JES2 spool configuration by providing the following functions:

- ▶ Increasing or reducing the total number of spool volumes
- ▶ Increasing or reducing the size of spool volumes
- ▶ Removing spool volumes without altering any spool pointers (MTTRs or MQTRs)
- ▶ Copying of data from one spool volume to another while address spaces are actively reading and writing data to the spool volumes
- ▶ Merging of volumes and track group map onto another spool volume
- ▶ Creating a new spool volume

MERGE migration

A MERGE migration copies an existing source volume to free space on a target volume, as shown in Figure 3-8. Upon completion, the source volume becomes a mapped volume. The volume remains mapped until all jobs and SYSOUT that have space on the Source Volume are purged. It then goes away (no longer exists).

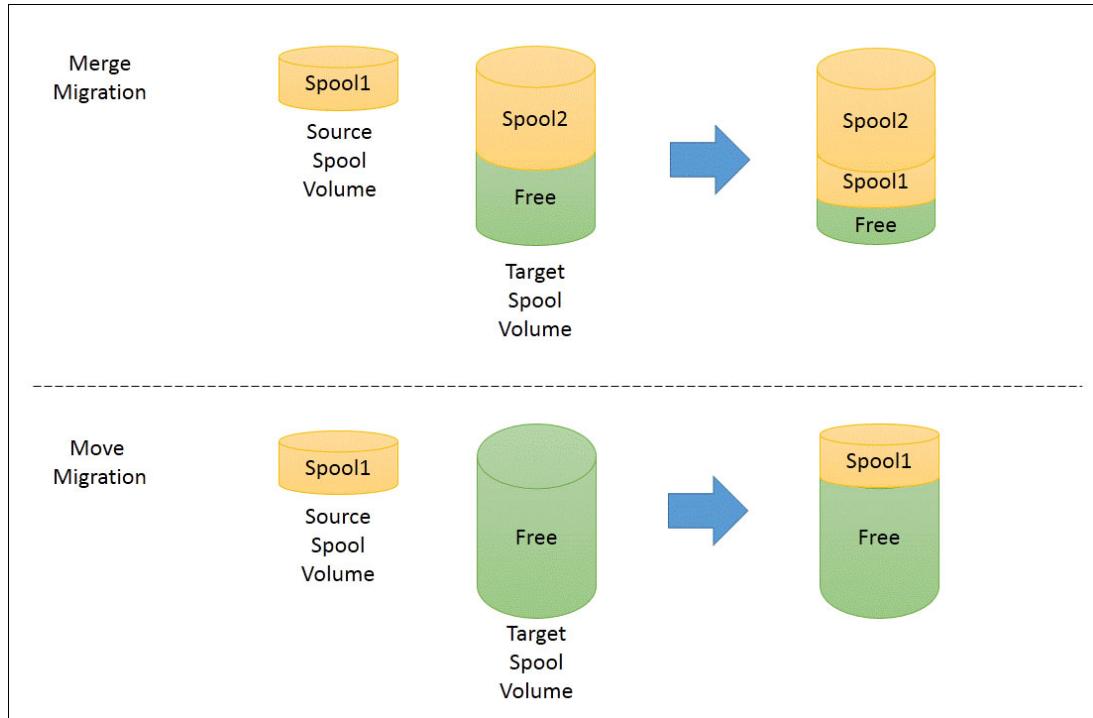


Figure 3-8 Spool migration types

Restrictions: Consider the following restrictions:

- ▶ The Target Volume must be Active (can be Reserved).
- ▶ The Records Per Track of the Target Volume cannot be less than the Source Volume.
- ▶ The Target Volume must use relative addressing.
- ▶ The Target Volume cannot be actively extending.

COPY migration

A MOVE migration copies a single Inactive (halted) source volume to a new target volume. Upon completion, the target volume inherits the \$DAS structure of the source volume.

Restriction: The source volume must be Inactive. You cannot move an absolute volume (instead, do a merge). You need to use the SPACE= parameter. The target volume inherits the source volume tracks per track group value.

JES2 managing SPOOL migration

To manage the migration of spool volume data and the access to that data, JES2 runs various operations on all members of a MAS. These operations communicate messages through JESXCF services. JESXCF services require one XCF group per active migration to identify the messages that are specific to each active migration.

Restriction note: JES2 limits the number of simultaneous migrations to five due to XCF considerations. If a spool migration is requested and an XCF group is not available, JES2 refuses the request and issues a \$HASP808 message with return code 57 or 58. To determine the number of available XCF groups, run the **D XCF,COUPLE** command.

Be aware of the following restrictions:

- ▶ The source volume cannot be a mapped target.
- ▶ The source volume cannot be actively migrating or extending.
- ▶ The source volume cannot be stunted.

Figure 3-9 shows the spool migration phases.

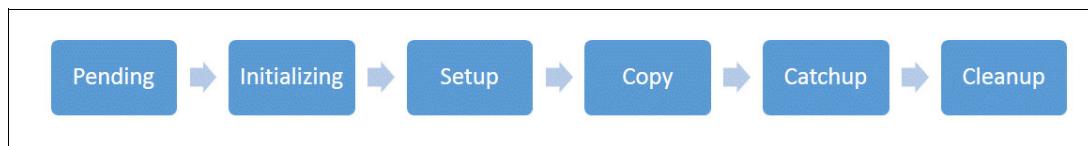


Figure 3-9 Spool migration phases

Spool migration states with the **\$DSPL** command The new spool volume proceeds through the following states:

MIGRATING	The spool volume is a source of an active migration.
MAPPED	The spool volume has been migrated, and the corresponding data set is eligible for deletion. The volume remains MAPPED until all jobs and SYSOUT that have space on the volume are purged.
The \$DSPL command filters the migrating volumes by the following current phase types:	
PENDING	Awaiting start of migration. INITIALIZING General migration configuration work is being done, such as creating subtasks, data structures, and XCF mailboxes.
SETUP	Setup for a migration is being done. All MAS members participate in this process.
COPY	The data set on the source spool volume was migrated to the target spool volume. Runtime changes are coordinated and tracked by the migration.
CATCHUP	Tracks that were changed by runtime operations during the COPY phase are being recopied.
CLEANUP	General cleanup is being done at the end of the migration.
CANCEL	Migration subtask cleanup is being done because an operator cancelled the active migration, or the migration process encountered an error.
BACKOUT	Updates are being backed out because an operator cancelled the active migration, or the migration process encountered an error.

Note: An active Migration can be cancelled using the **\$MSPL(source),CANCEL** command. However, a migration cannot be cancelled during the CATCHUP phase or any CLEANUP phase. Upon completion of a cancel, the source volume remains in DRAINING state.

Check volume space for migration

You can use the **\$D SPOOL,MIGDATA** command to determine if enough space exists on a target volume to accommodate a planned source volume or volumes. The command displays the largest contiguous free area on each spool volume, as shown here:

```
$D SPOOL,MIGDATA
$HASP893 VOLUME (SPOL3)
$HASP893 VOLUME (SPOL3) MIGDATA=(SPACE_USED=40000,LARGEST_FREE=10000)
$HASP893 VOLUME (SPOL5)
$HASP893 VOLUME (SPOL5) MIGDATA=(SPACE_USED=20000,LARGEST_FREE=10000)
```

To display all volumes having contiguous free space greater than 3000 cylinders, issue the following command:

```
$D SPOOL,MIGDATA=LARGEST_FREE>3000,MIGDATA
$HASP893 VOLUME (SPOL2)
$HASP893 VOLUME (SPOL2) MIGDATA=(LARGEST_FREE=4000)
$HASP893 VOLUME (SPOL3)
$HASP893 VOLUME (SPOL3) MIGDATA=(LARGEST_FREE=7000)
```

Single spool migration examples

The example shown in Figure 3-10 demonstrates the use of the **\$MSPL(volser,)** command. The **\$MSPL(volser,)** command can be used to migrate one or more existing spool volumes (VOLSERs) to a new spool volume. Note the following points regarding this example:

- ▶ When you issue the **\$MSPL** command, JES2 creates a new data set on the new spool volume J2SPL1. The default data set name is defined by the **\$SPOOLDEF** command.
- ▶ The data set size is specified by the **SPACE=MAX** parameter, indicating the largest size possible considering the available disk space and JES2 architecture limits.
- ▶ After the spool volume J2SPL1 is formatted, its state is **RESERVED**, and it remains in that state until the **\$TSPOLL RESERVED=NO** command is executed.
- ▶ The SPOOL6 spool volume is in the **INACTIVE** state prior to the migration and transitions to the **MIGRATING** state during the migration. SPOOL6 is removed from the SPOOL configuration (drained) at the completion of the migration.
- ▶ The data set previously associated with the spool volume SPOOL6 can now be deleted.

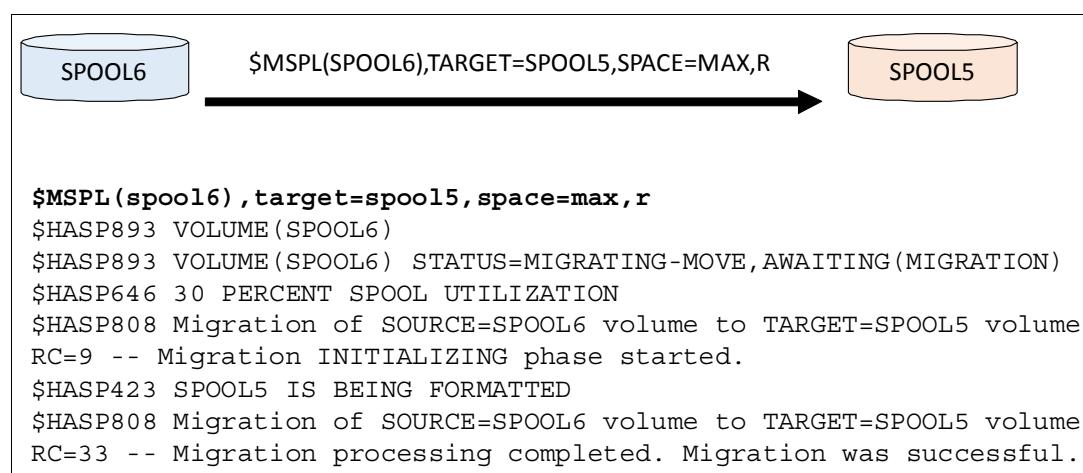


Figure 3-10 Example of the **\$MSPL** command

In Figure 3-10 on page 44, the \$HASP808 message displays the migration progress. The \$DSPL command displays the status of the target volume:

```
$DSPL(spool5),status,percent,reserved  
$HASP893 VOLUME(SPOOL5) STATUS=ACTIVE,PERCENT=0,RESERVED=YES  
$HASP646 30 PERCENT SPOOL UTILIZATION
```

Multiple spool migration example

You can use the \$MSPL(volser,) command to merge one or more existing spool volumes (VOLSERs) with a single existing spool volume. This scenario merges three spool volumes (Spool2, Spool3 and Spool4) into another (Spool5), as shown in Figure 3-11. When the \$MSPL(SPOOL2,SPOOL3,SPOOL4),target=SPOOL5 command is used (as shown in Figure 3-11), JES2 merges spool volumes Spool2, Spool3, and Spool4 with the existing active spool volume Spool5.

In this example, the following process occurs:

- ▶ Spool volume Spool2 is merged first, followed by Spool3, and then Spool4.
- ▶ Spool volumes Spool3 and Spool4 migration phases are pending while Spool2 is migrating.
- ▶ After migration is complete, Spool2, Spool3, and Spool4 are left in the MAPPED state.
- ▶ At this point, the data sets previously associated with volumes Spool2, Spool3, and Spool4 can be deleted.
- ▶ Both spool volumes remain in a MAPPED state until all jobs using the volumes are purged. At that point Spool2, Spool3, and Spool4 are removed from the spool configuration (drained), and the corresponding spool extent number can be reused.

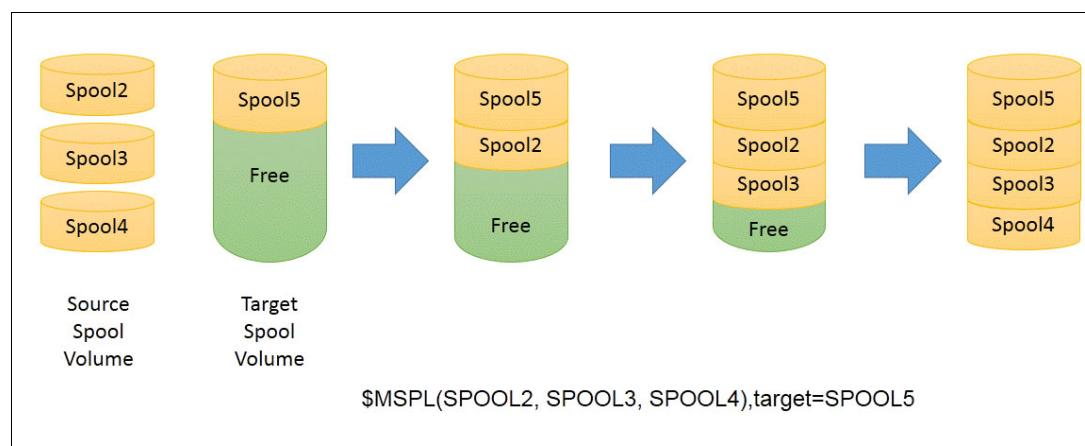


Figure 3-11 Spool migration process view

Figure 3-12 shows the messages that are displayed during the migration process.

```
$MSPL(spool12,spool13,spool14),target=spool15
$HASP808 Migration of SOURCE=SPPOOL2 volume to TARGET=SPPOOL5 volume
RC=9 -- Migration INITIALIZING phase started.
$HASP893 VOLUME(SPOOL2)
$HASP893 VOLUME(SPOOL2) STATUS=MIGRATING-MERGE,AWAITING(MIGRATION)
$HASP893 VOLUME(SPOOL3) STATUS=MIGRATING-MERGE,AWAITING(MIGRATION)
$HASP893 VOLUME(SPOOL4)
$HASP893 VOLUME(SPOOL4) STATUS=MIGRATING-MERGE,AWAITING(MIGRATION)
$HASP646 20 PERCENT SPOOL UTILIZATION
$HASP808 Migration of SOURCE=SPPOOL2 volume to TARGET=SPPOOL5 volume
RC=33 -- Migration processing completed. Migration was successful.
$HASP808 Migration of SOURCE=SPPOOL3 volume to TARGET=SPPOOL5 volume
RC=9 -- Migration INITIALIZING phase started.
$HASP630 VOLUME SPOOL2 INACTIVE 2 PERCENT UTILIZATION
$HASP808 Migration of SOURCE=SPPOOL3 volume to TARGET=SPPOOL5 volume
RC=33 -- Migration processing completed. Migration was successful.
$HASP808 Migration of SOURCE=SPPOOL4 volume to TARGET=SPPOOL5 volume
RC=9 -- Migration INITIALIZING phase started.
$HASP630 VOLUME SPOOL3 INACTIVE 1 PERCENT UTILIZATION
$HASP808 Migration of SOURCE=SPPOOL4 volume to TARGET=SPPOOL5 volume
RC=33 -- Migration processing completed. Migration was successful.
```

Figure 3-12 Spool migration messages

Use the **\$DSPL** command to see the status of the target volume and the source volumes if they are in a MAPPED state. The **\$HASP893** messages, shown in Figure 3-13, display the status for each spool volume defined to JES2. The **\$HASP646** message displays the total percent spool utilization for the complex of all active spool volumes.

```
$dspl,status,target,maptarget
$HASP893 VOLUME(SPOOL1) STATUS=ACTIVE,MAPTARGET=NO
$HASP893 VOLUME(SPOOL2) STATUS=MAPPED,TARGET=SPPOOL5,MAPTARGET=NO
$HASP893 VOLUME(SPOOL3) STATUS=MAPPED,TARGET=SPPOOL5,MAPTARGET=NO
$HASP893 VOLUME(SPOOL4) STATUS=MAPPED,AWAITING(JOB),TARGET=SPPOOL5,
MAPTARGET=NO
$HASP893 VOLUME(SPOOL5) STATUS=ACTIVE,MAPTARGET=YES
$HASP646 30 PERCENT SPOOL UTILIZATION
```

Figure 3-13 \$DSPL SPOOL targets

Figure 3-13 shows SPOOL2, SPOOL3, and SPOOL4 volumes marked with the TARGET option, meaning that an ongoing spool migration is in process. The spool volume was marked with MAPTARGET=YES, which means that in the ongoing spool migration, the volume became the target.

The new parameters for the **\$DSPL** command are as follows:

- ▶ MAPTARGET=Yes | No

Indicates whether the volume is the target of a MAPPED spool volume.

- Yes indicates the volume is the target of a MAPPED volume.
- No indicates the volume is not the target of a MAPPED volume.

- ▶ MIGDATA=([LARGEST_FREE | SPACE_USED])

On a per-spool volume basis, displays the largest contiguous free space or the highest used location. The unit of measurement is tracked.
- ▶ MIGRATOR

Displays the name of the JES2 MAS member that is performing the migration of a MIGRATING spool volume.
- ▶ MPERCENT

Displays the percentage of the migration that has completed.
- ▶ STATUS=[ACTIVE | DRAINING | EXTENDING | HALTING | INACTIVE | MAPPED | MIGRATING | STARTING]
 - MAPPED

Indicates that the spool volume has been migrated and that the corresponding data set is eligible for deletion.
 - MIGRATING

Indicates that the spool volume is a source of an active migration. The associated target volume is also displayed.

Note: The spool volume extent number will persist until all jobs and SYSOUT that have space on the volume have been purged.

– MIGRATING

Indicates that the spool volume is a source of an active migration. The associated target volume is also displayed.

JES2 checkpoint

The JES2 checkpoint function performs the following separate functions:

- ▶ Job and output queue backup to ensure ease of JES2 restart
- ▶ MAS member-to-member workload communication to ensure efficient independent JES2 operation

The function or functions that the checkpoint data set performs in your configuration depends on whether you have a JES2 single-member MAS or a multi-member MAS with as many as 32 members.

Checkpointing is the periodic copying of a member's in-storage job and output queues to the checkpoint data set, which can reside on either a DASD volume or a coupling facility structure. Checkpointing ensures that information about a member's in-storage job and output queues is not lost if the member loses access to these queues as the result of either hardware or software errors. Because all members in a JES2 MAS configuration operate in a loosely-coupled manner, each capable of selecting work from the same job and output queues, checkpointing allows all members to be aware of current queue status. Within the single-member environment, the checkpoint function operates solely as a backup to the "in-storage" job and output (SYSOUT) queues that are maintained by JES2.

In a MAS environment, the checkpoint data set not only backs up the job and output queues, it also links all members. It is the commonly accessible repository of member activity that allows each member to communicate and be aware of the current work load. The checkpoint data set contains a record of member values that describe the overall configuration of the MAS environment and specific characteristics and information that describes the current status of each member. The checkpoint allows all members to access and update (write to) the checkpoint data set and also allows all members to refresh their in-storage queues by reading from the checkpoint data set.

Errors can occur while processing jobs and data. Some of these errors can result in the halting of all system activity. Other errors can result in the loss of jobs or the invalidation of jobs and data. If such errors occur, it is preferable to stop JES2 in a way that allows processing to be restarted with minimal loss of jobs and data.

Important: Information in the checkpoint data set is critical to JES2 for normal processing and also if a JES2 or system failure occurs. Therefore, it is strongly recommended to duplex JES2 checkpoint data set.

JES2 periodically updates the checkpoint data set by copying the changed information from the in-storage copy to the checkpoint data set copy. Example 3-6 shows an example of an checkpoint definition designed for high availability and maximum performance.

Example 3-6 Example of checkpoint definitions

CKPTDEF CKPT1=(STRNAME=JES2CKPT_1,INUSE=YES)	(1)
CKPTDEF NEWCKPT1=(STRNAME=JES2CKPT_1_NEW)	(2)
CKPTDEF CKPT2=(DSNAME=JES2#A.&SYSNODE..&JESENV.O.CKPT2, VOLSER=SYA210,INUSE=YES)	(3)
CKPTDEF NEWCKPT2=(DSNAME=JES2#A.&SYSNODE..&JESENV.O.CKPT2NEW, VOLSER=SYA211)	(4)

This sample configuration uses one primary checkpoint in an external coupling facility and a copy on DASD. The numbers in bold font in the example correspond to the following information:

1. Assigns CKPT1 to a structure that is defined in the coupling facility and makes sure to achieve the maximum performance for accessing the checkpoint.
2. In case of failure of primary CKPT1, JES2 automatically switches to the structure that is defined in NEWCKPT1. This new structure should be located in another coupling facility than CKPT1.
3. Defines the secondary CKPT2 on a DASD with the given data set name, which avoids data loss in case of an outage of coupling facility because they are volatile.
4. Defines a backup for CKPT2 and is used automatically by JES2 in case of failures with CKPT2.

During JES2 startup, messages indicate the use of checkpoint. Example 3-7 show some checkpoint information from JES2 that point to the coupling facility and use of a structure named JES2CKPT_1.

Example 3-7 JES2 startup checkpoint messages

```
IXL014I IXLCONN REQUEST FOR STRUCTURE JES2CKPT_1 798
    WAS SUCCESSFUL.  JOBNAME: JES2 ASID: 002B
    CONNECTOR NAME: JES2_R21 CFNAME: CF15R1
    $HASP478 INITIAL CHECKPOINT READ IS FROM CKPT1 801
        (STRNAME JES2CKPT_1)
        LAST WRITTEN THURSDAY, 5 OCT 2017 AT 00:09:19 (GMT)
*$HASP493 JES2 ALL-MEMBER WARM START IS IN PROGRESS - z22 MODE
$HASP537 THE CURRENT CHECKPOINT USES 5280 4K RECORDS
```

During runtime you can use KES2 \$DCKPTDEF system command to get the current active checkpoint configuration of your system. Example 3-8 shows the output of the JES2 checkpoint configuration that is defined in Example 3-6 on page 48.

Example 3-8 JES2 display of checkpoint configuration

```
$DCKPTDEF
$HASP829 CKPTDEF
$HASP829 CKPTDEF CKPT1=(STRNAME=JES2CKPT_1,INUSE=YES,VOLATILE=YES),
$HASP829           CKPT2=(DSNAME=JES2#A.RR2.P0.CKPT2,VOLSER=SYA210,
$HASP829           INUSE=YES,VOLATILE=NO),
$HASP829           NEWCKPT1=(STRNAME=JES2CKPT_1_NEW),
$HASP829           NEWCKPT2=(DSNAME=JES2#A.RR2.P0.CKPT2NEW,
$HASP829           VOLSER=SYA211),MODE=DUPLEX,DUPLEX=ON,LOGSIZE=1,
$HASP829           VERSIONS=(STATUS=ACTIVE,NUMBER=50,WARN=80,MAXFAIL=0,
$HASP829           NUMFAIL=0,VERSFREE=50,MAXUSED=2),RECONFIG=NO,
$HASP829           VOLATILE=(ONECKPT=IGNORE,ALLCKPT=WTOR),OPVERIFY=NO
```

JES2 provides a dynamic means by which the current checkpoint configuration can be changed. Example 3-9 shows the checkpoint reconfiguration, which can be initiated by the **\$TCKPTDEF, RECONFIG=YES** operator command or by JES2. JES2 enters a checkpoint reconfiguration for any of the following reasons:

- ▶ To complete initialization processing:
 - Either JES2 cannot determine the checkpoint data set specifications, or JES2 requires operator verification of the data set specifications
 - JES2 startup option
 - Checkpoint statement definition
- ▶ To correct an I/O error to the checkpoint data set
- ▶ To remove the checkpoint from a volatile coupling facility structure

An operator initiated reconfiguration is started with a **\$TCKPTDEF, RECONFIG=YES** command. This command is entered on any member of the MAS and places all members into the reconfiguration. This form of the reconfiguration is used to move, suspend, or resume a checkpoint data set. Either data set can be processed at this time. Canceling out of this reconfiguration returns JES2 to normal operation. An error reconfiguration is entered when JES2 detects an error on the checkpoint data set (either with the device or with the data that was read). This reconfiguration is used to process an error with a specific data set. The data set can be moved to a new location or use of the data set can be suspended. Optionally, an installation can set up to automatically forward the error data set in the event of an error. Canceling out of an error reconfiguration causes all members that encountered the error to ABEND.

Example 3-9 JES2 checkpoint reconfiguration

```
$TCKPTDEF, RECONFIG=YES
*$HASP285 JES2 CHECKPOINT RECONFIGURATION STARTING
*$HASP233 REASON FOR JES2 CHECKPOINT RECONFIGURATION IS OPERATOR 813
                  REQUEST
*$HASP285 JES2 CHECKPOINT RECONFIGURATION STARTED - DRIVEN BY 814
                  MEMBER SC81
*$HASP271 CHECKPOINT RECONFIGURATION OPTIONS 815
*
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
*002 $HASP272 ENTER RESPONSE (ISSUE D R,MSG=$HASP271 FOR RELATED MSG)

```

With option 2, as shown in Figure 3-14, CKPT2 was to be forwarded. *Forwarding* is the process of creating a new copy of the checkpoint, writing all the current checkpoint data to the new checkpoint, updating all internal pointers to the new checkpoint, and suspending the use of the old data set.

This process is coordinated with all members of the MAS. JES2 ensures that all MAS members can access the new checkpoint data set before completing the process. Any problem results in an error message, and this message being redisplayed. In this case, there was a NEWCKPT2 data set specified. Before JES2 allocates the new data set, it confirms that this is the data set to which you want to forward. At this point, you can change the target for the forwarding.

A **CKPTDEF** command with NEWCKPT2 (the file name from the HASP273 message) can be used as a response to change the target of the forwarding. To modify the checkpoint definition, use the **\$T CKPTDEF,NEWCKPT2=(...)** command to allocate the data set that is specified by the command. Upon activation of the **NEWCKPT2** command, the data set is allocated by JES2.

```

r 5,2
IEE600I REPLY TO 05 IS;2
*$HASP273 JES2 CKPT2 DATA SET WILL BE ASSIGNED TO NEWCKPT2
JES2#A.RR2.PO.CKPT2NEW ON SYSA211
VALID RESPONSES ARE:
'CONT' - PROCEED WITH ASSIGNMENT
'CANCEL' - EXIT FROM RECONFIGURATION
CKPTDEF (NO OPERANDS) - DISPLAY MODIFIABLE
                        SPECIFICATIONS
CKPTDEF (WITH OPERANDS) - ALTER MODIFIABLE
                        SPECIFICATIONS
*06 $HASP272 ENTER RESPONSE (ISSUE D R,MSG=$HASP273 FOR RELATED MSG)

```

Figure 3-14 Forward CKPT2 to NEWCKPT2

Even though the data set was specified as the **NEWCKPT2** command, the data set does not exist on the volume JES2, and JES2 issues an error, as shown in Figure 3-15.

```
r 6,cont
IEE600I REPLY TO 06 IS;CONT
$HASP414 MEMBER IBM1 -- OBTAIN FAILED FOR NEWCKPT1 JES2#A.RR2.PO.CKPT2NEW ON
SYA211 WITH CC 8
*$HASP278 UNABLE TO LOCATE OR UNABLE TO USE NEWCKPT1
JES2#A.RR2.PO.CKPT2NEW ON SYA211
DOES NOT EXIST OR IS NOT USABLE - REFER TO PREVIOUS
MESSAGE(S)
VALID RESPONSES ARE:
'CANCEL' - EXIT FROM RECONFIGURATION
'CREATE' - CREATE DATA SET
CKPTDEF (NO OPERANDS) - DISPLAY MODIFIABLE
SPECIFICATIONS
CKPTDEF (WITH OPERANDS) - ALTER MODIFIABLE
SPECIFICATIONS
*07 $HASP272 ENTER RESPONSE (ISSUE D R,MSG=$HASP278 FOR RELATED MSG)
```

Figure 3-15 Failed allocation of **NEWCKPT2**

Before JES2 creates the data set, it again confirms that it is using the correct data set specification. If a response of CREATE is given, JES2 allocates a new data set with the correct amount of space for the checkpoint. (This process works for DASD specifications only.) This process is also useful if you need to place a checkpoint in an unexpected location.

However, it is recommended that sufficiently large NEWCKPT1 and NEWCKPT2 data sets exist so that the space on the volume is not accidentally used by some other data set. At this point, you can also change the target data set specification. If you do, JES2 confirms the new specifications before allocating a new data set.

In general, if you respond to any reconfiguration message with a CKPTDEF operand, you get another WTO/WTOR to confirm the new values you have specified, as shown in Figure 3-16 on page 52. One thing to note, the \$HASP414 message is not a highlighted message and scrolls off the screen if there is a large amount of message traffic on this console, which should be considered before issuing the **\$T CKPTDEF** command.

```

r 07,create
IEE600I REPLY TO 07 IS;CREATE
$HASP280 JES2 CKPT2 DATA SET (JES2#A.RR2.P0.CKPT2NEW ON SYA211) IS NOW IN USE
IEE400I THESE MESSAGES CANCELLED - 08.
*$HASP256 FUTURE AUTOMATIC FORWARDING OF CKPT2 IS SUSPENDED UNTIL
NEWCKPT2 IS RESPECIFIED.
ISSUE $T CKPTDEF,NEWCKPT2=(...) TO RESPECIFY
$HASP255 JES2 CHECKPOINT RECONFIGURATION COMPLETE
$dckptdef
$HASP829 CKPTDEF
$HASP829 CKPTDEF CKPT1=(DSNAME=SYS2.JESCKPT1,VOLSER=CKPTPK,
$HASP829           INUSE=YES,VOLATILE=NO),
$HASP829           CKPT2=(DSNAME=JES2#A.RR2.P0.CKPT2NEW,VOLSER=SYA211,
$HASP829           INUSE=YES,VOLATILE=NO),NEWCKPT1=(DSNAME=,
$HASP829           VOLSER=),NEWCKPT2=(DSNAME=JES2#A.RR2.P0.CKPT2,
$HASP829           VOLSER=SYA210),MODE=DUAL,DUPLEX=ON,LOGSIZE=4,

```

Figure 3-16 Successful allocation of CKPT2NEW

Checkpoint tuning

The goal in checkpoint tuning is to reduce the MAS penalty. Because processes need to access the checkpoint to perform certain functions, a latency is introduced that is proportional to the checkpoint cycle time (how long it takes a member to acquire access to the checkpoint, hold the checkpoint, release access, and then re-acquire access).

You can reduce the MAS penalty either by reducing cycle time or by increasing parallelism. By increasing parallelism, you increase the amount of work done after the checkpoint is obtained and thus lessening the impact of the checkpoint latency. But there is a cost to pushing cycle time too low. An overhead is associated in acquiring and releasing the checkpoint. This costs in both CPU and other resources and in an increase in the ratio of non-useful time to productive time in the checkpoint process. As machines and devices get faster, there are issues where an old hold and dormancy value can throw things out of balance.

The JES2 checkpoint is a serially shared resource. Essentially it is time sliced among all the active MAS members. The sharing is controlled by the MASDEF HOLD and DORMANCY values. HOLD controls how long a member will hold the checkpoint once it has read all the information in it. After the hold time expires and the checkpoint activity has subsided, the checkpoint is released.

Dormancy controls when a member will try to get the checkpoint back. Dormancy values include min and max dormancy. *Min dormancy* is the minimum amount of time a member must wait before attempting to re-acquire the checkpoint. No attempt to obtain the checkpoint will be made until that interval expires. After the min dormancy expires, a member attempts to access the checkpoint *if* it believes that access is needed. If there are no processes that require the checkpoint, the member does not attempt to re-acquire the checkpoint until max dormancy has expired.

In reality those values for HOLD and DORMANCY are depended from the kind of workload that you have on that particular system and how many systems participating the JES2 MAS. Also if you plan move workload between systems in the sysplex the settings have to be adjusted manually. So its difficult to determine the best values for your environment.

Table 3-3 lists the recommendations for both values, depending on the installation.

Table 3-3 HOLD and DORMANCY starting values

System workload	Single system	Two members	Three members	Four members	Five or more members
BATCH, NJE, RJE, TSO, PRINT	HOLD=60000 DORM=(5 ,500)	HOLD=50 DORM=(50 ,500)	HOLD=40 DORM=(80 ,500)	HOLD=30 DORM=(90 ,500)	HOLD=20 DORM=(100 ,500)
Heavy SSI use (Limit to as few members as possible)	HOLD=60000 DORM=(5 ,500)	HOLD=80 DORM=(20 ,500)	HOLD=80 DORM=(20 ,500)	HOLD=80 DORM=(20 ,500)	HOLD=80 DORM=(20 ,500)
Little JES2 activity	HOLD=60000 DORM=(5 ,500)	HOLD=30 DORM=(80 ,500)	HOLD=20 DORM=(100 ,500)	HOLD=20 DORM=(100 ,500)	HOLD=20 DORM=(100 ,500)

Another and better way to avoid manual tuning of JES2 checkpoint is to let JES2 decide the settings based on the workload on that particular system. To use that function, add the CYCLE MGMT parameter from the MASDEF initialization statement (as shown in Example 3-10). Another way is the usage of the **\$T MASDEF, CYCLEMGMT=AUTO** system command.

Example 3-10 Sample MASDEF for CYCLE MGMT

```

-$DMASDEF
$HASP843 MASDEF
$HASP843 MASDEF OWNMEMB=SC80,AUTOEMEM=ON,CKPTLOCK=ACTION,
$HASP843          COLDTIME=(2007.274,21:41:14),COLDVRSN=z/OS 1.9,
$HASP843          ENFSCOPE=SYSPLEX,DORMANCY=(0,300),HOLD=80,
$HASP843          LOCKOUT=1000,RESTART=YES,SHARED=NOCHECK,
$HASP843          SYNCCTL=120,WARMTIME=(2017.229,01:59:01),
$HASP843          XCFGPNM=WTSCPLX8,QREBUILD=0,CYCLEMGMT=AUTO,
$HASP843          ESUBSYS=HASP

```

After CYCLE MGMT is set to AUTO, you can no longer set HOLD and DORMANCY manually, as shown in Figure 3-17.

```

$TMASDEF,HOLD=50
$HASP003 RC=(168),T
$HASP003 RC=(168),T MASDEF - HOLD and DORMANCY cannot be
$HASP003           changed when CYCLEMGMT=AUTO

```

Figure 3-17 Manual set of HOLD

Checkpoint altering

During the time the workload increases, what used to seem big suddenly seems small. The same is true of the checkpoint data sets. It might be a **\$ACTIVATE** command or an increase in the number of spool volumes in the system, or just installing a new release. Someday, you will discover your checkpoints are too small.

The traditional way to increase the checkpoint size is to use the reconfiguration process shown in Example 3-9 on page 49 to move the checkpoint to a new place (either data set or CF) that is larger than the current specification. This works, but what if you don't want to move the checkpoint, just make it bigger without moving it? There are some creative ways that can be used that do not require the checkpoint to be moved and involve little to no use of the reconfiguration process.

The method you use will depend on your availability requirements and where you have placed your checkpoint. These procedures involve running with one checkpoint for some period of

time. This exposes you to a loss of all CKPT data if there is a major crash at this time. Consider doing this at a time when the system is fairly quiet usually outside prime times of your business.

Checkpoint located in the Coupling Facility

Increasing the size of a checkpoint on a CF is the easiest and least error prone to do. Most of the work is done by MVS. The trick is to do this when JES2 is not looking (not connected to the structure). First, update the CFRM policy to increase the size of the checkpoint structure. Do not forget to activate the new policy (common error). When you activate the policy, at least one policy change shows as pending. If you use the **DISPLAY XCF,STRUCTURE,STRNAME=ckpt_structure** command, it indicates a POLICY CHANGE PENDING - CHANGE in the output.

Next you have to get JES2 to disconnect from the structure either by issuing a JES2 ABEND (**\$PJES2,ABEND**) or by using the reconfiguration dialog to suspend the use of the checkpoint structure. Either option might impact the performance of the system. Ensure that you pick a time when this impact is not going to be a problem. Rebuild the structure using the **SETEXCF START,REBUILD,STRNAME=ckpt_structure** command, which allows MVS to update the size of the structure. At this point, a **DISPLAY XCF** of the structure should not show any pending changes. Reconnect to the structure by either hot starting JES2 or using the reconfiguration dialog to resume using the checkpoint data set. At this point, a **\$D CKPTSPACE** command reflects the larger checkpoint.

Checkpoint on DASD

In older releases of JES2, the process to reconfigure checkpoint data sets included reviewing complex dialog panels, even for simple changes. The size of a checkpoint data set cannot be altered without deleting it. The new fast path checkpoint reconfiguration process and support for dynamic changing of checkpoint sizes, including ALTER processing on a coupling facility (CF), and extending a DASD data set into adjacent free space.

To validate the current checkpoint size, the **\$DCKPTSPACE** shown in Example 3-11 now includes several tracks for DASD data sets or current and maximum size of CF structure in 1 K blocks. shows the new CKPTSPACE output, including space information.

Example 3-11 Output of \$DCKPTSPACE

```
$DCKPTSPACE
$HASP852 CKPTSPACE
$HASP852 CKPTSPACE  BERTNUM=6100,BERTFREE=5655,BERTWARN=80,
$HASP852          CKPT1=(CAPACITY=888,UNUSED=192,TRACKS=75),
$HASP852          CKPT2=(CAPACITY=888,UNUSED=192,TRACKS=75)
```

The **\$T CKPTDEF** command was updated to include DSNAME, VOLSER, and STRNAME to define the checkpoint data set that is resized and SIZE and SPACE attributes to define the new checkpoint size.

To alter the checkpoint size, you can use new keywords SPACE and SIZE for DASD and CF structures. The checkpoint resize function is available only when JES2 level z22 is run. The following keyword options are available:

- ▶ **SPACE=(TRK|CYL|MAX,nnnn):**
 - TRK: Define the space allocation is specific in tracks
 - CYL: Define the space allocation is specific in cylinder
 - MAX: Specify to use the largest CKPT or all free space available, whichever is smaller
 - *nnnn*: The size of the CKPT data set, in CYLs or TRKs

- ▶ SIZE=*nnnn* | MAX:
 - *nnnn*: Size of the CKPT CF structure, in 1K blocks
 - MAX: Largest CKPT or all the policy limit, whichever is smaller

Example 3-12 alters CKPT2 from an initial allocation size of 75 TRKs to 90 TRKs.

Example 3-12 Successful altering of CKPT2

```
$DCCKPTSPACE
$HASP852 CKPTSPACE 638
$HASP852 CKPTSPACE BERTNUM=6100,BERTFREE=5655,BERTWARN=80,
$HASP852           CKPT1=(CAPACITY=888,UNUSED=192,TRACKS=75),
$HASP852           CKPT2=(CAPACITY=888,UNUSED=192,TRACKS=75)
$TCKPTDEF,CKPT2=SPACE=(TRK,90)
$HASP829 CKPTDEF 640
$HASP829 CKPTDEF CKPT1=(DSNAME=SYS1.HASPCKPT,VOLSER=BH8SP1,
$HASP829           INUSE=YES,VOLATILE=NO),
$HASP829           CKPT2=(DSNAME=SYS1.HASPCKP2,VOLSER=BH8ST1,
$HASP829           INUSE=YES,VOLATILE=NO),NEWCKPT1=(DSNAME=,
$HASP829           VOLSER=),NEWCKPT2=(DSNAME=,VOLSER=),
$HASP829           MODE=DUPLEX,DUPLEX=ON,LOGSIZE=1,
$HASP829           VERSIONS=(STATUS=ACTIVE,NUMBER=50,WARN=80,
$HASP829           MAXFAIL=0,NUMFAIL=0,VERSFREE=50,MAXUSED=1),
$HASP829           RECONFIG=NO,VOLATILE=(ONECKPT=WTOR,
$HASP829           ALLCKPT=WTOR),OPVERIFY=YES
$HASP740 Volume BH8ST1 Data set SYS1.HASPCKP2 Extend successful.
```

If the physical CKPT data set has no secondary extends available the altering of CKPT fails and displays the \$HASP745 error message as shown in Example 3-13.

Example 3-13 Failure of altering CKPT2

```
$HASP745 Volume BH8ST1 Data set SYS1.HASPCKP2 Extend unsuccessful.
Error code = 60, Insufficient space.
```

3.5.4 JES2 configuration

JES2 includes the following possible configurations:

- ▶ Single-system configuration

A JES2 configuration can contain as few as one processor (one MVS and JES2 system) or as many as 32 processors linked together. A single processor is referred to as a *single-system configuration*. Such a system is suitable for an installation that has a relatively small work load, or possibly an installation that requires a single processor to be isolated from the remainder of the data processing complex (for example, to maintain a high degree of security).

- ▶ Multiple-system configuration (MAS)

Many installations take advantage of JES2's ability to link processors together to form a multiple-processor complex, which is generally referred to as a *MAS configuration*. A multi-access spool configuration consists of two or more JES2 (MAS) processors at the same physical location, all sharing the same spool and checkpoint data sets. There is no direct connection between the processors; the shared direct access data sets provide the communication link. Each JES2 processor can read jobs from local and remote card readers, select jobs for processing, print and punch results on local and remote output

devices, and communicate with the operator. Each JES2 processor in a multiple processor complex operates independently of the other JES2 processors in the configuration.

The JES2 processors share a common job queue and a common output queue, which reside on the checkpoint data sets. These common queues enable each JES2 processor to share in processing the installation's workload; jobs can run on whatever processor is available and print or punch output on whatever processor has an available device with the proper requirements. Users can also specifically request jobs to run on a particular processor and output to print or punch on a specific device. If one processor in the configuration fails, the others can continue processing by selecting work from the shared queues and optionally take over for the failed processor. Only work in process on the failed processor is interrupted; the other JES2 systems continue processing.

- ▶ **Running multiple copies of JES2 (Poly-JES)**

MVS allows more than one JES2 subsystem to operate concurrently, if one subsystem is designated as the primary subsystem and all others are defined as secondary subsystems. A secondary JES2 does not have the same capabilities as the primary JES2, and some restrictions apply to its use. Most notably, TSO/E users can only access the primary subsystem. The restrictions are necessary to maintain the isolation from the MVS-JES2 production system, but the convenience for testing is a valuable function. Operation of multiple copies of JES2 is referred to as poly-JES. Secondary JES2s can be useful in testing a new release or installation-written exit routines. This isolation prevents potential disruption to the primary JES2 and base control program necessary for normal installation production work.

- ▶ **JES2 remote job entry (RJE)**

An RJE workstation is a workstation that is connected to a member by means of data transmission facilities. The workstation can be a single I/O device or group of I/O devices or can include a processor, such as a IBM Z® machine. Generally, RJE workstations include a programmable workstation (such as a personal computer) connected to the MVS system through a telecommunication link. Such a link uses synchronous data link control (SDLC) or binary synchronous communication (BSC) protocols for communicating between JES2 and remote devices. The remote device is either a system network architecture (SNA) remote, which uses SDLC, or a BSC remote, which uses BSC.

- ▶ **JES2 network job entry (NJE)**

The JES2 network job entry facility is similar to remote job entry (RJE) in that both provide extensions to a computer system. In its simplest terms, NJE is “networking” between systems that interact as peers, whereas RJE is networking between JES2 and workstations. The main difference between them is one of overall computing power and processor location. Remember, RJE is an extension of a single computer system (that is, either a single-processor or multi-access spool complex) that allows jobs to be submitted from, and output routed back to, sites that are remote to the location of that system. NJE provides a capability to link many such single-processor systems or multi-access spool complexes into a processing network. Each system can be located on the same physical processor, side-by-side in a single room, or across the world in a network of thousands of nodes. The important difference is that a processor and its local and remote devices make up a node. Two or more attached nodes make up an NJE network.

JES2 customization

JES2 is designed to be tailored to meet an installation's particular processing needs. You can address basic customization needs when you create the JES2 initialization data set. If you find this control over JES2 processing to be insufficient, JES2 also provides exits and table pairs to change many JES2 functions or processes. A general discussion of each is provided later in this chapter. Refer to *z/OS JES2 Installation Exits*, SA22-7534, for a complete description of each IBM-defined exit.

If you need to modify JES2 processing beyond the capability provided by initialization statements, and elect to do so through installation-written code, such code should be isolated from the IBM-shipped source code. Changes to JES2 processing that are implemented through direct source code modification are error prone, counter-productive during migration to future releases, and can prove to be time consuming when debugging, diagnosing, and applying IBM-written fixes to code, such as program temporary fixes (PTFs) and authorized program analysis report (APARs) fixes. Further, alteration of JES2 processing in this manner complicates IBM service assistance. Therefore, JES2 provides several means of allowing you to tailor its processing without direct source code modification.

The HASPARM member

Because every installation that uses JES2 to manage its work input and output is unique, so too are the requirements each installation has on JES2. To meet these needs, JES2 is highly tailorabile, and with minimal effort, a system programmer can customize most JES2 functions by changing and using the JES2 initialization data set that is provided with the product in the HASIPARM member of the SYS1.SHASAMP data set. Although this data set will not run as shipped without some installation additions, it is a valuable model that can save hours of system programmer input time. See Example 3-14 for an example.

Example 3-14 A HASPPARM member example

```
//JES2      PROC
//IEFPROC  EXEC PGM=HASJES20,PARM=(NOREQ),TIME=1440,DPRTY=(15,14)
//HASPLIST  DD DDNAME=IEFRDER
//HASPPARM  DD DSN=SYS1.PARMLIB(J2DFAULT),DISP=SHR
//PROC00    DD DSN=SYS1.PROCLIB,DISP=SHR
//*        DD DSN=CPAC.ZOSR1D.PROCLIB,DISP=SHR
//        DD DSN=SYS1.IBM.PROCLIB,DISP=SHR
```

With a set of approximately 70 initialization statements, you can control all JES2 functions. The JES2 initialization data set provides all the specifications that are required to define output devices (printers and punches), job classes, the JES2 spool environment, the checkpoint data sets, the trace facility, and virtually every other JES2 facility and function.

Tip: It's useful to separate JES2 HASPARM statements into at least two members—one for all parameters that are valid for all systems and one with all local definitions. You might also consider separating NJE and printer definitions into a separate data set member.

Figure 3-18 illustrates the HASPPARM member using the INCLUDE statement.

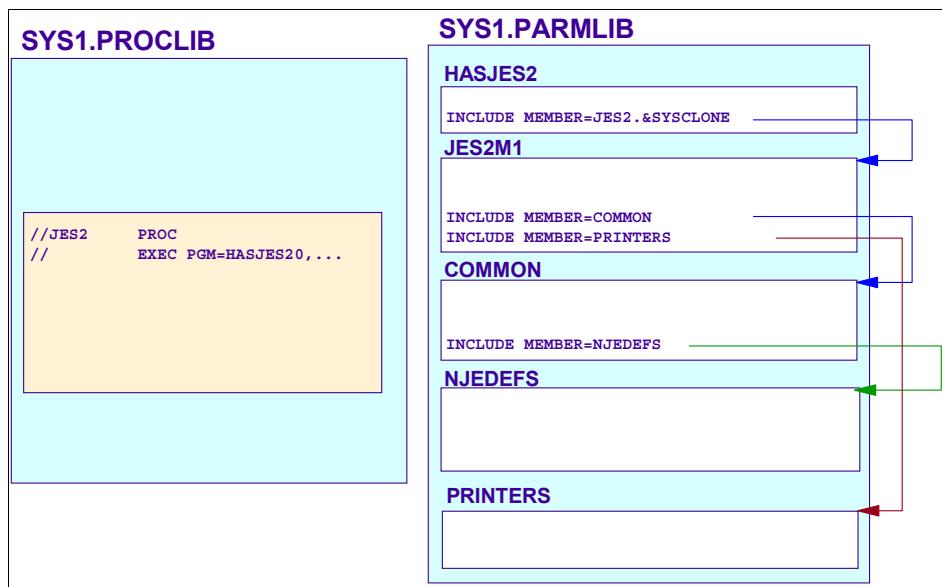


Figure 3-18 The HASPPARM member using the INCLUDE statement

JES2 initialization statements

Each initialization statement groups initialization parameters that define a function. The use of most JES2 initialization statements is optional. That is, you need not define them unless you need to implement or tailor a particular function. Further, many of the parameters provide default specifications that allow your installation to perform basic JES2 processing with no explicit definition on your part. JES2 requires only a minimal set of initialization statements (and/or parameters) that you define when first installing JES2.

The JES2 initialization data set provides all the specifications that are required to define:

- ▶ Output devices (printers and punches)
- ▶ Job classes
- ▶ Job groups
- ▶ The JES2 spool environment
- ▶ The checkpoint data sets
- ▶ The trace facility
- ▶ Every other JES2 facility and function

In the IBM-provided sample data set, SYS1.SHASSAMP, the HASI* members are samples you can tailor to meet your installation's needs.

JES2 initialization statements give the system programmer a single point of control to define an installation's policies regarding the degree of control users have over their jobs. For example, consider the confusion an installation might experience if users defined their own job classes and priorities. Likely, all users would define all their jobs as top priority, resulting in no effective priority classifications at all.

JES2 provides an assortment of commands you can use to dynamically alter JES2 customization whenever your processing needs change. Table 3-4 identifies the available JES2 initialization statements.

Table 3-4 JES2 initialization statements

Initialization statement	Function
APPL(avvvvvv)	Defines an SNA NJE application to JES2.
BADTRACK	Specifies an address or range of addresses of defective spool volume tracks JES2 is not to use.
BUFDEF	Defines the local JES2 buffers to be created.
CKPTDEF	Defines the JES2 checkpoint data set or data sets and the checkpointing mode.
COMPACT	Defines a compaction table for use in remote terminal communications.
CONDEF	Defines the JES2 console communication environment.
CONNECT	Specifies a static connection between the nodes identified.
DEBUG	Specifies whether debugging information is to be gathered by JES2 during its operation for use in testing.
DESTDEF	Defines how JES2 processing interprets and displays both job and SYSOUT destinations.
DESTID(jxxxxxxxx)	Defines a destination name (mostly for user use, as on a JCL statement) for a remote terminal, another NJE node, or a local device.
D MODULE(jxxxxxxxx)	Displays diagnostic information for specified JES2 assembly modules and installation exit assembly modules.
ESTBYTE	Specifies, in thousands of bytes, the default estimated output (SYSOUT) for a job at which the BYTES EXCEEDED message is issued and the subsequent action taken.
ESTIME	Specifies, in minutes, the default elapsed estimated execution time for a job, the interval at which the TIME EXCEEDED message is issued and it is determined whether the elapsed time job monitor feature is supported.
ESTLNCT	Specifies, in thousands of lines, the default estimated print line output for a job, the interval at which the LINES EXCEEDED message is issued and the subsequent action is taken.
ESTPAGE	Specifies the default estimated page output (in logical pages) for a job, the interval at which the PAGES EXCEEDED message is issued and the subsequent action is taken.
EXIT(nn)	Associates the exit points defined in JES2 with installation exit routines.
FSS(accccccc)	Specifies the functional subsystem for printers that are supported by an FSS (for example PSF).
GRPDEF	Defines the characteristics that are assigned to jobs groups that enter the JES2 member.
INCLUDE	Allows new initialization data sets to be processed.
INIT(nn)	Specifies the characteristics of a JES2 logical initiator.
INITDEF	Specifies the number of JES2 logical initiators to be defined.

Initialization statement	Function
INPUTDEF	Controls how JES2 input processing is performed for JES3 JECL.
INTRDR	Specifies the characteristics of all JES2 internal readers.
JECLDEF	Controls how JES2 input processing handles various JES2 and JES3 JECL statements.
JOBCLASS(v)	Specifies the characteristics associated with job classes, started tasks, and time sharing users.
JOBDEF	Specifies the characteristics that are assigned to jobs that enter the JES2 member.
JOBPTY(n)	Specifies the relationship between job scheduling priorities and job execution time.
LINE(nn nn)	Specifies the characteristics of one teleprocessing line or logical line (for SNA) to be used during remote job entry or network job entry.
L(nn nn).JT(m)	Specifies the characteristics for a job transmitter on an NJE line.
L(nn nn).ST(m)	Specifies the characteristics for a SYSOUT transmitter on a line defined for network job entry.
LOADMOD(jxxxxxxxx)	Specifies the name of a load module of installation exit routines to be loaded.
LOGON(n)	Identifies JES2 as an application program to VTAM.
MASDEF	Defines the JES2 multi-access spool configuration.
MEMBER(n)	Defines the members of a JES2 multi-access spool configuration.
NAME	Specifies the module or control section to be modified through subsequent VER and REP initialization statements.
NETACCT	Specifies a network account number and an associated local account number.
NJEDEF	Defines the network job entry characteristics of this JES2 node.
NODE(nn nn)	Specifies the characteristics of the node to be defined.
OFF(n).JR	Specifies the characteristics of the offload job receiver associated with an individual offload device.
OFF(n).JT	Specifies the characteristics of the offload job transmitter associated with an individual offload device.
OFF(n).SR	Specifies the characteristics of the offload SYSOUT receiver associated with an individual offload device.
OFF(n).ST	Specifies the characteristics of the offload SYSOUT transmitter associated with an individual offload device.
OFFLOAD(n)	Specifies the characteristics of the logical offload device.
OPTSDEF	Defines the options that are currently in effect.
OUTCLASS(v)	Specifies the SYSOUT class characteristics for one or all output classes.
OUTDEF	Defines the job output characteristics of the JES2 member.

Initialization statement	Function
OUTPRTY(n)	Defines the association between the job output scheduling priorities and the quantity (records or pages) of output.
PCEDEF	Specifies the definition for various JES2 processes.
PRINTDEF	Defines the JES2 print environment.
PROCLIB	Ensures that data sets specified can be allocated.
PRT(nn nn)	Specifies the characteristics of a local printer.
PUNCHDEF	Defines the JES2 punch environment.
PUN(nn)	Specifies the characteristics of a local card punch.
R(nn nn).PR(m)	Specifies the characteristics of a remote printer.
R(nn nn).PU(m)	Specifies the characteristics of a remote punch.
R(nn nn).RD(m)	Specifies the characteristics of a remote card reader.
RDR(nn)	Specifies the characteristics of a local card reader.
RECOVOPTS(type)	Specifies the error rate below which the operator will not be involved in the recovery process.
REDIRECT(vvvvvvvv)	Specifies where JES2 directs the response to certain display commands entered at a console.
REP	Specifies replacement patches for JES2 modules during initialization.
REQJOBID	Describes attributes to be assigned to Request JOBID address spaces.
RMT(nn nn)	Specifies the characteristics of a BSC or SNA remote terminal.
SMFDEF	Specifies the system management facilities (SMF) buffers to JES2.
SPOOLDEF	Defines the JES2 spool environment.
SSI(nn)	Specifies the characteristics associated with individual subsystem interface definitions.
SUBTDEF	Specifies the number of general purpose subtasks you wish JES2 to attach during initialization.
TPDEF	Defines the JES2 teleprocessing environment.
TRACE(n)	Specifies whether a specific trace ID or IDs is to be started.
TRACEDEF	Defines the JES2 trace environment.
VER	Specifies verification of replacement patches for JES2 modules during initialization.

For more information, refer to *z/OS JES2 Initialization and Tuning Reference*, SA22-7533 and *z/OS JES2 Initialization and Tuning Guide*, SA22-7532.

When you are first getting started, you need not define or even be concerned about some of the more sophisticated processing environments such as a multi-access spool complex, nodes, or remote workstations. You are simply building a base on which your installation can grow. There is no need to be overwhelmed with the potential complexity of your JES2 system.

As your installation grows, you will likely use more and more of JES2's available functions. The sample data set shipped in SYS1.PARMLIB contains all default values and requires only the addition of installation-defined devices and installation-specific values. It contains all the JES2 initialization statements and the defaults for all parameters for which they are provided.

The JES2 initialization checker

The initialization data set checker allows installations to verify their initialization data sets without having to start a JES2 subsystem. The process can detect syntax errors in initialization statements and problems with settings that might prevent JES2 from starting. The checks can verify that the statements are valid for a cold start or a warm start.

If verifying parameters on a warm start, you must run the checker within a SYSPLEX with an active member of the MAS. The checker uses XCF messaging to extract information from the active MAS member to verify that the parameters are valid on a warm start and to perform more analysis of resource usage.

Important: Because the checker was added in z/OS V2R3 to obtain checkpoint (runtime) data, the checker and the other JESPLEX member must both be running z/OS V2R3 or later.

The output of the initialization data set checker is placed in the HASPLIST DD. The requirements for this DD are the same as the HASPLIST DD for the normal JES2 PROC. The sections in the HASPLIST data set that are produced by the checker are:

- ▶ Pre-initialization data set checks and messages. These are related to exit 0 processing and errors in loading the various JES2 load modules.
- ▶ Initialization statements with any syntax errors.
- ▶ Post initialization statement validation. This includes validating the initialization statements that are read against a running JES2 subsystem (if one in the correct MAS is available).
- ▶ A resource usage analysis using values from the running MAS (if data is available).
- ▶ Summary of problems found.

Because it is running the same code as a normal JES2 subsystem, you must indicate that you want to perform an initialization check and not a normal JES2 start. This can be done a number of ways (any of which can be used to run the initialization data set checker).

- ▶ Specify CHECK on the PARM= when starting JES2 (with or without other parameter values).
- ▶ Run JES2 as a batch job.
- ▶ Specify PGM=HASJESCK on the EXEC statement.

The next examples explain the use of the HASJESCK program.

Figure 3-19 shows a sample JCL for the JES2 initialization checker invoked by a BATCH job.

```
//LUTZCHK JOB , 'JES2 CHECK', NOTIFY=LUTZ,REGIONX=(8M,128M),  
//           CLASS=A,MSGCLASS=X,MSGLEVEL=(1,1),TIME=1440  
/*JOBPARM S=SC80  
//JES2CHK EXEC PGM=HASJESCK  
//HASPLIST DD SYSOUT=*  
//HASPPARM DD DISP=SHR,DSN=SYS1.PARMLIB(J2DFAULT)  
//
```

Figure 3-19 HASJESCK JCL

The program itself uses the following data sets:

HASPLIST

The produced report is written to that data set.

HASPPARM

The input parmlib that contains the JES2 startup configuration member.

After the initialization statements are processed, the processing attempts to access the runtime data. If available, the normal verification of initialization settings against the runtime data.

After normal initialization processing completes, a number of reports are generated. The first is the data set read report, which lists the initialization data sets that were read and the number of records that are processed from each data set as shown in Figure 3-20.

Initialization data sets read:

Data set name	VOLSER	Unit	Records
SYS1.PARMLIB(J2DFAULT)	BH8CAT		206

Figure 3-20 JES2 check data sets read

If runtime data was obtained, the resource usage information is summarized, as shown in Figure 3-21. The information contained in this summary is based on the current running system at the time that the initialization data set checker was run. The details of which system supplied the data is provided in the summary report.

Resource usage information:										
JQEs	TYPE	ACTIVE	COMPLETE	JOEs	TYPE	COUNT	TGs	TYPE	COUNT	INUSE
	BATCH	2	232		WORK	3,132		DEFINED	30,030	6,620
	STC	267	171		CHAR	2		ACTIVE	30,030	6,920
	TSU	2	13		INDEX	0		FREE	23,110	
	JOBGROUP	0	0		FREE	1,866				
	INTERNAL	1								
	FREE	4,312								
BERTs	TYPE	COUNT	CB COUNT	ZJCs	TYPE	COUNT	Jobnum	Description	Value	
	INTERNAL	34	4		JOBGROUP	0		Low Range	1	
	JQE	298	286		DEP JOB	0		High Range	9,999	
	CAT	114	38		DEPENDNT	0		In Use	688	
	WSCQ	0	0		FREE	1,000				
	DJBQ	0	0							
	JOE	0	0							
	DAS	0	0							
	GRP	0	0							
	FREE	5,654								

Figure 3-21 Resource usage information summary

The next section gives recommendations for minimum settings for a number of resources, as shown in Figure 3-22. This value is based on looking at the current usage ration of resources per job and projecting what would be needed if the job limit is reached.

Recommendations:						
	Current Limit	Current Usage	Percent Usage	Usage per JQE/JOE	Max with max JQE/JOE	Recommended min limit
JQEs	5,000	688	13.76			5,000
Job Numbers	9,999	688	6.88	1.00	5,000	5,000
JOEs	5,000	3,134	62.68	7.52	37,600	40,000
Active TGs	30,030	6,920	23.04	9.62	48,100	50,000
BERTs	6,100	446	7.31			3,000
JQE BERTs		298		0.43	2,150	
JOE BERTs		0		0.00	0	

Figure 3-22 JES2 check recommendations

The recommended minimum level is displayed and is based on the current use and projection to the maximum JQEs. This recommendation gives a point in time estimate if there are enough resources for the maximum number of jobs that are defined. No check is made to see whether the recommendation is below the current limit (since this is a minimum value).

The summary report returns information about the JES2 instance that was verified, as shown in Figure 3-23. It includes the JES2 member name, node name, and XCF group name that is derived from the initialization data sets. The MVS name and SYSPLEX name are from the system where the checker was run. If checkpoint (runtime) data is obtained, that information is indicated here. If it was not obtained but you want the runtime data reports, you must run the checker in a SYSPLEX where at least one member of the JESPLEX is active. The version of the checker is also displayed. The version of the checker does not need to match the current z/OS level or the level of the active JESPLEX member. However, there must be a JES2 subsystem active on the system that runs the checker.

Summary report:

Member name	SC80
NJE node name	WTSCPLX8
JESXCF group name	WTSCPLX8
MVS system name	SC80
MVS SYSPLEX name	WTSCPLX8
Checkpoint data	Obtained
Checker version	z/OS 2.3

Figure 3-23 JES2 check summary report

The error summary gives a summary of any errors that are found during processing (as shown in Figure 3-24).

Error Summary:

Type	Count
Warnings	3
Init statement errors	0
Validation errors	0
Read/OPEN errors	0
Configuration errors	0
Exit requested termination	0
Total error count	3

Figure 3-24 JES2 check error summary

Errors are divided into many categories:

- ▶ *Warnings*: As shown in Figure 3-25, problems encountered that do not impact JES2 initialization but should be corrected.
- ▶ *Initialization statement errors*: Initialization statements that encountered an error. These statements are ignored and might prevent JES2 from initializing correctly.
- ▶ *Validation errors*: Problems encountered validating the consistency of the initialization statement after all initialization data sets are processed. These problems might prevent JES2 from initializing properly.
- ▶ *Read/OPEN errors*: Error reading records from the initialization data sets.
- ▶ *Configuration errors*: Initialization settings are inconsistent with the running system. Initialization cannot be completed unless these are corrected.
- ▶ *Exit requested termination*: An exit has requested that initialization terminates.

When an error is encountered, the appropriate count is updated. However, some errors or warnings might result in multiple counts being updated.

DIAGNOSTIC	INFO	\$HASP442 INITIALIZATION STATEMENTS CONFLICTING WITH SAVED VALUES FOLLOW:
DIAGNOSTIC	WARNING	\$HASP496 JOBDEF JOBNUM=3000 SAVED VALUE OF 5000 WILL BE USED
DIAGNOSTIC	WARNING	\$HASP496 OUTDEF JOENUM=3000 SAVED VALUE OF 5000 WILL BE USED
DIAGNOSTIC	WARNING	\$HASP496 CKPTSPAC BERTNUM=3850 SAVED VALUE OF 6100 WILL BE USED
DIAGNOSTIC	INFO	\$HASP537 THE CURRENT CHECKPOINT USES 700 4K RECORDS

Figure 3-25 JES2 check warnings

The JES2 initialization statements support system symbol substitution. This can alter the statements themselves, including altering what data set is read. When running the checker, system symbol substitution is performed as normal. The symbols that are used are from the system on which the checker is run, which might not yield the same results as the target system whose initialization data sets are being checked.

Important: If symbol substitution is important in your environment, ensure that the checker is run on the same system as the JES2 subsystem that is being checked.

3.5.5 JES2 exits

JES2 might not satisfy all installation-specific needs at a given installation. When you modify JES2 code to accomplish specific functions, you are susceptible to the migration and maintenance implications that result from installing new versions of JES2. JES2 exits allow you to modify JES2 processing without directly affecting JES2 code. In this way, you keep your modifications independent of JES2 code, making migration to new JES2 versions easier and making maintenance less troublesome.

The following JES2 processing modifications can be made through the use of exits:

- ▶ Initialization processing

You can modify the JES2 initialization process and incorporate your own installation-defined initialization statements in the initialization process. Also, you can change JES2 control blocks prior to the end of JES2 initialization.

- ▶ Job input processing

You can modify how JES2 scans and interprets a job's JCL and JES2 control statements. Also, you can establish a job's affinity, execution node, and priority assignments before the job actually runs.

- ▶ Subsystem interface (SSI) processing

You can control how JES2 performs SSI processing in the following areas: job selection and termination, subsystem data set OPEN, RESTART, allocation, CLOSE, unallocation, end-of-task, and end-of-memory.

- ▶ JES2-to-operator communications

You can tailor how JES2 communicates with the operator and implement additional operator communications for various installation-specific conditions. Also, you can preprocess operator commands and alter, if necessary, subsequent processing.

- ▶ Spool processing

You can alter how JES2 allocates spool space for jobs.

- ▶ Output processing

You can selectively create your own unique print and punch separator pages for installation output on a job, copy, or data set basis.

- ▶ JES2-SMF processing

You can supply to SMF added information in SMF records.

- ▶ RJE processing

You can implement additional security checks to control your RJE processing and gather statistics about signons and signoffs.

JES2 provides various strategic locations, called exit points, from where an installation-written exit routine can be invoked. JES2 can have up to 256 exits, each identified by a number from 0 to 255. JES2 code includes a number of *IBM-defined exits*, which have already been strategically placed in the code. For these IBM-defined exits you need only write your own exit routines and incorporate them via the EXIT(nn) and LOAD(xxxxxx) initialization statements, where *nnn* is the exit point number and *xxxxxx* is the load module name.

If the IBM-defined exits are not sufficient, you can define your own exit points. However, exits established by you are modifications to JES2 code, and you must remember that you run a greater risk of disruption when installing a new version of JES2 code. The new JES2 code into which you have placed your exits may have significantly changed since the insertion of your exit point.

Attention: It's strongly recommended to install any custom made JES2 exit using SMP/E USERMOD to avoid conflicts with IBM shipped modifications.

The IBM-defined exits can be classified into the following categories:

- ▶ Not job-related exits: These are exits taken during functions not necessarily related to individual jobs (for example, JES2 initialization, JES2 termination, RJE, and JES2 command processing).
- ▶ Job-related exits: These exits are described in further detail in the following section.

Often the use of more than one exit is required, and sometimes a combination of JES2 and other exits (such as Systems Management Facilities (SMF) exits) must be used. Table 3-5 and Table 3-6 on page 69 list the job-related exits to help you decide which exits to choose to control certain processes or functions during the life of a job.

Many installations use input service exits to control installation standards, tailor accounting information, and provide additional controls. When choosing an exit to act in this phase, it is important to consider all sources of jobs, especially if you want to select jobs from some sources to follow standards. For more details, refer to *JES2 Installations Exits*, SC28-1793.

Table 3-5 JES2 job-related exits

Exit	Exit title	Comments and some specific uses
1	Print/punch job separator	Taken when a job's data sets are selected for printing or punching prior to the check for the standard separator page.
2	JOB statement scan	The first exit taken for a job and before the statement is processed.
3	JOB statement accounting field scan	Taken after JOB statement is processed. Normally used to replace or supplement JES2's accounting field scanning routine (HASPRSCN) but also used as a post job card exit.
4	JCL and JES2 control statement scan	Taken for each JCL and JECL statement submitted but not for PROCLIB JCL statements.
6	Converter/Interpreter Text scan	A good exit for scanning JCL because of structured text and single record for each statement (no continuation).
7	\$JCT Read/Write (JES2 environment)	Receives control when JES2 main task reads or writes the \$JCT.
8	Control Block Read/Write (user or Subtask environment)	Taken from the user address space or a JES2 subtask each time a spool resident control block (\$JCT, \$IOT, \$SWBIT, \$OCR) is read from or written to spool.
15	Output Data Set/Copy Select	Taken once for each data set where the data set's \$PDDB matches the selected job output element (\$JOE) and once for each copy of these data sets.
20	End of Job Input	Taken at the end of input processing and before \$JCT is written. This is usually a good place to make final alterations to the job before conversion.
28	SSI Job Termination	Taken at the end of job execution before the \$JCT is written to spool.
30	SSI Data Set Open/Restart	Taken for SYSIN, SYSOUT, or internal reader Open or Restart processing.

Exit	Exit title	Comments and some specific uses
31	SSI Data set Allocation	Taken for SYSIN, SYSOUT, or internal reader Allocation processing. Uses: <ul style="list-style-type: none">▶ Fail an allocation.▶ Affect how JES2 processes data set characteristics.
32	SSI Job Selection	Taken after all job selection processing is complete. Uses: <ul style="list-style-type: none">▶ Suppress job selection-related messages.▶ Perform job-related processing such as allocation of resources and I/O for installation-defined control blocks.
33	SSI Data Set Close	Taken for SYSIN, SYSOUT, or internal reader Close processing. Uses: <ul style="list-style-type: none">▶ Free resources obtained at OPEN.
34	SSI Data Set Unallocation - Early	Taken for SYSIN, SYSOUT, or internal reader early in allocate processing. Uses: <ul style="list-style-type: none">▶ Free resources obtained by Exit 30.
35	SSI End-of-Task	Taken at end of each task during job execution. Uses: <ul style="list-style-type: none">▶ Free task-related resources.
36	Pre-SAF	Taken just prior to JES2 call to SAF. Uses: <ul style="list-style-type: none">▶ Provide/change additional information to SAF.▶ Eliminate call to SAF.
37	Post-SAF	Taken just after the return from the JES call to SAF. Uses: <ul style="list-style-type: none">▶ Change the result of SAF verification.▶ Perform additional security authorization checking above what SAF provides.
40	Modifying SYSOUT	Taken during OUTPUT processing for each SYSOUT data set before JES2 gathers data sets with like attributes into a \$JOE. Uses: <ul style="list-style-type: none">▶ Change the destination of a SYSOUT data set.▶ Change the class of a SYSOUT data set to affect grouping.
44	Post Conversion (JES2 environment)	Taken after job conversion processing and before the \$JCT and \$JQE are checkpointed. Uses: <ul style="list-style-type: none">▶ Change fields in the \$JQE and \$JCT.
46	NJE Transmission	Taken for NJE header, trailer, and data set header during NJE job transmission. Uses: <ul style="list-style-type: none">▶ Remove/add/change installation-defined sections to an NJE data area before transmission.
47	NJE Reception	Taken for NJE header, trailer, and data set header during NJE job reception. Uses: <ul style="list-style-type: none">▶ Remove/change installation-defined sections that were previously added to an NJE data area.

Exit	Exit title	Comments and some specific uses
48	SYSOUT unallocation - Late	More suitable than exit 34 when modifying SYSOUT characteristics or affecting SPIN processing.
49	Job Queue Work Select	Taken whenever JES2 has located a pre-execution job for a device. Uses: <ul style="list-style-type: none"> ▶ Provide an algorithm to accept or not accept a JES2-selected job. ▶ Control WLM initiator job selection.

Table 3-6 Some SMF job-related exits

Exit	Exit Title	Comments and some specific uses
IEFUJV	SMF Job Validation	Receives control: <ul style="list-style-type: none"> ▶ Before each JCL statement is interpreted, and ▶ After all the JCL is converted, and again ▶ After all the JCL is interpreted
IEFUJI	SMF Job Initiation	Receives control before a job on the input queue is selected for initiation. Uses: <ul style="list-style-type: none"> ▶ Selectively cancel the job.
IEFUSI	SMF Step Initiation	Receives control before each job step is started (before allocation). Uses: <ul style="list-style-type: none"> ▶ Limit the user region size. Often the use of more than one exit is required, and sometimes a combination of JES2 and other exits (such as Systems Management Facilities (SMF) exits) must be used. Many installations use input service exits to control installation standards, tailor accounting information, and provide additional controls. When choosing an exit to act in this phase, it is important to consider all sources of jobs, especially if you want to select jobs from some sources to follow standards. For more details, refer to <i>JES2 Installations Exits</i> , SC28-179
IEFACTRT	SMF Job Termination	Receives control on the termination of each step or job. Uses: <ul style="list-style-type: none"> ▶ Decide whether the system is to continue the job (for step job). ▶ Decide whether SMF termination records are to be written to SMF data set.
IEFUJP	SMF Purge	Receives control when a job is ready to be purged from the system, after the job has terminated and all its SYSOUTS have been written. Uses: <ul style="list-style-type: none"> ▶ Selectively decide whether the SMF job purge record (type 26) is to be written to the SMF data set.

IEFUSI

The IEFUSI installation exit control region size and memory above the bar. Changes made by IEFUSI to the region limit and hi memory limit are used if a RACF® user profile is not used to define these values for the user. The IEFUSI exit can change the values used by the system for virtual storage available above and below the 16M line, and high memory. The IEFUSI is written in assembler language and changes are not very simple to deploy.

Another new parmlib member calls SMFLIMxx where introduced to get rid of IEFUSI for controlling region sizes. An SMFLIMxx parmlib member consists of an ordered set of rules, each starting with the word REGION to begin the rule. The keywords that follow the REGION statement are composed of filters and attributes. Figure 3-26 shows a simple example of using SMFLIMxx that assigns a MEMLIMIT of 1 TB to the TSO user Lutz.

```
REGION
JOBNAME(LUTZ) SUBSYS(TSO)
MEMLIMIT(1T)
```

Figure 3-26 Sample of SMFLIMxx

Filter keywords are used by the system to match the jobstep being initiated to the rule. If any filter does not match, the attributes are not used and processing continues with the next rule. Filter keywords allow up to eight values to be specified for each, applied in an OR fashion. For example, a REGION statement with SUBSYS(JES*,STC) would match any jobstep that started as an STC or a JES-initiated job. Each filter keyword that is used must have at least one matching string for the rule to match.

The following filter keywords are available:

JOBNAME	Matches a REGION statement to the JOBNAME specified in the JCL.
JOBCLASS	Matches a REGION statement to the JES JOBCLASS for the job.
JOBACCT	Matches the accounting information that is specified on the JCL JOB statement.
STEPNAME	Matches a REGION statement to the STEPNAME specified in the JCL.
STEPACCT	Matches the accounting information that is specified on the JCL EXEC statement.
PGMNAME	Matches the JOBSTEP program that is specified on the PGM= keyword of the EXEC statement.
USER	Matches the user ID that is associated with the job, either specified on the USER keyword on the JOB statement, or the user that submitted the job. For more information about the USER keyword, see the <i>z/OS MVS JCL User's Guide</i> , SA23-1386.
SUBSYS	Matches the subsystem that is associated with the job. This subsystem is often JES2 or JES3, but it might also be STC or some other subsystem name.
SYSNAME	Matches the name of the system. The SYSNAME keyword is handled differently than the other filter keywords, as detailed in <i>z/OS MVS Initialization and Tuning Reference</i> , SA23-1380.
SYSRESVABOVE	
SYSRESV BELOW	Use these keywords to set aside part of the private area for system-key functions and services, which prevents the user-key JOBSTEP program from obtaining all available private storage and causing system functions to fail.
REGION ABOVE	
REGION BELOW	Use these keywords to override the REGION or REGIONX specification on a job and change its requested region size. In today's systems with large amounts of real storage, it is less important to enforce a restrictive REGION size on a JOBSTEP. However, it might be useful to

	code REGIONABOVE(NOLIMIT) REGIONBELOW(NOLIMIT). To give the JOBSTEP program access to the entire below-the-bar private area without altering the default for the JOBCLASS or altering the REGION statements of many JCL jobs.
MEMLIMIT	Use this keyword to override the MEMLIMIT value that is provided as a default in SMFPRMxx or with the JCL MEMLIMIT= keyword. NOLIMIT is accepted for MEMLIMIT. Using NOLIMIT might leave your system exposed to storage shortages if the JOBSTEP program attempts to obtain and use all of the above-the-bar virtual storage in its address space.

Attention: The filter keywords are applied in an AND fashion.

After logging on the TSO Lutz user ID, Figure 3-27 shows the additional messages in the user's job log about the assigned storage controlled by SMFLIMxx.

IEF043I Actions taken by SMFLIMxx parmlib policy for LUTZ IKJACCT Step MEMLIMIT set to 1T by policy - SMFLIM00 0001
--

Figure 3-27 Output of SMFLIMxx

3.5.6 JES2 start

JES2 uses start options to determine how it will perform the current initialization. You can specify start options in either of the following ways:

- ▶ As parameters on the EXEC statement in the JES2 procedure
- ▶ As options specified at the console

If you do not specify the options on the EXEC statement, JES2 requests them from the operator by issuing the \$HASP426 (SPECIFY OPTIONS) message.

The operator then enters the options using the standard reply format as described in z/OS JES2 commands. The operator can enter options in upper or lower case; they must be separated by commas. If the operator enters conflicting options (for example, WARM or COLD), the last option specified is the one JES2 uses.

If the options are specified on the EXEC statement, JES2 suppresses the \$HASP426 (SPECIFY OPTIONS) message and completes initialization without operator intervention unless CONSOLE control statements are added to the JES2 initialization data set or an initialization statement is in error.

If you let JES2 prompt for the options, JES2 issues the following \$HASP426 message:

```
*id $HASP426 SPECIFY OPTIONS - jes option
```

Respond using the z/OS REPLY command to specify the JES2 options determined by your installation procedures, as listed in Table 3-7 on page 72.

Note: If you specify the NORREQ option, JES2 automatically begins processing after the initialization. Otherwise, you must enter the \$\$ command in response to the \$HASP400 ENTER REQUESTS message to begin JES2 processing.

Table 3-7 JES start options

Option	Explanation
FORMAT NOFMT	<p>Specifies that JES2 is to format all existing spool volumes. If you add unformatted spool volumes, JES2 automatically formats them whether or not the FORMAT option is specified. With the FORMAT option, JES2 automatically performs a cold start.</p> <p>Default: NOFMT specifies that JES2 is not to format existing spool volumes unless JES2 determines that formatting is required.</p>
COLD WARM	<p>Specifies that JES2 is to be cold started. All jobs in the system are purged, and all job data on the spool volumes are scratched.</p> <p>Default: The WARM option specifies that JES2 is to continue processing jobs from where they were stopped. If the FORMAT option was also coded, JES2 ignores the WARM specification and performs a cold start.</p>
SPOOL = VALIDATE NOVALIDATE	<p>Specifies that the track group map is validated on a JES2 all-member warm start.</p> <p>Default: The SPOOL=NOVALIDATE option specifies that the track group map is not validated when JES2 restarts.</p>
NOREQ REQ	<p>Specifies that the \$HASP400 (ENTER REQUESTS) message is suppressed and JES2 automatically begins processing when initialization is complete.</p> <p>Default: The REQ option specifies that the \$HASP400 (ENTER REQUESTS) message is written at the console. This message allows you to start JES2 processing with the \$S command.</p>
NOLIST LIST	<p>Specifies that JES2 is not to print the contents of the initialization data set or any error flags that occur during initialization. If you specify NOLIST, JES2 ignores any LIST control statements in the initialization data set. <i>z/OS JES2 Initialization and Tuning Reference</i>, SA32-0992, presents an example of an initialization data set listing produced by using the list option.</p> <p>Default: The LIST option specifies that JES2 is to print all the statements in the initialization data set and any error flags that occur during initialization. (JES2 prints these statements if a printer is defined for that purpose when JES2 is started.) The LIST option will not print any statements that follow a NOLIST control statement in the initialization data set.</p>
NOLOG LOG	<p>Specifies that JES2 is not to copy initialization statements or initialization errors to the HARDCPY console. If you specify the NOLOG option, JES2 ignores LOG control statements in the initialization data set.</p> <p>Default: The LOG option specifies that JES2 is to honor any LOG statements in the initialization data set.</p>
CKPT1 CKPT2	<p>Specifies what checkpoint data set JES2 must use for building the JES2 work queues.</p> <p>Default: If you do not specify, JES2 automatically determines which checkpoint data set to use.</p>

Option	Explanation
RECONFIG	<p>Specifies that JES2 uses the checkpoint data set definitions as specified on the CKPTDEF statement in the initialization data set. JES2 overrides any modifications to the checkpoint data set definitions previously made either by the \$T CKPTDEF command or through the use of the checkpoint reconfiguration dialog. Specifying the RECONFIG option also causes JES2 to enter the reconfiguration dialog during initialization and issues the \$HASP289 CKPT1 AND/OR CKPT2 SPECIFICATIONS ARE REQUIRED message.</p> <p>If you previously reconfigured the checkpoint configuration through the checkpoint reconfiguration dialog box, the CKPTDEF statement definition might not contain the most current checkpoint definition. Changes made through the checkpoint reconfiguration dialog box are not saved in the input stream data set.</p>
HASPPARM=ddname	<p>Specifies the name of the data definition (DD) statement that defines the data set containing the initialization statements that JES2 uses for this initialization.</p> <p>Default: The HASPPARM=HASPPARM option specifies that JES2 is initialized using the initialization statements in the data set that is defined by the HASPPARM DD statement in the JES2 procedure.</p>
CONSOLE	<p>Causes JES2 to simulate receiving a CONSOLE initialization statement after all initialization statements are processed. That is, if CONSOLE is specified, JES2 diverts to the operator console for further parameter information after the input stream data set is exhausted.</p>
MEMBER=	<p>Specifies the member of the default PARMLIB concatenation that contains the initialization statements that JES2 is to use for this initialization.</p> <p>PARMLIB_MEMBER= is a synonym for MEMBER=. MEMBER is mutually exclusive with HASPPARM=. The order of processing is as follows:</p> <ol style="list-style-type: none"> 1. MEMBER= if specified 2. HASPPARM= if specified 3. Default HASPPARM=HASPPARM if DD can be opened 4. Default MEMBER=HASjesx member of the default PARMLIB concatenation
NONE U N	<p>NONE, U, or N character specifies that JES2 uses all of the default start options. There is no difference between these options; they are equivalent. When NONE, U, or N is specified, JES2 uses the default start options, which are:</p> <ul style="list-style-type: none"> ► NOFMT ► WARM ► REQ ► LIST ► LOG
UNACT	<p>UNACT inactivates the new JES2 functions that are introduced in z/OS Version 2 Release 2. This option reverses the ACTIVATE level=z22 setting. It can be used on an all-member WARM or HOT start or a COLD start.</p> <p>Default: On a WARM or HOT start, JES2 takes no action and the checkpoint release level stays at the previous setting. On a COLD start, JES2 starts in z/OS Version 2 Release 2 (z22) mode.</p>
CHECK	<p>If specified, instructs JES2 to run the initialization data set checker and not start a JES2 subsystem. The initialization data set checker runs JES2 initialization processing producing a number of reports. For more information, see JES2 initialization data set checker.</p>

3.5.7 JES2 start options

JES2 can be either cold started or warm started, depending on the current environment. You can specify the type of start (COLD or WARM) on the z/OS **START** command or in response to the \$HASP426 SPECIFY OPTIONS message. (See Table 3-7 on page 72 for information about how to specify JES2 options.) Specifying WARM can result in a hot start or quick start, as described in the section that follows.

Cold start

If you want to specify the COLD or FORMAT initialization options (both of which result in a cold start of JES2), you must complete the IPL process on the system before you issue the MVS **START** command, unless you stopped the previous JES2 with a **\$P JES2** operator command. Implement a cold start with caution, because it results in the re-initialization of all JES2 job and output queues, which means that all job queue entries and job output elements are cleared and formatted.

If you specify the FORMAT initialization option, JES2 formats all existing spool volumes. Otherwise, JES2 formats any spool volumes it determines are unformatted. No other member in a MAS configuration can be active during a cold start, or JES2 terminates with an error message.

You invoke one of the remaining initialization processes as a result of specifying the WARM initialization option without the FORMAT option. If you specify the FORMAT option, JES2 automatically completes a cold start.

An IPL process must precede a JES2 cold start, unless JES2 was stopped with a **\$P JES2** operator command.

Warm start

During a warm-start initialization, JES2 reads through its job queues and handles each job according to its status, as follows:

- ▶ Jobs in input readers are lost and must be reentered.
- ▶ Jobs in output (print/punch) are requeued and later restarted. The checkpoint for jobs on the 3800 printer points to the end of the page being stacked at the time of the checkpoint. Jobs that were sent to the 3800 printer, but that did not reach the stacker, are reprinted from the checkpoint. If no checkpoint exists, it is reprinted from the beginning.
- ▶ Jobs in execution are either requeued for execution processing or are queued for output processing.
- ▶ All other jobs remain on their current queues.

A warm start can be completed in a multi-access spool configuration in the following ways:

- ▶ All-member warm start

An all-member warm start is performed if a warm start is specified by the operator and JES2 determines that no other members of the configuration are active or there is only one member in the configuration. All in-process work in the MAS will be recovered. After an all-member warm start, other members entering the configuration for the first time will perform a quick start.

- ▶ Single-member warm start

This type of start is performed when WARM is specified and others members of the configuration are active. The warm-starting member joins the active configuration and recovers only work in process on that member when it failed or was stopped.

► Quick start

This type of start is performed when you specified a warm start and JES2 determines that the job queue and job output table do not need to be updated. In this case, the member being started is not the first member being started in the MAS, which occurs in the following cases:

- After **\$P JES2** is issued to quiesce the member. Because all work is quiesced, there is no need to update the job queue or job output table before restarting.
- After an all-member warm start is performed and no work is waiting to be processed; therefore, the job and output queues are empty.
- When a **\$E MEM** command was entered at a processor within the MAS configuration other than the member being started.

► Hot start

This is a warm start of an abnormally terminated JES2 member without an intervening IPL. When it happens, all address spaces continue to execute as if JES2 never terminated. JES 2 validates (and rebuilds, when necessary) the job and output queues and the job queue index. Damaged or corrupted job output elements (JOEs) and job queue elements (JQEs) are placed on the rebuild queue.

3.5.8 JES2 stop

There are instances when JES2 must be stopped and restarted either by a warm or cold start. For example, redefining the number of systems in a network job environment requires a warm start. You can stop and restart JES2 in a system at any time by using operator commands, which allows you to:

- Quiesce job processing in preparation for an orderly system shutdown.
- Restart JES2 to perform an initialization with different initialization parameter specifications.

You can dynamically change JES2 initialization statements by using the **\$T** operator command for most parameters. Before stopping JES2 for the purpose of changing initialization statement parameters, refer to *z/OS JES2 Initialization and Tuning Reference*, SA22-7533.

To stop JES2, complete the following steps:

1. Issue the **\$P** command to stop all JES2 processing. System initiators, printers, punches, job transmitters, and SYSOUT transmitters will not accept any new work and will become inactive after completing their current activity. However, new jobs will be accepted through input devices. When all TSO users log off, and all JES2 started tasks, logical initiators, printers, and punches complete their current activities and become inactive, JES2 notifies you with the following message:
\$HASP099 ALL AVAILABLE FUNCTIONS COMPLETE
2. Stop all started tasks.
3. Enter the **\$P JES2** command to withdraw JES2 from the system. If any jobs are being processed or any devices are active, the **\$P JES2** command is processed as a **\$P** command and drains JES2 work from the system.

If it is not possible or reasonable to drain the JES2 member (for example, due to large numbers of lines, jobs, and remote, or if you plan to restart JES2 using a hot start), you can specify:

\$P JES2,ABEND

The ABEND parameter forces JES2 termination regardless of any JES2 or system activity. If the checkpoint resides on a Coupling Facility structure and the member is processing a write request, JES2 issues the \$HASP552 message and delays the \$P command until the checkpoint write completes.

If the **\$P JES2,ABEND** command does not successfully terminate JES2, you can also specify the FORCE parameter. The **\$P JES2,ABEND,FORCE** command results in a call to the recovery termination manager (RTM) to terminate the JES2 address space. Because the FORCE parameter can cause unpredictable results, always attempt to enter the **\$P JES2,ABEND** command first.

To withdraw JES2 from a system involved in cross-system activity, you can issue the **\$P JES2,QUICK** command. Cross-system activity occurs when a user on one JES2 subsystem requests a cross-system function from another JES2 subsystem within the same poly-JES2 environment. This option deletes the control blocks for the request submitted by the user who requested cross-system function. Before using the QUICK keyword on the **\$P JES2** command, send a message to the user asking them to end cross-system activity.

4. Issue the **HALT EOD** command. It ensures that important statistics and data records in storage are not permanently lost.

3.6 JES3

With the z/OS MVS JES3 system, resource management and workflow management are shared between MVS and its JES3 component. Generally speaking, JES3 does resource management and workflow management *before* and *after* job execution, while MVS does resource and workflow management *during* job execution.

JES3 considers job priorities, device and processor alternatives, and installation-specified preferences in preparing jobs for processing job output. The features of the JES3 design include:

- ▶ Single-system image
- ▶ Workload balancing
- ▶ Availability
- ▶ Control flexibility
- ▶ Physical planning flexibility

Figure 3-28 shows the JES3 topics discussed in this chapter.

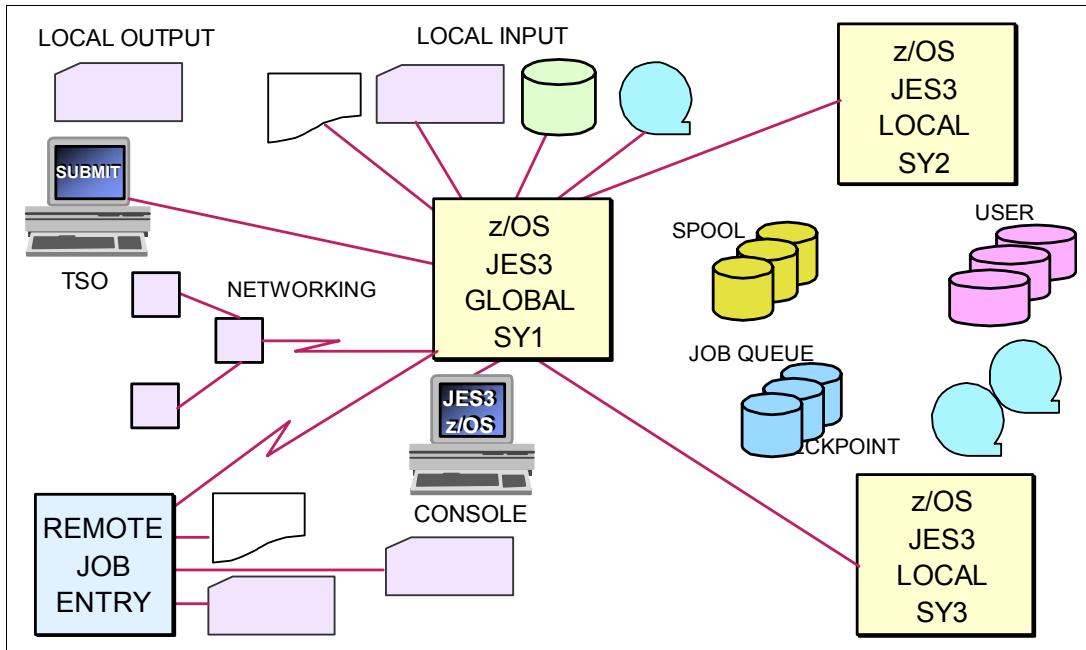


Figure 3-28 JES3 overview

JES3 configuration

JES3 can run on a single processor, or on multiple processors, up to 32 processors in a *sysplex*. A *sysplex* is a set of MVS systems communicating and cooperating with each other through certain multisystem hardware components and software services to process customer workloads.

The following hardware components are used for connecting the systems:

- ▶ A sysplex timer to synchronize the TOD clocks of the processors of a sysplex (not required if a sysplex is located on only one processor)
- ▶ Channel-to-channel connections to connect the members of a sysplex to house a coupling data set

In a sysplex, your installation must designate one processor as the focal point for the entry and distribution of jobs and for the control of resources needed by the jobs. That processor, called the *global processor*, distributes work to the processors, called *local processors*.

It is from the global processor that JES3 manages jobs and resources for the entire complex, matching jobs with available resources. JES3 manages processors, I/O devices, volumes, and data. To avoid delays that result when these resources are not available, JES3 ensures that they are available before selecting the job for processing.

JES3 keeps track of I/O resources, and manages workflow in conjunction with the workload management component of MVS by scheduling jobs for processing on the processors where the jobs can run most efficiently. At the same time, JES3 maintains data integrity. JES3 will not schedule two jobs to run simultaneously anywhere in the complex if they are going to update the same data.

JES3 can be operated from any console that is attached to any system in the sysplex. From any console, an operator can direct a command to any system and receive the response to that command. In addition, any console can be set up to receive messages from all systems,

or a subset of the systems in the sysplex. Thus, there is no need to station operators at consoles attached to each processor in the sysplex.

If you want to share I/O devices among processors, JES3 manages the sharing. Operators do not have to manually switch devices to keep up with changing processor needs for the devices.

The JES3 architecture of a global processor, centralized resource and workflow management, and centralized operator control is meant to convey a single-system image, rather than one of separate and independently-operated computers.

JES3 runs in the following environments:

- ▶ Single-processor environment
- ▶ Multiprocessor environment
- ▶ Remote job processing environment
- ▶ JES3 networking environment
- ▶ APPC environment

A JES3 complex can involve any combination of these environments.

JES3 complex

z/OS uses JES3 to control the input, processing, and output of jobs. JES3 services the job processing requirements of from 1-32 physically connected z/OS processors, called *mains*. Viewed as a whole, the 1-32 main environments that are serviced by JES3 are called a *complex*. In Figure 3-29 illustrates an overview how JES3 works.

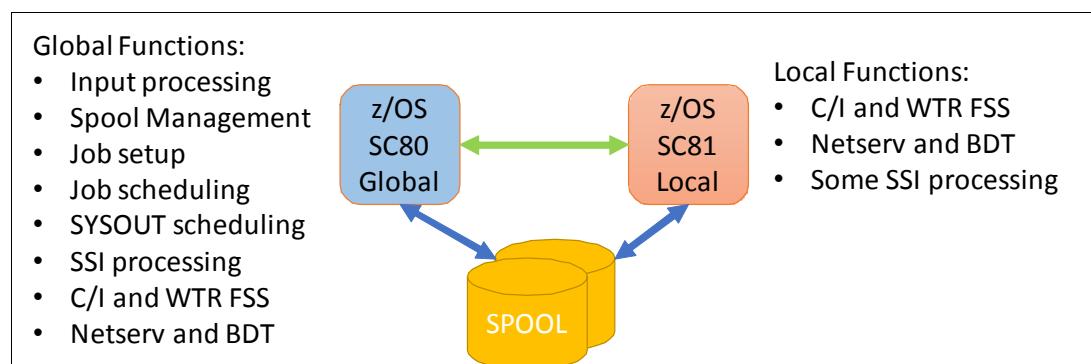


Figure 3-29 JES3 complex

JES3 has its own private address space in each of the mains in the complex. One main, the JES3 global main, is in control of the entire complex. There must be a *global main*; if there is only one main in the complex, that main is the global. In a complex with more than one main, the other mains in which JES3 resides are called *local mains*. There can be as many as 31 local mains.

JES3 is designed so that if the global fails, any properly-configured local within the complex can assume the function of the global through a process called dynamic system interchange (DSI). Figure 3-29 illustrates the basic structure of a JES3 complex.

As the primary subsystem, the global JES3 plays an important role, and the following functions provided by JES3 indicate why communication is needed between MVS and JES3. The global JES3:

- ▶ Introduces all jobs into the system, no matter what the source.
- ▶ Handles scheduling of conversion and interpretation of JCL.

- ▶ Completes pre-execution setup of devices.
- ▶ Schedules MVS jobs to all main processors.
- ▶ Maintains awareness of all jobs in execution.
- ▶ Handles the scheduling of all SYSOUT data sets.
- ▶ Manages the allocation and deallocation of space on the shared-spool devices.

When carrying out some of these responsibilities, global JES3 needs the assistance of local JES3. This is true during the scheduling of work on the local processor.

3.6.1 JES3 functions

It is from the global processor that JES3 manages jobs and resources for the entire complex, matching jobs with available resources. JES3 manages processors, I/O devices, volumes, and data. To avoid delays that result when these resources are not available, JES3 ensures that resources are available before selecting the job for processing.

JES3 keeps track of I/O resources and manages workflow in conjunction with the workload management component of MVS by scheduling jobs for processing on the processors where the jobs can run most efficiently. At the same time, JES3 maintains data integrity. JES3 will not schedule two jobs to run simultaneously anywhere in the complex if they are going to use the same serially reuseable resources. If you want to share I/O devices among processors, JES3 manages the sharing. Operators do not have to manually control the online/offline status of devices to keep up with changing processor needs for the devices.

JES3 global functions

The following JES3 global functions control all work in a JES3 complex:

- ▶ *Workflow manager*: JES3 provides installation benefits from the distribution of work among processors as a workflow manager. The entry of all jobs through a central point means that control of the actions needed to prepare jobs for execution can be centralized. The distribution and publication of job management functions becomes unnecessary, and an awareness of the status of all jobs entering or leaving the system can be easily maintained.
- ▶ *Resource manager*: Another benefit is resource management. All jobs, all input required for the jobs, and all output produced by the jobs enters or leaves the system at a single point. This single point, JES3, can coordinate the subsystem, the allocation of devices, volumes, and data sets. Centralized resource control expands the opportunity for full resource utilization. If you are using the storage management subsystem (SMS), you can allow SMS to coordinate the allocation of permanently resident catalog volumes and catalogued data sets. When SMS is activated, JES3 will not manage units and volumes for SMS-managed data sets.
- ▶ *Operations aid*: Operator control also benefits from improved resource utilization and centralized job management. With all system resources known to JES3 and with one job management mechanism, providing control over the entire system is relatively simply. And yet, the need for operator control can be minimized because JES3 is aware of job mix and resource availability and can coordinate them with little need for operator intervention and decision-making.

JES3 DSP

Each small piece of work that JES3 performs when processing a job is accomplished with a JES3 program called a dynamic support program, or DSP. Each DSP is presented on the FCT chain by one or more FCT entries or elements. The elements on the FCT chain are executed according to their priority, and are placed on the FCT chain with the high priority element first. The higher priority elements are executed before the lower priority elements.

JES3 dynamic support programs (DSPs) control JES3 processing. Some primary DSPs (such as MAIN) directly relate to a job's execution. Other DSPs (such as Dump Job and Disk Reader DSPs) provide functions or services that can be called by a system operator.

In addition to the JES3 DSPs, you can create your own DSPs and add them to the system. Such DSPs run enabled in supervisor state, and under protection key 1. They become part of JES3, and JES3 expects them to use the same programming conventions as JES3 DSPs supplied by IBM. Examples of DSPs provided by JES3 are readers, writers, converters, and so on.

To execute a DSP:

1. Issue the ***CALL** command to make the DSP available to the system (that is, added to the FCT chain).
2. Issue the ***START** command to start processing the DSP.

To stop a DSP, issue the ***CANCEL** command.

3.6.2 JES3 job flow

Whatever the source of the input, JES3 is signalled that an input stream is to be read, which begins a chain of events that includes:

- ▶ Creating and scheduling of a card reader job.
- ▶ Reading the input stream by a DSP.
- ▶ Building JCT entries for each job in the input stream.
- ▶ Executing DSPs represented by scheduler elements in the JCT entries for each job.

The modules that provide this service control the processing at the beginning of a typical MVS job. Input routines create scheduler elements that represent jobs to JES3, process control statements, and group jobs into batches.

Figure 3-30 illustrates the JES3 job flow.

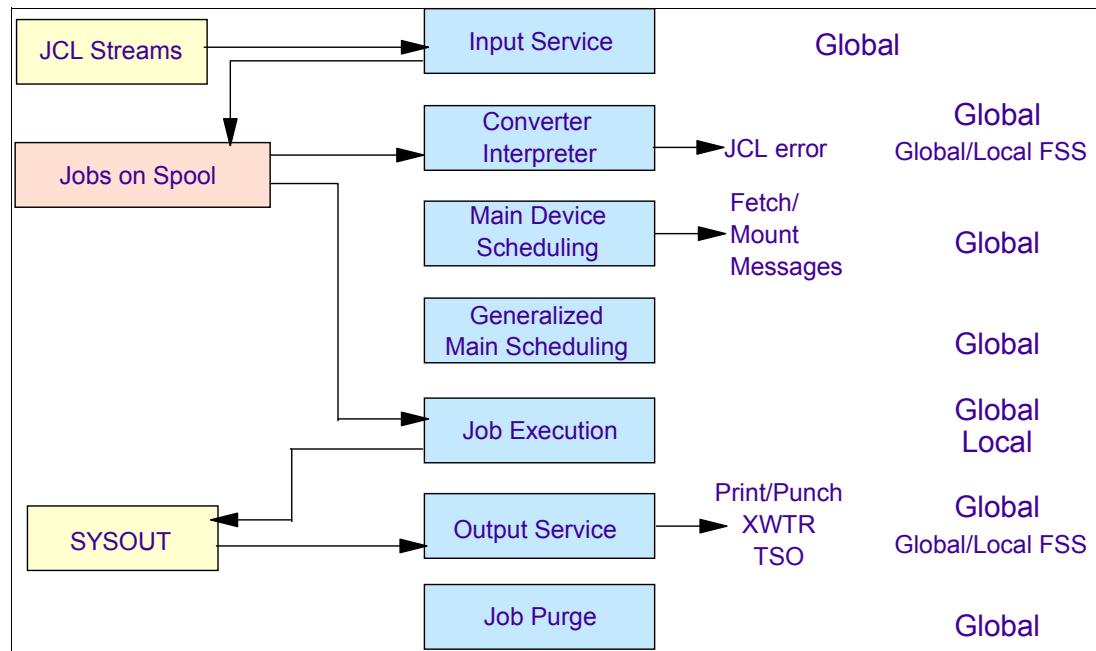


Figure 3-30 JES3 job flow

Input service accepts and queues all jobs entering the JES3 system. The global processor reads the job into the system from:

- ▶ A TSO **SUBMIT** command
- ▶ A local card reader (CR DSP)
- ▶ A local tape reader (TR DSP)
- ▶ A disk reader (DR DSP)
- ▶ A remote work station (RJP/SNARJP DSPs)
- ▶ Another node in a job entry network (NJE DSPs)
- ▶ The internal reader (INTRDR DSP)

This service reads from the input source and adds jobs to the job queue.

JES3 converter interpreter

The converter/interpreter (C/I) is the first scheduler element for every standard job. After a job passes through this first segment of processing, JES3 understands the resources that the job requires during execution. C/I routines provide input to main device scheduling (MDS) routines by determining available devices, volumes, and data sets. These service routines process the job's JCL to create control blocks for setup and also prevent jobs with JCL errors from continuing in the system. Main device scheduling provides for the effective use of system resources. JES3 MDS, commonly referred to as *setup*, ensures the operative use of non-sharable mountable volumes, eliminates operator intervention during job execution, and performs data set serialization. It oversees specific types of pre-execution job setup and generally prepares all necessary resources to process the job. The main device scheduler routines use resource tables and allocation algorithms to satisfy a job's requirements through the allocation of volumes and devices, and, if necessary, the serialization of data sets.

JES3 GMS processing

JES3 generalized main scheduling (GMS) is the group of routines that govern where and when MVS execution of a JES3 job occurs. Job scheduling controls the order and execution of jobs running within the JES3 complex.

Job execution

Job execution is under the control of JES3 main service, which selects jobs to be processed by MVS initiators. Main service selects a job for execution using the job selection algorithms established at JES3 initialization. The MAINPROC, SELECT, CLASS, and GROUP initialization statements control the key variables in the job scheduling and job execution process.

Output processing

Output service routines operate in various phases to process SYSOUT data sets destined for print or punch devices, TSO users, internal readers, external writers, and writer functional subsystems.

Purge processing

Purge processing represents the last scheduler element for any JES3 job (that is, the last processing step for any job). It releases the resources used during the job and uses the System Management Facility (SMF) to record statistics.

3.6.3 JES3 checkpoint

The JES3 checkpoint data set or data set allow you to warm or hot start the JES3 system with little or no loss of system information.

The checkpoint must be allocated in the JES3 start procedure (one cylinder is enough). The DDNAMES must be CHKPNT and CHKPNT2. At least one of these data sets must be available to JES3 during initialization processing. The checkpoint is not heavily used, but do not place it on a spool volume.

You can add or replace either checkpoint data set over a JES3 restart of any kind with no effect on JES3 processing. Both checkpoint data sets contain identical information; to ensure against loss of checkpointed data, allocate both data sets.

The following rules and requirements apply to the checkpoint data set:

- ▶ The checkpoint data set must be allocated as a single extent.
- ▶ The extent must begin and end on a cylinder boundary.
- ▶ The number of 4K records must be calculated for each record type.
- ▶ The number of tracks must be calculated for each record type.
- ▶ Additional tracks should be added to the total for all record types for error recovery and possible expansion of a record type over time.

Accessing the checkpoint data set

Input or output to the checkpoint data set may be done only through the IATXCKPT macro interface. The following rules must be heeded when accessing the checkpoint record:

- ▶ Access to the checkpoint from the JES3 address space is serialized by the access method itself.
- ▶ The IATXCKPT macro to READ, WRITE, or PURGE records must be used.
- ▶ Creation of new records can be done only by the JES3 address space on the global processor.
- ▶ Requests to read or write existing records may be done in any address space or processor.
- ▶ Requests to PURGE existing records can be done only from the global JES3 address space.

Note: If you allocate only one checkpoint data set and it develops a severe permanent I/O error, you must complete a cold start. If you allocate both checkpoint data sets and one develops a severe permanent I/O error, JES3 can continue. For recovery procedures, see *z/OS JES3 Diagnosis*, GA22-7547.

Checkpoint data set space

To determine how much space your installation's checkpoint data set or data sets require, consider the following factors:

- ▶ Each checkpoint record type begins on a track boundary. Each track contains a 128-byte track header record.
- ▶ The checkpoint data set track map, the complex status record, the initialization checkpoint record, and the JESCKPNT checkpoint record each need one track.
- ▶ The dynamic allocation checkpoint record requires 44 bytes plus an additional 92 bytes for each DYNALLOC initialization statement.

- ▶ The spool volumes checkpoint record requires 64 bytes plus an additional 80 bytes for each TRACK and FORMAT initialization statement. Add additional bytes as reserve for spool expansion.
- ▶ The spool partition checkpoint record requires 64 bytes plus an additional 96 bytes for each SPART initialization statement.
- ▶ The partition TAT checkpoint record space requirement is calculated using a complex algorithm involving many different factors. Allow about 512 bytes for each spool data set.
- ▶ The BADTRACK checkpoint record requires 44 bytes plus an additional 64 bytes for each BADTRACK initialization statement. Every entry in the BADTRACK checkpoint record requires an additional 64 bytes. Thus, the size of this data area varies with the number of tracks having I/O errors at one time.

Allocate enough free space to permit growth in the complex without reallocating the checkpoint data sets.

Checkpoint problems at initialization

JES3 creates a copy of each checkpoint record on the duplex checkpoint data set. The primary use of the duplex copy is to enhance error recovery from checkpoint errors in the primary checkpoint data set. The TOD, time of day, is saved in all records and track headers on the checkpoint data sets. The following logic is used when two copies of the checkpoint exist.

- ▶ On read requests, the most current copy of a checkpoint record is used. When it occurs that the time stamps do not match, the most current copy is then used to update the other one.
- ▶ If an I/O error occurs in accessing a checkpoint record, then the duplex checkpoint record is accessed to satisfy the request.
- ▶ If one checkpoint data set is lost, it can be replaced with a new one on the next restart. The remaining checkpoint data set is used to update the new one.
- ▶ If both checkpoint data sets are not usable, a COLDSTART is required.

3.6.4 JES3 spool

Spooling refers to process of communicating data to another program by placing it in a temporary working area, where the other program can access it at some later point in time. Spooling is often used when devices access data at different rates. The temporary working area provides a waiting station where data can reside while a slower device can process it at its own rate.

In a JES3 system, spool serves the following functions:

- ▶ As a buffer between input devices and routines that read input data, and between routines that write output data and output devices
- ▶ As a storage place for the control blocks and data that JES3 builds to process jobs.

A spool device must be a DASD. *Spooling* refers only to use of DASD by JES3 for storage of jobs and job-related data. Use of the same device or devices for other purposes is not considered spooling.

Spooling allows reading or writing to take place continuously at or near the full speed of I/O devices. Without spooling, there would be frequent delays (and processing overhead) during reading or writing of data.

The JES3 checkpoint data set contains the information that is required to initialize either a global or local JES3 processor, which lets JES3 start either a global or local JES3 processor with little or no loss of system information.

The JES3 checkpoint facility writes job-related control block information to the JES3 checkpoint data set or data sets at appropriate points in time during system processing, that is as information changes in the system. This control block information is restored to the system after performing a hot or warm start. All other information is lost.

The JES3 spool (Figure 3-31) consists of one or more single extent DASD data sets. Each data set is typically a full volume that starts and ends on a cylinder boundary. The data sets are logically divided into track groups, the unit of spool space allocation for jobs.

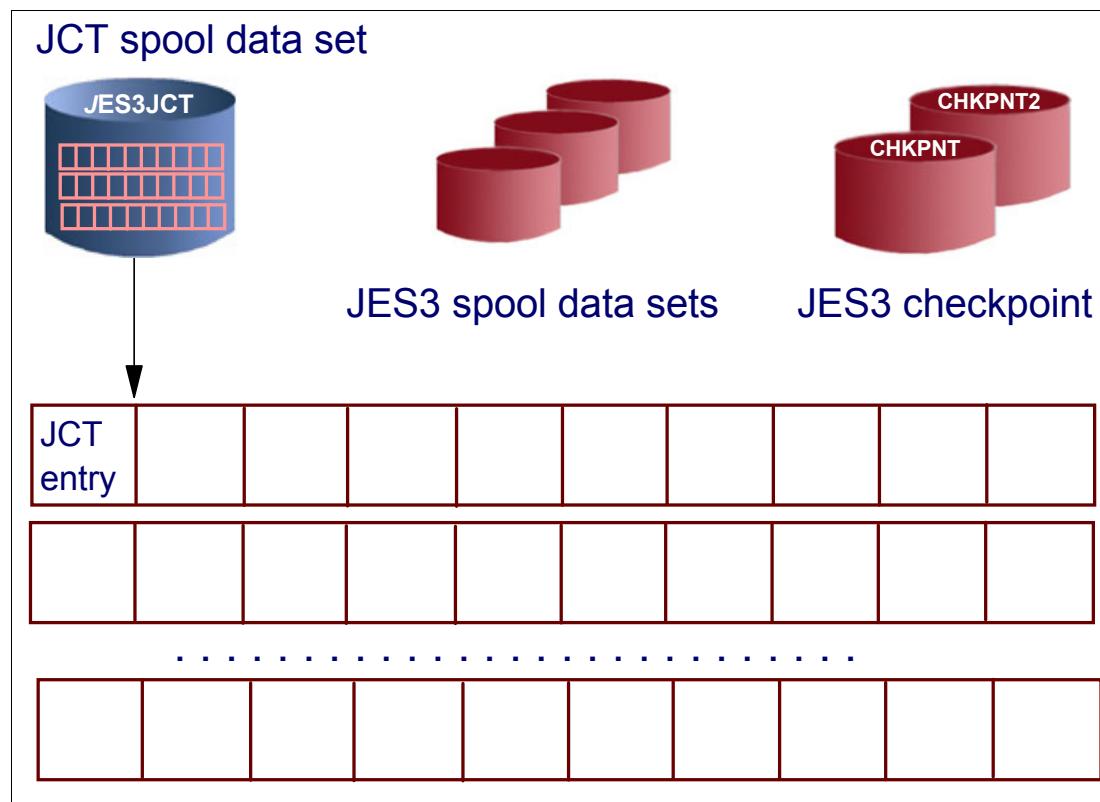


Figure 3-31 JES3 spool

The system programmer defines a spool data set to JES3 by a TRACK or FORMAT statement in the JES3 initialization stream.

Each TRACK or FORMAT statement defines a unique ddname of the DD statement or the ddname on a DYNALLOC statement that defines the spool data set. Spool data sets must be MVS allocated to each JES3 processor so that any main in the JES3 complex can perform I/O to any spool data set.

The spool data set can be allocated either by using JCL DD statements in the JES3 start procedure or by the JES3 DYNALLOC initialization statement. To manage spool data sets, JES3 assigns a unique internal extent number to each data set during JES3 initialization.

Figure 3-32 provides a JES3 sample of SPOOL definitions.

```
DYNALLOC,DDN=SP00L01,DSN=JES3#A.RZ4.P0.SP00L01,UNIT=SYSDA,VOLSER=SYA431
DYNALLOC,DDN=SP00L02,DSN=JES3#A.RZ4.P0.SP00L02,UNIT=SYSDA,VOLSER=SYA432
DYNALLOC,DDN=SP00L03,DSN=JES3#A.RZ4.P0.SP00L03,UNIT=SYSDA,VOLSER=SYA433
FORMAT,DDNAME=SP00L01,SPART=SP00LP1
TRACK,DDNAME=SP00L01,SPART=SP00LP1,STT=(1660,1664)
FORMAT,DDNAME=SP00L02,SPART=SP00LP1
TRACK,DDNAME=SP00L02,SPART=SP00LP1
```

Figure 3-32 JES3 sample of SPOOL definitions

Formatting spool data sets

Before JES3 can use a spool data set, you must format the spool data set. You can use one of the following methods:

- ▶ Format it during JES3 initialization by including a FORMAT statement in the JES3 initialization stream.
- ▶ Format it by executing the utility program IEBDG as a batch job, which fills the extent with hexadecimal FF data.

If you format a spool data set during JES3 initialization, JES3 can use the spool data set after initialization completes. If you use IEBDG to format a spool data set, you must then do a warm start or cold start so JES3 can use the data set.

Formatting during JES3 initialization

To format a spool data set during JES3 initialization, include a FORMAT statement for the spool data set in the JES3 initialization stream. Then start JES3 using that initialization stream.

The type of start you use depends on why you are formatting the spool data set:

- ▶ If you have changed the BUFSIZE= parameter on the BUFFER statement, use a cold start (C). (In this case, you must format all spool data sets.)
- ▶ If you are replacing a spool data set, use a warm start to replace a spool data set (WR). If you also want an analysis of the jobs in the job queue, use a warm start with analysis to replace a spool data set (WAR).
- ▶ If you are adding a spool data set, use a warm start (W) or a warm start with analysis (WA).

After JES3 processes the initialization stream, replace the FORMAT statement with a TRACK statement. If the FORMAT statement contained the STT or STTL parameter, also code this parameter on the TRACK statement.

If you use a warm start and the initialization stream contains a FORMAT statement for a spool data set that is already formatted, JES3 issues a warning message. JES3 continues with initialization, however, and does not reformat the spool data set.

Formatting with IEVDG

You can use the utility program IEVDG to format a data set that you plan to use as a spool data set. *z/OS DFSMSdfp Utilities*, SC23-6864, explains how to use IEVDG. An JCL example is shown in Figure 3-33.

```
//JOB1      JOB    ....
//FORMAT    EXEC   PGM=IEVDG
//SPXTNT    DD     DSN=spool_data_set_name,DISP=OLD,
//           UNIT=SYSDA,VOL=SER=serial_number,
//           DCB=(RECFM=U,BLKSIZE=nnn)
//SYSPRINT  DD     SYSOUT=A
//SYSIN     DD     *
DSD        OUTPUT=(SPXTNT)
FD         NAME=SPOOL,FILL=X'FF',LENGTH=nnn
CREATE     NAME=(SPOOL),QUANTITY=2000000000
END
/*
```

Figure 3-33 JES3 SPOOL formatting JCL

The value of the variable *nnn* must equal the value of the BUFSIZE= parameter on the BUFFER initialization statement. The variable *nnn* appears on both the SPXTNT DD statement and on the FD utility program control statement.

Specify the value of the QUANTITY=2000000000 on the CREATE statement to ensure that IEVDG formats the entire data set.

If IEVDG successfully formats the entire spool data set, the formatting job ends with an ABEND code of D37. In addition, MVS issues message IEC031I. Ignore the corrective action specified in the message.

After the spool data set has been formatted, include a TRACK statement for it in the initialization stream. To make the spool data set available to JES3, restart JES3. The type of restart you use depends on why you are formatting the spool data set:

- ▶ If you have changed the BUFSIZE= parameter on the BUFFER statement, use a cold start (C). In this case, you must format all spool data sets.
- ▶ If you are replacing a spool data set, use a warm start to replace a spool data set (WR) or a warm start with analysis to replace a spool data set (WAR).
- ▶ If you are adding a spool data set, use a warm start (W) or a warm start with analysis (WA).

Reformatting a spool data set

When reformatted, reformat a data set only when one of the following conditions is true:

- ▶ The spool data set has been damaged.
- ▶ You change the BUFSIZE= parameter on the BUFFER initialization statement, in which case you must reformat all spool data sets.

To reformat a spool data set, use either of the procedures described previously.

The checkpoint data set or data sets lets JES3 store key information while the system is running. One or both JES3 checkpoint data sets must be defined in the JES3 start procedure using ddnames CHKPNT and CHKPNT2.

Single track table

The single track table is a section of spool space used exclusively for JES3 control blocks not associated with a particular job, such as control blocks used to track JES3 functions and to save status. The allocation mechanism of the single track table is by record as opposed to track group, in contrast to the rest of JES3 spool space allocation.

The following types of STTs are used in the JES3 environment:

- ▶ The job control table (JCT) STT
- ▶ The system STT

Job control table single track table (JCT STT)

The job control table single track table (JCT STT) is built during JES3 initialization. The JCT STT is a fixed length table and contains a bit map used to allocate JCTs.

System single track table (STT)

The system STT is used to allocate single record files (SRFs) on a individual record basis. The system programmer specifies the spool data sets that contain the STTs to be allocated using the STT= or STTL= parameter on the TRACK or FORMAT statement, as shown in the following example:

```
FORMAT,DDNAME=SP00L1,STT=(400,401)  
FORMAT,DDNAME=SP00L2,STTL=(400,2)
```

The first system STT is called the *primary STT*, and STTs that are built later are called *expansion STTs*. The primary STT and any expansion STTs are linked together to form a single linked list.

The STT and STTL statements are defined as follows:

STT	Specifies the range of cylinders that you want allocated to the STT. This range must be within the extent allocated to the data set. The value of cylnum specifies an absolute cylinder number. (Absolute cylinder numbers are device-dependent; the component description for the device describes the numbering scheme.)
STTL	Specifies the location and number of <i>track groups</i> to allocate to the STT. These track groups must be within the extent allocated to the data set. The value for cylnum specifies an absolute cylinder number indicating the beginning cylinder number of the STT allocation in this extent. (Absolute cylinder numbers are device-dependent; the component description for the device describes the numbering scheme.) The value for numtrkgps specifies the number of track groups to allocate to this extent, beginning with the first track group that is located completely in cylinder cylnum. The maximum number of track groups that may be allocated to the STT is 9999.

STT dynamic expansion

If all STT space is used, the STT is expanded dynamically. STT space is accounted for on a record basis via a control block called the STT. In the STT control block, there is one bit for each STT record.

Replacing or deleting a spool data set

If you replace a spool data set, JES3 cancels all jobs with data on the replaced spool data set. If the replaced data set contains STT records, JES3 might lose information that could result in the loss of jobs in the system. STT records include information, such as the status of devices, DJC network data, deadline queue data, volume unavailable data, dynamic allocation checkpoint data, output service checkpoint data, JESNEWS, device fencing data, virtual unit

status, GMS status, and FSS checkpoint data. If STT data is lost, JES3 issues messages that allow you to take the appropriate recovery actions.

If you delete a spool data set, JES3 cancels all jobs in the system that have spool data or allocation tables on the affected data set. Try not to delete a data set that contains important information (for example, the single track table (STT) or the JESNEWS data set). If this information is lost, the system issues messages giving you the opportunity to take appropriate actions.

Adding or deleting a spool data set

You can increase or decrease the spool capacity in your environment without performing a cold start by adding or deleting a spool data set. To determine whether the current spool capacity is appropriate for your environment, you can monitor spool usage using the ***I,Q,S** operator command. To monitor the use of channel paths, control units, and spool data sets, use a system monitoring facility such as system management facilities (SMF), resource measurement facility (RMF™), or JES3 monitoring facility (JMF). To add a spool data set by using the ***MODIFY CONFIG** command, a hot start with refresh (HR) or a warm start (W), follow the guidelines found in the *z/OS JES2 Initialization and Tuning Guide*, SA32-0991.

Deleting a spool data set requires more planning. You can delete a spool data set using one of the following methods:

- ▶ Warm start
- ▶ Hot start with refresh
- ▶ The ***MODIFY CONFIG** command

The general steps for each method are:

1. Prepare an updated spool data set definition.
2. Stop new track group allocations from the spool data set.
3. Empty the spool data set (for example, release currently allocated track groups).

As you work through the process of clearing the data set, use the ***I Q,DD=ddname,U** command to check the progress. For example, if you want to delete a spool data set, SPOOL3, on a system, before you begin the process, issue ***I Q,DD=SPOOL3,U** to view the jobs with allocations on the data set, as shown in Figure 3-34.

```
*i Q,DD=SPOOL3,u
IAT8752 SPOOL3      TOTAL IN USE          10 TRKGPS
IAT8752 SPOOL3      TOTAL IN USE BY JES3    4 TRKGPS
IAT8752 SPOOL3      TOTAL IN USE BY JOBS    6 TRKGPS
IAT8753 SPOOL3      USERS FOUND=   2 JES3  5 JOBS; 7 DISPLAYED
IAT8754 SPOOL3 : JOB IBMUSER (JOB00016)  2 TRKGPS,  2%
IAT8754 SPOOL3 : JOB JES3STT (JOB00000)  2 TRKGPS,  2%
IAT8754 SPOOL3 : JOB JES3INIT (JOB00000) 2 TRKGPS,  2%
IAT8754 SPOOL3 : JOB SDSF   (JOB00004)   1 TRKGPS,  1%
IAT8754 SPOOL3 : JOB CREATEHC (JOB00009) 1 TRKGPS,  1%
IAT8754 SPOOL3 : JOB MAP0421 (JOB00021)  1 TRKGPS,  1%
IAT8754 SPOOL3 : JOB MAP0913 (JOB00022)  1 TRKGPS,  1%
IAT8755 INQUIRY ON SPOOL DATA SET USAGE COMPLETE
```

Figure 3-34 JES3 list of allocations for SPOOL3

The display shows that JES3 (JOB00000) occupies space on the data set. JES3STT is the job name that is given to spool space used internally by JES3 for information that persists across a JES3 restart or an IPL. JES3INIT represents spool space that is used to contain the data contained in your JES3 initialization statements.

The first step is to prepare an updated spool definition. You can use the ***MODIFY CONFIG** command to delete the SPOOL3 data set, and you can define your spool data sets with DYNALLOC statements. Remove the DYNALLOC statement for SPOOL3 and clean up references to it, such as TRACK or FORMAT statements. The updates for a hot start with refresh and warm start are similar. For a warm start, you might need to remove the appropriate DD statement from the JES3 cataloged start procedure. See *z/OS JES3 Commands*, SA32-1008, for more information about the operator activities that are required to add or delete a spool data set.

The next step in the delete process is to drain the target spool data set by issuing a ***F Q,DD=ddname,DRAIN** command. This command prevents new allocations from the data set and relocates the JES3STT records. Although a subsequent ***I Q,DD=ddname,U** command might display the JES3STT space on the spool, all of the STT records in that space are moved.

The next step is to clear the spool data set. You can use one of the following methods to reclaim the spool space:

- ▶ CANCEL jobs, if practical. For example, the ***F J=9,C** command reclaims the space for job CREATEHC.
- ▶ Dump eligible jobs by using the JES3 dump job facility (see Dump job facility in *z/OS JES3 Commands*, SA32-1008). In this example, this action reclaimed the space that was used by the MAP0421 and MAP0913 jobs.
- ▶ Log off or CANCEL TSO user jobs. The IBMUSER logged off, and its space was released.
- ▶ Process Job0 output (use ***I U,Q=WTR,J=0** or ***I Q,Q=HOLD,J=0** to see if any output exists).
- ▶ Cancel remaining jobs by running a ***F Q,DD=ddname,CANCEL** command.
- ▶ End started tasks. The process varies depending on the properties of the started task. For example, SDSF was stopped (P SDSF) to release its space.

Delete the spool by running the ***F CONFIG** command. Alternatively, perform a hot start with refresh by using an updated initialization stream. The hot start with refresh is unsuccessful if the initialization data (JES3INIT) resides on the volume. In this case, you have the following options:

1. Use the ***F CONFIG** command to delete the spool data set (recommended). The command moves the initialization files to other spool data sets.
2. Use a hot start with refresh with the original set of spools (while the spool to be deleted is in drained status). Then, do another hot start with refresh to delete the spools.

EAV considerations

With z/OS V1R10 and higher releases, z/OS added support for DASD volumes that have more than 65,520 cylinders. To expand the capacity of DASD storage volumes beyond 65,520 cylinders, z/OS extended the track address format. Thus the name EAV is used for a volume of more than 65,520 cylinders, as illustrated in Figure 3-35 on page 90.

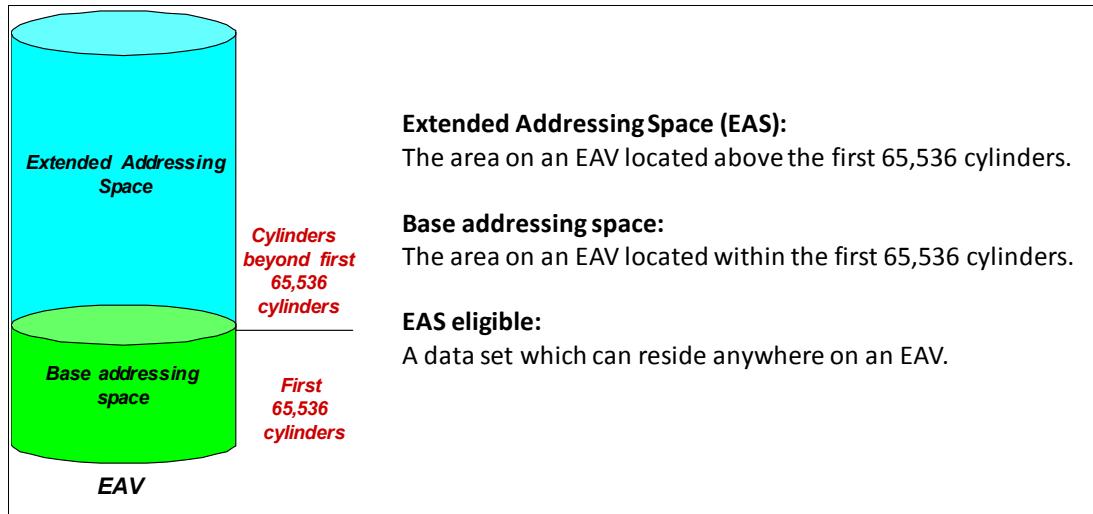


Figure 3-35 JES3 EAV

EAVs provide increased z/OS addressable disk storage. EAVs help to relieve storage constraints and simplify storage management by providing the ability to manage fewer, large volumes as opposed to many small volumes.

DFSMS added support for base and large format sequential data sets that now can be exploited for JES3 data sets. EAS eligible data sets are defined to be those that can be allocated in the EAS and have extended attributes. EAS is sometimes referred to as *cylinder-managed space*. Below EAS is sometimes referred to as track-managed space.

A new data set attribute, EATTR, allows a user to control whether a data set can have extended attribute DSCBs and thus control whether it can be allocated in EAS.

JES3 exploitation of the EAV support customers to define EAS eligible data sets for use as SPOOL extents, checkpoint data sets, and the Job Control Table (JCT) data set.

An added benefit is that large sequential data sets, DSNTYPE=LARGE, when EAS is eligible on an EAV, are no longer limited to the volume size of 65,520 cylinders, which allows JES3 customers to have much larger SPOOL data sets. Each SPOOL data set must be contained in a single extent.

Attention: You cannot allocate any secondary extents, and you cannot allocate more than 1024 spool data sets.

To allow a JCT data set to be copied without a cold start, JES3 provides a program called the JCT utility, or IATUTJCT. This utility can be used to migrate existing JCT and checkpoint data sets to EAS eligible data sets.

Managing the JES3 SPOOL space does not change with the support added to exploit EAVs. See *z/OS JES2 Initialization and Tuning Guide*, SA32-0991 for details about adding and replacing SPOOL data sets.

Figure 3-36 shows how to allocate and then format a JES3 spool extent in EAS on an EAV.

```
//ALLOC EXEC PGM=IEFBR14
//SPXTNT DD DSN=SYS1.JESPACE,DISP=(NEW,KEEP,KEEP),
//           UNIT=3390,VOL=SER=J3SPL1,
//           DCB=(RECFM=U,BLKSIZE=2048),
//           SPACE=(CYL,80000),EATTR=OPT,
//           DSNTYPE=LARGE
//SPLFRMT EXEC PGM=IEBDG
//SPXTNT DD DSN=SYS1.JESPACE,DISP=SHR,
//           UNIT=3390,VOL=SER=J3SPL1,
//           DCB=(RECFM=U,BLKSIZE=2048,BUFNO=255),
//           DSNTYPE=LARGE
//SYSPRINT DD SYSOUT=*
//SYSIN DD *
DSD OUTPUT=(SPXTNT)
FD NAME=SPOOL,FILL=X'FF',LENGTH=4084
CREATE NAME=(SPOOL),QUANTITY=2147483647
END
```

Figure 3-36 JES3 sample JCL for SPOOL on EAV

3.6.5 JES3 JCT

JES3JCT is the ddname given to the DD statement of job control table (JCT) data set. Other spool data sets allocated to JES3 cannot use this ddname.

The JCT data set is the JES3 job queue. It is accessed by JES3 functions through spool data management services. The JCT data set holds JCT entries that represent jobs. The JCT data set will not contain any SYSIN or SYSOUT data.

Figure 3-37 shows the JES3 initialization statement for JES3JCT.

```
DYNALLOC,DDN=JES3JCT,DSN=JES3#A.RZ4.P0.JCTSPL,UNIT=SYSDA,VOLSER=SYA411
```

Figure 3-37 JES3 initialization statement for JES3JCT

One of the JES3 job control blocks is the JCT entry. The JCT resides in storage until JES3 writes it into the JES3JCT data set on DASD and into a data space. This data set is called the JCT data set and it contains all of the JCT entries. MVS allocates this data set during JES3 initialization. The JCT data does not reside within the DASD space allocated for the other spool volumes.

The JCT entries on the JCT data set are a fixed-length, unblocked format. Note that JCT entries vary in size: called jobs have two scheduler elements (SE), standard jobs have four (or more), and non-standard jobs have a variable number up to an installation-defined maximum.

The maximum number of scheduler elements is specified on the SE parameter of the OPTIONS initialization statement. (10-90 is the valid range; 10 is the default.)

Even though the JCT data set is not contained within the DASD space allocated for spool, it is read from and written to the JES I/O buffer pool by means of JSAM single-record file processing techniques. The JCT data set, depending on how much space was allocated to it, contains a fixed number of blocks, which determines the maximum number of jobs simultaneously in the JES3 complex.

JCT size calculation

The JCT data set contains information about the status of jobs in the system. You must make the size of the JCT data set large enough to accommodate the maximum number of jobs that can be in your JES3 complex simultaneously. If the data set is larger than required, JES3 uses only the portion that is needed to hold the maximum number of jobs. (The remainder of the JCT is used for write error recovery.) However, the larger the JCT data set, the longer JES3 initialization takes. For optimum performance, minimize the size of the JCT data set as much as possible.

To calculate the size of the JCT data set, complete the following steps:

1. Calculate the size, in bytes, of the JCT entry within the JCT data set as follows:

$xxx = (4 \times max\ SEs) + zzz + 28$ bytes (prefix)

where:

- xxx is the size, in bytes, of a single JCT
- 4 is the size of one JCT scheduler element in the JCT
- $max\ SEs$ is the maximum number of scheduler elements as specified
- zzz is the size, in bytes, of the fixed JCT. Field JCTFSIZE in the IATYJCT macro defines this value. If you have not modified the size of the JCT, the value of JCTFSIZE is 668 bytes.
- 28 is the size, in bytes, of the SRF header prefix mapped using the IATYSRF macro.

2. Determine the number of JCTs per cylinder. For example, the IBM 3390 device contains 41 standard size JCT entries on a track and 615 JCT entries on 1 cylinder.

Calculation example

You want to process 40,000 JES3 jobs concurrently. The size of the standard IBM-supplied JCT has not been modified by making changes to IATYJCT, IATYCND, or IATYFDB. You have specified SE=10 in the OPTIONS initialization statement (the default) and are using IBM 3390 Direct Access Devices for the JCT data set. Follow these steps:

1. Calculate the size, in bytes, of the JCT entry within the JCT data set as follows:

$xxx = (4 \times max\ SEs) + zzz + 28$ bytes for prefix

$736 = (4 \times 10) + 668 + 28$

2. Determine the number of JCTs per cylinder.

Using the 3390 capacity table, a 736 byte record falls within the 719 - 752 data length range, and 615 of these records will fit in a 3390 cylinder.

3. Determine the required size of the JCT data set:

JCT data set	$(64 + max\ # \text{ of jobs})$
size	-----
in cylinders	number of JCTs per cylinder
$65.15 = (64 + 40,000) / 615$	

Because the result is not a whole number, it must be incremented to the next whole number, 66. You require a JCT data set size of 66 cylinders.

You can allocate the JCT data set by including the //JES3JCT DD statement in the JES3 catalogued start procedure or by using a DYNALLOC initialization statement.

JCT utility IATUTJCT

For various reasons you might need to replace an existing JCT data set. In order to allow a JCT data set to be copied without a cold start, JES3 provides a program called the JCT utility.

To complete the copy using IATUTJCT, you must:

- ▶ Allocate a new set of checkpoint data sets in addition to the new JCT data set.
- ▶ Identify the data set names for the new data sets used in the JCL for the JCT utility.
- ▶ Arrange your system such that JES3 can be started with either the new or old data sets. You can use symbolic parameters on the JES3 procedure to allow use of either the new or old data sets.
- ▶ Bring JES3 and all Converter Interpreter Functional Subsystems (CIFSS) down on the global processor and all local processors.

Attention: If you do not bring JES3 down, JES3 might update the old JCT and the checkpoint data sets after you run IATUTJCT. These updates will be lost when you switch to the new data sets and will cause corruption to the JES3 job queue. This corruption then might lead to conditions from which a cold start is required to recover.

- ▶ Restart JES3, after running IATUTJCT, with the DD definitions that point to the new JCT data sets. For the checkpoint data set, these DD statements are in the JES3 procedure. For the JCT data set, these DD statements are in the JES3 procedure or identified by the DYNALLOC statement in the JES3 initialization stream. An example for that is shown in Figure 3-38 on page 93. To start migration after JES3 is down use the following start command:

S IATUTJCT,P=MIGRATE,SUB=MSTR
- ▶ Restart the C/I FSS procedures with the new checkpoint data sets.

```
//IATUTJCT PROC P=
//*
//IEFPROC EXEC PGM=IATUTJCT,      Invoke the IATUTJCT Utility
//    TIME=1440,
//    PARM='&P'
//*
//STEPLIB DD DISP=SHR,           Required if IATGRCK and IATUTJCT
//    DSN=SYS1.SIATLIB   are not in LNKLSTxx
//*
//JES3JCT DD DISP=SHR,          Current JCT data set
//    DSN=SYS1.JES3JCT
//*
//NJES3JCT DD DISP=SHR,         New JCT data set
//    DSN=SYS1.NJES3JCT
//*
//CHKPNT DD DISP=SHR,           Current primary checkpoint data set
//    DSN=SYS1.CHKPNT
//*
//CHKPNT2 DD DISP=SHR,          Current alternate checkpoint data set
//    DSN=SYS1.CHKPNT2
//*
//NCHKPNT DD DISP=SHR,          New primary checkpoint data set
//    DSN=SYS1.NCHKPNT
//*
```

Figure 3-38 Sample JCL for IATUTJCT

When using IATUTJCT to move the JCT data set, you can pick up the new data set with any type of JES3 start; however, if you perform a hot start with refresh or a warm start, and you use the DYNALLOC statement to define the JES3JCT DD name, you must change the DYNALLOC statement to point to the new JCT data set.

3.6.6 JES3 system configuration

An *initialization stream* is a collection of JES3 initialization statements that are used to define the JES3 system configuration. It also defines how JES3 manages resources and jobs. These statements tell JES3 how to manage the following resources and jobs:

- ▶ Jobs, job classes, and job class groups
- ▶ Mains (global and local)
- ▶ I/O devices
- ▶ Main and external storage
- ▶ The system log
- ▶ Communication lines or protocols
- ▶ Operator communication

The following statements must be included in all initialization streams:

- ▶ ENDINISH
- ▶ ENDJSAM
- ▶ FORMAT or TRACK
- ▶ MAINPROC

The job management information that you code on initialization statements specify how you want JES3 to process job input and interpret JES3 control statements and to select and schedule jobs for execution and process job output, as shown in Figure 3-39.

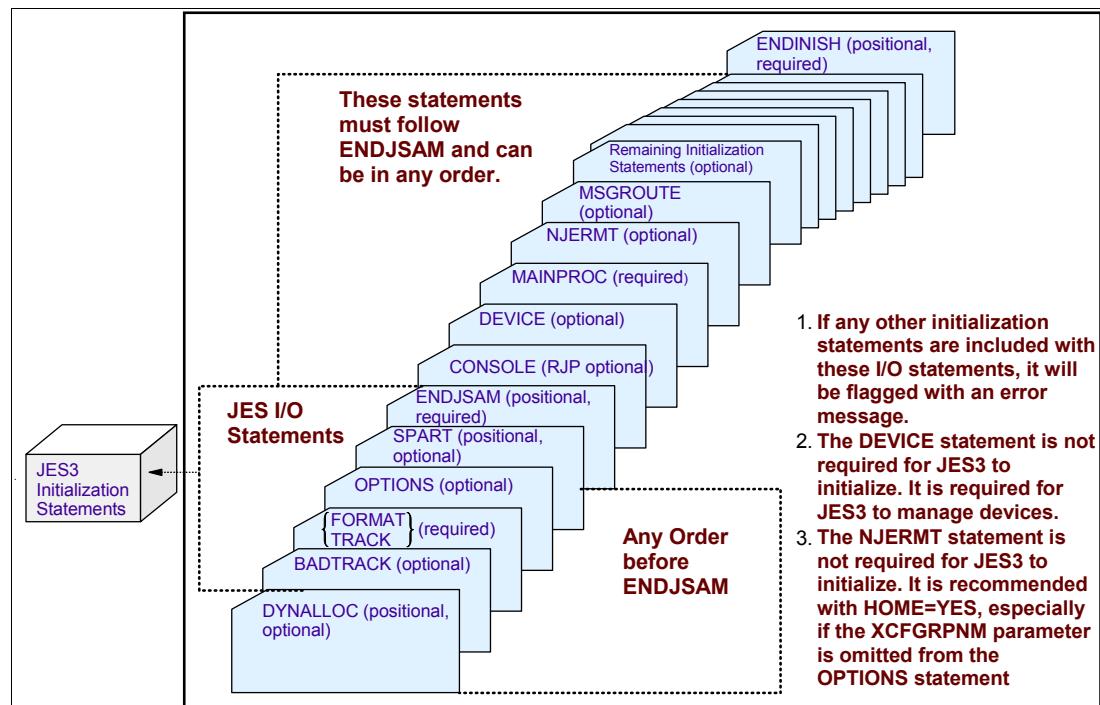


Figure 3-39 JES3 initialization statements

You must observe the following rules when coding initialization statements:

- ▶ Code the statement name and parameters in columns 1-71. Column 72 must be blank if the statement is complete; column 72 can be blank or non-blank if the statement is continued. JES3 ignores columns 73-80.
- ▶ Use commas to separate the statement name from the first parameter and to separate one parameter from another.
- ▶ If you code a keyword parameter more than once, JES3 usually uses the last value coded. In some cases, however, JES3 treats a duplicate keyword as a continuation of a previously coded keyword of the same name. In other cases, when JES3 encounters a duplicate keyword, it issues a diagnostic message.
- ▶ If you want to continue an initialization statement (individual statement descriptions specify whether a statement can be continued).

Phases of JES3 initialization

Depending on the type of start the operator specifies, JES3 initialization is a three or four phase process, as follows:

- Phase 1** Phase 1 determines the types of starts that are allowed for the main and asks the operator to select a start type.
- Phase 2** Phase 2 reads the statements from the initialization stream that initialize the control blocks to manage spool space and, in the event of a restart, validates jobs that are in the JES3 job queue.
- Phase 3** Phase 3 reads the initialization stream and converts information supplied on each initialization statement to intermediate spool files. These intermediate spool files are written to spool.
- Phase 4** Phase 4 uses the intermediate spool files built in phase 3 to build the final control blocks necessary for job execution. Phase 4 informs the operator that JES3 initialization is complete.

When modifying or creating an initialization stream, you must be aware of related initialization statement parameters. With related parameters, the value you code for one parameter influences the value you code for another parameter. *z/OS JES2 Initialization and Tuning Guide*, SA32-0991 contains tables that list the interdependent parameters.

JES3 initialization stream

The JES3IN DD statement in the JES3 start procedure defines the data set that contains the JES3 initialization stream. This data set must be a blocked or unblocked partitioned data set.

The following list defines some JES3 initialization statements:

- ▶ ACCOUNT (Job Accounting): Use the ACCOUNT initialization statement to define default job accounting information. JES3 assigns this default if the operator omits the ACCT parameter on a JES3 *CALL command.
- ▶ BADTRACK (Bypass Defective Tracks): Use the BADTRACK statement to identify defective tracks on a spool volume. JES3 dynamically adds an entry to the BADTRACK table when a defective track is discovered and issues a message to the console operator that identifies the defective track. If possible, add a BADTRACK statement to your initialization stream at that time so that JES3 keeps a record of the defective track across a warm or cold start. If you cannot add a BADTRACK statement immediately, ensure that you add a BADTRACK statement before the next warm or cold start.
- ▶ BUFFER (JES3 Spool Work Buffers): Use the BUFFER statement to define the size of the JES3 buffer pool and the length of JES3 buffers and spool data set records.

- ▶ CIPARM (Converter/Interpreter Parameters): Use the CIPARM statement to specify the options to be used by the MVS converter/interpreter (C/I). These options are used as system defaults applied to certain JCL statement parameters and other options for jobs scheduled on any main.
- ▶ CLASS (JES3 Job Class Definition): Use the CLASS initialization statement to define the characteristics of JES3 job classes. A CLASS statement must define each class that can appear on the //MAIN control statement.
- ▶ COMMDEFN (Communication SSI Definition): Use the COMMDEFN statement to specify the optional user communication subsystem interface (VTAM) parameters.
- ▶ COMMENT (*): Use the comment statement to include comments in a JES3 initialization stream.
- ▶ COMPACT (Compaction Table Definition): Use the COMPACT statement to define a compaction table to JES3. The compaction table is a set of characters which can be transmitted as a compacted character string.
- ▶ CONSOLE (RJP Operator Consoles): Use this CONSOLE statement to define the characteristics of a console for an RJP workstation. This statement assigns message destinations to these type of consoles.
- ▶ CONSTD (Console Service Standards): Use the CONSTD statement to define standards for your console configuration. These standards include JES3 command prefix characters, hardcopy log configuration, and special characters to be used in editing commands processed by JES3 console services.
- ▶ DEADLINE (Deadline Type Definition): Use the DEADLINE statement to define a deadline type for job scheduling; each type determines how JES3 increases the priority of a job so the job is scheduled within a specified time limit.
- ▶ DESTDEF (NJE Define Destinations for SYSOUT): Use the DESTDEF initialization statement to specify how inbound SYSOUT data sets from other NJE nodes are to be processed at this node.
- ▶ DEVICE (Processor Status, BSC line, I/O Device): Use the *Define Processor Status* form of the DEVICE statement (there are two other forms of the DEVICE statement) to define the initial status of mains in a JES3 complex. If you omit this DEVICE, the processor in question is initialized online to every processor in the complex.

Use the *Define a Network BSC line or CTC Connection* form of the DEVICE statement to define a BSC line or a CTC connection that connects your node to another node in a network. You must code a DEVICE statement for each such line. or connection.

Use the *Define I/O Devices* form of the DEVICE statement to define a device that JES3 can use:

- To satisfy its own functions (JES3 device).
- To satisfy the needs of a job (execution device).
- As a JES3 device or as an execution device (shared device).
- ▶ DYNALDSN (Dynamically Allocated Data Set Integrity): Use the DYNALDSN statement to specify which data sets on permanently resident or reserved DASD volumes require data set integrity protection when the data set is dynamically allocated.
- ▶ DYNALLOC (Dynamically Allocate Data Sets and Devices): Use the DYNALLOC statement to specify a data set or a device that you want dynamically allocated to JES3 during initialization. The DYNALLOC statement allows you to allocate a data set or device without changing the JES3 catalogued procedure.
- ▶ ENDINIT (End of Initialization Stream): Use the ENDINIT statement to identify the end of the initialization statements in the initialization stream.

- ▶ ENDJSAM (End of JES3 I/O Statements): Use the ENDJSAM initialization statement to indicate the end of the JES3 spool initialization statements.
- ▶ FORMAT (Format Spool Data Set): Use the FORMAT statement to specify formatting for a data set residing on a direct-access spool volume during initialization. Specify this statement only when introducing an unformatted volume into a JES3 system or when you change the BUFSIZE parameter on the JES3 BUFFER initialization statement.
- ▶ FSSDEF (Functional Subsystem Definition): Use the FSSDEF statement to define the characteristics of a functional subsystem (FSS), which operates in its own address space. Use a FSSDEF statement for either of the following situations:
 - To define one or more C/I FSSs.
 - To define one or more output writer FSSs for printers that you define to run in FSS mode (via the DEVICE initialization statement). You can define more than one printer to run under the control of a single output writer FSS. If you do not define an output writer FSS for each printer that requires one, JES3 creates an FSS using default values.
- ▶ GROUP (Job-Class Group Definition): Use the GROUP initialization statement to define the characteristics of a JES3 job-class group and whether the initiators managed by this group are WLM managed or JES3 managed. A GROUP statement must define each job class group (except for the default group, JS3BATCH) named on a CLASS initialization statement.
- ▶ HWSNAME (High Watermark Setup Names): Use the HWSNAME statement to:
 - Define, to JES3, all names by which users can reference a given device type to enable high watermark setup (HWS) processing.
 - Identify the characteristics of each device name for the specific JES3 complex. This statement can be used to define which device names are subsets of other device names. In general, the fewer the number of alternate names, the more restrictive the device name being defined. This ensures that initial allocation for devices that are reused from step to step is the most restrictive device. It also ensures that attempts to override passed or cataloged unit names are processed correctly. Non-HWS users are encouraged to supply HWSNAME information to take advantage of this function.
- ▶ INCLUDE (Include Initialization Stream Member): Use the INCLUDE statement to include a member in the initialization stream member. Different sections of the initialization stream can be put into different members and included in the primary initialization stream member. The member is the PDS member name within the data set specified on the JES3IN DD statement in the JES3 procedure to be included. Up to four member levels can be used (the primary initialization stream member and up to three INCLUDE level members). Use the INCLUDE statement anywhere after the DYNALLOC statements.
- ▶ INTDEBUG (Initialization Debugging Facility): Use the INTDEBUG statement to specify error message text and an index value. If the specified message text is issued the number of times indicated by the index value, JES3 issues a U005 JES3 user abnormal end and takes a storage dump.
- ▶ MAINPROC (Define a JES3 Main): Use the MAINPROC initialization statement to define a processor as a JES3 main. The initialization stream must include one MAINPROC statement for each main that you wish to define to JES3.
- ▶ MSGROUTE (MVS Message Route Table): Use the MSGROUTE statement to control the routing of subsystem modifiable messages (such as most MVS-issued messages). If you do not include a MSGROUTE statement, the routing attributes of the messages that originate from that processor are not modified by JES3 MSGROUTE processing. Even though MSGROUTE processing might not make modifications, a message is still eligible for other forms of JES3 message routing.

- ▶ NETSERV: Use the NETSERV initialization statement to define the attributes of a TCP/IP/NJE Network Server (NETSERV) address space.
- ▶ NJECONS (Console for NJE): Use the NJECONS initialization statement to specify the message class to which JES3 is to send messages about the JES3 job entry network.
- ▶ NJERMT (JES3 Network Node Definition): Use the NJERMT initialization statement to define a node in the JES3 job entry network.
- ▶ OPTIONS (JES3 Options): Use the OPTIONS initialization statement to specify:
 - The type of MVS system dump to be taken, if needed.
 - Whether or not a dump should be taken when a termination condition exists.
 - The job numbering limits for JES3 jobs.
 - Whether you want the writer output multitasking facility enabled or disabled.
 - The number of scheduler elements needed to support the largest job that will be run in the JES3 complex.
- ▶ OUTSERV (Output Service Defaults and Standards): The OUTSERV initialization statement specifies default values and standards for the output service element (OSE) to be used on output devices; for example printers, punches, or RJP (remote job processing). These defaults apply to every built OSE, regardless of the device that handles the output, provided other overrides do not take effect.
- ▶ RESCLBK (Resident Control Block): Use the RESCLBK initialization statement to preallocate storage for the highly used JES3 function control table (FCT) entries.
- ▶ RESDSN (Resident Data Set Names): Use the RESDSN statement to name permanently resident data sets for which JES3 is to bypass setup processing. JES3 bypasses setup processing whenever the named data sets appear as catalogued references (no UNIT or VOLUME parameters are specified) on the DD statement of a job.
- ▶ RJPLINE (BSC Remote Job Processing Line): Use the RJPLINE initialization statement to define the characteristics of a single BSC line (and its respective adapter) that will be used by the JES3 global for remote job processing. You can also use this statement to assign a specific RJP work station, defined by the N parameter of an RJPTERM statement, to this line.
- ▶ RJPTERM (BSC Remote Job Processing Terminal): Use the RJPTERM initialization statement to define a single remote BSC work station to the JES3 system. This statement causes a default description to be provided for each work station device (printer, punch, or card reader) indicated by the PR, PU, or RD parameters along with the operating characteristics of the work station. If the JES3 default characteristics for a remote printer or punch device are not acceptable, a DEVICE statement should be coded to indicate desired characteristics. If a work station is to have the facilities of a JES3 operator console, then a CONSOLE statement must be coded.
- ▶ RJPWS (SNA Work Station Characteristics): Use the RJPWS initialization statement to describe each SNA work station's characteristics to the JES3 system. This statement causes a default description to be provided for each work station device (printer, punch, or card reader) indicated by the PR, PU, or RD parameter along with the operating characteristics of the work station.
- ▶ SELECT (Job Selection Mode): Use the SELECT initialization statement to define scheduling controls you want associated with a particular job selection mode. The initial job selection mode is assigned to a JES3 main using the SELECT parameter on the JES3 MAINPROC initialization statement. If a MAINPROC statement does not indicate a selection mode, the SELECT statement default values are assigned to that main. Each select mode defined can be dynamically changed using the *MODIFY,G,main,S operator command. In addition, the commands *MODIFY,G,main,G or *MODIFY,G,main,C can indirectly affect the select mode.

A SELECT statement must be specified for each select mode indicated on a MAINPROC statement or in a *MODIFY,G,main,S command.

- ▶ SETACC (Accessibility to Direct-Access Volumes): Use the SETACC initialization statement to identify those mains that normally have access to a permanently resident direct-access volume. The SETACC statement identifies the location of a volume on the uninitialized mains in a JES3 complex. SETACC prevents JES3 from setting up a job that needs the mounted volume until the main is initialized. When all mains are initialized or the volume is found, the SETACC definition is no longer used and normal JES3 management of the volume and device occurs. The devices on which the volumes reside are defined on a DEVICE statement with an XTYPE parameter and PR subparameter.
- ▶ SETNAME (Set JES3 Device Names): Use the SETNAME initialization statement to specify all user-assigned names and device type names associated with MDS-managed devices.
- ▶ SETPARAM (Set MDS Parameters): Use the SETPARAM initialization statement to specify parameters that the JES3 main device scheduler (MDS) and the DYNAL DSP uses in allocation, mounting, and deallocation of devices for jobs run on all mains. The SETNAME and DEVICE statements are used with the SETPARAM statements. SETNAME and DEVICE identify the devices to be managed by MDS. SETPARAM also indicates how MDS is to manage devices.
- ▶ SETRES (Mount Direct-Access Volumes): The SETRES statement identifies frequently used direct-access volumes which are not permanently resident. The SETRES statement specifies volumes which may reside on devices at main initialization time. When a specified volume is found to be present on an MDS-managed, removable, direct-access device during main initialization, the volume is considered mounted by MDS, without a MOUNT command being necessary.
- ▶ SOCKET: Use the SOCKET initialization statement to describe a TCP/IP socket connection that is used to communicate with an NJE node using the TCP/IP protocol.
- ▶ SPART (Spool Partition Definition): The SPART statement defines one spool partition and specifies:
 - The name of the partition
 - Whether JES3 is to use the partition as the default partition
 - Whether JES3 is to write initialization information to the named partition
 - Whether the partition is to overflow into another partition
 - The number of records in each track group
- ▶ STANDARDS (Installation Defaults and Standards): Use the STANDARDS initialization statement to specify default values for information not provided on other JES3 initialization statements or on the //FORMAT JES3 control statement. It also provides standards to be applied to all jobs entering the system.
- ▶ SYSID (Define the Default MVS/BDT Node): Use the SYSID initialization statement to define the default MVS/Bulk Data Transfer (BDT) node for this JES3 complex. If the JES3 complex includes one or more MVS/BDT facilities (program product 5665-302), you must include this statement in the JES3 initialization stream. JES3 submits MVS/BDT commands and transactions to the MVS/BDT node defined by this statement unless otherwise specified on the command or transaction.
- ▶ SYSOUT (SYSOUT Class Characteristics): Use the SYSOUT initialization statement to define SYSOUT class characteristics. The SYSOUT statement is required for each JES3 output class that requires other than TYPE=PRINT processing (JES3 initially sets all SYSOUT classes to TYPE=PRINT).
- ▶ TRACK (Preformatted Spool Data Set): Use the TRACK initialization statement to replace a corresponding FORMAT statement in an initialization stream after the spool data set specified by the FORMAT statement has been formatted. The TRACK statement indicates that the corresponding data set has been formatted.

SYSOUT data sets

The SYSOUT statement parameters are applicable to all SYSOUT data sets created by JES3. Also, be aware that if the SYSOUT is associated with an output descriptor that is defined by the OUTPUT JCL statement or TSO **OUTDES** command, and then the output characteristics are merged for SYSOUT on the HOLD queue.

Initialization stream checker (IATUTIS)

You use the JES3 initialization stream checker utility (IATUTIS) to test your JES3 initialization statements before you perform a hot start with refresh, warm, or cold start of JES3. The initialization stream checker detects most syntax errors and some logical errors in the initialization stream. You can run this utility as a batch job or as a TSO job using a command list (CLIST).

Steps to run the checker

To help avoid errors during JES3 initialization, JES3 provides a utility called the initialization stream checker. This utility simulates the JES3 initialization process and enables you to verify your initialization stream before actually initializing JES3. This utility scans the entire initialization stream for syntax errors and certain inconsistencies in the statements which would cause JES3 to either fail to initialize or to initialize with errors.

A sample JES3 initialization stream is shipped with JES3. The sample stream can be used to help a new JES3 user get started on creating an initialization stream.

Complete the following steps to use the initialization stream checker:

1. Gather data for the initialization stream checker.

Obtain hardware configuration data that the initialization stream checker uses to detect logical errors in the DEVICE, HWSNAME, RJPLINE, and SETNAME initialization statements. If you omit this step, the initialization stream checker performs only syntax checking.

If you want the initialization stream checker to check for logical errors in your initialization stream, first obtain the MVS configuration data by running the hardware configuration definition (HCD) program. Otherwise, you can omit this step.

2. Run the initialization stream checker.

The initialization stream checker examines all initialization statements for correct syntax, except the DYNALLOC statements, and creates the JES3 intermediate initialization tables.

When you have obtained the configuration data for each system in your complex, submit the JCL shown in Figure 3-40, which runs the initialization stream 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
//
```

Figure 3-40 JES3 sample JCL for IATUTIS

You must create this data for all processors that are defined in the initialization stream. Each member has the same name as one of these processors.

Dynamically changing the JES3 configuration

You can make changes to the JES3 configuration dynamically after the system is initialized and running by using the ***MODIFY,CONFIG** operator command.

The ***MODIFY,CONFIG** command allows you to add the following definitions to JES3 without having to restart the JES3 global and local address spaces:

- ▶ SNA RJP work stations and their associated consoles and devices
- ▶ Non-channel attached printers

You use ***MODIFY,CONFIG** command to specify the name of a member in the data set that is allocated to the JES3IN DD statement in the JES3 cataloged start procedure. This member contains the initialization statements that are associated with definitions that you want added to the JES3 configuration. The following initialization statements can be coded in the member specified on the ***MODIFY,CONFIG** command:

- ▶ RJPWS to define SNA/RJP workstation characteristics
- ▶ CONSOLE to define SNA/RJP console
- ▶ DEVICE to define SNA/RJP devices and non-channel attached FSS managed printers
- ▶ FSSDEF to define writer FSSs
- ▶ INTDEBUG to establish the Initialization Debugging Facility
- ▶ INCLUDE to include another initialization stream member

Note: If the JES3IN DD data set is concatenated, only the members in the first data set of the concatenation are used in processing the INCLUDE statement.

- ▶ DYNALLOC to dynamically allocate data sets
- ▶ BUFFER to define buffer spool parameters
- ▶ FORMAT to format data sets on a direct-access spool volume
- ▶ TRACK to define a pre-formatted data set on a direct-access spool volume or to replace a corresponding FORMAT statement
- ▶ SPART to define a spool partition
- ▶ OPTIONS to specify dump and job parameters
- ▶ ENDJSAM to indicate the end of JES3 spool initialization statements.

Attention: The ***MODIFY,CONFIG** command can be used only to dynamically change statements in the JSAM portion of the initialization stream (DYNALLOC, BUFFER, FORMAT/TRACK, SPART, and OPTIONS) as part of an update that includes the entire JSAM portion of the initialization stream, up to and including the ENDJSAM statement.

3.6.7 JES3 exits

Installation exits enable an installation to tailor JES3 without having to modify JES3 code. By not modifying JES3 code, you reduce the amount of work necessary to install JES3 maintenance or new JES3 function. Thus, installation exit routines can be very useful in helping you develop an operating system tailored to your needs. IBM supplies a module for each JES3 installation exit.

Note: Some IBM shipped exits perform a default function, but most are dummy modules.

You can, of course, write your own installation exit routine in place of any exit that is provided by IBM. For each exit, either the IBM-supplied routine or an installation exit routine must reside in the JES3 module library. Otherwise, JES3 issues the IAT3020 and IAT3102 warning messages.

Attention: All user modification made for exits should be implemented by using SMP/E standard installation program to avoid conflicts.

Table 3-8 provides a list of all available JES3 exits and their major functions in the system.

Table 3-8 JES3 available exits

Exit	Function
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
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

Exit	Function
IATUX30	Examine authority level for TSO/E terminal commands
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
IATUX42	TSO interactive data transmission facility screening and notification
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
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
IATUX73	STT record deletion disposition

3.6.8 JES3 start

Before jobs enter the system, the job entry subsystem (JES3) must be initialized to process work. Initialization of JES3 is the process of establishing control blocks used for JES3 processing. JES3 initialization takes place between the time that the **START JES3** command is issued and the time that the processing of jobs begins.

JES3 start procedure

The JES3 cataloged start procedure contains the job control language (JCL) statements that are needed to allocate the data sets required by JES3. IBM provides a basic cataloged start procedure that is shipped with JES3 and that is stored in the JES3 member of the SYS1.PROCLIB data set. Figure 3-41 shows a sample of the JES3 procedure, which contains all of the required JCL statements.

If you introduce an error while changing the procedure, JES3 cannot be restarted. In this case, you must use another system (for example, the starter system) to change the procedure.

```
//JES3 PROC JES=JES3, ID=01
//JES3 EXEC PGM=IATINTK,DPRTY=(15,15),TIME=1440,REGION=0M
//* -----
//* JES3 PROCEDURE: JES3
//*      CHKPNT:    SYS1.JES3CKPT
//*              SYS1.JES3CKP2
//*      DISK RDR:   SYS1.JES3DR
//*      JCT:       SYS1.JES3JCT
//*      SPOOL:     SYS1.JES3SPL1
//*      JES3OUT:   SYS1.JES3OUR
//*      DUMPS:     SYS1.JES3DUMP
//* -----
//CHKPNT   DD DSN=SYS1.&JES.CKPT,DISP=SHR          R,S
//CHKPNT2  DD DSN=SYS1.&JES.CKP2,DISP=SHR          O,S
//JES3DRDS DD DSN=SYS1.JES3DR,DISP=SHR            0
//JES3JCT  DD DSN=SYS1.&JES.JCT,DISP=SHR          R,D,S
//SP00L1   DD DSN=SYS1.&JES.SPL1,DISP=SHR          R,D,S
//JES3OUT  DD DSN=SYS1.&JES.OUT,DISP=SHR           R,D
//JES3SNAP DD DUMMY                               0,D
//SYSMDUMP DD DSN=SYS1.JES3DUMP,DISP=SHR
//IATPLBST DD DSN=SYS1.&PROCLIB1..PROCLIB,DISP=SHR R,D
//          DD DSN=SYS1.PROCLIB,DISP=SHR          R,D
//          DD DSN=ESA.SYS1.PROCLIB,DISP=SHR        R,D
//JESABEND DD DUMMY                               0,D
//JES3IN   DD DSN=SYS1.PARMLIB(JES3IN&ID),DISP=SHR R
//*
//*  R-Required, D-DYNALLOC eligible, S-Global/Locals share,
//*  O-Optional
//
```

Figure 3-41 JES3 start JCL

DD statements in procedure

JES3 includes the following cataloged start procedure DD-statements:

- ▶ CHKPNT and CHKPNT2: Defines the JES3 checkpoint data set or data sets. At least one of the two checkpoint data sets must be allocated and catalogued prior to JES3 operation. Each checkpoint data set must be allocated as a single extent that begins and ends on a cylinder boundary.
- ▶ JES3JCT: Defines the JES3 job control table (JCT) data set. This data set must be allocated and catalogued prior to JES3 operation. The data set must be large enough to accommodate the maximum number of JCT records to be allocated concurrently during normal system operation.
- ▶ spool1 to spoolnn: Defines the spool data sets. The installation selects the ddnames and data set names for these statements. The ddname for this statement must be the same ddname specified on the BADTRACK, FORMAT, or TRACK initialization statements. Spool data sets must be allocated and catalogued prior to JES3 operation. (These data sets may be any size; however, a minimum of 100 cylinders is recommended.)
- ▶ JES3OUT: Defines the data set upon which the JES3 initialization stream and initialization error messages are printed. This data set is de-allocated after initialization completes. You can tailor the block size (BLKSIZE) and logical record length (LRECL) values to improve performance. The values you can specify are device-dependent.
- ▶ JES3SNAP: Defines the data set used if JES3 produces a dump during a hot start, hot start with analysis, hot start with refresh, hot start with refresh and analysis, warm start, or warm start with analysis. This data set contains important diagnostic information. The information will not be available if you define JES3SNAP as a dummy data set.
- ▶ JESABEND: Defines the data set used for a JES3 formatted dump. If omitted, a formatted dump of JES3 control information will not be produced.
- ▶ SYSABEND, SYSUDUMP, or SYSMDUMP: Defines the data set for JES3 system dumps.
- ▶ IATPLB_n: Defines the installation's standard procedure library concatenation.

Note: C/I functional subsystems (C/I FSS) and the PROCLIB update function obtains unit and volume information for the procedure libraries from the catalog. For these functions, JES3 ignores unit and volume information that you specify in the JES3 start-up procedure or on a DYNALLOC initialization statement.

- ▶ IATPLB_n: Defines the installation's other procedure library concatenation.

Note: If a data set is dynamically allocated as both a JES3 DISKRDR data set and a JES3 PROCLIB data set, the UPDATE= parameter on the JES3 ///*MAIN statement (JES3 procedure library update facility) cannot be used to move the data set.

- ▶ JES3IN: Defines the data set containing the JES3 initialization stream. This data set must be a blocked or unblocked partitioned data set. The default initialization stream is read from SYS1.SIATSAMP(member JES3IN00).
- ▶ JES3DRDS: Defines the partitioned data sets containing input for the JES3 disk reader facility. The maximum block size for this data set is 3200. Concatenated data sets can be used.

Be aware that JES3 does not hold any data set ENQUEUE (major name=SYSDSN, minor name=dsname) while it is running regardless of the type of allocation (JCL or dynamic). JCL allocation ENQUEUEs are based on the DSI subparameter of the PPT parameter of the

SCCHEDxx member of SYS1.PARMLIB. Dynamic allocations by JES3 use an equivalent parameter to prevent a data set ENQUEUE.

Depending on the type of start the operator specifies, JES3 initialization is a three or four phase process as follows:

- ▶ Phase 1 determines the types of starts that are allowed for the main and asks the operator to select a start type.
- ▶ Phase 2 reads the statements from the initialization stream that initialize the control blocks to manage spool space and, in the event of a restart, validates jobs that are in the JES3 job queue.
- ▶ Phase 3 reads the initialization stream and converts information supplied on each initialization statement to intermediate spool files. These intermediate spool files are written to spool.
- ▶ Phase 4 uses the intermediate spool files built in phase 3 to build the final control blocks necessary for job execution. Phase 4 informs the operator that JES3 initialization is complete.

3.6.9 JES3 start types

The type of start you select depends on why you need to start or restart JES3. During phase 1 of JES3 initialization, a message is issued prompting the operator to enter the start type. The operator responds with a C, W, WA, WR, WAR, H or HA to initialize the processor that will contain the JES3 global address space.

```
IAT3011 SPECIFY JES3 START TYPE  
*100 IAT3011 (C, L, H, HA, HR, HAR, W, WA, WR, WAR, OR CANCEL)
```

After the global is initialized, each additional processor can be initialized by issuing the **START JES3** command on that processor. When JES3 issues a message for the start type, the operator enters an L to initialize each subsequent processor as a local.

Hot start (H)

Use a hot start to start JES3 on the global:

- ▶ After an orderly shutdown
- ▶ After a JES3 failure on the global from which JES3 cannot automatically recover
- ▶ After an MVS failure that terminates all functions in the global
- ▶ To replace one of your checkpoint data sets.

Initialization is used to restart the JES3 global processor after an orderly shutdown or after JES3 ends abnormally. The initialization stream is not reread, therefore the initialization parameters remain the same as before JES3 ended. A hot start need not be preceded by an MVS IPL of the global and local processors. Most job processing resumes after a hot start (IPL) or continues through the hot start (no IPL).

Hot start with analysis (HA)

Use hot start with analysis after a hot start fails or if you suspect problems with the JES3 job queue. A hot start with analysis performs the same functions as a hot start. In addition, JES3 also:

- ▶ Performs further job validation
- ▶ Gives the operator an opportunity to delete invalid jobs.

You are not required to IPL or to restart the local mains, although you can optionally do so.

Hot start with refresh (HR)

Use hot start with refresh when you want to change the initialization stream without having to IPL the entire complex. Performing a hot start with refresh avoids disrupting your system since you need only restart the JES3 global address space. You do not need to complete the IPL process on all processors in the complex as you do with a warm start.

Hot start with refresh and analysis (HAR)

Use hot start with refresh and analysis after a hot start with refresh fails and/or you suspect problems with the JES3 job queue. A hot start with refresh and analysis performs the same functions as a hot start with refresh.

Warm start (W)

Use a warm start to restart JES3 on the global:

- ▶ After either type of hot start or hot start with refresh fails.
- ▶ After a failure of the global because of a software/hardware failure.
- ▶ When you want to change the initialization stream and the changes cannot be performed with a hot start with refresh.

Warm start with analysis (WA)

Use warm start with analysis when JES3 terminates abnormally and you suspect problems with the JES3 job queue, or when you change the level of JES3 and want to verify the integrity of the JES3 job queue across the change.

Warm start to replace a spool data set (WR)

Use warm start to replace a spool data set when you want to replace a spool data set. This type of start performs the same function as a warm start in addition to allowing you to replace a spool data set.

Warm start with analysis (WAR)

Use a warm start with analysis to replace a spool data set when you suspect problems with the JES3 job queue and you want to replace a spool data set. This type of start performs the same function as a warm start.

Note: During JES3 hot starts and warm starts, JES3 performs validation checking on the jobs that were active when the system ended. JES3 examines the control blocks for the active jobs and if the information in the control blocks is sufficient for the jobs to continue, JES3 allows the jobs to resume processing. If not, JES3 asks the system operator whether or not the job should be cancelled. Depending on how critical the job is, the operator might have to stop JES3 initialization and then restart JES3.

Cold start (C)

Use a cold start to start JES3 on the global when all types of warm starts are unsuccessful.

Local start (L)

Use a local start to start JES3 on a local main.

Hot start considerations

You need to be careful when changing the initialization stream during a hot start with refresh, because not all of the initialization statements are processed. If you add or change one of the initialization statements that is not processed during a hot start with refresh, errors can occur

when it is referenced by another initialization statement that is processed during a hot start with refresh.

Figure 3-42 shows that during a hot start with refresh, because of dependencies between statements, many statements and parameters on statements cannot be changed. This summary shows the initialization statements that are processed during a hot start with refresh. The A, S, and N indicate what statements are affected.

Initialization statements that can change during HR JES3 start			
ACCOUNT (A)	BUFFER (S,N)	CIPARM (A,N)	CLASS (A,N)
COMMDEFN (A)	COMPACT (A)	CONSOLE (A,N)	DEADLINE (A)
DESTDEF (A)	DEVICE (A,N)	DYNALDSN (A)	DYNALLOC (A)
ENDINISH (A)	ENDJSAM (A)	FSSDEF (A,N)	GROUP (A,N)
HWSNAME (A,N)	INCLUDE (A)	INTDEBUG (A)	MAINPROC (S,N)
MSGROUTE (A)	NJECONS (A)	NJERMT (S,N)	NETSERV (A)
OPTIONS (S,N)	OUTSERV (A)	RESCLBK (A)	RESDSN (A,N)
RJPLINE (A)	RJPTERM (A)	RJPWS (A)	SELECT (A)
SETACC (A,N)	SETNAME (A,N)	SETPARAM (S,N)	SETRES (A)
SOCKET (A)	STANDARDS (S,N)	SYSID (A)	SYSOUT (A,N)

A - All parameters S - Subset of parameters N - See notes in JES3 Initialization and Tuning Reference

Statements that require C or W JES3 start to change
BADTRACK (Bypass Defective Tracks)
CONSTD (Console Service Standards)
FORMAT (Format Spool Data Set)
SPART (Spool Partition Definition)
TRACK (Preformatted Spool Data Set)

Figure 3-42 JES3 HR start changes

See *z/OS JES3 Initialization and Tuning Reference*, SA22-7550 for the specific statements and the restrictions in place concerning the dependencies.

3.6.10 JES3 Stop

Before you remove a local main for maintenance or other reasons, allow processing jobs to complete normally. Use the following steps:

- ▶ Enter a *F, V,main,OFF command for the local main to prevent JES3 from scheduling any further jobs on the processor.
- ▶ Enter the *RETURN command to end JES3 after all jobs on the local main have completed processing.
- ▶ Enter the HALT EOD command on the local main to ensure that important statistics and data records in storage are not permanently lost.

The installation might not want to wait for all jobs to complete normally. Jobs will not end because of system problems (hardware and software). An IPL or JES3 restart is scheduled to take place at a predetermined time and jobs will not be able to complete. In this case, you must make the decision to delay the IPL or to cancel the jobs. (Your installation procedures should tell you whether to cancel jobs or delay an IPL.)

Note: Enter the **Z EOD** command only if you are performing an IPL or SYSTEM RESET, not to restart JES3.

Before stopping the global, you should stop the local mains as described above. You should stop all JES3 processing by entering the *F,Q,H command, which puts all jobs in hold status before stopping JES3. System initiators, printers, and punches do not begin any new work and become inactive after completing their current activity. Jobs in JES3 queues remain in their current position. You should also queue the system log for printing by entering the WRITELOG command. This prevents log messages from being lost if you later restart JES3 with a hot start. After all system activity has completed, enter the ***RETURN** command to end JES3.

After you enter the ***RETURN** command, enter the **HALT EOD** command to ensure that important statistics and data records in storage are not permanently lost. As a result of this command, the internal I/O device error counts are stored in the SYS1.LOGREC data set; the SMF buffers are emptied onto one of the SYS1.MANx data sets; and the system log, if still active, is closed and put on the print queue. When these actions are complete, the system issues message IEE334I, stating that the HALT EOD operation was successful. See *z/OS MVS System Commands*, SA38-0666 for further information about stopping MVS™.



LPA, LNKLST, and authorized libraries

To maximize the performance for retrieving executable programs, z/OS is designed to maintain in memory those executable programs that are needed for fast response to requests coming from z/OS components and from critical applications. Link pack area (LPA), LNKLST, and authorized program facility (APF) libraries described here are the cornerstone of the executable program fetching process. Also the roles of Virtual Lookaside Facility (VLF) and Linklist Lookaside (LLA) components are described at the appropriate level.

Executable programs, whether stored as load modules (coming from a PDS member) or program objects (coming from a PDSE member), must be loaded into both virtual storage and central storage before they can be executed. When one executable program calls another one, either directly by asking for it to be executed or indirectly by requesting a system service that uses it, it does not begin to run instantly. How long it takes before a requested executable program begins to run depends on where in its search order z/OS finds a usable copy, and on how long it takes z/OS to make the copy it finds available.

You should consider the following factors when deciding where to place individual executable programs or libraries containing multiple such programs in the system-wide search order:

- ▶ The search order z/OS uses for executable programs
- ▶ How placement affects virtual storage boundaries
- ▶ How placement affects system performance
- ▶ How placement affects application performance

4.1 The creation of an executable program

Figure 4-1 shows the steps when creating an executable program.

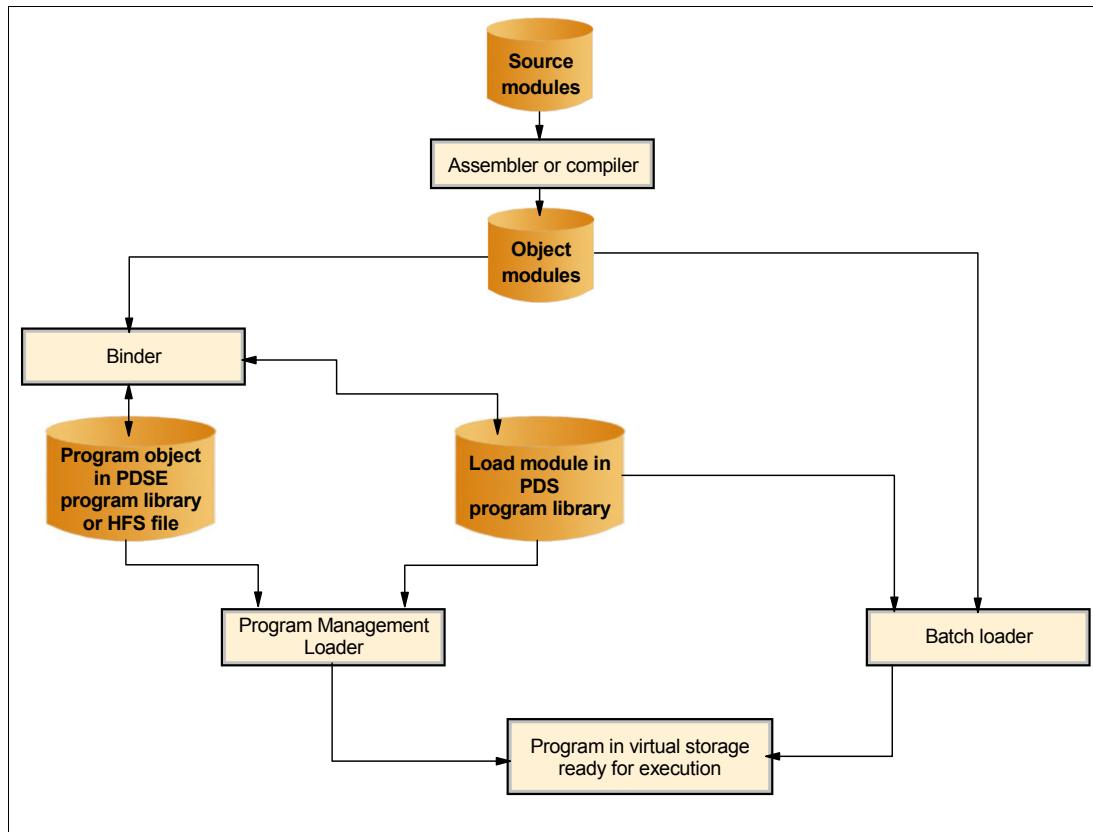


Figure 4-1 The creation of an executable program

The first step in creating an executable program is to write the source code. This code can use a high-level language compiler, such as Cobol, PL/I, C, or the assembler, usually referred to as *high-level assembler*. The output of the compiler or the assembler is called an object module. The source statements are already converted to machine instructions, but such code is not ready to be executed.

The second step is to process the object module by a z/OS utility program called the *Binder*. The Binder, following information contained in the object module, creates an executable program. If required, the binder links the module with other object modules, and also resolves relative address references between object modules. The format and the name of the Binder's output, the executable program, depends on the type of library used to store the program, as follows:

- ▶ Load module, if stored in a PDS
- ▶ Program object, if stored in a PDSE

Generally, use PDSEs instead of a PDS. Program object format has several advantages when compared with load modules, and PDSEs are more flexible and efficient than a PDS.

In this chapter and in many manuals the expression *executable program* is translated into load module, even if the code is stored in a PDSE. Here, we use the expression *program object* only to make a remark about that specific type of executable program.

The last step is to load or fetch the executable program into virtual and central storage. This task is done by the a z/OS program named *Loader*, which belongs to the z/OS program management component. During the load, some address variables are resolved by adding to them the virtual address where they are loaded.

Program execution considerations

One variation of this process is during testing the logic of the code. You can use the batch loader where the executable program is stored directly into virtual storage, and it is then ready to be executed.

If the language that you select is interpretative, such as Java, the picture is different. There is no compilation, and the running program is the Java interpreter, also called *JVM*. It reads the Java source and translates it into machine code. It then executes the machine instructions. When a CPU gets control back, it translates the next source statement and so on.

4.1.1 Program load order

When a program is requested through a system service (such as LINK, LOAD, XCTL, or ATTACH) using default options, the system searches for it in the following sequence:

1. Job pack area (JPA)

A program in a JPA is already loaded in the requesting address space. If the copy in the JPA can be used, it is used. Otherwise, the system either searches for a new copy or defers the request until the copy in the JPA becomes available. (For example, the system defers a request until a previous caller is finished before reusing a serially-reusable module that is already in the JPA.)

2. TASKLIB

A program can allocate one or more data sets to a TASKLIB concatenation. Data sets concatenated to TASKLIB are searched for after the JPA but before any specified STEPLIB or JOBLIB. Modules loaded by unauthorized tasks that are found in TASKLIB must be brought into a private area virtual storage before they can run. Modules that have previously been loaded in common area virtual storage (LPA modules or those loaded by an authorized program into CSA) must be loaded into a common area virtual storage before they can run.

For more information about TASKLIB, see *z/OS MVS Programming: Assembler Services Guide*, SA23-1368.

3. STEPLIB or JOBLIB

STEPLIB and JOBLIB are specific DD names that can be used to allocate data sets to be searched ahead of the default system search order for programs. Data sets can be allocated to both the STEPLIB and JOBLIB concatenations in JCL or by a program using dynamic allocation. However, only one or the other is searched for modules. If both STEPLIB and JOBLIB are allocated for a particular jobstep, the system searches STEPLIB and ignores JOBLIB. Any data sets concatenated to STEPLIB or JOBLIB are searched after any TASKLIB but before the LPA. Modules found in STEPLIB or JOBLIB must be brought into private area virtual storage before they can run. Modules that have previously been loaded in common area virtual storage (LPA modules or those loaded by an authorized program into CSA) must be loaded into common area virtual storage before they can run. For more information about JOBLIB and STEPLIB, see *z/OS MVS JCL Reference*.

4. LPA, which is searched in this order:
 - a. Dynamic LPA modules, as specified in PROGxx members
 - b. Fixed LPA (FLPA) modules, as specified in IEAFIXxx members
 - c. Modified LPA (MLPA) modules, as specified in IEALPAXxx members
 - d. Pageable LPA (PLPA) modules, loaded from libraries specified in LPALSTxx (LPA library list) or PROGxx

LPA modules are loaded in common storage, shared by all address spaces in the system. Because these modules are reentrant and are not self-modifying, each can be used by any number of tasks in any number of address spaces at the same time. Modules found in LPA do not need to be brought into virtual storage, because they are already in virtual storage.

5. Libraries in the linklist, as specified in PROGxx and LNKLSTxx.

By default, the linklist begins with SYS1.LINKLIB, SYS1.MIGLIB, SYS1.CSSLIB, SYS1.SIEALNKE, and SYS1.SIEAMIGE. However, you can change this order using SYSLIB in PROGxx and add other libraries to the linklist concatenation. The system must bring modules found in the linklist into private area virtual storage before the programs can run.

Programs already loaded into your memory

Before this search order, the system searches among the programs that have already been loaded into the job's memory. The TSO session counts as a job. There can be multiple tasks running within the job. In that case, the system first searches the program modules that belong to the task where the request was issued, and after that it searches those that belong to the job itself. Relevant jargon: a task has a Load List, which has Load List Elements (LLE); the job itself has Job pack area (JPA).

If a copy of the module is found in your job's memory, in either the task's LLE or the job's JPA, that copy of the module will be used if it is available.

4.2 Link pack area

The *link pack area* (LPA) is a part of an address space's common area storage and is divided into pageable, fixed, and modified sections (see Figure 4-2). It is built at IPL time by loading load modules contained in SYS1.LPALIB and its concatenations. There are two LPAs (one above and another below the 16M line), depending on the residence mode of the load module. Because the load modules are in the common area, they are shared by all address spaces in the system. The load modules must be *refreshable*, meaning that they are not self-modifying. In this context, a *reentrant* load module means that it can be executed concurrently by several tasks, but it cannot be temporarily modified. Then, each copy can be used by any number of tasks running in any number of CPUs in any number of address spaces at the same time. Modules found in LPA do not need to be brought into virtual storage on request because they are already in virtual storage. The benefit of having lots of load modules in LPA is that it decreases the use of the private areas.

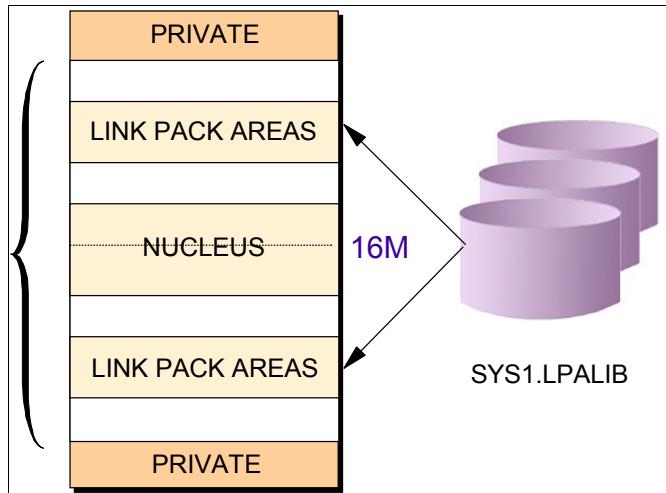


Figure 4-2 The link pack area

Reentrant considerations

Basically all module placed in any LPA must be linked with the reentrant Attribute. If the module is marked as reentrant if the program has a flag set to promise that it does not modify itself at all. Then any number of tasks can use the same copy simultaneously, so you always get it. Figure 4-3 shows an example of modules that are linked reentrant.

Name	Prompt	Alias-of	---- Attributes ----
ACYAPCIP		ACYAPCNP	RN RU
ACYAPCNP			RN RU

Figure 4-3 Example of reentrant Modules

FLPA and central storage

Modules placed anywhere in LPA are always in virtual storage and sometimes in central storage. Observe that the load module page that contains the instructions in execution must be in a central storage frame. Whether modules in LPA are in central storage depends on how often they are used by all the users of the system, and on how much central storage is available. The more often an LPA module is used, and the more central storage is available on the system, the more likely it is that the pages containing the copy of the module are in central storage at any given time, without suffering page faults. When all the pages that contain an LPA module (or its first page) are not in central storage when the module is called, the module begins to run only after its first page has been brought into central storage.

LPA pages can be stolen but never paged out because there are always valid copies in the LPA page data set (loaded at IPL time). Recall that LPA pages are not altered.

Many z/OS components' code is loaded in PLPA, such as SVC type 3 and 4 routines, access methods (VSAM and QSAM), initiators, and JES load modules.

Modules in CSA

Modules can also be loaded into CSA by authorized programs. An example of this is the implementation of dynamic LPA, where after an IPL, new load modules are added to LPA. Because the borders of LPA cannot be modified and CSA is a getmainable common virtual storage area, the new load modules are fetched into CSA. When modules are loaded into CSA and shared by multiple address spaces, the performance considerations are similar to those for modules placed in LPA. There are pieces of CSA that are pageable and pieces of

CSA that are fixed, depending on the subpool number used during a getmain of the storage area.

4.3 LPA subareas

The link pack area is a section of the common area of an address space. It exists below the system queue area (SQA) and consists of the pageable link pack area (PLPA), then the fixed link pack area (FLPA) if one exists, and finally the modified link pack area (MLPA).

For more information, see *z/OS MVS Initialization and Tuning Reference*, SA22-7592.

Each component of LPA has a counterpart in the extended common area (that is, above the 16M line) as follows:

FLPA For integrity reasons, there are z/OS functions that cannot suffer a page fault. If this function is implemented through LPA load modules, these load modules' pages must be page fixed (no page steals). At IPL time, the installation can declare those load modules' names, in the IEAFIXxx parmlib member, and the IPL process loads them in the fixed link pack area (FLPA), which is a non pageable area of storage. The FLPA exists only for the duration of an IPL. Therefore, if an FLPA is desired, the modules in the FLPA must be specified for each IPL (including quick-start and warm-start IPLs).

Modules placed in the FLPA must be refreshable, that means they are never altered during execution.

It is the responsibility of the installation to determine which modules, if any, to place in the FLPA. Note that if a module is heavily used and is in PLPA, the system's paging algorithms will tend to keep that module in central storage.

PLPA During an IPL, if the CLPA option is on, the SYS1.LPALIB member and its concatenated data sets, as defined in the LPALSTxx parmlib member, are copied into the PLPA page data set and consequently into virtual storage.

In the PLPA page data set, ASM maintains records that have pointers to the PLPA and extended PLPA pages (above 16M). During quick-start or warm-start IPLs, the system uses the pointers to locate the PLPA and extended PLPA pages. The system then rebuilds the PLPA and extended PLPA page tables, and uses them for the current IPL.

If CLPA is specified at the operator console during the IPL, indicating a cold start is to be performed, the PLPA storage is deleted and made available for system paging use. A new PLPA and extended PLPA are then loaded, and pointers to the PLPA and extended PLPA pages are recorded in the PLPA page data set.

MLPA This LPA area can be used to contain reentrant load modules from APF-authorized libraries that are part of the PLPA during the current IPL. The MLPA exists only for the duration of an IPL. Therefore, if an MLPA is desired, the modules in the MLPA must be specified in the IEALPAXxx parmlib member for each IPL, including quick-start and warm-start IPLs. MLPA is allocated just below the FLPA, or the PLPA if there is no FLPA. The extended MLPA is allocated above the extended PLPA, or the extended PLPA if there is no extended PLPA. When the system searches for a load module, MLPA is searched before PLPA.

Note: The MLPA can be used at IPL time to temporarily modify or update the PLPA with new or replacement modules. No actual modification is made to the quick start PLPA stored in the system's paging data sets. The MLPA is read-only, unless NOPROT is specified on the MLPA system parameter.

4.4 Pageable link pack area (PLPA/extended PLPA)

The PLPA (Figure 4-4) is an area of common storage that is loaded at IPL time (when you perform a cold start, specify CLPA in your IPL).

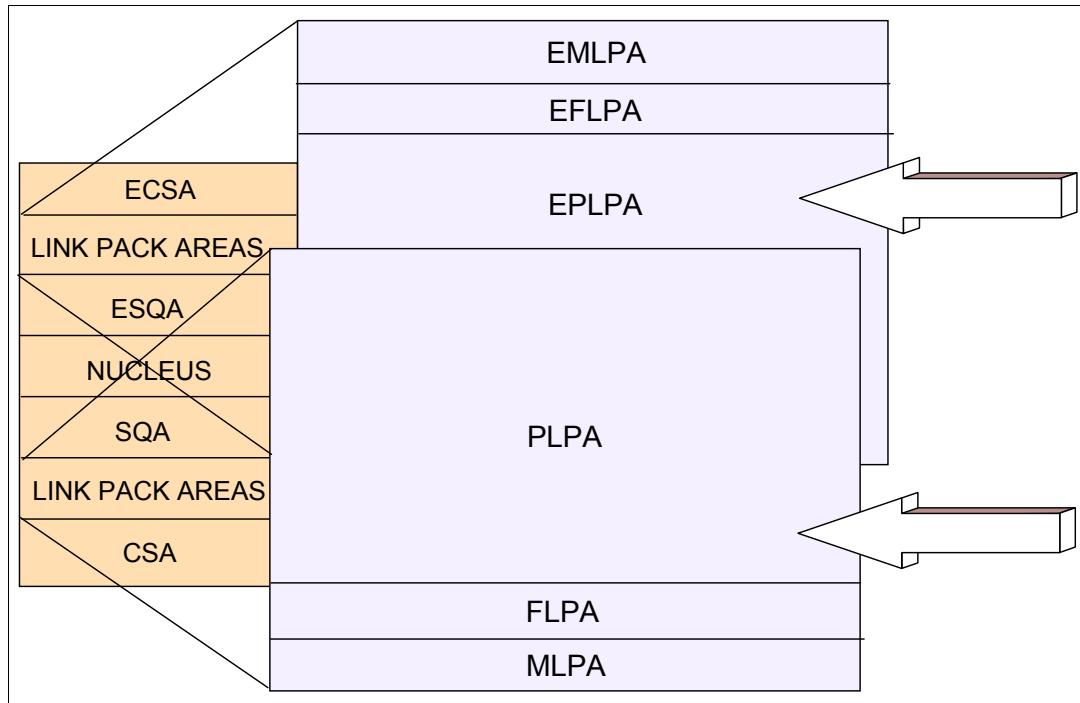


Figure 4-4 Pageable link pack area

This area contains SVC routines, access methods, and other read-only system programs, along with any read-only reentrant user programs selected by an installation that can be shared among users of the system. It is also desirable to place all frequently used refreshable SYS1.LINKLIB and SYS1.CMDLIB modules in the PLPA.

Load modules AMODE and RMODE

A program module has a residence mode, RMODE, assigned to it, and each entry point and alias has an addressing mode, AMODE, assigned to it. You can specify one or both of these modes when creating a program module or you can allow the binder to assign default values.

Contents of the PLPA or extended PLPA

The PLPA and extended PLPA contain all members of SYS1.LPALIB and any other libraries that are specified in the active LPALSTxx through the LPA parameter in the IEASYSxx or from the operator's console at system initialization. The modules in the PLPA or extended PLPA must be refreshable. Load modules are placed in the PLPA or extended PLPA, depending on the RMODE of these load modules. Load modules with an RMODE of 24 are placed in the PLPA, while those with an RMODE of ANY are placed in the extended PLPA.

4.5 LPA parmlib definitions

The LPA library list, or LPALSTxx member, is used to add read-only, reentrantable user programs to the pageable link pack area. To identify which LPALSTxx members the system should use, specify the member suffixes on the LPA parameter in IEASYSxx or as a parameter entered during IPL. Example 4-1 shows LPA parmlib definitions.

The two characters (A through Z, 0 through 9, @, #, or \$) represented by 00 (or aa and so forth) in Example 4-1 are appended to LPALST to form the name of the LPALSTxx parmlib members.

If the L option is specified, the system displays the contents of the LPALSTxx parmlib members at the operator's console as the system processes the members.

Example 4-1 LPA parmlib definitions

```
LPA= {aa          }
      {(aa,bb,...[,L])}
(YEASYSxx parameter)
```

The LPA parameter is effective only during cold starts or during IPLs in which you specify the CLPA option. The LPA parameter does not apply to modules requested through the MLPA option.

Note: The z/OS load option CLPA is now the default because of the performance of the most current disk subsystems.

An example of overriding the value of the LPA parameter in the IEASYSxx during system initialization can be seen in Example 4-2.

Example 4-2 Overriding the LPA parameter value

```
IEA101A SPECIFY SYSTEM PARAMETERS FOR z/OS 02.03.00 HBB77B0
r 00,LPA=03
IEE600I REPLY TO 00 IS;LPA=03
```

LPALST03 (shown as LPA-03) was selected during system initialization.

How to specify modules in the PLPA or extended PLPA

Use one or more LPALSTxx members in the SYS1.PARMLIB to concatenate your installation's program library data sets to SYS1.LPALIB. You can also use the LPALSTxx member to add your installation's read-only reentrantable user programs to the pageable link pack area (PLPA). The system uses this concatenation, which is referred to as the LPALST concatenation, to build PLPA. The word "library" commonly used in z/OS manuals means a PDS or a PDSE data set organization used to contain executable programs.

Note: The system does not check user catalogs when it searches for data sets to be added to the LPALST concatenation. Therefore, the data sets in the concatenation must be catalogued in the system master catalog.

During nucleus initializing process (NIP), the system opens and concatenates each data set specified in LPALSTxx in the order in which the data set names are listed, starting with the first specified LPALSTxx member.

If one or more LPALSTxx members exist, and the system can open the specified data sets successfully, the system uses the LPALST concatenation to build the PLPA (during cold starts and IPLs that included the CLPA option). Otherwise, the system builds the PLPA from only those modules named in the SYS1.LPALIB.

Note: The LPALST concatenation can have up 255 extents. If you specify more data sets than the concatenation can contain, the system truncates the LPALST concatenation and issues messages that indicate which data sets are excluded in the concatenation.

4.6 Coding a LPALSTxx parmlib member

Figure 4-5 presents an example of parmlib members of LPALSTxx.

Some important syntax rules for the creation of LPALSTxx are:

- ▶ On each record, place a string of data set names separated by commas.
- ▶ If a data set is not cataloged in the system master catalog, but is cataloged in a user catalog, specify in parentheses immediately following the data set name the one to six character VOLSER of the pack on which the data set resides.
- ▶ Indicate continuation by placing a comma followed by at least one blank after the last data set name on a record.

SYS1.SICELPA,	* SORT
SYS1.SORTLPA,	* SORT
CEE.SCEELPA,	* LE
TCPIP.SEZALPA,	* TCP/IP
SYS1.REXX.SEAGLPA,	* REXX RUNT
ISP.SISPLPA	* ISPF

Figure 4-5 LPALSTxx parmlib member example

To become you active during IPL of the system you have to add LPA=xx statement to your active IEASYSxx member. The name of one or more parmlib members (LPALSTxx) that contain names of data sets that are to be concatenated to SYS1.LPALIB for building the pageable LPA (PLPA and extended PLPA).

4.7 Fixed link pack area

The FLPA (Figure 4-6 on page 120) is loaded at IPL time, with the load modules listed in the active IEAFIXxx member of SYS1.PARMLIB. These load modules must be kept in fixed storage frames (that is, they cannot be paged out).

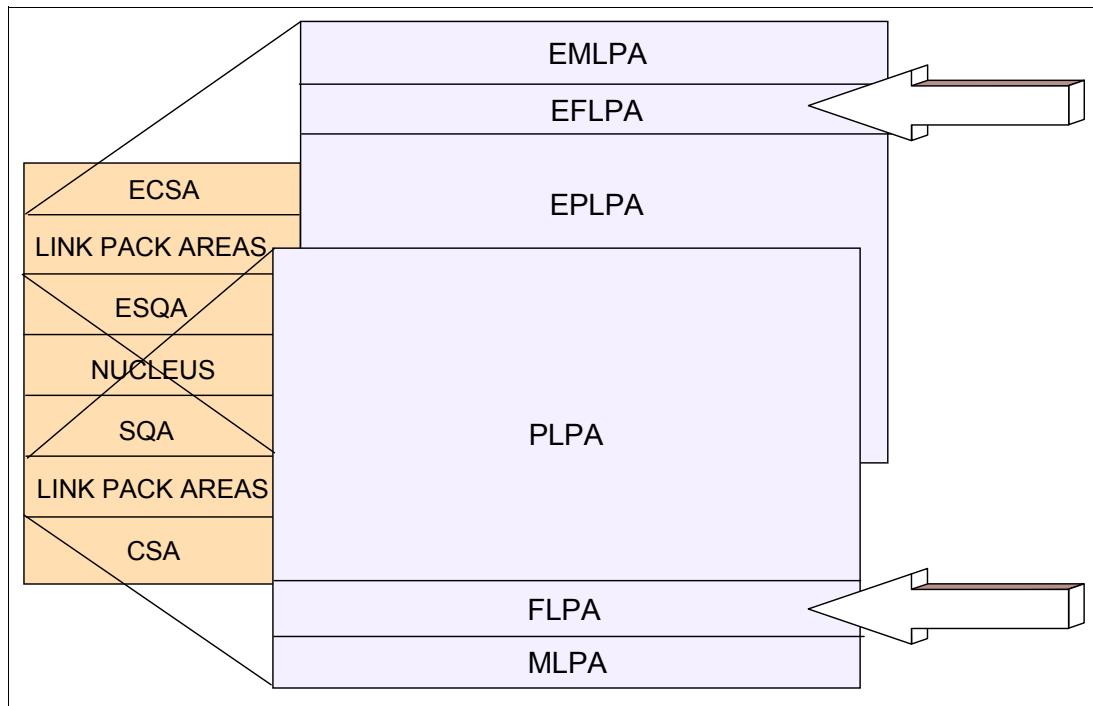


Figure 4-6 Fixed link pack area

This area should be used only for load modules that should not allow page faults because of integrity considerations. Modules placed in the FLPA must be reentrant and refreshable.

Contents of the FLPA or extended FLPA

Load modules from the LPALST concatenation, the LNKLST concatenation, SYS1.MIGLIB, and SYS1.SVCLIB can be included in the FLPA. FLPA is selected through specification of the FIX parameter in IEASYSxx, which is appended to IEAFIX to form the IEAFIXxx parmlib member, or from the operator's console at system initialization.

Load modules specified in IEAFIXxx are placed in the FLPA or extended FLPA, depending on the RMODE of these load modules: load modules with an RMODE of 24 are placed in the FLPA, while those with an RMODE of ANY are placed in the extended FLPA.

4.8 Coding the IEAFIXxx member

Use the IEAFIXxx member of SYS1.PARMLIB to specify the names of modules to be fixed in storage for the duration of an IPL (as described in the sections that follow).

Specifying modules in the FLPA or extended FLPA

Figure 4-7 on page 121 shows an example of IEAFIXxx parmlib member. Use the IEAFIXxx member in SYS1.PARMLIB to specify the names of the load modules that are to be fixed in central storage for the duration of an IPL. Load modules specified in IEAFIXxx are loaded and fixed in the order in which they are specified in the member. To keep search time within reasonable limits, do not allow the FLPA to become excessively large. For more information, see *z/OS MVS Initialization and Tuning Reference*, SA22-7592.

```

INCLUDE LIBRARY(SYS1.LPALIB) MODULES(
    IEAVAR00          /* RCT INIT/TERM           */
    IEAVAR06          /* RCT INIT/TERM ALIAS     */
    IGC0001G          /* RESTORE(SVC17)          */
)
INCLUDE LIBRARY(FFST.SEPWMOD2) MODULES(
    EPWSTUB
)

```

Figure 4-7 The IEAFIXxx parmlib member

Note: Catalog these libraries in the system master catalog.

Coding IEAFIXxx in SYS1.PARMLIB

The statements or parameters used in IEAFIXxx are:

INCLUDE	Specifies load modules to be loaded as temporary extensions to the existing PLPA.
LIBRARY	Specifies a qualified data set name. The specified library must be cataloged in the system master catalog.
MODULES	Specifies a list of 1 to 8 character load module names.

Note: The advantage of using the FLPA is the reduction in the central storage available for paging old jobs and starting new jobs. Remember that pages referenced frequently tend to remain in central storage, even when they are not fixed.

4.9 Specifying the IEAFIXxx member

The two characters (A through Z, 0 through 9, @, #, or \$) represented by aa (or bb and so forth), are appended to IEAFIX to form the name of the IEAFIXxx parmlib members. The options are:

L	If the L option is specified, z/OS displays the contents of IEAFIXxx parmlib members at the operator's console as the system processes the members.
NOPROT	By default, the LPA load modules in the IEAFIXxx parmlib members are page-protected in storage. It means that the protection key in central storage is B'0000' (four zero-bits), and usually the PSW key is B'1000'; you have a protection exception when altering the code. However, the NOPROT option allows an installation to override the page protection default.

Syntax format for FIX parameter: Use the following syntax for the FIX parameter:

```
FIX= {aa} {(aa,[,L][,NOPROT]) } {(aa,bb,...[,L][,NOPROT])}
```

Figure 4-8 shows an example of overriding the value of the FIX parameter that was specified in the IEASYSxx during system initialization. Member 01 (FIX-01) is selected during system initialization.

```
IEA101A SPECIFY SYSTEM PARAMETERS FOR z/OS 02.03.00 HBB77B0
R 00, FIX=01
IEE600I REPLY TO 00 IS;FIX=01
```

Figure 4-8 Overriding the FIX parameter

4.10 Modified link pack area (MLPA)

This area (shown in Figure 4-9) can be used to contain reentrantable load modules from APF-authorized libraries (see 4.21, “Authorized program facility (APF)” on page 138) that are to be part of the pageable extension to the link pack area during the current IPL.

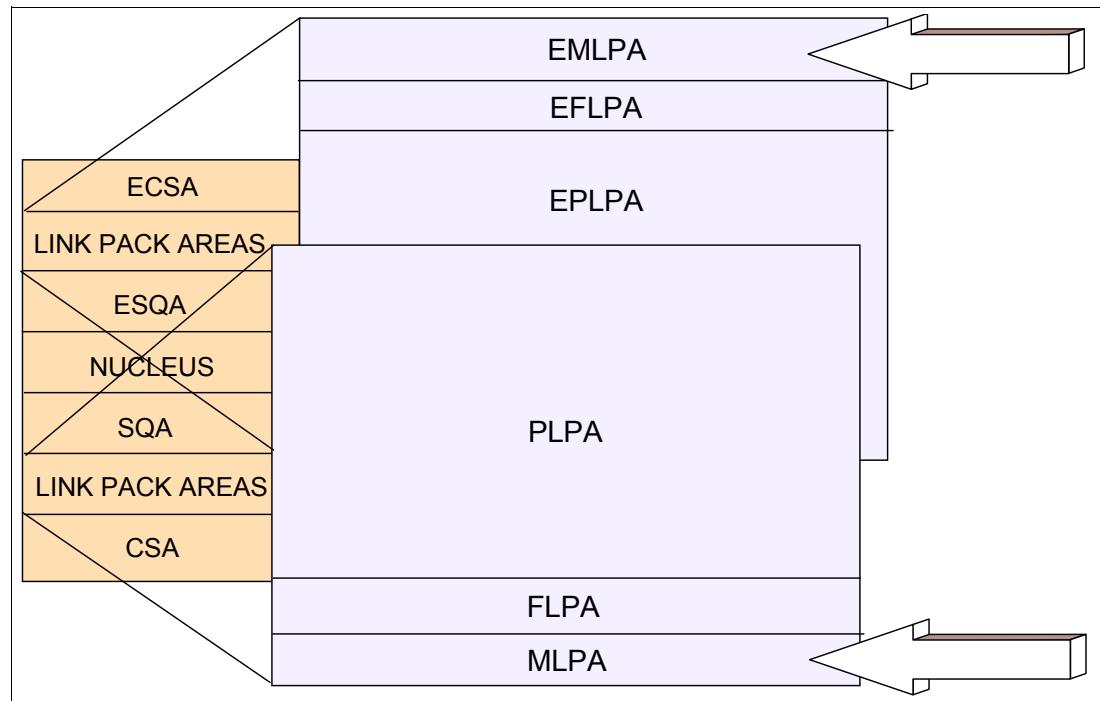


Figure 4-9 Modified Link Pack Area

The MLPA exists only for the duration of the IPL. Therefore, if an MLPA is desired, the load modules in the MLPA must be specified for each IPL (including quick start and warm start IPLs).

Contents of the MLPA or extended MLPA

MLPA contains load modules listed in the active IEALPAXx member of SYS1.PARMLIB, through the specification of the MLPA parameter in the IEASYSxx or from the operator's console at system initialization.

LPA load modules specified in the IEALPAXx are placed in the MLPA or the extended MLPA, depending on the RMODE of the load modules. Load modules with an RMODE of 24 are placed in the MLPA, and those with an RMODE of ANY are placed in the extended MLPA.

4.11 Coding the IEALPAxx member

Figure 4-10 is an example of the IEALPAxx parmlib member. Use the IEALPAxx member to specify the reenterable modules that are to be added as a temporary extension to the pageable link pack area (PLPA). The modules are cataloged in the system master catalog.

```
INCLUDE LIBRARY(SYS1.MLPA.LOAD)
MODULES(IGC0001C,
        IEAVTABG,
        IEAVTABQ,
        IEAVTABR,
        DFSAFMDO)
```

Figure 4-10 Coding the IEALPAxx parmlib member

This extension is temporary because the modules will remain in paging data sets and will be listed on the active LPA queue only for the duration of the IPL.

The system will not automatically quick-start or warm-start these modules (that is, reinstate the modules without re-specification of the MLPA parameter). Both the modified LPA (MLPA) and the fixed LPA (FLPA) modules (those named in IEAFIXxx) do not require the system to search the LPA directory when one of the modules is requested. The modified LPA, unlike the fixed LPA, however, contains pageable modules that behave in most respects like PLPA modules.

Note: A library that includes modules for the MLPA must be a PDS. You cannot use PDSEs in the LPALST concatenation.

The data sets in the LPALST can be a mixture of APF-authorized and non-APF-authorized data sets. However, any module in the modified link pack area will be treated by the system as though it came from an APF-authorized library. Ensure that you have properly protected any library that contributes modules to the modified link pack area to avoid system security and integrity exposures, just as you would protect any APF-authorized library.

You can use IEALPAxx to temporarily add or replace SVC or ERP routines. Another possible application would be the testing of replacement LPA modules that have been altered by PTFs.

LPA modules specified in IEALPAxx are placed in the MLPA or in the extended MLPA, based on the RMODE of the modules: modules with an RMODE of 24 are placed in the MLPA, while those with an RMODE of ANY are placed in the extended MLPA. By default, the MLPA and the extended MLPA are page protected, which means that a protection exception will occur if there is an attempt to store data into either area. To override page protection, use the NOPROT option on the MLPA system parameter.

LPA modules that are replaced through IEALPAxx are not physically removed from the PLPA or from the LPA directory. They are, however, logically replaced because, when one of them is requested, MLPA is searched first and the system does not examine the LPA directory that contains the name of the replaced module.

The system searches the fixed LPA before the modified LPA for a particular module and selects the module from the modified LPA only if the module is not also in the fixed LPA.

4.12 Specifying the IEALPAxx member

The two characters (A through Z, 0 through 9, @, #, or \$) represented by aa (or bb and so forth), are appended to IEALPA to form the name of the IEALPAxx parmlib members. The options are as follows:

- | | |
|--------|--|
| L | If the L option is specified, the system displays the contents of the IEALPAxx parmlib members at the operator's console as the system processes the members. |
| NOPROT | By default, the LPA load modules in the IEALPAxx parmlib members are page-protected in storage. However, the NOPROT option allows an installation to override the page protection default. |

Syntax format for MLPA parameter: Use the following syntax for the MLPA parameter:

```
MLPA= {aa} { (aa[,L][,NOPROT]) } { (aa,bb,...[,L][,NOPROT]) }
```

Figure 4-11 shows an example of overriding the value of the MLPA parameter in the IEASYSxx during system initialization; IEALPA02 was selected during system initialization.

```
IEA101A SPECIFY SYSTEM PARAMETERS FOR z/OS 02.03.00 HBB77B0
R 00,MLPA=02
IEE600I REPLY TO 00 IS;MLPA=02
```

Figure 4-11 Overriding the value of the MLPA parameter in the IEASYSxx during system initialization

4.13 Dynamic LPA functions

Dynamic link pack area has the ability to dynamically add or delete load modules from the LPA after the IPL. This allows optional and new products to be loaded into the system without requiring an IPL. It also enables you to display load modules residing in LPA.

A load module added dynamically is found before one of the same name added during the IPL. This allows you to test new load modules before you put them into production. Load modules added dynamically to the LPA are loaded into the common area (CSA) or extended common area (ECSA). The parameter CSAMIN guarantees a minimum value of free CSA after the load of the PLPA load module; if this is not honored the load fails.

How to perform dynamic LPA functions

The dynamic LPA functions can be invoked in one of the following ways:

- ▶ The PROGxx parmlib member includes the LPA statements, which are used to define what load modules can be added to or deleted from LPA after the IPL. You use the SET command to execute the functions described in the PROGxx parmlib member; for example, SET PROG=xx.
- ▶ The SETPROG LPA command can be used to initiate a change (add or delete) to the LPA.
- ▶ The DISPLAY PROG,LPA command can be used to display information about load modules that have been added to LPA.
- ▶ The CSVDYLPA macro allows an authorized program to initiate dynamic LPA services.

Note: CSVDYLPA is the only method currently available to place modules into 64-Bit CSA.

For more information, see *z/OS MVS Initialization and Tuning Reference*, SA22-7592.

64-bit considerations

The binder might create a module that is RMODE 64. That modules can be in a PDS or PDSE. A new binder control statement RMODEX(64TRUE) in combination with RMODE=64 is also available, as shown in Figure 4-12.

DSLST		LUTZ.LOADLIB		Row 0000001 of 0000001				Scroll ==> PAGE		
Command ==>		Name	Prompt	Alias-of	Size	TTR	AC	AM	RM	
		IEEU83			000000C0	000010	01	64	64	
		End								

Figure 4-12 RMODE=64 module

z/OS allows you to load modules above the 2 GB bar with the CSVDYLPA, which saves memory below the bar. The CSVDYLPA (dynamic LPA) allows now to get information about 64-bit modules loaded above the bar.

Restriction: A program object can have CSECTs that are of the three RMODEs (24, 31, 64) but the resulting program object, even with RMODE=SPLIT, will have only two and the user can control which two.

Just because you can get the system to put a module above 2G does not mean that it will work there. Some restrictions are:

- ▶ If you have 4-byte address constants: no
- ▶ If you invoke any service that is not entered by SVC or PC: maybe
- ▶ WAIT and Pause/Release do work
- ▶ If you're using z/OS recovery routines such as ESTAE, ARR, FRR: the routines themselves must be below 2G
- ▶ If you have an asynchronous exit (for example, timer exit), the exit routine must be below 2G
- ▶ Judicious use of RMODE=SPLIT can be helpful
- ▶ If you see a module address of x'7FFFFBAD' it probably means that the display is not utilizing new fields that contain the full 64-bit address

Attention: This support is available only for assembler modules at this time.

4.14 The LNKLST

Program management is a z/OS component that deals with load modules. Performance of programs can be improved by using LNKLST and other mechanisms that are to locate and fetch load modules, such as those described in 4.16, “Library lookaside” on page 129 and the 4.19, “Virtual lookaside facility” on page 134. See Figure 4-13 for an overview of the LNKLST.

Initially, we can say that the LNKLST allows the installation to concatenate load module libraries (PDS or PDSE) with the SYS1.LINKLIB. Concatenation means that the access method processes all the concatenated data sets as though they were other extents of the first data set, that is, a continuation and not a different data set.

LLA is a program management z/OS component that keeps in virtual storage directory entries from load module libraries, thereby speeding up the fetch of a load modules. VLF is another z/OS component that keeps objects in virtual storage to save I/O operations that would load such objects. One of these object types is load modules.

When determining which data sets LLA and VLF are to manage, try to limit the choices to production load libraries that are rarely changed. This avoids a refresh of LLA or VLF operations. Because LLA manages LNKLST libraries by default, you only need to determine which non-LNKLST libraries LLA is to manage. If, for some reason, you do not want LLA to manage particular LNKLST libraries, you must explicitly remove such libraries from LLA management.

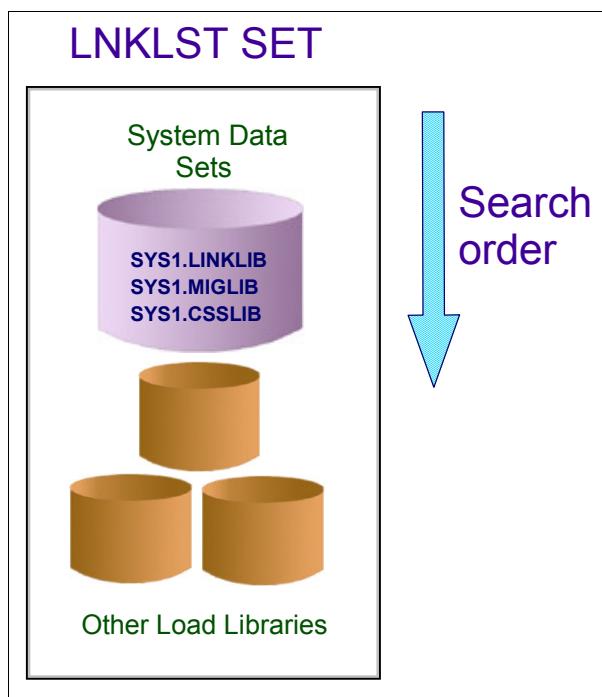


Figure 4-13 Overview of the LNKLST

To obtain the most performance benefit, plan to use both LLA and VLF. This method reduces the I/O required to find directory entries and to fetch load modules from DASD. Selected load modules can be staged in VLF dataspaces. LLA does not use VLF to manage library directories. When used without VLF, LLA eliminates only the I/O the system would use in searching library directories on DASD.

All LLA-managed libraries must be cataloged, including LNKLST libraries. A library must remain cataloged for the entire time it is managed by LLA. The benefits of LLA apply only to load modules that are retrieved through the LINK, LINKX, LOAD, ATTACH, ATTACHX, XCTL, and XCTLX macros. These macros are issued by a load module running under a task in order to load another load module. For example, the initiator attaches the load module name described in the EXEC PGM card in order to start a step task. The LNKLST is a group of system and user-defined load libraries that is a continuation of SYS1.LINKLIB and also indicates the search order the system uses for programs.

Executable programs, whether stored as load modules (PDS) or program objects (PDSE), must be loaded into both virtual storage and central storage before they can be executed.

By default, the LNKLST begins with the following system data sets:

- ▶ SYS1.LINKLIB
- ▶ SYS1.MIGLIB
- ▶ SYS1.CSSLIB

The LNKLST concatenation

You can change this order and add other load libraries to the LNKLST concatenation. The LNKLST concatenation is established at IPL time and can optionally be modified dynamically. It consists of SYS1.LINKLIB, followed by the libraries specified in one or more LNKLSTxx members of SYS1.PARMLIB. The LNKLSTxx member is selected through the LNK parameter in the IEASYSxx member of SYS1.PARMLIB.

The building of the LNKLST concatenation happens during an early stage in the IPL process, before any user catalogs are accessible, so only those data sets whose catalog entries are in the system master catalog can be included in the linklist. To include a data set that is cataloged in a user catalog in the LNKLST, you have to specify both the name of the data set and the volume serial number (VOLSER) of the DASD volume on which the data set resides.

Note: The number of data sets that you can concatenate to form the LNKLST concatenation is limited by the total number of DASD extents the data sets will occupy. The total number of extents must not exceed 255. When the limit is exceeded, the system writes the IEA328E error message to the operator's console.

These data sets are concatenated in the order in which they appear in the LNKLSTxx members, and the system creates a data extent block (DEB) describing the data sets concatenated to SYS1.LINKLIB and their extents. The DEB contains details of each physical extent allocated. These extents remain for the duration of the IPL. After the LNKLST process, LLA starts managing the LNKLST data sets and can be used to control updates to them.

Specifying the LNK parameter

An example of overriding the value of LNK parameter in IEASYSxx along IPL is shown in Figure 4-14.

```
IEA101A SPECIFY SYSTEM PARAMETERS FOR z/OS 02.03.00 HBB77B0
R 00,LNK=03
IEE600I REPLY TO 00 IS;LNK=03
```

Figure 4-14 Overriding the LNK parameter

For more information, see *z/OS MVS Initialization and Tuning Reference*, SA22-7592, and *z/OS MVS System Commands*, SA22-7627.

4.15 Dynamic LNKLST functions

You can change the current LNKLST set dynamically through the **SET PROG=xx** and **SETPROG LNKLST** commands. In reality, you do not *change* a LNKLST, you *replace* the full LNKLST.

A LNKLST set remains allocated until there are no longer any jobs or address spaces associated with its libraries. If the current LNKLST set is dynamically changed, any job or address space associated with a previous LNKLST set continues to use the data sets until the job or address space finishes. Thus, a previously current LNKLST set might be active or in use by a job or address space even though a new current LNKLST set has been activated. Jobs or address spaces started after a new current LNKLST set is activated use this set.

The **SET PROG=xx** and **SETPROG LNKLST** commands remove the definition of a LNKLST set from z/OS, associate a job or address space with the current LNKLST set, and locate a specific load module associated with a data set in LNKLST set. You can set the content of the LNKLST dynamically using one of the following optional methods:

- You can create a PROGxx parmlib member with the new changes to the LNKLST set, then issue the **SET PROG=xx** operator command to activate the changes.

```
LNKLST DEFINE NAME(LNKLST47) COPYFROM(CURRENT)  
LNKLST ADD      NAME(LNKLST47) DSN(LUTZ.LOADLIB)  
LNKLST ACTIVATE NAME(LNKLST47)
```

Attention: The name for the new LNKLST must differ from the current active one; otherwise, you will receive the following error message:

```
-SET PROG=LK  
CSV530I ERROR IN PARMLIB MEMBER=PROGLK ON LINE 1:  
LNKLST SET LNKLST00 WAS NOT CHANGED. IT IS IN USE
```

After successfully activation, the following messages display on the system console:

```
-SET PROG=LK  
CSV500I LNKLST SET LNKLST47 HAS BEEN DEFINED  
CSV501I DATA SET LUTZ.LOADLIB  
HAS BEEN ADDED TO LNKLST SET LNKLST47  
CSV500I LNKLST SET LNKLST47 HAS BEEN ACTIVATED  
IEE536I PROG      VALUE LK NOW IN EFFECT
```

- Use the CSVDYNL macro programming service in an authorized program to change the LNKLST concatenation for associated jobs and address spaces.

- Use the **SETPROG LNKLST** operator command to update the LNKLST directly.

Use the **D PROG,LNKLST** command to display information about the LNKLST set:

D PROG,LNKLST

```
CSV470I 14.06.00 LNKLST DISPLAY 802  
LNKLST SET LNKLST00  LNKAUTH=LNKLST  
ENTRY APF VOLUME DSNAME  
    1   A  BH8CAT  SYS1.SC80.LINKLIB  
    2       BH8CAT  SYS1.SC80.MIGLIB  
    3       BH8CAT  SYS1.SC80.PDSE  
    4   A  Z23RA1  SYS1.SIEALNKE  
    5   A  Z23RA1  SYS1.SIEAMIGE  
    6   A  Z23RA1  SYS1.LINKLIB  
    7   A  Z23RA1  SYS1.MIGLIB  
    8   A  Z23RA1  SYS1.CSSLIB
```

```

9     A   Z23RA1  SYS1.SERBLINK
10          Z23RA1  APK.SAPKMOD1

```

4.16 Library lookaside

Library lookaside (LLA) is a z/OS function implemented in an address space that maintains in the private area a copy of the directory entries of the libraries that it manages (shown in Example 4-15). Because the entries are cached, the system does not need to read the data set directory entries to determine where a load module is stored before fetching it from DASD. This greatly reduces I/O operations.

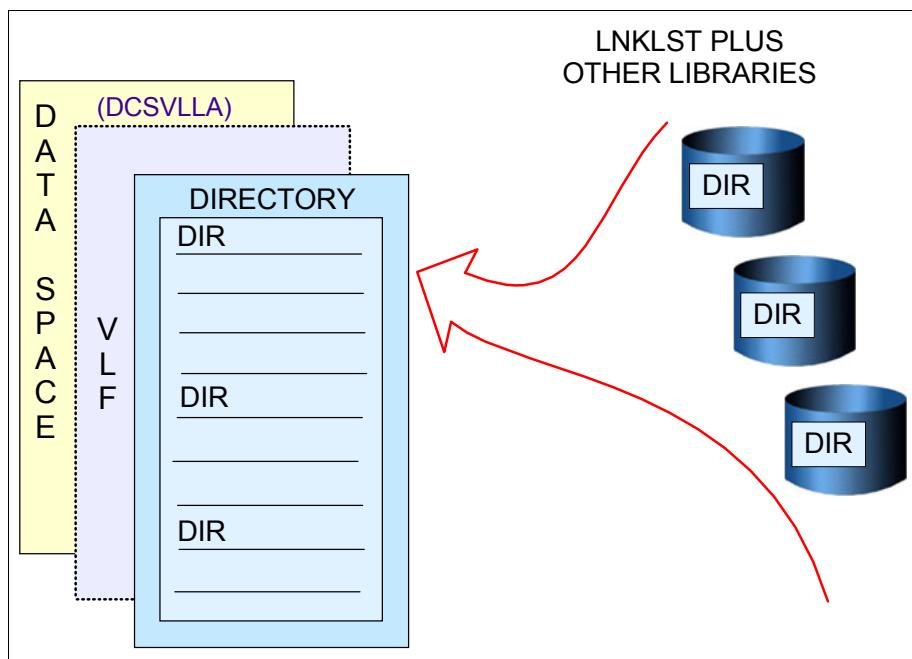


Figure 4-15 Library lookaside (LLA)

The main purpose of using LLA is to improve the performance of load module directory searching on your system. In addition, LLA can be a user of VLF facilities to keep copies of load modules in VLF data spaces.

How LLA improves performance

LLA improves load module fetch performance in the following ways:

- ▶ By maintaining (in the LLA address space) copies of the library directories, program management uses this to locate load modules. Program management can quickly search the LLA copy of a directory in virtual storage instead of using costly I/O to search the PDS directories on DASD.
- ▶ By placing (or staging) copies of selected load modules in a virtual lookaside facility (VLF) data space named DCSVLLA when you define the LLA class to VLF, and start VLF. The system can quickly fetch load modules from virtual storage, rather than using slower I/O to fetch the load modules from the DASD libraries.
- ▶ By determining which load modules, if VLF staged, would provide the most benefit to load module fetch performance. LLA evaluates load modules as candidates for staging based on statistics it collects about the members of the libraries it manages such as load module size, frequency of fetches per load module (fetch count), and the time required to fetch a

particular load module. If necessary, you can directly influence LLA staging decisions through installation exit routines CSVLLIX1 and CSVLLIX2.

LLA-managed libraries

By default, LLA address space caches directory entries for all the load modules in the data sets included in the linklist concatenation defined in either the LNKLSTxx or PROGxx parmlib members.

You can also identify other user-defined load libraries that contain frequently used load modules to be managed by LLA.

The issue with caching to boost performance is the possibility that the data kept in the virtual storage buffers to avoid I/O operations might be out of date because of updates done in the DASD data set. LLA can avoid that through the GET_LIB_ENQ (YES) option in the CSVLLAxx parmlib member or by the installation F LLA,REFRESH command. Refer to 4.17, "CSVLLAxx SYS1.PARMLIB member" on page 130 for more information.

Planning LLA use

Before you can use LLA, complete the following steps:

1. Determine which libraries should be LLA-managed libraries.
2. Code the CSVLLAxx parmlib member to include, modify, and remove the LLA-managed libraries and to optimize the performance of the search processing and selectively refresh LLA directory entries.
3. Learn how to control the LLA through the START, STOP, and MODIFY LLA operator commands. LLA directory entries can be selectively refreshed via the F LLA,REFRESH operator command. Refer to the next section for more information about the refresh concept.

4.17 CSVLLAxx SYS1.PARMLIB member

You use the CSVLLAxx parmlib member to specify which libraries, in addition to the LNKLST concatenation, that LLA is to manage.

Note: If you do not supply a CSVLLAxx member, LLA by default only manages the directories of those libraries that defined in the LNKLST concatenation.

You can also use CSVLLAxx to specify:

- ▶ Libraries to be added or removed from LLA management while LLA is active.
- ▶ Whether LLA is to hold an enqueue for the libraries it manages. If you specify GET_LIB_ENQ (YES), which is the default, LLA obtains a shared enqueue for the libraries it manages. The shared enqueue allows your job to read the libraries, but not to move or erase them. To update these libraries, you must first remove them from LLA management through the REMOVE keyword. This function guarantees data integrity.
- ▶ Use a FREEZE|NOFREEZE option per library. The NOFREEZE option prevents directory entries being kept in the LLA address space, so no performance improvements are achieved for directory access. The reason might be that lots of updates are done in the library.
- ▶ Members of libraries, or whole libraries, to be selectively refreshed in the LLA directory.

- ▶ Additional CSVLLAxx members to be used to control LLA processing. These members can reside in data sets other than SYS1.PARMLIB.
- ▶ Whether exit routines are to be called during LLA processing.

For more information, see *z/OS MVS Initialization and Tuning Reference*, SA22-7592.

How to start and stop LLA

To start LLA, use the **START LLA,LLA=xx** command. This command identifies the CSVLLAxx parmlib member to be used to build the LLA directory. This command is issued during system initialization by the IBM-supplied IEACMD00 parmlib member and can be entered by the operator.

The **S LLA** command uses the following parameters:

LLA	Invokes the LLA procedure and creates the LLA address space.
LLA=xx	Indicates which CSVLLAnn parmlib member LLA is to use. If you omit LLA=xx, LLA builds its directory using only the LNKLST libraries.
SUB=MSTR	Indicates that the name of the subsystem that processes the task is the master subsystem. If you omit this parameter, the JES subsystem starts LLA. The resulting dependency on JES requires LLA to be stopped when stopping JES.

Recommendation: Specify SUB=MSTR on the **START LLA** command to prevent LLA from failing if JES fails.

Syntax for S LLA command: S LLA[,SUB=MSTR] [,LLA=xx]

To stop LLA, use the **P LLA** operator command.

The parameter for the **P LLA** command is:

LLA The job name assigned to the LLA address space.

Syntax for P LLA command: P LLA

For more information, see *z/OS MVS Initialization and Tuning Reference*, SA22-7592, and *z/OS MVS System Commands*, SA22-7627.

How to refresh LLA

Caching to boost performance might cause integrity problems because the data kept in the virtual storage buffers to avoid I/O operations can be out of date because of updates done in the DASD data set. LLA accepts an installation command refresh that can refresh totally all the libraries managed by LLA, which is bad for performance, or a selective member refresh, which is recommended. You can use the **MODIFY LLA** command to change LLA dynamically, in one of the following ways:

- ▶ **MODIFY LLA,REFRESH**

This command rebuilds LLA's directory for the entire set of libraries managed by LLA. This action is often called the *complete refresh* of LLA.

```
-F LLA,REFRESH
CSV210I LIBRARY LOOKASIDE REFRESHED
```

► **MODIFY LLA,UPDATE=xx**

This command rebuilds LLA's directory only for specified libraries or load modules. *xx* identifies the CSVLLAxx member that contains the names of the libraries for which directory information is to be refreshed. This action is often called a *selective refresh* of LLA.

If the library is shared by several z/OS systems, as in a Parallel Sysplex, the **Modify LLA** command must be issued in all systems.

Note: There are several situations where you need to refresh the LLA with the latest version of the directory information from the DASD. Whenever you update a load module in a library that LLA manages, make sure you follow the update by issuing the appropriate form of the **F LLA** command. Otherwise, LLA continues to use an older version of a load module.

If you delete a data set from the current LNKLST set, LLA continues working and finding load modules in the deleted library, because the physical addresses of the members are still held by LLA although it is no longer possible to open the directory. This method only works until the physical space on the DASD is reused by something else.

You also find yourself in the situation where you need to compress a data set from the LNKLST set. Refresh LLA's cache after the compress; otherwise, the directory entries in the LLA for every load module moved during the compress process will be invalid, which can lead to abends whenever a user attempts to load one of these modules until a refresh is completed.

4.18 Compressing LLA-managed libraries

Observe that a PDS library sometimes needs to be compressed to reclaim the free space caused by member deletions. The recommended procedure for compressing an LLA-managed library is as follows:

1. Create a new CSVLLAxx member that includes a REMOVE statement that identifies the library that needs to be compressed, as shown in Figure 4-16.

```
Menu Utilities Compilers Help
-----
BROWSE SYS1.PARMLIB(CSVLLA02) - 01.01      Line 00000000 Col 001 080
***** Top of Data *****
REMOVE(SYS1.PRODLIB)
***** Bottom of Data *****
```

Figure 4-16 Create a new CSVLLAxx member that includes a REMOVE statement

Then issue the **F LLA,UPDATE=xx** command, as shown in Figure 4-17.

```
F LLA,UPDATE=02
IEE252I MEMBER CSVLLA02 FOUND IN SYS1.PARMLIB
CSV210I LIBRARY LOOKASIDE UPDATED
```

Figure 4-17 Issue the F LLA,UPDATE=xx command

2. Compress the library.

You can compress a data set using one of the following methods:

- You can issue the **Z** line command against the data set you want to compress from the ISPF panel, as shown in Figure 4-18.

The screenshot shows the ISPF menu with the following options: Menu, Options, View, Utilities, Compilers, Help. Below the menu is a dashed line. The main display area shows the following text:
DSLIST - Data Sets Matching SYS1.LINKLIB
Command ==> Row 1 of 1
Command - Enter "/" to select action Message Volume
Z SYS1.LINKLIB MPRES1
***** End of Data Set list *****

Figure 4-18 Issue the Z line command

- Alternatively, you can submit a job to compress the data set using the **IEBCOPY** utility as shown in the sample JCL in Figure 4-19.

```
//COMPRES JOB (), 'MVSSP', NOTIFY=&SYSUID,CLASS=A,MSGLEVEL=(1,1),  
//          MSGCLASS=X  
//*****  
//**          *  
//**      COMPRESSING DATA SETS USING IEBCOPY *  
//**          *  
//*****  
//STEP1 EXEC PGM=IEBCOPY  
//SYSPRINT DD SYSOUT=*  
//INPUT    DD DSNAME=SYS1.LINKLIB,DISP=SHR  
//OUTPUT   DD DSNAME=SYS1.LINKLIB,DISP=SHR  
//SYSIN    DD *  
          COPY INDD=INPUT,OUTDD=OUTPUT  
/*
```

Figure 4-19 Compress the data set using the IEBCOPY utility

3. Create a new CSVLLAxx member that includes a LIBRARIES statement to return the compressed library to LLA management, as shown in Figure 4-20.

The screenshot shows the ISPF menu with the following options: Menu, Utilities, Compilers, Help. Below the menu is a dashed line. The main display area shows the following text:
BROWSE SYS1.PARMLIB(CSVLLA03) - 01.01 Line 00000000 Col 001 080
***** Top of Data *****
LIBRARIES(SYS1. PRODLIB)
***** Bottom of Data *****

Figure 4-20 Create a new CSVLLAxx member

Then issue the **F LLA,UPDATE=xx** command, as shown in Figure 4-21.

```
F LLA,UPDATE=03
IEE252I MEMBER CSVLLA03 FOUND IN SYS1.PARMLIB
CSV210I LIBRARY LOOKASIDE UPDATED
```

Figure 4-21 Issue the **F LLA,UPDATE=xx** command

4.19 Virtual lookaside facility

The virtual lookaside facility (VLF) is a z/OS component that enables an authorized program, a VLF exploiter, to store named data objects in dataspaces managed by VLF and to retrieve these objects by name on behalf of users in multiple address spaces. VLF, shown in Figure 4-22, is designed primarily to improve performance by retrieving the most frequently used objects from virtual storage rather than performing repetitive I/O operations from DASD. VLF runs in its own address space. When the application makes a request for such objects, VLF checks its data space to see if it is there. If affirmative, VLF can rapidly retrieve it without needing a DASD I/O. These data objects can contain a copy of programs.

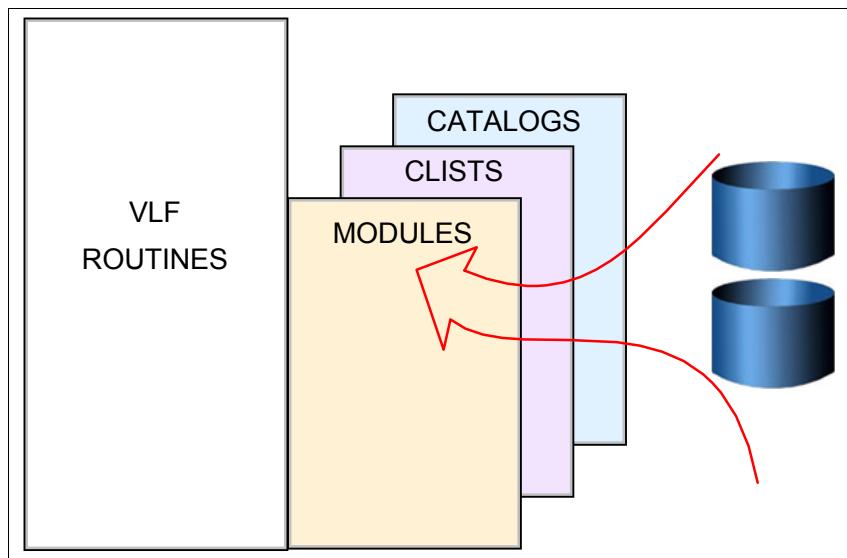


Figure 4-22 Virtual lookaside facility (VLF) overview

Data objects should be small-to-moderate in size, named according to the VLF naming convention, and associated with an installation-defined class of data objects.

Not all types of data objects can be stored in the VLF dataspaces. Only certain IBM products or components are VLF exploiters, such as:

- ▶ LLA (for load modules)
- ▶ TSO/E (for REXX and CLIST procedures)
- ▶ Catalog address space (CAS) for user catalog entries
- ▶ RACF records from its database

Because VLF uses virtual storage for its dataspaces, there are performance considerations that each installation must weigh when planning for the resources required by VLF.

Note: VLF is intended for use with major applications. Because VLF runs as a started task that the operator can stop or cancel, it cannot take the place of any existing means of accessing data on DASD. Any application that uses VLF must also be able to run without it.

Using VLF with LLA

You obtain the most benefit from LLA when you have both LLA and VLF. You should plan to use both. When used with VLF, LLA reduces the I/O required to fetch load modules from the DASD by causing selected load modules to be staged in VLF dataspaces.

LLA does not use VLF to manage library directories. When using LLA without VLF, LLA eliminates only the I/O that the system would use in searching for library directories on DASD.

VLF considerations

Before you can take full advantage of implementing VLF to improve system performance, consider the following factors:

- ▶ VLF works best with the following types of data:
 - Data objects that are members of partitioned data sets, located through a partitioned data set (PDS) concatenation
 - Data objects that, although not PDS members, can be easily described as a collection of named objects that are repeatedly retrieved by many users
- If neither description fits your data objects, it is likely that you would not obtain any performance benefit from VLF. Remember, there are storage overhead costs associated with using VLF.
- ▶ VLF works best with relatively small objects because less virtual storage is expended to reduce the number of I/O operations.
- ▶ Like data in private storage, data stored through VLF is subject to page stealing. That is, VLF works best when a significant portion of the data is likely to remain in central storage and not be paged out to auxiliary storage. VLF is designed to improve performance by increasing the use of virtual storage to reduce the number of I/O operations. For a system that is already experiencing a central storage constraint, this strategy is probably not a good choice. However, we should state that in the majority of installations, currently there is plenty of available central storage.

VLF planning

Before you can use VLF, complete these steps:

1. Start VLF using the IBM-supplied default COFVLFx member and a JCL procedure.
2. Update COFVLFx to include the VLF classes associated with the applications or products.

4.20 COFVLFx parmlib member

You use the COFVLFx member of the SYS1.PARMLIB to define classes of VLF objects. To activate a class of VLF objects, VLF requires that a CLASS statement describing that group of objects be present in the active COFVLFx parmlib member (the member named on the START command used for VLF).

A VLF class is a group of related objects made available to users of an application or component. To get the most benefit from using VLF, consider its use for objects that are:

- ▶ Used frequently
- ▶ Changed infrequently
- ▶ Used by multiple users concurrently

Some of the more important statements and parameters for COFVLFx are:

CLASS	Indicates that the following parameters define that particular group of objects to VLF. Each group of objects that VLF processes must have a CLASS statement defining it.
NAME(classname)	Specifies the name of the VLF class. For example, LLA uses the class of VLF objects named CSVLLA. If the CLASS statement for CSVLLA is not included in the active COFVLFx parmlib member, LLA cannot use VLF, and many of the performance and operational benefits of LLA will not be available. The class name can be 1 to 7 alphanumeric characters including @, #, and \$.
EDSN(dsn)] VOL(vol)]	Identifies a partitioned data set name whose members are eligible to be the source for VLF objects in the class for a PDS class, such as LLA, EDSN(dsn). The dsn can be 1 to 44 alphanumeric characters, including @, #, and periods (.). You do not need to specify the volume if the cataloged data set is the desired one. If the data set is not cataloged, or if you have multiple data sets with the same name on different volumes, you must specify VOL(vol). Multiple occurrences of the same data set name with a different volume are acceptable. However, if duplicate entries of the same data set name and the same volume occur, the system issues an informational message and ignores the redundant information. For the LLA class of objects it is recommended that the same PDSs defined in CSVLLAx also be defined for VLF.
EMAJ(majname)	Identifies an eligible major name (majname) for a non-PDS class. For an IBM-supplied class, use the product information to determine if anything other than the names specified in the IBM-supplied default COFVLFx member are eligible. The majname can be 1 to 64 alphanumeric characters except comma(,), blank, or right parenthesis (), for example EMAJ(LLA).
<p>Important: Do not use the EMAJ and EDSN parameter on the same CLASS statement.</p>	
MAXVIRT(nn)	Specifies the maximum amount of virtual storage that the installation wants VLF to use for the objects in the class. Unless you supply this value on the optional MAXVIRT parameter of the CLASS statement, VLF will use a default value. When you specify the MAXVIRT value, ensure that it is large enough to hold most or all of the frequently-used objects in a VLF class. An excessively small value tends to cause thrashing of the data in that VLF class, and an excessively large MAXVIRT value tends to increase the consumption of auxiliary storage because infrequently-used data is paged out, rather than discarded.

For more information, see *z/OS MVS Initialization and Tuning Reference*, SA22-7592.

Figure 4-23 shows an example of a COFVLFx member.

```
Menu Utilities Compilers Help
-----
BROWSE SYS1.PARMLIB(COFVLF04) - 01.01      Line 00000000 Col 001 080
***** Top of Data *****
**
CLASS NAME(CSVLLA)          /* CLASS NAME FOR LIBRARY LOOKASIDE */
    EMAJ(LLA)             /* MAJOR NAME FOR LIBRARY LOOKASIDE */
                           /* Default MAXVIRT = 4096 4K blocks */
                           /*                      = 16Mb           */
/*
CLASS NAME(IKJEXEC)         /* TSO/E VERSION 2 CLIST/REXX INTERFACE */
    EDSN(SYS1.OS390.CLIST)
    EDSN(MVSTOOLS.SHARED.CLIST)
    MAXVIRT(1024)          /* MAXVIRT = 512 4K blocks */
                           /*                      = 4Mb           */
/*
CLASS NAME(IGGCAS)          /* MVS/DFP 3.1 CATALOG in Data space */
    EMAJ(CATALOG.SC54.MCAT)
    EMAJ(CATALOG.TOTICF1.VITS001)
    MAXVIRT(1024)          /* MAXVIRT = 512 4K blocks */
/*
                           /*                      = 2Mb (minimum value)
*****
Bottom of Data *****
```

Figure 4-23 Example of parmlib member COFVLFx

Starting and stopping VLF

To start VLF, use the **START VLF, SUB=MSTR, N=xx** command. This command identifies the COFVLFx parmlib member to build VLF and is issued by the IBM-supplied COMMND00 parmlib member during system initialization. The command can also be entered by the operator.

The **S VLF** command uses the following parameters:

S VLF, SUB=MSTR[, NN=xx]

This command invokes the VLF procedure that starts the VLF, where:

NN=xx Indicates that the system is to start VLF using the COFVLFx member of the SYS1.PARMLIB (default COFVLF00).

To stop VLF use the **P VLF** command.

The parameter for the **P VLF** command is:

VLF The job name assigned to the virtual lookaside facility. Using this parameter stops VLF with the message COF033I.

Important: Stopping VLF can degrade system performance.

4.21 Authorized program facility (APF)

Each CPU can run in supervisor or problem state as indicated by bit 15 in the current PSW. When in problem mode state (PSW bit 15 ON), the CPU refuses to execute a privileged instruction, generating a program interrupt with the X'0002' interrupt code. This interrupt causes the running task to abend with an X'0C2' completion code. All privileged instructions share the fact that if they are misused, it might jeopardize the integrity and the security of the multi-transaction z/OS system. Authorized libraries are shown in Figure 4-24.

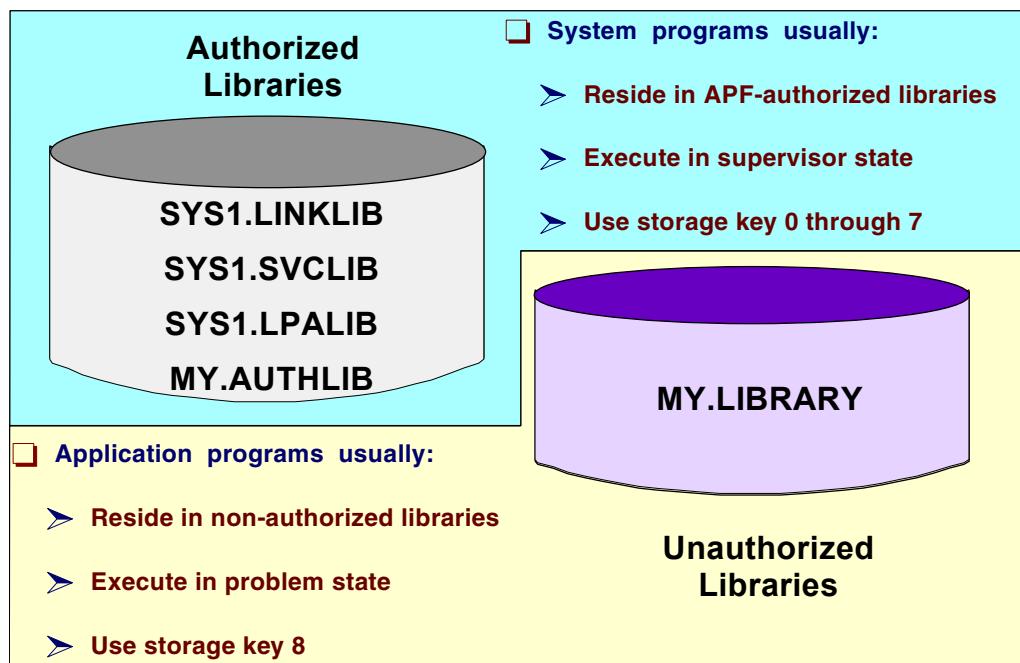


Figure 4-24 Authorized libraries

Also, certain SVC routines when misused by the caller program might compromise system integrity and system security. Access to these routines must be restricted to only authorized programs.

To protect the system, z/OS introduced the concept of APF. Only load modules from an APF task can invoke the APF protected SVCs. A task is APF protected when all of its load modules are APF authorized. If just one load module running in a task is non-APF authorized, the task loses its APF state forever. The reason for this is to prevent programs from counterfeiting a load module in the load module flow of an authorized job step task. Thus, if a load module from a non-authorized task tries to execute SVCs that require APF authorization, z/OS abnormally ends the task and issues an X'306' completion code and a X'04' reason code. Logically, if the load module is running in supervisor state or PSW keys from 0 to 7, it can invoke any SVC routine. Recall that all the load modules in the MLPA must be APF authorized.

A load module needs to fulfill the following condition to be APF authorized:

- ▶ Use a linkedit step with the loader utility program with authorization code (AC) equal to one. This state is presented in a bit setting in the PDS/PDSE directory entry for the load module.
- ▶ The module must be stored in an APF library.

Authorized (APF) libraries overview

Libraries that contain authorized programs are known as *authorized libraries*. The following APF libraries are authorized libraries:

- ▶ SYS1.LINKLIB
- ▶ SYS1.SVCLIB
- ▶ SYS1.LPALIB
- ▶ An authorized library specified by your installation

Note: By default, the libraries in the LNKLST concatenation are considered authorized unless you specified LNKAUTH=APFTAB in the IEASYSxx parameter list. However, if the system accesses the libraries in the LNKLST concatenation through J0BLIB or STEPLIB DD statements, the system does not consider those libraries authorized unless you specified the library name in the APF list by using either the IEAAPFx or the PROGxx parmlib member. If a JCL DD statement concatenates an authorized library in any order with an unauthorized library, the entire set of concatenated libraries is treated as unauthorized. SYS1.LPALIB is treated as an authorized library only while the system builds the pageable link pack area (PLPA) using the LPALSTxx parmlib member. All load modules in PLPA are marked as coming from an authorized library. When accessed through a STEPLIB, J0BLIB, or TASKLIB DD statement, SYS1.LPALIB is considered an authorized library only if you have included it in the APF list.

Link-editing load modules with AC=1

You can use the PARM field on the link-edit step to assign the APF authorization code to a load module. This method causes the Binder to consider every load module created by the step to be authorized. If no authorization code is assigned in the Binder step, the AC=0 default is considered. To assign an authorization code using JCL, code AC=1 in the operand field of the PARM parameter of the EXEC statement.

```
//LKED EXEC PGM=HEWL,PARM='AC=1',...
```

Guidelines for using APF

When you use APF authorization, you must control which programs are stored in the authorized libraries. If the first load module in a program sequence is authorized, the system assumes that the flow of control to all subsequent load modules is known and secure as long as these subsequent load modules come from authorized libraries.

To ensure that this assumption is valid:

- ▶ Ensure that all programs that run as authorized programs adhere to your installation's integrity guidelines.
- ▶ Ensure that no two load modules with the same name exist across the set of authorized libraries. Two load modules with the same name could lead to an accidental or deliberate mix-up in load module flow, possibly introducing an integrity exposure.
- ▶ Linkedit with the authorization code (AC=1) only the first load module in a program sequence. Do not use the authorization code for subsequent load modules; they must be naturally APF authorized. If they are not, an x'306' abend is produced.

4.22 Authorizing libraries

The APF list is built during an IPL using those libraries specified in the IEAAPFx (not recommended) or PROGxx parmlib members, as indicated by the APF and PROG parameters in IEASYSxx or from the operator's console at system initialization.

Note: You can also dynamically modify the APF list after IPL, but only when you have used the dynamic APF format in the PROGxx.

IEAAPFxx parmlib member coding

The use of IEAAPFxx member in the SYS1.PARMLIB is not recommended, and it is not covered in this book.

PROGxx parmlib member coding

You can use the APF statement in PROGxx member to specify:

- ▶ The format of the APF-authorized library list, whether it is dynamic or static.
- ▶ Program libraries to be added to the APF list.
- ▶ Program libraries to be deleted from the APF list.

Note: The system automatically adds SYS1.LINKLIB and SYS1.SVCLIB to the APF list at IPL. If you specify a dynamic APF list format in PROGxx, you can update the APF list at any time during normal processing or at IPL. You can also enter an unlimited number of libraries in the APF list.

If you specify a static APF list format in PROGxx, you can define the list only at IPL, and are limited to defining a maximum of 255 library names (SYS1.LINKLIB and SYS1.SVCLIB, which are automatically placed in the list at IPL, and up to 253 libraries specified by your installation).

The following statements and parameters are available for the APF statement:

- ▶ FORMAT(DYNAMIC|STATIC)
 - Specifies that the format of the APF list is dynamic or static.
- ▶ ADD|DELETE
 - Indicates whether you want to add or delete a library from the APF list.
- ▶ DSNAME(dsname)
 - Name of the library that you want to add to or delete from the APF list.
- ▶ SMS|VOLUME(volume)
 - The identifier for the volume that contains the library that is specified on the DSNAME parameter, which is one of the following options:
 - SMS, which indicates that the library is SMS-managed or it has a storage class associated
 - A six character identifier for the volume
 - *****, which indicates that the library is located on the current SYSRES volume
 - *MCAT*, which indicates that the library is located on the volume containing the master catalog

Syntax format for APF statement: Use the following syntax for the APF statement:

```
APF FORMAT(DYNAMIC|STATIC) APF ADD | DELETE DSNAME(dsname) SMS | VOLUME(volname)
```

Figure 4-25 shows a sample PROGxx definition.

```
Menu Utilities Compilers Help
-----
BROWSE SYS1.PARMLIB(PROGTT) - 01.01 Line 00000000 Col 001 080
Command ===> Scroll ===> PAGE
***** Top of Data *****
APF FORMAT(DYNAMIC)
APF ADD
  DSNAME(SYS1.VTAMLIB)
  VOLUME(*****)
APF ADD
  DSNAME(SYS1.SICELINK)
  VOLUME(*****)
APF ADD
  DSNAME(SYS1.LOCAL.VTAMLIB)
  VOLUME(TOTCAT)
APF ADD
  DSNAME(ISP.SISPLOAD)
  VOLUME(*MCAT*)
***** Bottom of Data *****
```

Figure 4-25 PROGxx definition

4.23 Dynamic APF functions

You can use the following ways to update the content of the APF table dynamically:

- ▶ Use PROGxx parmlib member which includes the appropriate APF statement to define the change.
- ▶ The **SETPROG APF** operator can also initiate a change to the APF table.
- ▶ The **DISPLAY APF** command can be used to display the list of libraries authorized by APF.

Using the PROGxx parmlib member

As mentioned in the previous section, you can use the APF statement to add or delete libraries from the APF list. To delete a library from the APF list use the following command:

Example of the APF DELETE: APF DELETE DSNAME(SCRIPT.R40.DCFLOAD)
VOLUME(MPRES2)

Activate the change by using a **SET PROG=xx** command as shown in Figure 4-26.

```
SET PROG=T4
IEE252I MEMBER PROGT4 FOUND IN SYS1.PARMLIB
CSV410I DATA SET SCRIPT.R40.DCFLOAD ON VOLUME MPRES2 DELETED FROM APF
LIST
IEE536I PROG      VALUE T4 NOW IN EFFECT
```

Figure 4-26 SET PROG example

To add a library to the APF list use the following command:

```
APF ADD DSNAME(MYPROG.LOADLIB) VOLUME(MPRES3)
```

Activate the change by using a **SET PROG=xx** command as follows:

```
SET PROG=T5  
IEE252I MEMBER PROGT5 FOUND IN SYS1.PARMLIB  
CSV410I DATA SET MYPROG.LOADLIB ON VOLUME MPRES2 ADDED TO APF LIST  
IEE536I PROG VALUE T5 NOW IN EFFECT
```

Using the SETPROG APF command

Use the **SETPROG APF** operator command to perform the following functions:

- ▶ Change the format of the authorized program facility (APF) list from static to dynamic, or static to dynamic.
- ▶ Add a library to a dynamic APF list.
- ▶ Delete a library from a dynamic APF list.

To change the format of the APF list to dynamic, use the following command:

```
SETPROG APF,FORMAT=DYNAMIC  
CSV410I APF FORMAT IS NOW DYNAMIC
```

To add a library to the APF list:

```
SETPROG APF,ADD,DSN=LUTZ.LOADLIB,SMS  
CSV410I SMS-MANAGED DATA SET LUTZ.LOADLIB ADDED TO APF LIST
```

To delete a library from the APF list:

```
SETPROG APF,DELETE,DSN=LUTZ.LOADLIB,SMS  
CSV410I SMS-MANAGED DATA SET LUTZ.LOADLIB DELETED FROM APF LIST
```

Note: If you try to add to or delete from an APF list that is in a static format, the system responds with a CSV411I message, as follows:

```
SETPROG APF,DELETE,DSNAME=SCRIPT.R40.DCFLOAD,VOLUME=MPRES2  
CSV411I ADD/DELETE IS NOT ALLOWED BECAUSE APF FORMAT IS STATIC
```

Using the DISPLAY APF command

You can use the **DISPLAY APF** command to display one or more entries in the list of APF-authorized libraries. Each entry in the APF list display contains:

- ▶ An entry number
- ▶ The name of an authorized library
- ▶ An identifier for the volume on which the authorized library resides (or *SMS*, if the library is SMS-managed)

Syntax for DISPLAY APF command: Use the following syntax for the DISPLAY APF command:

```
D PROG,APF[,ALL ] |,DSNAME=libname |,ENTRY=xxx |,ENTRY=(xxx-yyy) ],L={a|cc|cca|name|name-a} ]
```

To display the entire APF list, use the command shown in Figure 4-27.

```
D PROG,APF
CSV450I 05.18.16 PROG,APF DISPLAY 971
FORMAT=DYNAMIC
ENTRY VOLUME DSNAME
 1 MPRES1 SYS1.LINKLIB
 2 MPRES1 SYS1.SVCLIB
 3 MPRES1 CPAC.LINKLIB

.
.
.

36 MPRES2 NETVIEW.V2R4M0.USERLNK
37 MPRES2 NPM.V2R3M0.SFNMLMD1
38 MPRES2 EMS.V2R1M0.SEMSLMD0
```

Figure 4-27 Display the entire APF list

This number is the decimal entry number for each of the libraries defined in the APF list. To display the specific library at entry number 1, as shown in Figure 4-28.

```
D PROG,APF,ENTRY=1
CSV450I 05.24.55 PROG,APF DISPLAY 979
FORMAT=DYNAMIC
ENTRY VOLUME DSNAME
 1 MPRES1 SYS1.LINKLIB
```

Figure 4-28 Display the specific library at entry number 1

To display all the libraries in the range from 1 to 5, use the commands shown in Figure 4-29.

```
D PROG,APF,ENTRY=(1-5)
CSV450I 05.26.27 PROG,APF DISPLAY 981
FORMAT=DYNAMIC
ENTRY VOLUME DSNAME
 1 MPRES1 SYS1.LINKLIB
 2 MPRES1 SYS1.SVCLIB
 3 MPRES1 CPAC.LINKLIB
 4 MPRES1 ISF.SISFLOAD
 5 MPRES1 ISF.SISFLPA
```

Figure 4-29 Display all the libraries in the range from 1 to 5

Note: It is recommended that you use the PROGxx parmlib member instead of the IEAAPFx member to define the APF list, regardless of whether you plan to take advantage of the dynamic update capability. If you specify both the PROG=xx and the APF=xx parameters, the system places into the APF list those libraries listed in the IEAAPFx, followed by those libraries in the PROGxx. If you are currently using the IEAAPFx parmlib member to define the APF list, you can convert the format of IEAAPFx to a PROGxx format using an IEAAPFPR REXX exec provided by IBM. For more information, see *z/OS MVS Initialization and Tuning Reference*, SA22-7592.



SMP/E for z/OS

SMP/E for z/OS is a tool designed to manage the installation of software products on your z/OS system and to track the modifications you make to those products. Usually, it is the system programmer's responsibility to ensure that all software products and their modifications are properly installed on the system. The system programmer also has to guarantee that all products are installed at the proper level. The challenge of this tasks increase exponentially with complexity of the software configuration and the installation structure; so does the task of monitoring all the elements of the system. To better understand this, let's take a closer look at your z/OS system and see how SMP/E works.

This chapter covers the following topics:

- ▶ Basic concepts
- ▶ Libraries, zones, and data set allocation
- ▶ SMP/E data set allocation
- ▶ SYSMOD Processing and SMP/E Data Display

5.1 Basic concepts

SMP/E is the basic tool for installing and maintaining software in z/OS systems and subsystems. SMP/e handles the hardware interfaces as well. For example when installing or replacing a storage system, you will probably have to install new modules in order to support the new functions. It controls these changes at the element level by:

- ▶ Selecting the proper levels of elements to be installed from a large number of potential changes
- ▶ Calling system utility programs to install the changes
- ▶ Keeping records of the installed changes

SMP/E is an integral part of the installation, service, and maintenance processes and in addition, it can be used to install and service any software that is packaged in SMP/E system modification (SYSMOD) format.

It can be run either using batch jobs or using dialogs under Interactive System Productivity Facility/Program Development Facility (ISPF/PDF).

5.1.1 What is a SYSMOD?

Any software, whether it is a product or service, is formed by a group of *elements* such as macros, modules, source, and other types of data (such as CLISTS or sample procedures). When using the SMP/E to install a software, it must include *control information* for the elements. This information describes the elements and any relationships that the software has with other products or services that can also be installed on the same z/OS system.

The combination of *elements* and *control information* is called a *system modification*. The SYSMODs are used to make changes in your system to improve the usability or reliability of a product, such as to add some new functions to your system, upgrade some of the elements of your system, or modify some elements for a variety of reasons.

Installing a SYSMOD refers to the process of placing of all new elements in the system data sets or libraries.

5.1.2 SYSMOD package

The SYSMOD is the package that contains information that SMP/E needs to install and track system modifications. SYSMODs are composed of the following parts:

- ▶ *Modification control statements (MCS)*, when browsing the SYSMOD, which are designated by ++ as the first two characters, that tell SMP/E:
 - What elements are being updated or replaced
 - How the SYSMOD relates to product software and other SYSMODs
 - Other specific installation information
- ▶ *Modification text*, which is the object modules, macros, and other elements supplied by the SYSMOD

5.1.3 Types of SYSMODs

All SYSMODs have a seven character *identifier* and there are four types of SYSMODs:

- ▶ Function SYSMODs

These introduce a new product, a new version or release of a product, or updated functions for an existing product into the system. There are two types of function SYSMODs:

A *base function* either adds or replaces an entire functional area in the system. It is a collection of elements (such as source, macros, modules, and CLISTS) that provides a general user function and is packaged independently from other functions. A base function is packaged as a function SYSMOD on a RELFILE tape, identified by an FMID (function module identifier). Function SYSMODs for base functions are applicable to any z/OS environment, although they might have interface requirements that require the presence of other base functions. Examples of base functions are SMP/E and z/OS.

A *dependent function* provides an addition to an existing functional area in the system. It is a collection of elements (such as source, macros, modules, and CLISTS) that provides an enhancement to a base function. It is called dependent because its installation depends on a base function already being installed. A dependent function is packaged as a function SYSMOD on a RELFILE tape, identified by an FMID. Function SYSMODs for dependent functions are only applicable to the parent base function. Each dependent function specifies an FMID operand on the ++VER MCS to indicate the base function to which it is applicable. Examples of dependent functions are the language features for SMP/E.

- ▶ PTF

A program temporary fix (PTF) is a tested fix for reported problems. PTFs can be used as preventive service to avert certain known problems that might have not yet appeared on your system, or they might be used as corrective service to fix problems you have already encountered. The installation of a PTF must always be preceded by that of a function SYSMOD, and often other PTFs as well.

- ▶ APAR fixes

An authorized program analysis report (APAR) is a temporary fix that is designed to fix or bypass a problem for the first reporter of the problem. These fixes might not be applicable to your environment. The installation of an APAR must always be preceded by that of a function SYSMOD, and sometimes of a particular PTF. That is, an APAR is designed to be installed on a particular preventive-service level of an element.

- ▶ User modifications (USERMODs)

These are SYSMODs built by you, either to change IBM code or to add independent functions to the system. The installation of a USERMOD must always be preceded by that of a function SYSMOD, sometimes certain PTFs, APAR fixes, or other USERMODs.

Note: If you want to package a user application program or new system function in SMP/E format, the correct way is to build a base or dependent function SYSMOD, not a USERMOD.

5.1.4 SYSMOD prerequisites

As you have learned, PTF, APAR, and USERMOD SYSMODs all have the function SYSMOD as a prerequisite. In addition to their dependencies on the function SYSMOD:

- ▶ PTF SYSMODs might be dependent upon other PTF SYSMODs.
- ▶ APAR SYSMODs might be dependent upon PTF SYSMODs and other APAR SYSMODs.

- ▶ USERMOD SYSMODs might be dependent upon PTF SYSMODs, APAR SYSMODs, and other USERMOD SYSMODs.

Consider the complexity of these dependencies. When you multiply that complexity by hundreds of load modules in dozens of libraries, the need for a tool like SMP/E becomes apparent.

5.2 Libraries, zones, and data set allocation

SMP/E structure is made up with different data and data containers. In this topic we will take a quick glance at some of the basics components of that structure to help understand how the software components is handled by the system. Figure 5-1 provides an overview of the SMP/E zones and libraries.

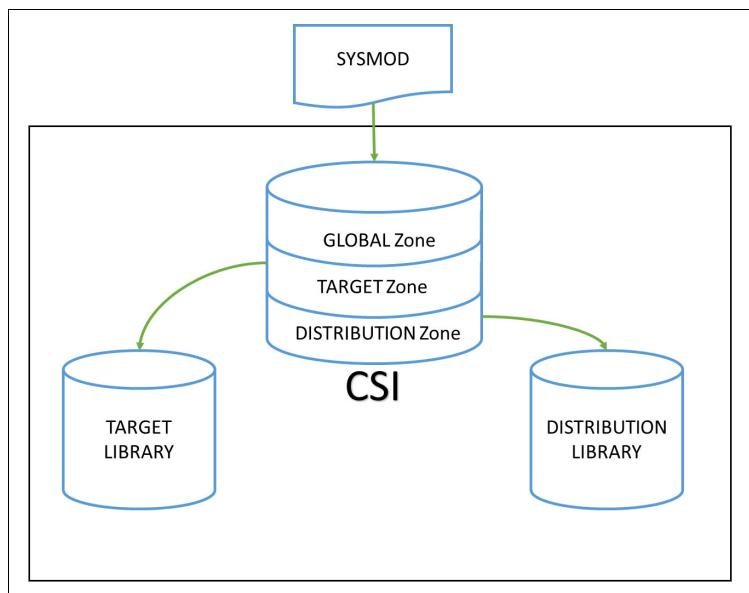


Figure 5-1 SMP/E zones and libraries

Other data sets and libraries that make part of the SMP/E infrastructure. For more information, refer to the SMP/E User's Guide.

5.2.1 SMP/E libraries

When SMP/E processes SYSMODs, it installs the elements in the appropriate libraries and updates its own records of the processing it has done. SMP/E installs program elements into two types of libraries:

- ▶ Distribution libraries (DLIBs) contain the master copy of each element for a system. They are used as input to the system generation process to build target libraries for a new system. They are also used by SMP/E for backup when elements in the target libraries have to be replaced or updated.
- ▶ Target libraries (TLIBs) contain the executable code needed to run your system (for example, the libraries from which you run your production system or your test system).

5.2.2 Consolidated software inventory

The consolidated software inventory (CSI), also known as the *SMPCSI*, data sets contain all the information SMP/E needs to track the distribution and target libraries. The CSI contains an entry for each element in its libraries and each one of the entries enclose the element name, type, history, how the element was introduced into the system, and a pointer to the element in the distribution and target libraries. It has a similar concept to the catalog, which was introduced in Chapter 1, “z/OS implementation and daily maintenance” on page 1.

Entries representing elements found in the distribution libraries are contained in the distribution zone. In the same way, the entries representing elements found in the target libraries are contained in the target zone.

5.2.3 SMP/e zones

Table 5-1 lists the zones in each SMP/E and the different information and pointers for each one.

Table 5-1 Three zones in each SMP/E

GLOBAL zone	TARGET zone	DISTRIBUTION zone
Identification and Description of Target and Distribution Zones	Content, Structure & status of Target Libraries	Content, Structure & status of Distribution Libraries
Processing Options	Pointer to the associated distribution zone	Pointer to the associated target zone
SYSMODs Status		
Exception Data for SYSMODs		

Exception data (SYSMOD HOLDDATA)

In SMP/E, when we speak of exception data, we are usually referring to HOLDDATA. HOLDDATA is often supplied for a product to indicate a specific SYSMOD should be held from installation. Reasons for holding a SYSMOD can be:

- ▶ A PTF is in error, normally known as PE, and should not be installed until the error is corrected (ERROR HOLD).
- ▶ Certain system actions might be required before SYSMOD installation (SYSTEM HOLD).
- ▶ The user might want to perform some actions before installing the SYSMOD (USER HOLD).

This HOLDDATA is also an entry in the Global Zone, created during the reception. The HOLDDATA is deleted when all the conditions for each HOLD are met.

5.3 SMP/E data set allocation

The processing of SMP/E commands requires a variety of data sets. You can either provide the data definition (DD) statements for these data sets (such as in a cataloged procedure) or have SMP/E allocate the data sets dynamically. Dynamic allocation has the advantage that data sets are allocated only as they are needed; DD statements must successfully allocate all data sets, regardless of whether they are needed for the command being processed.

There are some drawbacks to using DD statements. For example, all the data sets defined by DD statements must be successfully allocated, regardless of whether they are needed for the

command being processed. In addition, if you are running several SMP/E commands, you must be careful to use the correct DD statements for each command. If you are processing zones that are in different CSI data sets, you must make sure to provide a DD statement that points to each of those zones and their associated CSIs.

5.4 SYSMOD Processing and SMP/E Data Display

Figure 5-2 depicts SYSMOD processing. In order to install a SYSMOD, the first action is to RECEIVE it. This action copies the SYSMOD into data sets that are used by the SMP/E (SMPPTS, SMPLIB, and global zone within the CSI).

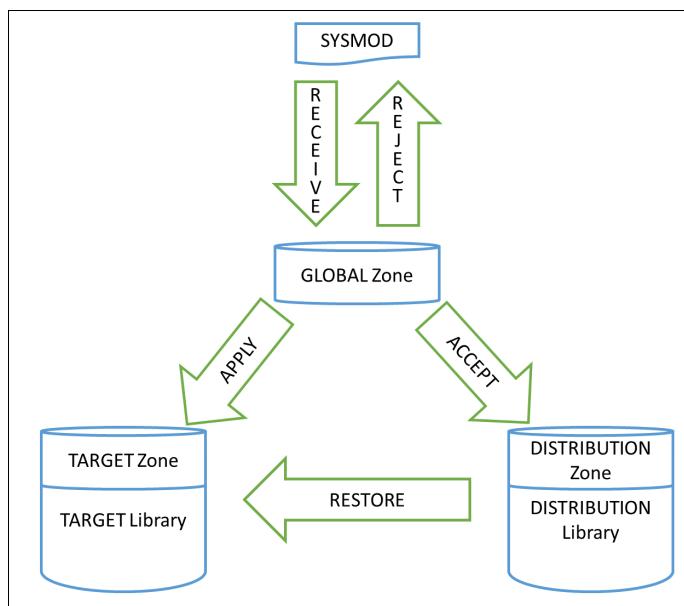


Figure 5-2 SYSMOD processing

The opposite action, shown in Figure 5-2, is the REJECT command. The REJECT command allows you to clean up the global zone, SMPPTS, and associated entries and data sets. The REJECT command is helpful if the SMPPTS is being used as a permanent database for all SYSMODs, including those that have been accepted. You can also use it to purge old data from the global zone and SMPPTS. However, sometimes you need to delete an already received SYSMOD.

Note: Use the CHECK operand while using the REJECT command to perform a trial run of the command without actually updating any zones or data sets. This action provides a way to test for errors that might occur during actual processing and to receive reports on the changes that would be made.

After a SYSMOD is received, the next command used is APPLY. This action instructs the SMP/E to install the elements supplied by the SYSMOD into the operating (or target) system libraries.

Note: Similar to the REJECT command, the CHECK operand can be used with the APPLY command to perform a trial run of the command without actually updating any zones or data sets.

In some cases you might want or need to remove a SYSMOD that was applied to target libraries (for example, an installed a fix for a problem got unexpected results). If you have not yet accepted the SYSMOD into the distribution libraries, you can use the **RESTORE** command to remove it from the target libraries.

The final step for a SYSMOD, after it is fully tested and results are as expected in terms of performance and stability, is to **ACCEPT** the SYSMOD. After you accept a SYSMOD, you cannot restore its element to a previous level. The **ACCEPT** command updates the distribution libraries so they are available for backup of any future SYSMODs. After you accept a SYSMOD into the distribution libraries, you cannot use the **RESTORE** command to remove it from the target libraries.

5.4.1 Displaying SMP/E data

The SMP/E CSI and other primary data sets contain a great deal of information you might find useful when installing new elements or functions, preparing user modifications, or debugging problems. You can use the following methods to display that information as well as information about modules, macros, and other elements:

- ▶ **SMP/E query dialogs**
- ▶ **LIST** commands
- ▶ **REPORT** commands

SMP/E query dialogs

Interacting directly with the SMP/E with the dialogs or panels, simplifies many tasks since you do not need to remember every single command and operator. SMP/E dialogs help you interactively query the SMP/E database and create and submit jobs to process SMP/E commands. Creating the JCLs is useful, for example, when you need to repeat an operation and you do not want to go through the dialogs every time.

Figure 5-3 shows an example of the SMP/E primary option menu.

```
----- SMP/E PRIMARY OPTION MENU ----- SMP/E 36.85
==> -
          More: +
```

0 SETTINGS	- Configure settings for the SMP/E dialogs
1 ADMINISTRATION	- Administer the SMPCSI contents
2 SYSMOD MANAGEMENT	- Receive SYSMODs and HOLDDATA and install SYSMODs
3 QUERY	- Display SMPCSI information
4 COMMAND GENERATION	- Generate SMP/E commands
5 RECEIVE	- Receive SYSMODs, HOLDDATA and support information
6 MIGRATION ASSISTANT	- Generate Planning and Migration Reports
7 ORDER MANAGEMENT	- Manage ORDER entries in the global zone
D DESCRIBE	- An overview of the dialogs
T TUTORIAL	- Details on using the dialogs
W WHAT IS NEW	- What is New in SMP/E

Specify the name of the CSI that contains the global zone:
SMPCSI DATA SET ==>
(Leave blank for a list of SMPCSI data set names.)

Figure 5-3 SMP/E Primary panel

SMP/E LIST command

The SMP/E data sets (the global zone, target zones, distribution zones, SMPPTS, SMPLOG, and SMPSCDS) contain a great deal of information that you might find useful when installing a new function, preparing a user modification, or debugging a problem. You can use the **LIST** command to display that information.

Note: To list entries in a data set, you must specify the name of the zone that contains the entries to be listed on the **SET BOUNDARY** command.

Suppose you need to determine whether target zone MVST100 contains entries for any elements owned by function XYZ1100. You can use the **LIST** command with the **FORFMID** operand, as shown in Example 5-1.

Example 5-1 Using the LIST command with the FORFMID operand

```
SET BDY(MVST100).      /* Set to target zone. */
LIST SYSMOD             /* List ALL SYSMOD      */
FORFMID(XYZ1100).       /* for this FMID.      */
```

5.4.2 SMP/E reports

This section includes a brief introduction of some of the reports that you can use with SMP/E. You can find the complete list and details in *SMP/E for z/OS Commands*, SA23-2275.

RECEIVE processing reports

The following **RECEIVE** command processing reports are available:

- ▶ A *Summary Report* provides a listing with all the SYSMODs that were processed during the **RECEIVE** command run. It shows which SYSMODs were received, which were not received, and why they were not received.
- ▶ The *Exception SYSMOD data report* provides a quick summary of the HOLDDATA information processed during the **RECEIVE** command run. It lists the SYSMODs that require special handling or that are in error, those SYSMODs that no longer require special handling, or those that have had an error fixed.
- ▶ The *File Allocation report* provides a list of data sets used for the **RECEIVE** command processing and supplies information about these data sets.

APPLY and ACCEPT reports

The following **APPLY** and **ACCEPT** command report are available:

- ▶ At the completion of the **APPLY** and **ACCEPT** command, the *Unresolved HOLD reason report* is generated to identify the SYSMODs that were terminated because of unresolved HOLD conditions.
- ▶ When a SYSMOD has a HOLD condition “bypassed,” it appears in the *Bypassed HOLD Reason Report*.
- ▶ The *Summary of bypassed and unresolved HOLD reason report* is produced at the completion of **APPLY** and **ACCEPT** command processing only if the *Unresolved HOLD Reason Report* or the *Bypassed HOLD Reason Report* is produced. This report can be useful in coding the proper **BYPASS** operands if you want to bypass error conditions.

Note: This report does *not* indicate whether the reason ID caused termination or was bypassed. That information is contained in the *Unresolved HOLD Reason Report* and the *Bypassed HOLD Reason Report*.

- ▶ The *Deleted SYSMOD* report shows the function SYSMODs that were deleted and all the service SYSMODs that were applicable to those functions.

Other reports

The following reports are also available:

- ▶ The *Cross-zone summary report*, produced during **APPLY** and **RESTORE** command processing, summarizes the cross-zone work that has been done, the cross-zone work that has not been done, and why.
- ▶ The *Reject summary report*, produced at the completion of the **REJECT** command, summarizes the processing that occurred for SYSMODs and other data.
- ▶ The *Causer SYSMOD Summary report* lists the causer SYSMODs and a summary of the related messages describing the errors that need to be fixed to successfully process the SYSMODs. The report is produced by the SMP/E that performs a root cause analysis for the **APPLY**, **ACCEPT** and **RESTORE** commands.



Language Environment

This chapter introduces the Language Environment architecture, which is a set of constructs and interfaces that provides a common runtime environment and runtime services for all Language Environment-conforming programming language products (those products that adhere to Language Environment's common interface). It includes an overview of Language Environment and descriptions of the Language Environment's full program model, callable services, storage management model, and debug information.

Language Environment provides a common runtime environment for IBM versions of certain high-level languages (HLLs), namely, C, C++, COBOL, Fortran, and PL/I, in which you can run existing applications written in previous versions of these languages as well as in the current Language Environment-conforming versions, without having to recompile or relink-edit the applications. Prior to Language Environment, each of the HLLs had to provide a separate runtime environment. POSIX-conforming C applications can use all Language Environment services.

Language Environment combines essential and commonly used runtime services, such as routines for runtime message handling, condition handling, storage management, date and time services, and math functions, and makes them available through a set of interfaces that are consistent across programming languages. With Language Environment, you can use one runtime environment for your applications, regardless of the application's programming language or system resource needs, because most system dependencies have been removed.

6.1 Language Environment

Today, enterprises need efficient, consistent, and less complex ways to develop quality applications software and to maintain their existing inventory of applications. The trend in application development is to create modules and share code. Language Environment gives you a common environment for all Language Environment-conforming high-level language (HLL) products. See 6.3, “High-level language and programs” on page 159.

In the past, programming languages also had limited ability to call each other and behave consistently across different operating systems. This has constrained those who wanted to use several languages in an application. Programming languages have had different rules for implementing data structures and condition handling, and for interfacing with system services and library routines.

6.2 Assembler language and programs

A computer can understand and interpret only machine language instructions. Machine language is in binary form and, thus, is difficult to write. The assembler language is a symbolic programming language that you can use to code instructions in a mnemonic format instead of coding in machine language in binary format. Figure 6-1 shows the basics of the assembler language.

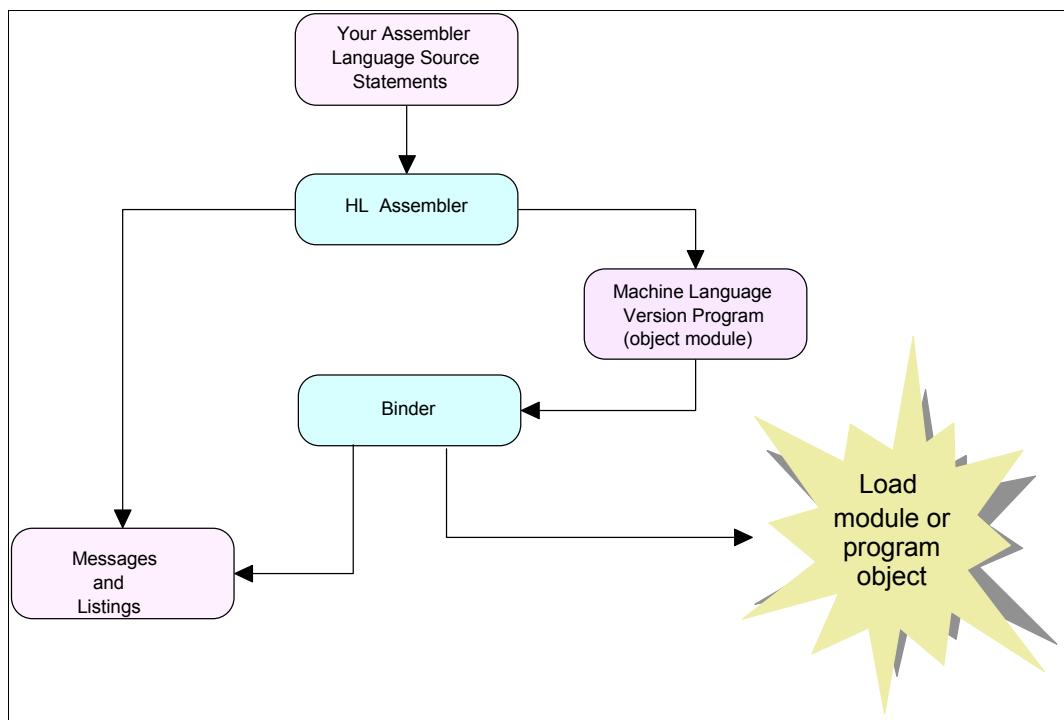


Figure 6-1 Assembler language

Assembler language lets you use meaningful symbols made up of alphabetic and numeric characters, instead of just the binary digits 0 and 1 used in machine language instructions. This makes your coding easier to read, understand, and change. Assembler is also enriched by a huge set of macros that make it a powerful language; however, its major quality is the performance of its object code at run time.

One of the key aspects of a computer language is its readability and capacity to document itself. This aspect is mandatory in a commercial environment, where applications should last for many years (to justify the investment). Longevity of an application means that different generations of programmers will be in charge of writing and maintaining the code, which implies that readability is a required quality of the language. Certain modern languages like C do not follow this rule. However, if you write an assembler program using all the documenting capability of its features, chances are that you will produce more readable code than some HLLs allow.

The assembler must translate the symbolic assembler language that you wrote into machine language before the computer can run your program. The specific procedures followed to complete this translation vary according to the operating system that you are using. However, the method is basically the same and consists of the following process:

Your program, written in the assembler language, becomes the *source module* that is input to the assembler program. The assembler processes your source module and produces an *object module* in machine language (called object code). The *object module* can be used as input to be processed by the binder. The binder produces a *load module* that can be loaded later into the main storage of the computer. When your program is loaded, it can then be run. Your source module and the object code produced are printed, along with other information, on a program listing.

Using assembler language

The assembler language is the symbolic programming language that lies closest to the machine language in form and content. You will, therefore, find the assembler language useful when:

- ▶ You need to control your program closely, down to the byte level and even to the bit level.
- ▶ You must write subroutines for functions that are not provided by other high-level programming languages, such as COBOL, FORTRAN, or PL/I.
- ▶ You need good performance at execution time.

The assembler language is made up of statements that represent either instructions or comments. The instruction statements are the working part of the language and are divided into the following three groups:

- ▶ Machine instructions

A machine instruction is the symbolic representation of a machine language instruction. It is called a machine instruction because the assembler translates it (one-to-one) into the machine language code that the computer can run.

- ▶ Assembler instructions

An assembler instruction is a request to the assembler to do certain operations during the assembly of a source module; for example, defining data constants, reserving storage areas, and defining the end of the source module. Except for the instructions that define constants, and the instruction used to generate no-operation instructions for alignment, the Assembler does not translate assembler instructions into object code.

- ▶ Macro instructions

A macro instruction is a request to the assembler program to process a predefined sequence of instructions called a macro definition. From this definition, the Assembler generates machine and assembler instructions, which it then processes as though they were part of the original input in the source module.

IBM supplies macro definitions for I/O, data management, and supervisor operations that you can call for processing by coding the required macro instruction.

You can also prepare your own macro definitions, and call them by coding the corresponding macro instructions. Rather than code all of this sequence each time it is needed, you can create a macro instruction to represent the sequence and then, each time the sequence is needed, simply code the macro instruction statement. During assembly, the sequence of instructions represented by the macro instruction is inserted into the source program.

Assembler program relationship to z/OS

The assembler program, also referred to as the assembler, processes: the machine, assembler instructions, and macro instructions you have coded (source statements) in the assembler language, and produces an object module in machine language.

The assembler in z/OS is called High Level Assembler. z/OS provides the assembler with services for:

- ▶ Assembling a source module
- ▶ Running the assembled object module (after being link-edited) as a program

In writing a source module, you must include instructions that request any required service functions from z/OS.

z/OS provides the following services:

- ▶ For assembling the source module:
 - A control program
 - Sequential data sets to contain source code
 - Libraries to contain source code and macro definitions
 - Utilities
- ▶ For preparing the execution of the Assembler program as represented by the object module:
 - A control program
 - Storage allocation
 - Input and output facilities

It can be difficult to write an assembler language program using only machine instructions. The assembler provides additional functions, not discussed here, that make this task easier.

The assembler-generated object module must be link-edited by the Binder utility program to be transformed into an executable program. This executable program is called a *load module* when it is stored in a PDS library, and a *program object* when it is stored in a PDSE.

6.3 High-level language and programs

A HLL is a *programming language* above the level of assembler language and below that of program generators and query languages. Figure 6-2 provides an overview of the HLL.

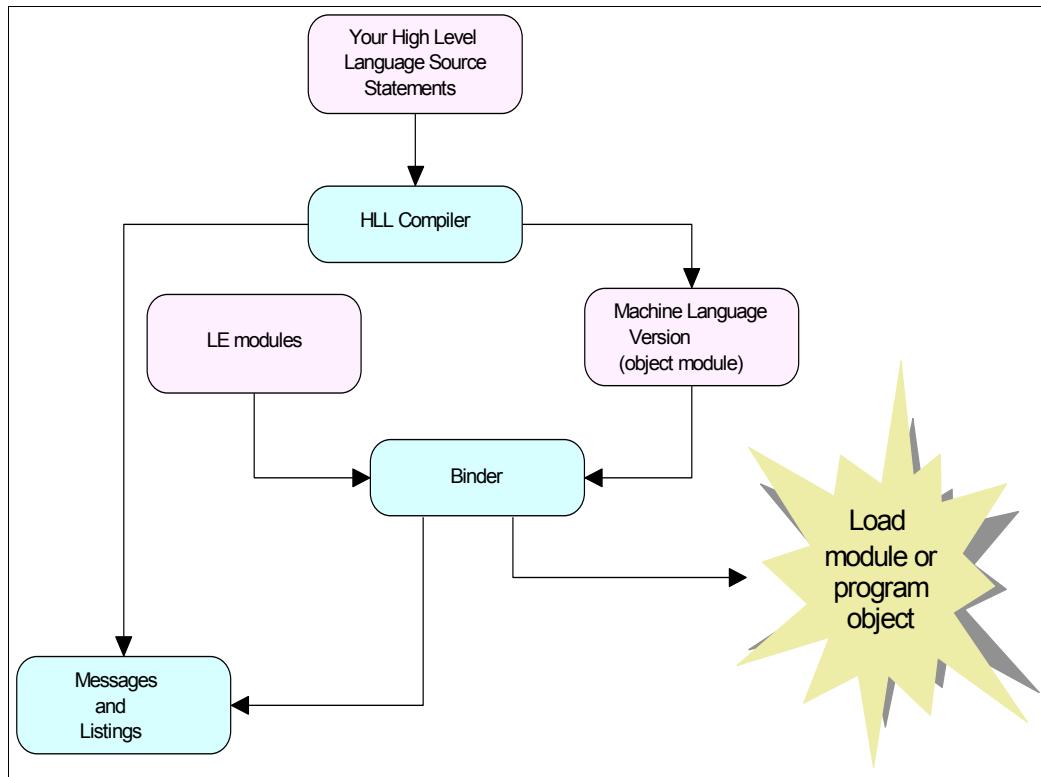


Figure 6-2 High-level language

The statements created in one of these languages can be the input to a computer program called a *compiler* that translates the HLL statements in machine language instructions, *object code*. It has instructions recognized by the processor of the specific platform where it is supposed to be executed. This object code is the fabric of the executable program, also called the *load module* or *program object*.

When we refer to HLL in this chapter, we are referring to any of the following languages:

- ▶ C or C++
- ▶ COBOL
- ▶ FORTRAN
- ▶ PL/I

6.4 Creating execution programs

In the 60s, the object module created from the source by the HLL compiler or by the assembler had all the machine code instructions, as declared in the source. Logically, the only code missing in the object module was the operating system instructions needed to execute certain functions such as an I/O operation. In this case, the compiler (or the assembler) generated an API call that at execution time would call the operating system for the required function.

In the 70s, we had the first revolution in these matters. To reduce the cost, increase the speed, and simplify the construction of compilers, the concept of pre-compiled routines was introduced. Then, together with the compiler code, IBM sent lots of object modules kept in libraries. These object modules had the logic of: data format convention, arithmetic functions, preparing an I/O request to be executed, and so on. When the compiler needed one of these routines it just made an external call to an existing object module. Later, when the Binder link-edited the source, those mentioned object modules were naturally included, making just one load module.

Finally, in the 90s the last revolution occurred. The pre-compiled object modules are replaced by pre link-edited load modules stored in runtime libraries sent with compile code. The external calls generated by the compilers are replaced by APIs that dynamically, through z/OS program management functions (as such LINK and XCTL), load and pass the CPU control to such load modules.

There are a few technical justification for such an implementation:

- ▶ The load module generated from the compilation is small (compared with the previous method) and, recalling that in a commercial program only 20% of the code (kernel) is executed frequently, we are saving virtual and central storage. However, there is a performance issue because when a needed function is implemented in a load module stored in a runtime library, an I/O operation (with an implicit wait) is needed during the task execution. LLA and VLF can be used to minimize this effect.
- ▶ Compiler manufacturers such as IBM can write common load modules for usual functions, such as: message handling, condition handling, program storage management, date and time services, math functions, abnormal termination, and others. Those load modules can be available in runtime libraries being shared and consistent across different programming languages, such as: Cobol, PL/I, Fortran, C, C++. For the customer, the advantage is to have the same behavior for a program independent of the language used. Also, it is much easier for a program written in Cobol to make a call to another in C.

For the manufacturer the cost of writing and maintaining different languages is much less.

The last revolution created the structure for IBM to implementing Language Environment. Language Environment uses runtime libraries for dynamic load of common load modules, but also uses common Syslibs, including shared object modules, during Binder link-edit.

So, restating the Language Environment definition: “Language Environment provides a common runtime environment for IBM versions of certain high-level languages (HLLs), namely, C, C++, COBOL, Fortran, and PL/I, in which you can run existing applications written in previous versions of these languages, as well as in the current Language Environment-conforming versions, without having to recompile or re-link-edit the applications. Prior to Language Environment, each of the HLLs had to provide a separate runtime environment. POSIX-conforming C applications can use all Language Environment services”

6.5 IBM compiler products and Language Environment

The z/OS Language Environment is the prerequisite runtime environment for applications generated with the following IBM compiler products:

- ▶ z/OS XL C/C++
- ▶ IBM OS/390® C/C++
- ▶ C/C++ for MVS/ESA
- ▶ COBOL for OS/390 & VM
- ▶ COBOL for MVS & VM (formerly COBOL/370)

- ▶ Enterprise COBOL for z/OS
- ▶ Enterprise COBOL for z/OS and OS/390
- ▶ AD/Cycle C/370 Compiler
- ▶ Enterprise PL/I for z/OS
- ▶ Enterprise PL/I for z/OS and OS/390
- ▶ IBM VisualAge® PL/I for OS/390
- ▶ PL/I for MVS & VM
- ▶ AD/Cycle PL/I for MVS & VM
- ▶ VisualAge for Java, Enterprise Edition for OS/390
- ▶ VS FORTRAN and FORTRAN IV (in compatibility mode)

Enterprise COBOL for z/OS is IBM's strategic COBOL compiler for the zSeries platform. Enterprise COBOL consists of features from IBM COBOL, VS COBOL II, and OS/VS COBOL, with additional features such as multithread enablement, Unicode, XML capabilities, object-oriented COBOL syntax for Java interoperability, integrated CICS translator, and integrated DB2 coprocessor (zIIP). Enterprise COBOL, as well as IBM COBOL and VS COBOL II, supports the COBOL 85 Standard.

Language Environment provides a single language runtime environment for COBOL, PL/I, C, and FORTRAN. In addition to support for existing applications, Language Environment also provides common condition handling, improved interlanguage communication (ILC), reusable libraries, and more efficient application development. Application development is simplified by the use of common conventions, common runtime facilities, and a set of shared callable services. Language Environment is required to run Enterprise COBOL programs.

In many cases, you can run compiled code generated from the previous versions of these compilers. A set of assembler macros is also provided to allow assembler routines to run with Language Environment.

6.6 Language Environment standards

The UNIX market is competitive, so each software house adds other functions to the kernel. This creates "UNIX proprietary" code. To address the problem, standards such as Portable Operating System Interface (POSIX) were introduced to define standard interfaces based on UNIX.

The IEEE POSIX standard is a series of industry standards for code and user interface portability. The UNIX System Services component in z/OS conforms with POSIX, so applications written for a UNIX-like operating system can run on z/OS. z/OS systems' UNIX System Services is also branded the UNIX by X/Open committee. C language programs can access operating system services through a set of standard language bindings. Programs written in C, due to UNIX System Services and z/OS Language Environment, can call C language functions defined in the POSIX standard from their C applications. Also they can run applications that conform to ISO/IEC 9899:2011 and conform to XPG4.2 specifications.

Applications that call POSIX functions can perform limited Interlanguage Communication (ILC) under Language Environment (see *z/OS Language Environment Writing Interlanguage Communication Applications*, SA22-7563, for details). In addition, C POSIX-conforming applications can use all Language Environment services.

6.7 Language Environment components

Language Environment consists of the following components:

- ▶ *Basic routines* that support starting and stopping programs, allocating storage, communicating with programs written in different languages, and indicating and handling conditions.
- ▶ *Common library* includes common services, such as messages, date and time functions, math functions, applications utilities, system services, and subsystem support, that are commonly needed by programs running on the system. These functions are supported through a library of callable services.
- ▶ *Language-specific portions of the runtime library* because many language-specific routines call Language Environment services. However, behavior is consistent across languages.

All these components are load modules stored in runtime libraries.

POSIX support is provided in the Language Environment base and in the C language-specific library.

Note: With AMODE 64 only the z/OS C/C++ language-specific portion of the runtime library is available.

6.8 Language Environment common runtime environment

Figure 6-3 on page 163 shows that each HLL has its specific run time (for load modules) and SYSLIB (for object modules), and shares with other HLLs a Common Execution Library (CEL). Further shows that the load modules produced in this way can be executed in different operating environments under z/OS, VSE and z/VM.

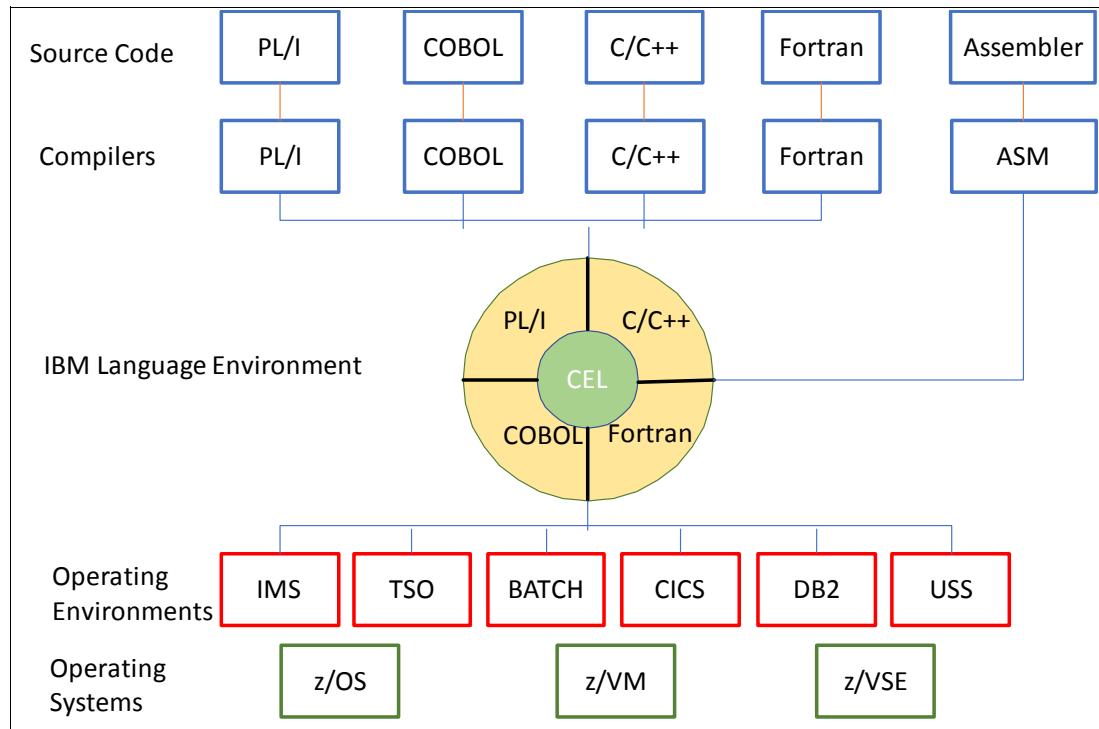


Figure 6-3 Language Environment overview

6.9 Language Environment runtime environment for AMODE 64

Figure 6-4 on page 164 illustrates the common environment that Language Environment AMODE 64 creates. It also shows that the load modules produced in this way can be executed only under z/OS. Those load modules are loaded below the bar (2-G) but can access addresses above such bar.

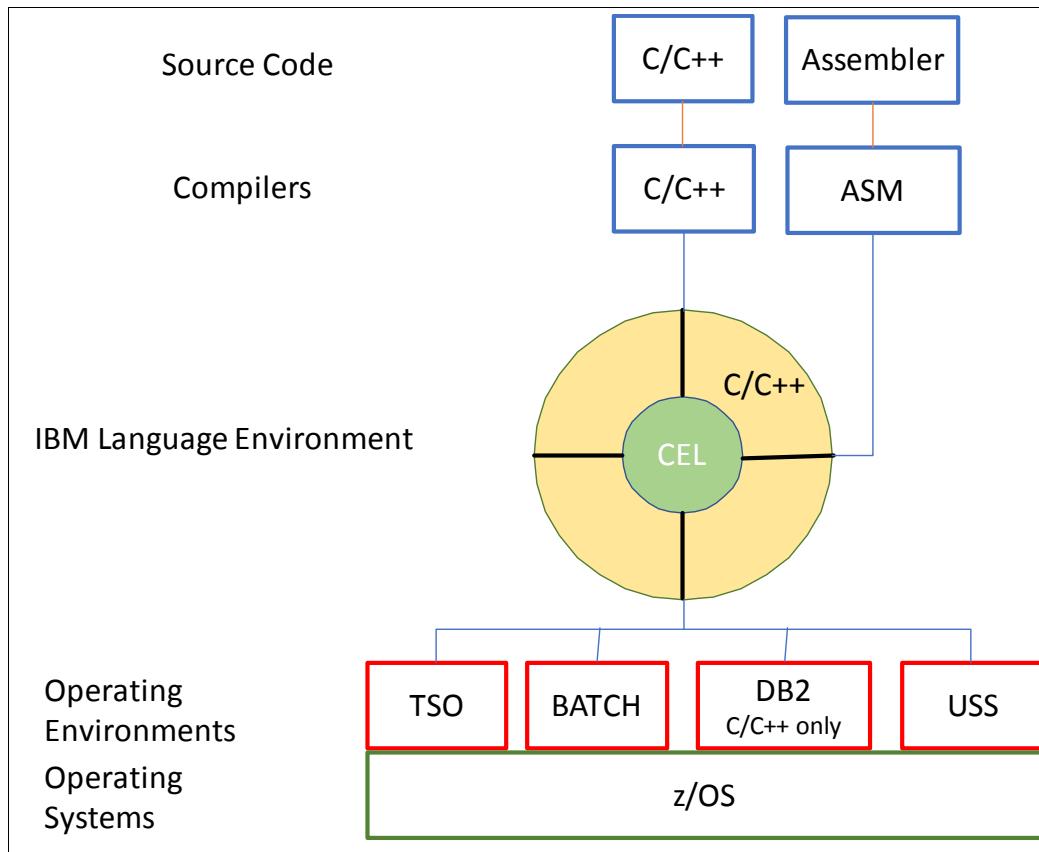


Figure 6-4 Language Environment common runtime environment for AMODE 64

In the 64-bit addressing mode (AMODE 64) supported by Language Environment, addresses are 64 bits in length, which allows access to virtual storage up to 16 exabytes. Although this is an extremely high address, consider a few important facts:

- ▶ Existing or new Language Environment applications that use AMODE 24 or AMODE 31 can continue to run without change. They run using the same Language Environment services that existed before 64-bit addressing was introduced, and these services will continue to be supported and enhanced.
- ▶ Language Environment applications that use AMODE 64 are not compatible with applications that use AMODE 24 or AMODE 31. The only means of communication between AMODE 64 and AMODE 24 or AMODE 31 applications is through mechanisms that can communicate across processes or address spaces. However, Language Environment applications that use AMODE 64 can run with existing applications that use AMODE 24 or AMODE 31 on the same physical zSeries system.
- ▶ Where necessary, there are new Language Environment runtime options to support AMODE 64 applications. The new runtime options primarily support the new stack and heap storage located above the bar. All other existing runtime options continue to be supported and enhanced for AMODE 24 and AMODE 31 applications.

The z/OS C/C++ compiler with the LP64 compiler option is the IBM Language Environment-conforming language compiler that currently supports 64-bit addressing.

Along with the change in addressing mode to use 64 bits, the other important consideration is that long data types also use 64 bits. The industry standard name for this data model is LP64, which translates roughly to “long and pointer data types use 64 bits.” The Language

Environment support for AMODE 24 and AMODE 31 applications is ILP32, meaning “integer, long, and pointer data types use 32 bits.”

Language Environment also provides new assembler macros that support creating Language Environment-conforming assembler applications that run AMODE 64.

6.10 Object code and Language Environment

The object code is included in the load module at three different times in z/OS:

- ▶ At compile time.
- ▶ At binder time, through the INCLUDE statement or resolving external references from the SYSLIB library.
- ▶ At execution time, through MVS dynamic link macros such as LINK and LOAD, which fetch to memory, or code from the load modules library. This approach is also called late binding.

The libraries containing the code invoked dynamically are called the runtime environment. Language Environment establishes a common runtime environment plus a common set of the SYSLIB libraries for all participating HLLs. However, due to many language-specific functions, there is still a need of some language-specific libraries as C/C++, COBOL, FORTRAN, PL/I.

6.11 Callable services

Language Environment helps you create mixed-language applications and gives you a consistent method of accessing common, frequently used services. Building mixed-language applications is easier with Language Environment-conforming routines because Language Environment establishes a consistent environment for all languages in the application.

Language Environment provides the base for future IBM language library enhancements in the z/OS environment. Language Environment callable services are divided into the following groups:

- ▶ Communicating Conditions Services
- ▶ Condition Handling Services
- ▶ Date and Time Services
- ▶ Dynamic Storage Services
- ▶ General Callable Services
- ▶ Initialization/Termination Services
- ▶ Locale Callable Services
- ▶ Math Services
- ▶ Message Handling Services
- ▶ National Language Support Services

Because Language Environment provides a common library, with services that you can call through a common callable interface, the behavior of your applications will be easier to predict. However, you do not need to call specifically such services, they are automatically included in your program depending on the statements you use in your HLL source code. Language Environment's common library includes common services such as messages, date and time functions, math functions, application utilities, system services, and subsystem support. The language-specific portions of Language Environment provide language interfaces and specific services that are supported for each individual language.

The following example illustrates how to invoke a Language Environment service in COBOL for z/OS:

```
CALL "CEEFDMDT" USING COUNTRY, PICSTR, FC.
```

You should use a CALL statement with the correct parameters for that particular service. See *z/OS Language Environment Programming Guide*, SA22-7561, for details.

The language-specific portions of Language Environment provide language interfaces and specific services that are supported for each individual language. Language Environment is accessed through defined common calling conventions.

Note: The callable services mentioned in 6.8, “Language Environment common runtime environment” on page 162 are for AMODE 31 only. For AMODE 64, none of the application writer interfaces (AWIs) will be supported in their present form. There can be C functions that provide similar functionality for some of the AWIs. A few nonstandard C functions have been added to provide the functionality of some of the AWIs. See *z/OS C/C++ Run-Time Library Reference-SA*, SA22-7821, for details.

6.12 Language Environment program management terms and terminology

The Language Environment program management model provides a framework within which an application runs. It is the foundation of all of the component models—condition handling, runtime message handling, and storage management—that comprise the Language Environment architecture. The program management model defines the effects of programming language semantics in mixed-language applications and integrates transaction processing and multithreading.

Some terms used to describe the program management model are common programming terms; other terms are described differently in other languages. It is important that you understand the meaning of the terminology in a Language Environment context as compared to other contexts.

General programming terms

Language Environment uses the following general programming terms:

- ▶ Application program: A collection of one or more programs cooperating to achieve particular objectives such as inventory control or payroll.
- ▶ Environment: In Language Environment, normally a reference to the runtime environment of HLLs at the enclave level.

Language Environment terms and HLL equivalents

The following terms are some Language Environment terms and HLL equivalents:

- ▶ *Routine*: In Language Environment, this term refers to either a *procedure*, *function*, or *subroutine*. The equivalent in HLL terms:
 - COBOL: *program*
 - C/C++: *function*
 - PL/I: *procedure*, *BEGIN block*

- *Enclave*: In Language Environment, a collection of routines, one of which is named as the main routine. The enclave contains at least one thread.

Equivalent HLL terms:

- COBOL: *run unit*
- C/C++: *program*, consisting of a main C function and its sub-functions
- PL/I: *main procedure* and its subroutines
- FORTRAN: *program* and its subroutines

- *Process*: The highest level of the Language Environment program management model. A process is a collection of resources, both program code and data, and consists of at least one enclave.
- *Thread*: An execution construct that consists of synchronous invocations and terminations of routines. The thread is the basic runtime path within the Language Environment program management model, and is dispatched by the system with its own runtime stack, instruction counter, and registers. Threads can exist concurrently with other threads.

Terminology for data

The following terminology describes the types of data used in a Language Environment environment:

- Automatic data

Data that is allocated with the same value on entry and re-entry into a routine if it has been initialized to that value in the semantics of the language used. The data does not persist across calls. The scope of automatic data is a routine invocation within an enclave.

- External data

Data that persists over the lifetime of an enclave and retains last-used values whenever a routine is re-entered. The scope of external data is that of the enclosing enclave; all routines invoked within the enclave recognize the external data.

- Local data

The scope of local data is that of the enclosing enclave; however, local data is recognized only by the routine that defines it.

Equivalent HLL terms:

- C/C++: *local data*
- COBOL: *WORKING-STORAGE data items* and *LOCAL-STORAGE data items*
- PL/I: data declared with the *PL/I INTERNAL attribute*

6.13 Language Environment program management model

Three entities — process, enclave, and thread — are at the core of the Language Environment program management model. Figure 6-6 shows the relationship between processes, enclaves, and threads. It illustrates the simplest form of the Language Environment program management model and how resources such as storage are managed.

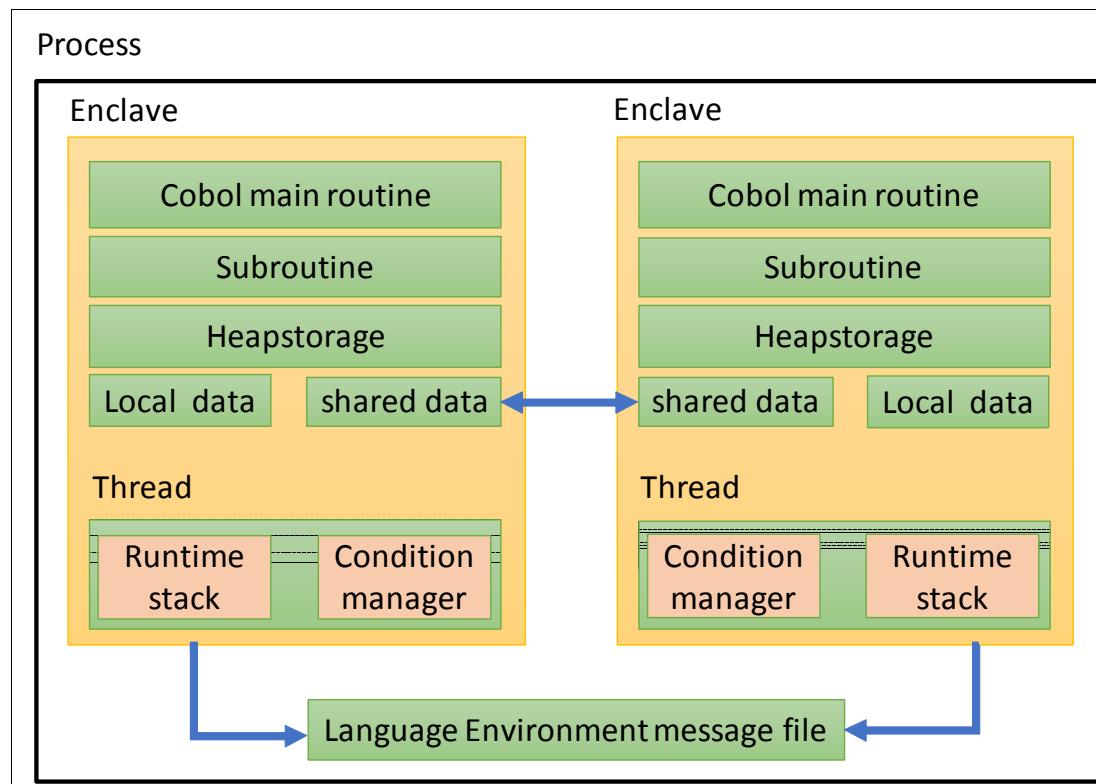


Figure 6-5 Language Environment program management

Process

A *process* is the highest level component of the Language Environment program model and consists of at least one enclave. Each process has an address space that is logically separate from those of other processes. Except for communications with each other using certain Language Environment mechanisms, no resources are shared between processes; processes do not share storage, for example. A process can create other processes. However, all processes are independent of one another; they are not hierarchically related.

Language Environment generally does not allow language file sharing across enclaves, nor does it provide the ability to access collections of externally stored data. However, the PL/I standard SYSPRINT file can be shared across enclaves. The Language Environment message file also can be shared across enclaves because it is managed at the process level. The Language Environment message file contains messages from all routines running within a process, making it a useful central location for messages generated during run time.

Processes can create new processes and communicate to each other using Language Environment-defined communication, for such things as indicating when a created process has been terminated.

Enclaves

Enclave is the key feature of the program management model, a collection of the routines that make up an application. The enclave is the equivalent of any of the following:

- ▶ A run unit in COBOL
- ▶ A program, consisting of a main function and its sub-functions, in C
- ▶ A main procedure and all of its subroutines in PL/I
- ▶ A program and its subroutines in FORTRAN

The enclave consists of one main routine and zero or more subroutines. The main routine is the first to execute in an enclave; all subsequent routines are named as subroutines.

Threads

Each enclave consists of at least one *thread*, the basic instance of a particular routine. A thread is created during enclave initialization, with its own runtime stack that keeps track of the thread's execution, as well as a unique instruction counter, registers, and condition-handling mechanisms. Each thread represents an independent instance of a routine running under an enclave's resources.

Threads share all of the resources of an enclave. A thread can address all storage within an enclave. All threads are equal and independent of one another and are not related hierarchically. A thread can create a new enclave. Because threads operate with unique runtime stacks, they can run concurrently within an enclave and allocate and free their own storage. Because they can execute concurrently, threads can be used to implement parallel processing applications and event-driven applications.

6.14 Language Environment program management full model

Figure 6-6 illustrates the full Language Environment program model, with its multiple processes, enclaves, and threads. It shows each process is within its own address space. An enclave consists of one main routine, with any number of subroutines. A main routine might not be active at all times in a POSIX application if the thread in which the main routine executes terminates before the other threads that it created.

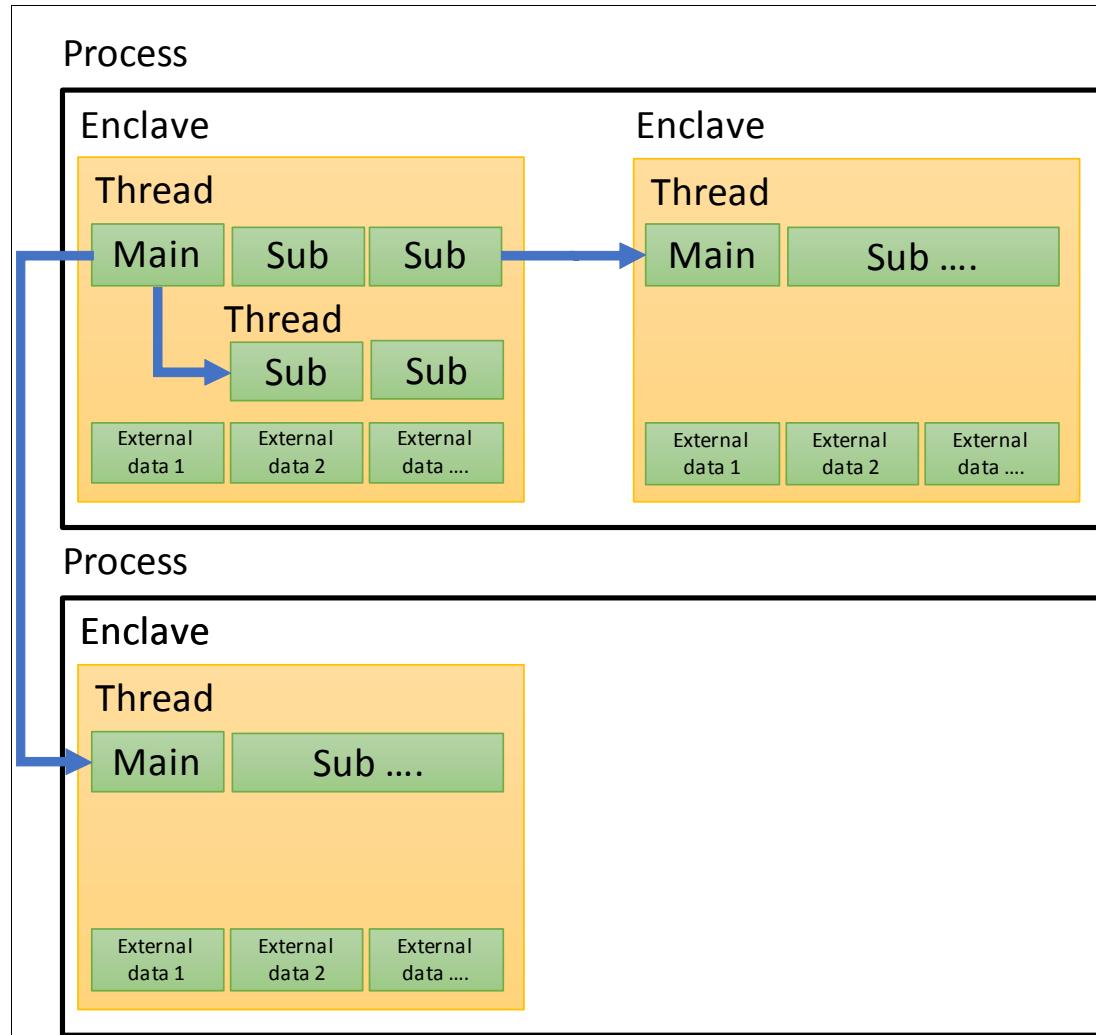


Figure 6-6 Program management full model

External data is available only within the enclave where it resides; notice that even though the external data can have identical names in different enclaves, the external data is unique to the enclave. The scope of external data, as described earlier, is the enclave. The threads can create enclaves, which can create more threads, and so on.

6.15 Language Environment condition handling model description and terminology

For single- and mixed-language applications, the Language Environment runtime library provides a consistent and predictable condition-handling facility. It does not replace current HLL condition handling, but instead allows each language to respond to its own unique environment as well as to a mixed-language environment.

Language Environment condition management gives you the flexibility to respond directly to conditions by providing callable services to signal conditions and to interrogate information about those conditions. It also provides functions for error diagnosis, reporting, and recovery.

Language Environment condition handling is based on the stack frame, an area of storage that is allocated when a routine runs and that represents the history of execution of that routine. It can contain automatic variables, information about program linkage and condition handling, and other information. Using the stack frame as the model for condition handling allows conditions to be handled in the stack frame in which they occur. This allows you to tailor condition handling according to a specific routine, rather than handle every possible condition that could occur within one global condition handler.

A unique feature of Language Environment condition handling is the condition token. The token is a 12-byte (AMODE 24/31) or 16-byte (AMODE 64) data type that contains an accumulation of information about each condition. The information can be returned to the user as a feedback code when calling Language Environment callable services. It can also be used as a communication vehicle within the runtime environment.

Condition handling terminology

The following terminology is used in application programs:

<i>Condition</i>	Any change to the normal programmed flow of a program. In Language Environment, a condition can be generated by an event that has historically been called an exception, interruption, or condition.
<i>Condition handler</i>	A routine invoked by Language Environment that responds to conditions in an application. Condition handlers are registered through the CEEHDLR callable service, or provided by the language libraries, by such constructs as PL/I ON statements.
<i>Condition token</i>	In Language Environment, a data type consisting of 12 bytes with structured fields that indicate various aspects of a condition, including severity, associated message number, and information that is specific to a given instance of the condition.
<i>Feedback code</i>	A condition token value used to communicate information when using the Language Environment callable services.
<i>Resume cursor</i>	Contains the address where execution resumes after a condition is handled. Initially, it will be the point in the application where a condition occurred when it is first reported to Language Environment.
<i>Stack frame</i>	The physical representation of the activation of a routine. The stack frame is allocated on a last in, first out (LIFO) basis and can contain automatic variables, information about program linkage and condition handling, and other information.
	A stack frame is conceptually equivalent to a dynamic save area (DSA) in PL/I, or a save area in assembler.

6.15.1 Language Environment condition handling steps

Language Environment condition handling is performed in the following distinct steps:

Enablement step	Refers to the determination that an exception should be processed as a condition. The enablement step begins at the time an exception occurs in your application. In general, you are not involved with the enablement step; Language Environment determines which exceptions should be enabled (treated as conditions) and which should be ignored, based on the languages currently active on the stack. If you do not specify explicitly or as a default any services or constructs, the default enablement of your HLL applies.
Condition step	Begins after the enablement step has completed and Language Environment determines that an exception in your application should be handled as a condition. In the simplest form of this step, Language Environment traverses the stack beginning with the stack frame for the routine in which the condition occurred and progresses towards earlier stack frames.
Termination step	You can use the TERMTHDACT runtime option to set the type of information you receive after your application terminates in response to a severity 2, 3, or 4 condition. For example, you can specify that a message or dump is to be generated if the application terminates.

6.15.2 Language Environment condition handling signaling

A condition is signaled within Language Environment as a result of one of the following occurrences:

- ▶ A hardware-detected interrupt
- ▶ An operating system-detected exception
- ▶ A condition generated by Language Environment callable services
- ▶ A condition explicitly signaled within a routine

The first three types of conditions are managed by Language Environment and signaled if appropriate. The last can be signaled by user-written code through a call to the service CEEESGL or signaled by HLL semantics such as SIGNAL in PL/I or raise in C.

When a condition is signaled, whether by a user routine, by Language Environment in response to an operating system or hardware-detected condition, or by a callable service, Language Environment directs the appropriate condition handlers in the stack frame to handle the condition.

Condition handling proceeds first with user-written condition handlers in the queue, if present, then with any HLL-specific condition handlers, such as a PL/I ON-unit or a C signal handler, that can be established. The process continues for each frame in the stack, from the most recently allocated to the least recently allocated. If a condition remains not handled after the stack is traversed, the condition is handled by either Language Environment or by the default semantics of the language where the condition occurred.

6.16 Language Environment storage management model

Common storage management services are provided for all Language Environment-conforming programming languages; Language Environment controls stack and heap storage used at run time. It allows single- and mixed-language applications to access a central set of storage management facilities, and offers a multiple heap storage model to languages that do not now provide one. The common storage model removes the need for each language to maintain a unique storage manager and avoids the incompatibilities between different storage mechanisms.

Storage management terminology

The following terminology is used when referencing common storage management services:

<i>Stack</i>	An area of storage in which stack frames can be allocated.
<i>Heap</i>	An area of storage used by Language Environment routines. The heap consists of the initial heap segment and zero or more increments.
<i>Heap element</i>	A contiguous area of storage allocated by a call to the CEEGTST service. Heap elements are always allocated within a single heap segment.
<i>Heap increment</i>	Additional heap segments allocated when the initial heap segment does not have enough free storage to satisfy a request for heap storage.
<i>Heap pool</i>	A storage pool that, when used by the storage manager, can be used to improve the performance of heap storage allocation. This can improve the performance of a multi-threaded application.
<i>Heap segment</i>	A contiguous area of storage obtained directly from the operating system.

Stack storage

Stack storage is the storage provided by Language Environment that is needed for routine linkage and any automatic storage. It is a contiguous area of storage obtained directly from the operating system. Stack storage is automatically provided at thread initialization.

A storage stack is a data structure that supports procedure or block invocation (call and return). It is used to provide both the storage required for the application initialization and any automatic storage used by the called routine. Each thread has a separate and distinct stack.

The storage stack is divided into smaller segments called stack frames, also known as dynamic storage areas (DSAs). A stack frame, or DSA, is dynamically acquired storage composed of a register save area and an area available for dynamic storage allocation for items such as program variables. Stack frames are added to the user stack when a routine is entered, and removed upon exit in a last in, first out (LIFO) manner. Stack frame storage is acquired during the execution of a program and is allocated every time a procedure, function, or block is entered, as, for example, when a call is made to a Language Environment callable service, and is freed when the procedure or block returns control.

For AMODE 64 support, users can specify a stack size above the bar, and can specify the maximum stack size.

Heap storage

Heap storage is used to allocate storage that has a lifetime not related to the execution of the current routine; it remains allocated until you explicitly free it or until the enclave terminates. Heap storage is typically controlled by the programmer through Language Environment runtime options and callable services.

For z/OS Language Environment, heap storage is made up of one or more heap segments consists of an initial heap segment, and, as needed, one or more heap increments, allocated as additional storage is required. The initial heap might or might not be preallocated prior to the start of the application code, depending on the type of heap.

Each heap segment is subdivided into individual heap elements. Heap elements are obtained by a call to one of the heap allocation functions, and are allocated within the initial heap segment by the z/OS Language Environment storage management routines. When the initial heap segment becomes full, Language Environment gets another segment, or increment, from the operating system.

See *z/OS Language Environment Programming Guide*, SA22-7561 and *Language Environment Programming Guide for 64-bit Virtual Addressing Mode*, SA22-7569, for details about the Language Environment storage management model.

6.17 Language Environment runtime options customization

If you do not want to customize Language Environment now, you can put it into production using the IBM-supplied defaults. Or, you can use the instructions in this information to customize Language Environment later, if you choose. For many of the runtime options, application programmers can override the defaults in their code. Application programmers at your site will be the primary users of Language Environment. Ask them what defaults they prefer for runtime options and user exits, which affect their work directly. Doing so ensures that the modifications you make will best support the application programs being developed at your site.

6.17.1 Language Environment system configuration

A systems programmer can modify the IBM-supplied defaults on a system-level or region-level basis, which can save time by reducing the need to override the runtime option defaults as often. An application programmer can further refine these options based on individual program needs. When an application runs, runtime options are merged in a specific order of precedence to determine the actual values in effect.

LNKLST

The Language Environment runtime libraries, SCEERUN, and SCEERUN2, can be placed in LNKLIST, as shown in Figure 6-7. In addition, heavily-used modules can be placed in LPA.

```
LNKLST ADD NAME(LNKLST00) DSN(CEE.SCEERUN)
LNKLST ADD NAME(LNKLST00) DSN(CEE.SCEERUN2)
```

Figure 6-7 Adding the Language Environment runtime libraries to the LNKLIST

After you add Language Environment to the LNKLIST/LPALST, Language Environment is available to all of your applications. To ensure that all applications are functioning correctly under Language Environment before adding Language Environment to your LNKLIST/LPALST, you can temporarily install Language Environment in LNKLIST/LPALST or use STEPLIB.

However, sometimes the SCEERUN data set cannot be placed in LNKLIST, because other applications require the pre-Language Environment runtime libraries. In that case, you can make the Language Environment runtime library available through STEPLIB as described in

Steps for making the runtime library available through STEPLIB. In addition, you can use this approach to test new levels of the runtime libraries.

LPA

Placing routines in the LPA/ELPA reduces the overall system storage requirement by making the routines shareable. Also, initialization/termination (init/term) time is reduced for each application, because load time decreases. For example, if Language Environment modules are not placed in LPA/ELPA, under z/OSUNIX, every fork() call requires approximately 4 MB to be copied into the user address space.

The SCEERUN data set has many modules that are not reentrant, so you cannot place the entire data set in the Link Pack Area (LPALSTxx parmlib). There is a data set called SCEELPA that contains a subset of the SCEERUN modules. All of those must be linked reentrant, reside above the line, and are heavily used by z/OS itself. If you put the SCEERUN data set in the LINKLIST (LNKLSTxx), you can place the SCEELPA data set in LPA list (LPALSTxx). Doing this will improve performance. You cannot place the SCEERUN2 data set as part of a LPALSTxx because it is a PDSE. You must use the Dynamic LPA capability to move individual members of SCEERUN2 into the Link Pack Area.

IBM provides some examples in your SCEESAMP data set for the following environments:

CEEWLPA	Language Environment base modules eligible for the LPA except callable service stubs. Uses Dynamic LPA
EDCWLP	All C/C++ component modules eligible for LPA from SCEERUN and SCEERUN2. Uses Dynamic LPA
IGZWMLP4	All Language Environment COBOL component modules eligible for LPA.
IBMALLP2	All Language Environment PL/I component modules eligible for LPA
IBMPLPA1	MLPA macro for Enterprise PL/I
AFHWMLP2	All Language Environment Fortran modules eligible for LPA

Language Environment system defaults using CEEPRMXX

System-level defaults can be established through a member in the system parmlib called CEEPRMxx or with a SETCEE operator command. Region-level defaults can be established with a CEEROPT (AMODE 31) or CELQROPT (AMODE 64) load module that is created by invoking the CEELOPT macro. For more information about the runtime options, default values, and syntax, see Language Environment runtime options. You might not need to change most default values.

Attention: The usage of Language Environment USERMODs to setup the default configuration will be discontinued in a future release.

Parmlib members are provided for specifying defaults for many system options. The CEEPRMxx parmlib member can be used for specifying system-level default runtime options that control various aspects of Language Environment. The CEEPRMxx parmlib member is identified during IPL by a CEE=xx statement, either in the IEASYSyy parmlib member or in the IPL parameters. After IPL, the operator can do the following tasks:

- ▶ Change the active CEEPRMxx parmlib member with the **SET CEE=xx** command.
- ▶ Change individual runtime options using the **SET CEE** command.
- ▶ Display current runtime option settings with the **D CEE** command.

- ▶ Clear all the system-level default runtime options and keywords using the **SETCEE CLEAR** command.
- ▶ Check existing CEEPRMxx parmlib members for valid syntax.

Note: Using this support is not required, so the default IEASY00 parmlib member does not specify a CEEPRMxx parmlib member. If you want to use this support, a sample CEEPRM00 member is included in CEE.SCEESAMP.

CEEPRMXX member contains several sections for different environments:

CEELOPT	Used to specify runtime options for CICS environments.
CEEDOPT	Used to specify runtime options for non-CICS environments excluding AMODE 64 environments.
CELQDOPT	Used to specify runtime options for AMODE 64 environments.
CEEROPT	Indicates whether to use region-level runtime options in a non-CICS or non-LRR environment.
COMPAT	Attempt a load and use of CEEROPT only in CICS or LRR environments. This is the default behavior.
ALL	Attempt a load and use of CEEROPT in all AMODE 31 and AMODE 24 environments.
CELQR0PT	Indicates whether to use region-level runtime options in an AMODE 64 environment.
NONE	Do not attempt a load or use of CELQR0PT in AMODE 64 environments. This is the default behavior.
ALL	

Runtime options in each section can be defined as overridable or non-overridable. After a runtime option is specified as nonoverrideable with the NONOVR attribute, it cannot later be overridden. This includes later specification in the same parmlib member or a **SETCEE** command. To remove the nonoverrideable setting, use the **SETCEE CLEAR** operator command, or the **SET CEE** command with a parmlib member that does not mark the runtime option as nonoverrideable. Using the OVR option for a single statement you see here:

```
ALL31=((ON),OVR),
ANYHEAP=((16K,8K,ANYWHERE,FREE),OVR),
```

Attention: Marking runtime options as NONOVR might cripple the capabilities of these programs, prevent them from being tuned properly, inhibit their ability to perform First Failure Data Capture, and prevent them from running. With few exceptions, IBM strongly discourages customers from marking any runtime option as NONOVR.

Operating with CEEPRMXX

Use the **SET CEE** command to change the active parmlib member after IPL. The **SET CEE** command parses the CEEPRMxx parmlib member and replaces the runtime options and keywords with the contents of the new member. Figure 6-8 provides an example of activating CEEPRM00 member.

```
-SET CEE=00
CEE3742I THE SET CEE COMMAND HAS COMPLETED.
```

Figure 6-8 Example of the **SET CEE** command

Use the **D CEE** command to display the values that were set in the current CEEPRMxx parmlib members and by the **SET CEE** command. Figure 6-9 shows the output of the **D CEE,ALL** system command. There are five sections of Language Environment settings for each particular supported environment.

```
-D CEE,ALL
CEE3745I 13.10.17 DISPLAY CEEOPT
CEE=(00)
LAST WHERE SET          OPTION
-----
PARMLIB(CEEPRM00)        ABPERC(NONE)
PARMLIB(CEEPRM00)        ABTERMENC(ABEND)
PARMLIB(CEEPRM00)        ALL31(ON)
.
.

CEE3745I 13.10.17 DISPLAY CEECOPT
CEE=(00)
LAST WHERE SET          OPTION
-----
PARMLIB(CEEPRM00)        ABPERC(NONE)
PARMLIB(CEEPRM00)        ABTERMENC(ABEND)
PARMLIB(CEEPRM00)        ALL31(ON)
.
.

CEE3745I 13.10.17 DISPLAY CELQDOPT
CEE=(00)
LAST WHERE SET          OPTION
-----
PARMLIB(CEEPRM00)        DYNDUMP(*USERID,NODYNAMIC,TDUMP)
PARMLIB(CEEPRM00)        ENVAR("")
PARMLIB(CEEPRM00)        FILETAG(NOAUTOCVT,NOAUTOTAG)
.
.

CEE3745I 13.10.17 DISPLAY CEEROPT
CEE=(00)
NO CEEROPT KEYWORD SPECIFIED
CEE3745I 13.10.17 DISPLAY CELQROPT
CEE=(00)
NO CELQROPT KEYWORD SPECIFIED
```

Figure 6-9 D CEE,ALL sample

Individual Language Environment configuration

Language Environment allows you to provide additional invocation-level runtime options using the CEEOPTS DD statement. The CEEOPTS DD statement can refer to an in-stream data set, regular sequential data set, or a member of a regular or extended partitioned data set. If specified, the data set must be available during initialization of the enclave so the options can be merged.

To specify the CEEOPTS DD statement, use the syntax found in Table 6-1 where appropriate.

Table 6-1 CEEOPTS DD statement

Method	Example
In-stream JCL	//CEE0PTS DD ALL31(OFF),STACK(,,BELOW)
Sequential data set	//CEE0PTS DD DSN=MY.CEEOPTS.DATASET,DISP=SHR
Partitioned data set	//CEE0PTS DD DSN=MY.CEEOPTS.DATASET(MYOPTS)
Ignore the DD statement	//CEE0PTS DD DUMMY

Another way of changing individual Language Environment parameter is the **SET CEE** system command. This command allows you to change any Language Environment runtime option as long they is not defined with N00VR option. Figure 6-10 demonstrates this use.

```
-SETCEE CEEDOPT,POSIX(ON)
CEE3743I THE SETCEE COMMAND HAS COMPLETED.
```

Figure 6-10 The SETCEE command example

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 IBM z/OS System Programming Volume 1*, SG24-6981
- ▶ *ABCs of IBM z/OS System Programming Volume 3*, SG24-6983
- ▶ *ABCs of z/OS System Programming: Volume 4*, SG24-6984
- ▶ *ABCs of z/OS System Programming: Volume 5*, SG24-6985
- ▶ *ABCs of IBM z/OS System Programming Volume 6*, SG24-6986
- ▶ *ABCs of z/OS System Programming Volume 7*, SG24-6987
- ▶ *ABCs of z/OS System Programming Volume 8*, SG24-6988
- ▶ *ABCs of z/OS System Programming: Volume 9*, SG24-6989
- ▶ *ABCs of z/OS System Programming Volume 10*, SG24-6990
- ▶ *ABCs of z/OS System Programming Volume 11*, SG24-6327
- ▶ *ABCs of z/OS System Programming Volume 12*, SG24-7621
- ▶ *ABCs of z/OS System Programming Volume 13*, SG24-7717
- ▶ *Introduction to the New Mainframe: z/OS Basics*, SG24-6366
- ▶ *JES3 to JES2 Migration Considerations*, SG24-8083
- ▶ *IBM z/OS V2R2: JES2, JES3, and SDSF*, SG24-8287
- ▶ *TCP/IP Tutorial and Technical Overview*, GG24-3376

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 Planning for Installation*, GA32-0890
- ▶ *z/OS MVS Planning: Operations*, SA23-1390
- ▶ *z/OS MVS Initialization and Tuning Reference*, SA23-1380
- ▶ *z/OS MVS Initialization and Tuning Guide*, SA23-1379
- ▶ *z/OS Migration*, GA32-0889
- ▶ *ServerPac: Using the Installation Dialog*, SA23-2278

- ▶ *IBM Health Checker for z/OS User's Guide*, SC23-6843
- ▶ *z/OS Batch Runtime Planning and User's Guide*, SA23-1376
- ▶ *z/OS MVS Diagnosis: Reference*, GA32-0904
- ▶ *z/OS MVS Diagnosis: Tools and Service Aids*, GA32-0905
- ▶ *z/OS MVS System Commands*, SA38-0666
- ▶ *z/OS MVS System Management Facilities (SMF)*, SA38-0667
- ▶ *z/OS MVS JCL Reference*, SA23-1385
- ▶ *z/OS MVS JCL User's Guide*, SA23-1386
- ▶ *z/OS JES2 Initialization and Tuning Guide*, SA32-0991
- ▶ *z/OS JES2 Installation Exits*, SA32-0995
- ▶ *z/OS MVS Using the Functional Subsystem Interface*, SA38-0678
- ▶ *z/OS MVS Using the Subsystem Interface*, SA38-0679
- ▶ *SMP/E for z/OS User's Guide*, SA23-2277
- ▶ *SMP/E for z/OS Reference*, SA23-2276
- ▶ *SMP/E for z/OS Commands*, SA23-2275
- ▶ *SMP/E for z/OS Messages, Codes, and Diagnosis*, GA32-0883
- ▶ *z/OS JES2 Macros*, SA32-0996
- ▶ *z/OS JES3 Initialization and Tuning Guide*, SA32-1003
- ▶ *z/OS MVS Capacity Provisioning User's Guide*, SC34-2661
- ▶ *EREP Reference*, GC35-0152
- ▶ *EREP User's Guide*, GC35-0151
- ▶ *z/OS MVS Data Areas Volume 1 (ABE - IAR)*, GA32-0935
- ▶ *z/OS MVS Data Areas Volume 2 (IAR - ISG)*, GA32-0936
- ▶ *z/OS MVS Data Areas Volume 3 (ISG - RSR)*, GA32-0937
- ▶ *z/OS MVS Data Areas Volume 4 (RTC - XTL)*, GA32-0938
- ▶ *z/OS MVS Installation Exits*, SA23-1381
- ▶ *z/OS MVS IPCS User's Guide*, SA23-1384

Help from IBM

IBM Support and downloads

ibm.com/support

IBM Global Services

ibm.com/services

Redbooks

ABCs of IBM z/OS System Programming Volume 2

(0.2"spine)
0.17" <-> 0.473"
90<->249 pages



SG24-6982-04

ISBN 0738443018

Printed in U.S.A.

Get connected

