

# ABCs of z/OS System Programming

## Volume 2

z/OS implementation and maintenance

Job management, JES2, JES3, SSI,  
LPA

LNKLST, SMP/E, LE



Paul Rogers  
Miriam Gelinski  
Sergio Munchen  
Monica Mataruco  
Julio Grecco Neto  
Joao Natalino Oliveira  
Antonio Orsi





International Technical Support Organization

**ABCs of z/OS System Programming Volume 2**

January 2006

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

## **Second Edition (January 2006)**

This edition applies to Version 1 Release 6 of z/OS (5694-A01), to Version 1 Release 6 of z/OS.e (5655-G52), and to all subsequent releases and modifications until otherwise indicated in new editions.

© Copyright International Business Machines Corporation 2003, 2006. 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

<b>Notices</b> .....	ix
Trademarks .....	x
 <b>Preface</b> .....	 xi
The team that wrote this redbook. ....	xii
Become a published author .....	xiii
Comments welcome. ....	xiii
 <b>Chapter 1. z/OS implementation and daily maintenance.</b> .....	 1
1.1 Basic aspects of z/OS implementation .....	2
1.2 SYS1.PARMLIB .....	3
1.3 Rules for parmlib member definitions .....	5
1.4 System symbols .....	6
1.5 System symbols: Types .....	8
1.6 Logical parmlib .....	10
1.7 Parmlib concatenation. ....	11
1.8 Using system symbols in parmlib .....	13
1.9 Where to use system symbols in parmlib .....	15
1.10 Member IEASYSxx .....	17
1.11 IEASYSxx and system symbols .....	19
1.12 IEASYMxx. ....	21
1.13 LOADxx parmlib member .....	22
1.14 Placement of LOADxx member. ....	23
1.15 Controlling parmlib .....	25
1.16 Initialization of z/OS .....	27
1.17 Types of IPLs .....	28
1.18 Initial Program Load (IPL) .....	30
1.19 PARMLIB commands .....	33
1.20 Parmlib commands (continued) .....	34
1.21 Use of SETLOAD command .....	35
1.22 Catalogs .....	36
1.23 Separating data from software .....	38
1.24 Placing data sets on specific volumes .....	39
1.25 The TLIBs volumes .....	41
1.26 The DLIB volumes. ....	43
1.27 The image-related volumes. ....	44
1.28 The cluster-related volumes .....	46
1.29 Naming conventions for data sets .....	48
1.30 DASD space utilization and performance .....	49
1.31 System data sets. ....	51
1.32 System administration. ....	52
1.33 User administration .....	53
1.34 DASD administration. ....	54
1.35 Adding a new DASD volume. ....	55
1.36 Implementing DFSMS. ....	57
1.37 Handling DASD problems .....	58
1.38 System data housekeeping. ....	60
1.39 SMF data. ....	61
1.40 SMFPRMxx parmlib member .....	63

1.41	Dumping SMF data . . . . .	65
1.42	LOGREC data . . . . .	66
1.43	SYSLOG data . . . . .	68
1.44	Other administration tasks . . . . .	70
1.45	Working with missing interrupt handler . . . . .	71
1.46	Adding a page data set . . . . .	73
1.47	Changing TSO timeout . . . . .	75
1.48	Adding spool volumes in JES2 . . . . .	77
1.49	Deletion of spool volumes in JES2 . . . . .	79
1.50	Verifying the system configuration . . . . .	82
1.51	HCD primary panel . . . . .	83
1.52	HCD activate or process configuration data . . . . .	84
1.53	Activate or verify configuration . . . . .	85
1.54	Identify system I/O configuration . . . . .	86
1.55	Build CONFIGxx Member panel . . . . .	87
1.56	Creating CONFIGxx member in batch . . . . .	88
1.57	View sysout using ISPF . . . . .	90
1.58	Changing your TSO profile . . . . .	92
1.59	Backup and restore of z/OS . . . . .	94
<b>Chapter 2. Subsystems and subsystem interface (SSI)</b> . . . . .		97
2.1	Defining subsystems and subsystem interface . . . . .	98
2.2	Subsystem initialization . . . . .	100
2.3	Types of subsystem requests . . . . .	102
2.4	IEFSSNxx parmlib member . . . . .	103
2.5	Subsystem definitions . . . . .	105
2.6	Subsystem interface (SSI) . . . . .	108
2.7	SSI control blocks and routines . . . . .	109
2.8	SSI request to master subsystem . . . . .	110
2.9	JES2 supported SSI functions . . . . .	111
2.10	JES3 supported SSI functions . . . . .	112
<b>Chapter 3. Job management</b> . . . . .		113
3.1	z/OS and job management . . . . .	114
3.2	Job management . . . . .	115
3.3	JCL-related actions . . . . .	116
3.4	JES2 and JES3 main differences . . . . .	117
3.5	JES2 primary job entry subsystem . . . . .	118
3.6	JES2 functions . . . . .	119
3.7	JES2 job flow . . . . .	122
3.8	JES2 spool data sets . . . . .	125
3.9	JES2 checkpoint data set . . . . .	127
3.10	JES2 configurations . . . . .	129
3.11	JES2 example of an RJE configuration . . . . .	131
3.12	JES2 example of an NJE configuration . . . . .	133
3.13	JES2 customization . . . . .	135
3.14	JES2 exits . . . . .	140
3.15	JES2 input-related exits . . . . .	142
3.16	JES2 exits in conversion phase . . . . .	146
3.17	JES2 exits in execution phase . . . . .	149
3.18	JES2 start procedure . . . . .	152
3.19	HASPPARM using INCLUDE statement . . . . .	154
3.20	Simplified procedure using the default parmlib member . . . . .	156

3.21	JES2 start parameters	158
3.22	Restarting JES2	162
3.23	Stopping JES2	164
3.24	JES2 operations	166
3.25	Controlling the JES2 environment	168
3.26	Controlling a MAS environment	169
3.27	Controlling JES2 spooling	170
3.28	Controlling JES2 jobs	171
3.29	Controlling JES2 printers	172
3.30	JES3 processing	174
3.31	JES3 configuration	175
3.32	JES3 complex	177
3.33	MVS and JES3 environment	178
3.34	JES3 global	180
3.35	JES3 sysplex components	182
3.36	JES3 multisystem sysplex	183
3.37	JES3 global functions	185
3.38	JES3 terminology	186
3.39	JES3 single LPAR	188
3.40	JES3 multiprocessing	189
3.41	JES3 spooling	191
3.42	JES3 job flow	193
3.43	JES3 job flow (2)	194
3.44	JES3 job flow review	196
3.45	JES3 job flow: Scheduler elements	198
3.46	JES3 standard job	200
3.47	JES3 non-standard job	201
3.48	Creating a batch job	202
3.49	Converter/interpreter processing	203
3.50	Main scheduler element	204
3.51	Main device scheduling (MDS)	205
3.52	Main scheduler element processing	206
3.53	OUTSERV scheduler element processing	207
3.54	Output service processing	208
3.55	Purge processing	209
3.56	JES3 functions not in JES2	210
3.57	Dynamic support program (DSP)	212
3.58	JES3 and consoles	213
3.59	Issuing commands	214
3.60	Consoles on each system	216
3.61	JES3 commands	217
3.62	Controlling JES3 jobs	218
3.63	Controlling job input	219
3.64	Commands for job queue status	221
3.65	Jobs in operator hold	222
3.66	Jobs in hold	223
3.67	Managing output service jobs	224
3.68	Start JES3 with a hot start	225
3.69	JES3 DLOG function	227
3.70	JES3 start procedure	228
3.71	JES3 initialization stream	230
3.72	JES3 startup messages	232
3.73	JES3 startup	233

3.74 JES3 messages following *S JSS . . . . .	234
3.75 JES3 start types . . . . .	235
3.76 TME 10 OPC . . . . .	239
3.77 TME 10 OPC platforms . . . . .	242
3.78 z/OS OPC configuration . . . . .	244
 <b>Chapter 4. LPA, LNKLST, and authorized libraries . . . . .</b>	<b>247</b>
4.1 Link pack area (LPA) . . . . .	248
4.2 LPA subareas . . . . .	250
4.3 Pageable link pack area . . . . .	252
4.4 LPA parmlib definitions . . . . .	253
4.5 Coding a LPALSTxx parmlib member . . . . .	255
4.6 Fixed link pack area . . . . .	256
4.7 Coding the IEAFIXxx member . . . . .	257
4.8 Specifying the IEAFIXxx member . . . . .	259
4.9 Modified link pack area . . . . .	260
4.10 Coding the IEALPAXx member . . . . .	261
4.11 Specifying the IEALPAXx member . . . . .	262
4.12 Dynamic LPA functions . . . . .	263
4.13 The LNKLST . . . . .	264
4.14 Dynamic LNKLST functions . . . . .	266
4.15 Library lookaside (LLA) . . . . .	268
4.16 CSVLLAxx SYS1.PARMLIB member . . . . .	270
4.17 Compressing LLA-managed libraries . . . . .	273
4.18 Virtual lookaside facility (VLF) . . . . .	275
4.19 COFVLFxx parmlib member . . . . .	277
4.20 Authorized libraries . . . . .	280
4.21 Authorizing libraries . . . . .	283
4.22 Dynamic APF functions . . . . .	286
 <b>Chapter 5. System Modification Program/Enhanced (SMP/E) . . . . .</b>	<b>289</b>
5.1 SMP/E overview . . . . .	290
5.2 SYSMODs . . . . .	292
5.3 Concept of SMP/E elements . . . . .	294
5.4 Changing the elements of the system . . . . .	296
5.5 Fixing problems with an element (PTF SYSMOD) . . . . .	298
5.6 Fixing a problem with an element (APAR SYSMOD) . . . . .	300
5.7 Customizing an element (USERMOD SYSMOD) . . . . .	302
5.8 PTF replacement . . . . .	304
5.9 PTF prerequisite . . . . .	305
5.10 Load module construction . . . . .	306
5.11 Data sets used by SMP/E . . . . .	308
5.12 Dynamic allocation of SMP/E data sets . . . . .	312
5.13 How dynamic allocation works . . . . .	314
5.14 Consolidated software inventory (CSI) . . . . .	316
5.15 How to organize CSI data sets . . . . .	318
5.16 Defining zones for your system . . . . .	320
5.17 SMP/E commands you need to know . . . . .	322
5.18 The RECEIVE process . . . . .	325
5.19 Sources of HOLDDATA . . . . .	327
5.20 Reports for RECEIVE processing . . . . .	328
5.21 RECEIVE examples . . . . .	331
5.22 The REJECT process . . . . .	334



5.23 REJECT examples . . . . .	336
5.24 The APPLY process . . . . .	339
5.25 APPLY examples . . . . .	342
5.26 The APPLY CHECK process . . . . .	344
5.27 The RESTORE process . . . . .	346
5.28 Restore examples . . . . .	349
5.29 The ACCEPT process . . . . .	351
5.30 ACCEPT examples . . . . .	354
5.31 Other useful SMP/E commands . . . . .	356
5.32 Using the LIST command . . . . .	358
5.33 Using the REPORT ERRSYSMOD command . . . . .	360
5.34 SMP/E dialogs . . . . .	364
5.35 Customize the SMP/E dialogs . . . . .	365
5.36 Query selection menu . . . . .	367
5.37 CSI query panel . . . . .	368
5.38 CSI query - Select entry . . . . .	369
5.39 CSI query - SYSMOD entry . . . . .	370
5.40 Building SMP/E jobs using the dialog . . . . .	371
5.41 Command generation - Selection menu . . . . .	372
5.42 Command generation - Select zone . . . . .	373
5.43 Command generation - LIST command . . . . .	374
5.44 Command generation - LIST global zone SYSMOD options . . . . .	375
5.45 Command generation - List FORFMID . . . . .	376
5.46 Command generation - Selection menu . . . . .	377
5.47 Command generation - SUBMIT . . . . .	378
5.48 The generated job . . . . .	379
5.49 Java archive (JAR) file support . . . . .	380
5.50 GIMZIP and GIMUNZIP service routines . . . . .	382
5.51 GIMXSID service routine . . . . .	390
5.52 SMP/E release levels . . . . .	392
5.53 SMP/E hints . . . . .	399
 <b>Chapter 6. Language Environment . . . . .</b>	 401
6.1 Language environment (LE) . . . . .	402
6.2 Assembler language and programs . . . . .	403
6.3 High-level language and programs . . . . .	406
6.4 IBM compiler products and LE . . . . .	407
6.5 LE standards . . . . .	408
6.6 LE components . . . . .	409
6.7 LE common run-time environment . . . . .	410
6.8 LE run-time environment for AMODE 64 . . . . .	411
6.9 Object code and LE . . . . .	413
6.10 Callable services . . . . .	414
6.11 LE program management terms and terminology . . . . .	416
6.12 LE program management model . . . . .	418
6.13 LE program management full model . . . . .	420
6.14 LE condition handling model description and terminology . . . . .	421
6.15 LE condition handling steps . . . . .	423
6.16 LE condition handling: Condition token . . . . .	424
6.17 LE condition handling stack frame . . . . .	425
6.18 LE condition handling signaling . . . . .	426
6.19 LE condition handling responses . . . . .	427
6.20 LE storage management model . . . . .	428

6.21	Debug Tool in the user environment . . . . .	430
6.22	Sample assembler routine . . . . .	431
6.23	LE run-time options customization . . . . .	432
6.24	LE user exits . . . . .	434
<b>Related publications . . . . .</b>		<b>435</b>
	IBM Redbooks . . . . .	435
	Other publications . . . . .	435
	How to get IBM Redbooks . . . . .	436

# 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 illustrates 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. 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 IBM's application programming interfaces.

# Trademarks

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

@server®  
@server®  
Redbooks (logo) ™  
z/Architecture™  
z/OS®  
zSeries®  
AFP™  
AIX®  
AS/400®  
CICS®  
DB2®  
DFSMSdfp™  
DFSMSdss™

ESCON®  
Infoprint®  
IBM®  
IMS™  
Language Environment®  
MVS™  
MVS/DFP™  
MVS/ESA™  
NetView®  
OS/2®  
OS/390®  
OS/400®  
Parallel Sysplex®

Print Services Facility™  
Redbooks™  
RACF®  
RMF™  
Sysplex Timer®  
System/370™  
System/390®  
Tivoli®  
TME 10™  
TME®  
VTAM®

The following terms are trademarks of other companies:

IQ, Java, JDK, Solaris, Sun, SunOS, and all Java-based trademarks are trademarks of Sun Microsystems, Inc. in the United States, other countries, or both.

Windows NT, Windows, and the Windows logo are trademarks of Microsoft Corporation in the United States, other countries, or both.

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 z/OS® System Programming is an eleven 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 would like 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 will 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 used to manage jobs and data. The topics covered are:

- ▶ 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, how to use the subsystem services provided by IBM® subsystems, and planning considerations for setting up and writing your own subsystem.
- ▶ Job management in the z/OS system using the job entry subsystems JES2 and JES3. Detailed discussion of 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), LNKLIST, authorized libraries, and the role of VLF and LLA components.
- ▶ SMP/E
- ▶ Language Environment®

The contents of the volumes are:

Volume 1: Introduction to z/OS and storage concepts, TSO/E, ISPF, JCL, SDSF, and z/OS delivery and installation

Volume 2: z/OS implementation and daily maintenance, defining subsystems, JES2 and JES3, LPA, LNKLIST, authorized libraries, SMP/E, and Language Environment

Volume 3: Introduction to DFSMS, data set basics, storage management hardware and software, VSAM, System-managed storage, catalogs, and DFSMSHVS

Volume 4: Communication Server, TCP/IP and VTAM®

Volume 5: Base and Parallel Sysplex®, System Logger, Resource Recovery Services (RRS), global resource serialization (GRS), z/OS system operations, automatic restart management (ARM), Geographically Dispersed Parallel Sysplex (GDPS), availability in the zSeries® environment

Volume 6: Introduction to security, RACF®, Digital certificates and PKI, Kerberos, cryptography and z990 integrated cryptography, zSeries firewall technologies, LDAP, Enterprise identity mapping (EIM), and firewall technologies

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 z/Architecture™, zSeries processor design, zSeries connectivity, LPAR concepts, HCD, and HMC

Volume 11: Capacity planning, performance management, WLM, RM, and SMF

## The team that wrote this redbook

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

**Paul Rogers** is a Consulting IT Specialist at the International Technical Support Organization, Poughkeepsie Center. He writes extensively and teaches IBM classes worldwide on various aspects of z/OS and OS/390®. Before joining the ITSO 18 years ago, Paul worked in the IBM Installation Support Center (ISC) in Greenford, England providing OS/390 and JES support for IBM EMEA and the Washington Systems Center. He has worked for IBM for 38 years.

**Miriam Gelinski** is a staff member of Maffei Consulting Group, where she is responsible for supporting customer planning and installing zSeries software. Miriam holds a bachelor degree in Information Systems from Universidade São Marcos. Before joining Maffei Consulting Group, she worked for IBM zSeries brand for two years, where she was responsible for implementing new workloads on zSeries.

**Sergio Munchen** is a system analyst at Banco do Brasil. He has four years of experience in the mainframe environment. His areas of expertise include assembler programming, customization of coupling facility exploiters, z/OS and HCD. Sergio holds a specialist degree in Computer Science from Universidade Federal de Santa Catarina.

**Joao Natalino Oliveira** is a certified I/T consulting specialist for IBM zSeries in Brazil, where he provides support for Brazil and Latin America. He has 28 years of experience in large systems, including MVS™, OS/390, and z/OS. His areas of expertise include performance and capacity planning, server consolidation, and system programming. He holds a bachelor degree in Mathematics and Data Processing from Fund. Santo Andre, Brazil.

**Monica Mataruco** is a system programmer at ABN AMRO Real Bank in Brazil. She has 18 years of experience in large systems, including MVS, OS/390, and z/OS. Her areas of expertise include installation and customization of the operating system and other z/OS components.

**Julio Grecco Neto** is a system programmer at ABN AMRO Real Bank in Brazil. He has 28 years of experience in large systems, including MVS, OS/390, and z/OS and DASD. His areas of expertise include installation and configuration of hardware and storage.

**Antonio Orsi** is a system programming manager at ABN AMRO Real Bank in Brazil. He has 31 years of experience in large systems, including MVS, OS/390, and z/OS. His areas of expertise include hardware and software configuration.

## Become a published author

Join us for a two- to six-week residency program! Help write an IBM Redbook dealing with specific products or solutions, while getting hands-on experience with leading-edge technologies. You'll team with IBM technical professionals, Business Partners and/or customers.

Your efforts will help increase product acceptance and customer satisfaction. As a bonus, you'll develop a network of contacts in IBM development labs, and increase your productivity and marketability.

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

[ibm.com/redbooks/residencies.html](http://ibm.com/redbooks/residencies.html)

## Comments welcome

Your comments are important to us!

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

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

[ibm.com/redbooks](http://ibm.com/redbooks)

- Send your comments in an email to:

[redbook@us.ibm.com](mailto:redbook@us.ibm.com)

- Mail your comments to:

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







# **z/OS implementation and daily maintenance**

Regardless of which method you use to install the z/OS operating system (ServerPac or CBPDO), it is highly recommended that you understand the basic aspects of the z/OS implementation, from installation to preparing for daily activities after IPL.

This chapter gives you an overview of the basics as well as the IPL process. It describes the main parameters and system data sets necessary to IPL and run a z/OS operating system, and the main daily tasks that a system programmer performs to maximize the advantages that a well-implemented operating system can offer to your IT structure.

## 1.1 Basic aspects of z/OS implementation

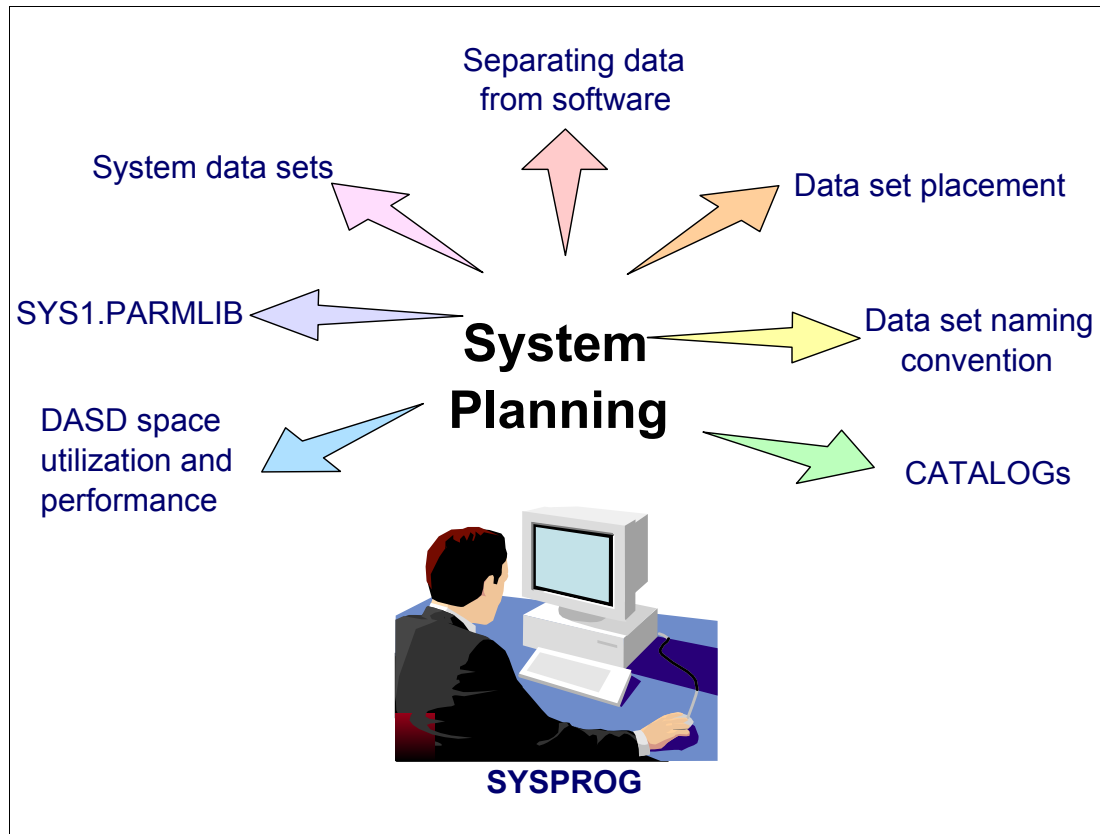


Figure 1-1 Basic aspects of z/OS implementation

### Basic aspects of z/OS implementation

When you build a z/OS system, you must balance the needs of your installation to build one that meets its needs well. While this will sometimes mean compromise, it more often 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. When applied well, using a well-planned structure, this flexibility can be used to control the time it takes to install and migrate new systems or new software levels throughout an installation.

A phased approach will often prove most feasible and can begin to control the installation and migration workload in the least time. This provides you benefits, starting with the next installation and migration cycle, while controlling the work involved in implementation.

Some aspects you might have to consider in adopting a structured approach to installation are:

- ▶ **SYS1.PARMLIB** and parmlib concatenation (logical parmlib)
- ▶ Catalogs and indirect catalog entries
- ▶ Separating data from software
- ▶ Placing data sets on specific volumes
- ▶ Choosing a naming convention for data sets
- ▶ DASD space utilization and performance
- ▶ System and installation requirements

## 1.2 SYS1.PARMLIB

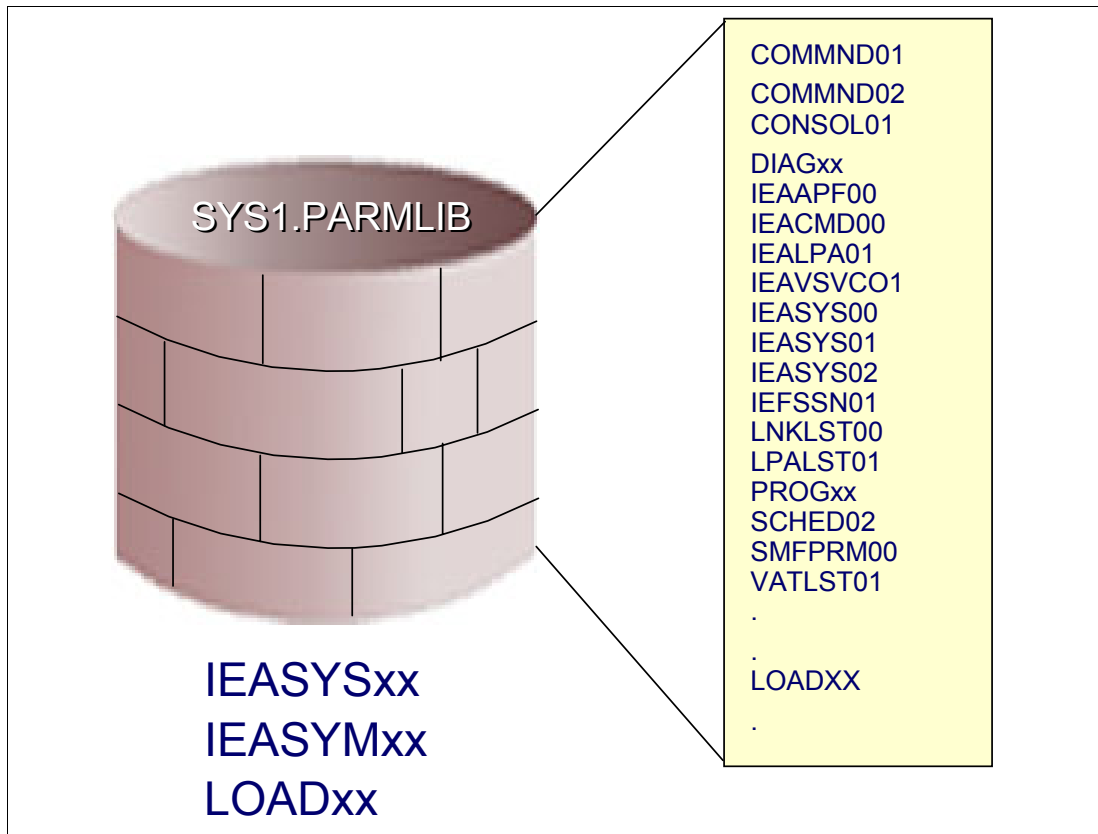


Figure 1-2 Defining the members of SYS1.PARMLIB

### Overview of parmlib members

SYS1.PARMLIB is a required partitioned data set that contains IBM-supplied and installation-created members, which contain lists of system parameter values. You can include user-written system parameter members in SYS1.PARMLIB before installing a product.

SYS1.PARMLIB is read by the system at IPL, and later by such components as the system resource manager, the TIOC, and GTF, which are invoked by operator commands. The purpose of parmlib is to provide many initialization parameters in a pre-specified form in a single data set, and thus minimize the need for the operator to enter parameters. The SYS1.PARMLIB data set can be blocked and can have multiple extents, but it must reside on a single volume.

**Note:** This is the most important data set in a z/OS operating system.

### Defining parmlib members

All parameters of the SYS1.PARMLIB data set are described in *z/OS MVS Initialization and Tuning Reference*, SA22-7592. Three of the most important parmlib members are discussed in this chapter. They are as follows:

**IEASYSxx** This parmlib member allows the specification of system parameters that are valid responses to the IEA101A SPECIFY SYSTEM PARAMETERS message, as shown in Figure 1-17 on page 28. Multiple system parameter lists are valid.

The list is chosen by the operator SYSP parameter or through the SYSPARM statement of the LOADxx parmlib member.

**IEASYMxx** Specifies, for one or more systems in a multisystem environment, the static system symbols and suffixes of IEASYSxx members that the system is to use. One or more IEASYMxx members are selected using the IEASYM parameter in the LOADxx parmlib member.

**LOADxx** Contains information about the name of the IODF data set, which master catalog to use, and which IEASYSxx members of SYS1.PARMLIB to use.

## 1.3 Rules for parmlib member definitions

- ❑ Use columns 1 through 71 to specify parameters, 72-80 ignored
  - ❑ Do not use blank lines
  - ❑ Leading blanks in records are acceptable, start in any column
  - ❑ Enter data in uppercase characters only
  - ❑ Use commas to separate multiple parameters in a record
  - ❑ Enclose multiple subparameters in parentheses
  - ❑ Indicate record continuation with a comma followed by a blank
  - ❑ The system ignores anything after a comma followed by a blank
  - ❑ End of a member is the first record that does not end in a comma
- ```
CON=01,MLPA=(00,01,02,03,L) , USE CONSOL01, USE IEALPA(00-03)
      COUPLE=&SYSCONE,           XCF SERIAL CTCS ARE DEFINED
      PLEXCFG=MULTISYSTEM,       TURN SYSPLEX ON
      DIAG=01                    USE DIAG01-LAST STMT; LIST ENDS HERE
/* CLOCK=SA          */          (this line is commented out)
```

Figure 1-3 Specifying the rules when defining parmlib members

### Syntax rules for parmlib members

The following rules apply to the creation of parmlib members:

- ▶ Use columns 1 through 71 to specify parameters. The system ignores columns 72 through 80.
- ▶ Do not use blank lines.
- ▶ Leading blanks in records are acceptable. Therefore, a parameter need not start at column 1.
- ▶ Enter data in *uppercase* characters only; the system does not recognize lowercase characters.
- ▶ Use commas to separate multiple parameters in a record, but do not leave blanks between commas and subsequent parameters.
- ▶ Enclose multiple subparameters in parentheses. The number of subparameters is not limited.
- ▶ Indicate record continuation with a comma followed by at least one blank.
- ▶ The system ignores anything after a comma followed by one or more blanks. You can use the remainder of the line for comments.
- ▶ The system considers the first record that does not end in a comma to be the end of the member and ignores subsequent lines. You can use the remainder of the record, which contains the last parameter, for comments, providing there is at least one blank between the last parameter and the comments. You can also use additional lines after the last parameter for comments.

Figure 1-3 shows an example of the syntax used in creating a parmlib member.

## 1.4 System symbols

- ❑ Dynamic system symbols
  - Substitution text changes often
- ❑ Static system symbols
  - Substitution text is fixed at system initialization
  - Two types of static symbols
    - System-defined
      - &SYSCclone - &SYSNAME - &SYSPLEX - &SYSR1
    - Installation-defined

Figure 1-4 Defining system symbols

### Defining system symbols

System symbols are elements that allow systems to share parmlib definitions while retaining unique values in those definitions. System symbols act like variables in a program; they can take on different values, based on the input to the program. When you specify a system symbol in a shared parmlib definition, the system symbol acts as a “place holder.” Each system that shares the definition replaces the system symbol with a unique value during initialization.

The following terms describe the elements of system symbols:

|                          |                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                  |
|--------------------------|--------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| <b>Symbol Name</b>       | The name that is assigned to a symbol. It begins with an ampersand (&) and optionally ends with a period (.).                                                                                                                                                                                                                                                                                                                                                                                                                                    |
| <b>Substitution Text</b> | 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. When you define static system symbols in the IEASYMxx parmlib member (see Figure 1-8 on page 13), the substitution text can contain other static system symbols; the resolved substitution text refers to the character string that is produced after all symbols in the substitution text are resolved. |

## Types of system symbols

The following terms describe the types of system symbols:

**Dynamic** 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 to the system; your installation cannot provide additional dynamic system symbols.

**Static** A symbol whose substitution text is defined at system initialization and remains fixed for the life of an IPL. 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.

There are two types of static system symbols:

- ▶ **System-defined:** System-defined static system symbols already have their names defined to the system. Your installation defines substitution texts or accepts system default texts for the static system symbols, which are:
  - &SYSCONE
  - &SYSNAME
  - &SYSPLEX
  - &SYSR1
- ▶ **Installation-defined:** Installation-defined static system symbols are defined by your installation. The system programmer specifies their names and substitution texts in the SYS1.PARMLIB data set.

## 1.5 System symbols: Types

| Static System Symbols                                                                                                                                                                                  | Dynamic System Symbols                                                                                                                                                                                                            | Symbols Reserved For System Use                                                                                                                                                                    |
|--------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|-----------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|----------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| <div><div>&amp;SYSALVL<br/>&amp;SYSCLONE<br/>&amp;SYSNAME<br/>&amp;SYSPLEX<br/>&amp;SYSR1</div><div>Installation<br/>Defined System<br/>Symbols</div><div>JCL Symbol</div><div>IPCS Symbol</div></div> | <div><div>&amp;DATE<br/>&amp;DAY<br/>&amp;HHMMSS<br/>&amp;HR<br/>&amp;JDAY<br/>&amp;JOBNAME<br/>&amp;MIN<br/>&amp;MON<br/>&amp;SEC<br/>&amp;SEQ<br/>&amp;TIME<br/>&amp;WDAY<br/>&amp;YR2<br/>&amp;YR4<br/>&amp;YYMMDD</div></div> | <div><div>&amp;DATE<br/>&amp;HR<br/>&amp;LYMMDD<br/>&amp;LHR<br/>&amp;LMON<br/>&amp;LWDAY<br/>&amp;LHHMMSS<br/>&amp;SEC<br/>&amp;SYSCLONE<br/>&amp;SYSR1<br/>&amp;WDAY<br/>&amp;YYMMDD</div></div> |

Figure 1-5 Types of system symbols that can be specified

### Static system symbols

The system substitutes text for static system symbols when it processes parmlib members. For static system symbols that the system provides, you can define substitution texts in parmlib or accept the default substitution texts. You can also define up to 99 additional static system symbols. You can enter the **DISPLAY SYMBOLS** operator command to display the static texts that are in effect for a system. See *z/OS MVS System Commands*, SA22-7627, for information about how to enter **DISPLAY SYMBOLS**. You can use the **SYMDEF** interactive problem control system (IPCS) subcommand to display an entry in the system symbol table. See *z/OS MVS Interactive Problem Control System (IPCS) Commands*, SA22-7594, for information about the **SYMDEF** subcommand.

In addition to the system symbols listed in Figure 1-5, the system allows you to define and use symbols in:

**JCL symbol** Symbols can be used in JCL. You can define JCL symbols on EXEC, PROC, and SET statements in JCL, and use them only in:

- JCL statements in the job stream
- Statements in cataloged or in-stream procedures
- DD statements that are added to a procedure

For more information about using JCL symbols, see *z/OS MVS JCL Reference*, SA22-7597.



**IPCS symbol** Symbols can be use by IPCS to represent data areas in dumps that are processed with IPCS subcommands.

For more information about using IPCS symbols, see *z/OS MVS Interactive Problem Control System (IPCS) User's Guide*, SA22-7596.

**Note:** Although IBM recommends the use of ending periods on system symbols, the text of this chapter does not specify them, except in examples, out of consideration for readability.

## Dynamic system symbols

You can specify dynamic system symbols in parmlib. However, be aware that the system substitutes text for dynamic system symbols when it processes parmlib members. For example, if you specify &HHMMSS in a parmlib member, its substitution text reflects the time when the member is processed.

This situation can also occur in other processing. For example, if you specify the &JOBNAME dynamic system symbol in a **START** command for a started task, the resolved substitution text for &JOBNAME is the name of the job assigned to the address space that calls the symbolic substitution service, not the address space of the started task.

## Symbols reserved for system use

When you define additional system symbols in the IEASYMxx parmlib member, ensure that you do not specify the names 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.

In addition to the symbols reserved for system use that are shown in Figure 1-5, the following symbols are also reserved for system use:

&JDAY, &JOBNAME, &LDAY, &LHHMMSS, &LJDAY, &LMIN, &LSEC, &LTIME, &LYR2, &LYR4, &MIN, &MON, &SEQ, &SID, &SYSNAME, &SYSPLEX, &SYSUID, &TIME, &YR2, &YR4, &SYSALVL

## 1.6 Logical parmlib

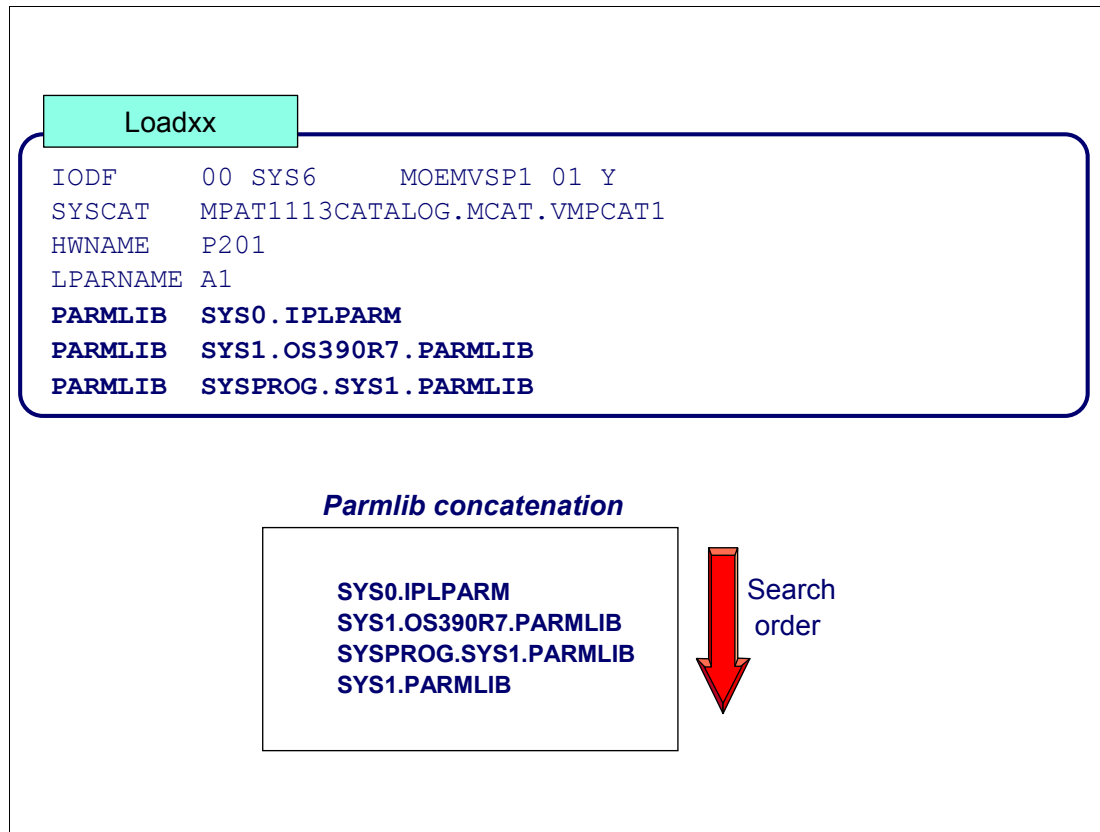


Figure 1-6 Creating a logical parmlib

### Using parmlib concatenation

Beginning with OS/390 V2R2, you can concatenate up to 10 data sets to SYS1.PARMLIB, in effect creating a *logical parmlib*. You define the concatenation in the LOADxx member of SYS1.PARMLIB or SYSn.IPLPARM. When there is more than one PARMLIB statement, the statements are concatenated and SYS1.PARMLIB, as cataloged in the Master Catalog, is added at the end of the concatenation. You can also use the **SETLOAD** operator command to switch from one logical parmlib to another without an IPL.

The benefit of using parmlib concatenation is that it gives you greater flexibility in managing parmlib and changes to parmlib members.

**Note:** If you do not specify at least one PARMLIB statement in the LOADxx, the parmlib concatenation will consist of only SYS1.PARMLIB and Master Scheduler processing will use the IEFPARM DD statement, if there is one in the Master JCL. However, if there is no PARMLIB statement in the parmlib concatenation and there is no IEFPARM DD statement, Master Scheduler processing will use only SYS1.PARMLIB.

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

## 1.7 Parmlib concatenation

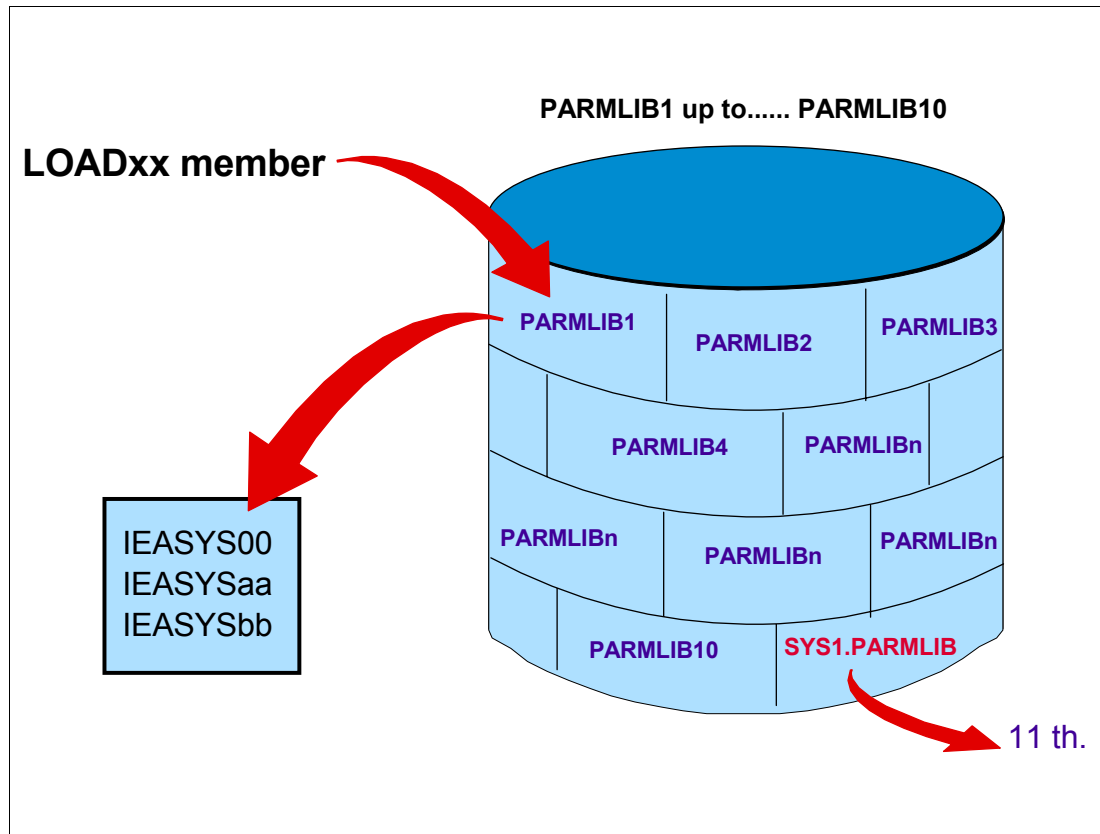


Figure 1-7 LOADxx member pointing to the parmli concatenation

### Description and use of the parmli concatenation

This section discusses the parmli concatenation, its purpose, ways to control the parmli data sets, and general syntax rules for creating most members of the data sets.

The parmli concatenation is a set of up to 10 partitioned data sets defined through PARMLIB statements in the LOADxx member of either SYSn.IPLPARM or SYS1.PARMLIB. These members contain many initialization parameters in a pre-specified form in a single logical data set, thus minimizing the need for the operator to enter parameters. SYS1.PARMLIB makes the eleventh or last data set in the concatenation and is the default parmli concatenation if no PARMLIB statements exist in LOADxx. For specific information on how to define a logical parmli concatenation, see “LOADxx (system configuration data sets)” on page 22. The SYS1.PARMLIB data set itself can be blocked and can have multiple extents, but it must reside on a single volume. The parmli concatenation used at IPL must be a PDS. However, after an IPL you can issue a **SETLOAD** command to switch to a different parmli concatenation that contains PDSEs. For information on processing of concatenated data sets see *z/OS DFSMS: Using Data Sets*, SC26-7410.

### IEASYSxx parmli members

Parmli contains both a basic or default general parameter list, IEASYS00, and possible alternate general parameter lists, called IEASYSaa, IEASYSbb, and so forth. Parmli also contains specialized members, such as COMMNDxx, and IEALPaxx. Any general parameter list can contain both parameter values and “directors.” The directors (such as MLPA=01) point or direct the system to one or more specialized members, such as IEALPA01.

The parmlib concatenation is read by the system at IPL, and later by other components such as the system resource manager (SRM), the TIOC, and SMF, which are invoked by operator commands. The TIOC is the terminal I/O coordinator, whose parameters are described under member IKJPRM00. SMF is the System Management Facility, whose parameters are described under member SMFPRMxx.

The system always reads member IEASYS00, the default parameter list. Your installation can override or augment the contents of IEASYS00 with one or more alternate general parameter lists. You can further supplement or partially override IEASYS00 with parameters in other IEASYSxx members or operator-entered parameters. You can specify the IEASYSxx members that the system is to use in:

- ▶ The IEASYMxx parmlib member
- ▶ The LOADxx parmlib member

The operator selects the IEASYSxx member using the SYSP parameter at IPL. The parameter values in IEASYS00 remain in effect for the life of an IPL unless they are overridden by parameters specified in alternate IEASYSxx members or by the operator.

## 1.8 Using system symbols in parmlib

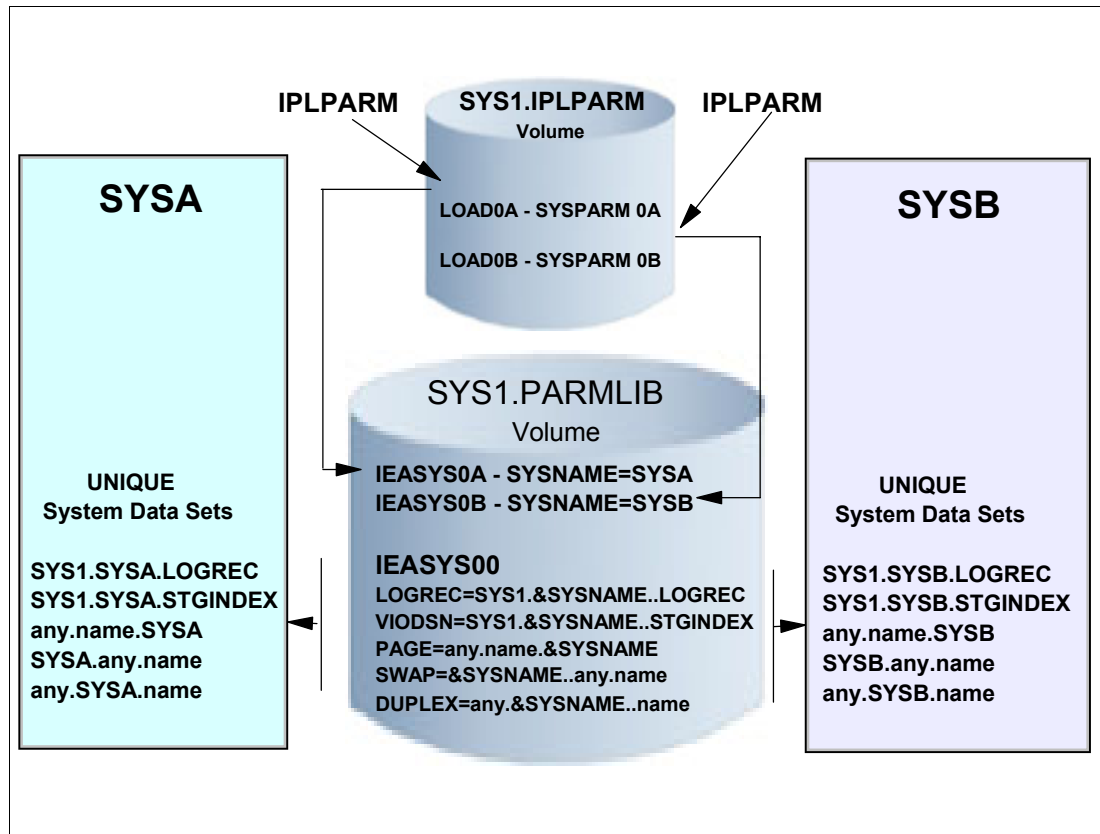


Figure 1-8 Defining parmlib members using system symbols

### Using system symbols in parmlib

After you set up parmlib for sharing, do the following to specify system symbols in parmlib definitions:

1. Know the rules for using system symbols in parmlib.
2. Determine where to use system symbols in parmlib.
3. Verify system symbols in parmlib.

### Know the rules for using system symbols in parmlib

Follow these rules and recommendations when using system symbols in parmlib:

1. Specify system symbols that:
  - Begin with an ampersand (&)
  - Optionally end with a period (.)
  - Contain 1 to 8 characters between the ampersand and the period (or the next character, if you do not specify a period)

If the system finds a system symbol that does not end with a period, it substitutes text for the system symbol when the next character is one of the following:

- Null (the end of the text is reached)
- A character that is not alphabetic, numeric, or special (@, #, or \$)

**Recommendation:** End all system symbols with a period. Omitting the period that ends a system symbol could produce unwanted results under certain circumstances. For example, if the character string (2) follows a system symbol that does not have an ending period, the system processes the (2) as substring syntax for the system symbol, regardless of how you intended to use the string in the command.

2. Use a small set of system symbols so they are easy to manage and remember.
3. Code two consecutive periods (..) if a period follows a system symbol. For example, code &DEPT..POK when the desired value is D58.POK and the substitution text D58 is defined to the system symbol &DEPT.
4. When using system symbols in data set name qualifiers, keep the rules for data set naming in mind. For example, if you use SC68 as a data set qualifier, ensure that the substitution text begins with an alphabetic character.
5. Ensure that resolved substitution texts do not extend parameter values beyond their maximum lengths. For example, suppose the following command is to start CICS®:

S CICS,JOBNAME=CICSSC68,...

The resolved substitution text for SC68 cannot exceed four characters because jobnames are limited to eight characters (the four characters in CICS plus up to four character in SC68). A substitution text of SYS1 is valid because it resolves to the jobname CICSSYS1. However, a substitution text of SYSTEM2 is not valid because it resolves to the jobname of CICSSYSTEM2, which exceeds the allowable maximum of eight characters.

6. If you use &SYSCclone, ensure that the LOADxx parmlib member indicates that the system is to check for a unique &SYSCclone substitution text on each system.
7. If you use SC68, ensure that its substitution text is unique on each system.
8. Do not specify system symbols in the values on the OPI and SYSP parameters in the IEASYSxx parmlib member.
9. Do not specify system symbols that were introduced in MVS/ESA™ SP 5.2 in parmlib members that are processed by pre-MVS/ESA SP 5.2 systems.
10. Do not specify any system symbols in parmlib members that do not support system symbol substitution.

## 1.9 Where to use system symbols in parmlib

- ❑ Symbols in data set names
  - SMFPRMxx parmlib member
    - SY&SYSCclone..SMF.DATA
- ❑ Symbols when systems share the same parmlib member
  - IEASYSxx parmlib member
    - CLOCK=&SYSCclone.
- ❑ Started task commands
  - S CICS,JOBNAME=CICS&SYSCclone.
    - CICSSYS1 on SYS1
    - CICSSYS2 on SYS2

Figure 1-9 Examples of using system symbols in parmlib

### System symbols in parmlib

System symbols offer the greatest advantage when two or more systems require different data sets, jobs, procedures, or entire parmlib members. This section provides examples of how to specify system symbols when naming certain resources in parmlib.

#### Data sets

A good example of using system symbols in data set names is the DSNNAME parameter in the SMFPRMxx parmlib member, which specifies data sets to be used for SMF recording. 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 could specify the following naming pattern to create different SMF recording data sets on each system:

```
SY&SYSCclone..SMF.DATA
```

When you IPL each system in the sysplex, the &SYSCclone system symbol resolves to the substitution text that is defined on the current system. For example, if a sysplex consists of two systems named SYS1 and SYS2, accepting the default value for &SYSCclone produces the following data sets:

```
SYS1.SMF.DATA on system SYS1  
SYS2.SMF.DATA on system SYS2
```

**Note:** The use of &SYSCclone provides unique data set names while establishing a consistent naming convention for SMF recording data sets.

## Parmlib members

You can apply the same logic to system images that require different parmlib members. For example, assume that system images SYS1 and SYS2 require different CLOCKxx parmlib members. If both systems share the same IEASYSxx parmlib member, you could specify &SYSCLOCK in the value on the CLOCK parameter:

```
CLOCK=&SYSCLOCK;
```

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

```
CLOCK=SYS1      (Specifies CLOCKS1 on system SYS1)
CLOCK=SYS2      (Specifies CLOCKS2 on system SYS2)
```

## Started task JCL

If JCL is for a started task, you can specify system symbols in the source JCL or in the **START** command for the task. You cannot specify system symbols in JCL for batch jobs, so you might want to change those jobs to run as started tasks.

If a started task is to have multiple instances, determine if you want the started task to have a different name for each instance. Started tasks that can be restarted at later times are good candidates. The different names allow you to easily identify and restart only those instances that require a restart. For example, you might assign different names to instances of CICS because those instances might be restarted at later points in time. However, instances of VTAM, which are generally not restarted, might have the same name on different systems.

When you start a task in the COMMNDxx parmlib member, you can specify system symbols as part of the job name. Assume that system images SYS1 and SYS2 both need to start customer information control system (CICS). If both system images share the same COMMNDxx parmlib member, you could specify the SC68 system symbol on a **START** command in COMMNDxx to start unique instances of CICS:

```
S CICS,JOBNAME=CICSSC68,...
```

When each system in the sysplex initializes with the same COMMNDxx member, SC68 resolves to the substitution text that is defined on each system. If SC68 is defined to SYS1 and SYS2 on the respective systems, the systems start CICS with the following jobnames:

```
CICSSYS1 on system SYS1
CICSSYS2 on system SYS2
```

## Verify system symbols in parmlib

IBM provides you with tools to verify symbol usage in parmlib. The parmlib symbolic preprocessor tool allows you to test symbol definitions before you IPL the system to use them. This tool shows how a parmlib member will appear after the system performs symbolic substitution.

If you only need to verify a new parmlib member's use of the current system symbols, you can run the IEASYMCK sample program to see how the contents of the parmlib member will appear after symbolic substitution occurs.

The IEASYMCK program is located in SYS1.SAMPLIB. See the program prolog for details.



## 1.10 Member IEASYSxx

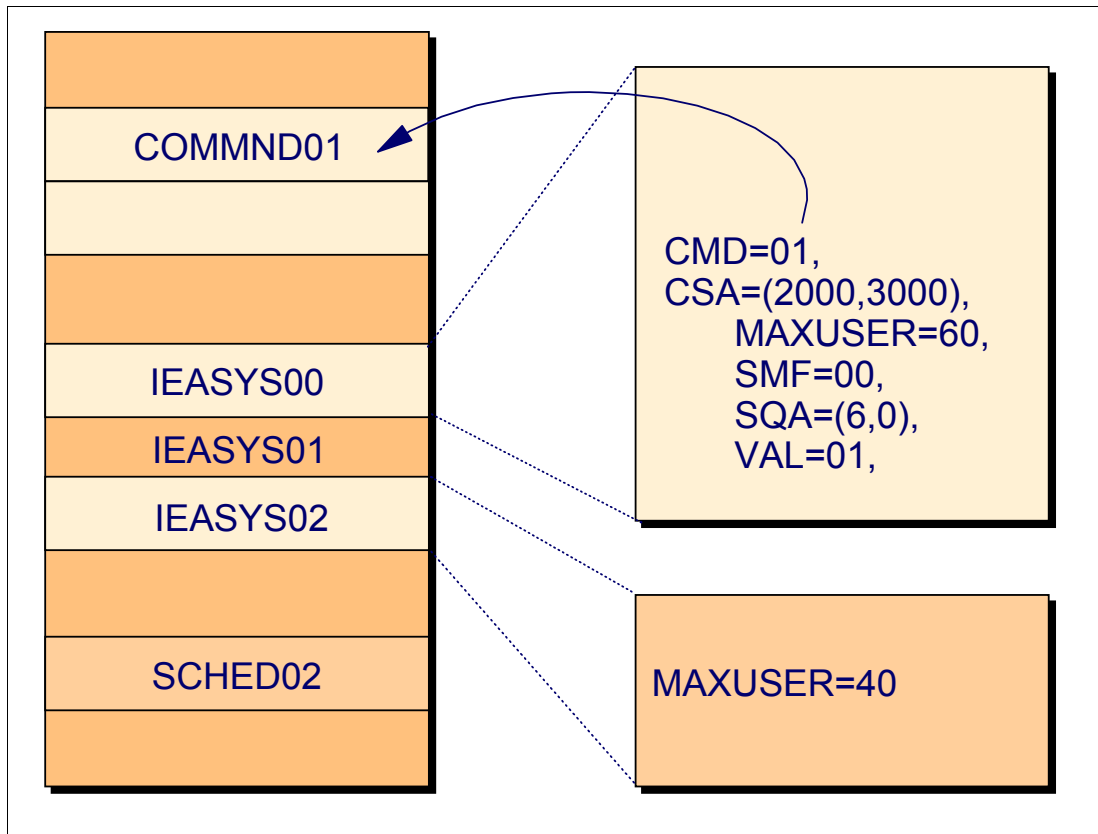


Figure 1-10 Parmlib member IEASYSxx

### IEASYSxx (system parameter list)

You can specify system parameters using a combination of IEASYSxx parmlib members and operator responses to the SPECIFY SYSTEM PARAMETERS message. You can place system parameters in the IEASYS00 member or in one or more alternate system parameter lists (IEASYSxx) to provide a fast initialization that requires little or no operator intervention.

IEASYS00 is the most likely place to put installation defaults or parameters that will not change from IPL to IPL. The system programmer can add to or modify parameters in IEASYS00. The alternate IEASYSxx members, in contrast, should contain parameters that are subject to change, possibly from one work shift to another.

Use of the IEASYS00 or IEASYSxx members can minimize operator intervention at IPL. Because IEASYS00 is read automatically, the operator can respond to SPECIFY SYSTEM PARAMETERS with ENTER or U and need not enter parameters unless an error occurs and prompting ensues.

### Using parameter lists

The use of system parameter lists in parmlib offers two main advantages:

- ▶ The parameter lists shorten and simplify the IPL process by allowing the installation to preselect system parameters.
- ▶ The parameter lists provide flexibility in the choice of system parameters.

You can do one of the following to specify a parameter list other than IEASYS00 for an IPL:

- Have the operator specify the suffix of an alternate IEASYSxx member by replying SYSP=xx in response to the SPECIFY SYSTEM PARAMETERS message.

R 00,SYSP=xx

The operator specifies this parameter to specify an alternate system parameter list in addition to IEASYS00.

- Specify one or more suffixes of alternate IEASYSxx members on the SYSPARM parameter in the LOADxx or in the IEASYMxx parmlib member.

## Overview of IEASYSxx parameters

See *z/OS MVS Initialization and Tuning Reference*, SA22-7592 for a list of all the system parameters that can be placed in an IEASYSxx or IEASYS00 member (or specified by the operator). Detailed discussions of these parameters are provided in other sections of the IEASYSxx topic.

**Note:** PAGE and GRS are the only mandatory parameters that have no default. They must be specified.

The GRSRNL parameter is mandatory when the GRS= parameter is specified as JOIN, TRYJOIN, START, or STAR. The GRSRNL parameter is ignored when GRS=NONE.

## Specifying the list option for IEASYSxx parameters

Certain parameters in IEASYSxx (such as CLOCK, CON, and MLPA) allow you to specify the list option (L). If you specify the L option, and message suppression is not active, the system writes all of the statements read from the associated parmlib member to the operator's console during system initialization. If message suppression is active (the default), the system writes the list to the console log only.

To ensure that messages are not suppressed, specify an appropriate initialization message suppression indicator (IMSI character) on the LOAD parameter. The IMSI characters that do not suppress messages are A, C, D, M, and T.

For more information on the LOAD parameter, see the section on loading the system software in *z/OS MVS System Commands*, SA22-7627.

## Statements/parameters for IEASYSxx

For detailed information about statements/parameters, see *z/OS MVS Initialization and Tuning Reference*, SA22-7592.

## 1.11 IEASYSxx and system symbols

### ❑ Understand the rules

IEASYMxx

`SYMDEF (&PAGTOTL=' (10,5) ')`

IEASYSxx

`PAGTOTL=&PAGTOTL.,`

### ❑ Can be specified in IEASYSxx

`&SYSNAME = SYSA`

`&SYSCLOCK = SA`

`&SYSPLEX = PX01`

`LNK= (&SYSPLEX (-2:2) ., 03, 00, L) ,  
CLOCK=&SYSCLOCK.,  
PROG=&SYSNAME (1:2) ,  
LOGREC=&SYSNAME..LOGREC`

`LNK= (01, 03, 00, L) ,  
CLOCK=SA,  
PROG=SY,  
LOGREC=SYSA.LOGREC`

Figure 1-11 Specifying system symbols in IEASYSxx

## IEASYSxx and system symbols

You can specify system symbols in all parameter values in IEASYSxx except in the values for the SYSP and OPI parameters and in specifying CLPA and OPI.

If you intend to use system symbols to represent parmlib member suffixes in IEASYSxx, be careful when defining, in the IEASYMxx parmlib member, parentheses (such as in the case of list notation) or commas as part of the substitution text:

- Specify system symbols only to the right of the equals sign and before the comma in the IEASYSxx notation.
- Specify only *balanced* parentheses in either the defined substitution text or the hard-coded values.

For example, the following notation for IEASYMxx and IEASYSxx is valid because the left and right parentheses both appear in the system symbol definition:

| IEASYMxx                                      | IEASYSxx                            |
|-----------------------------------------------|-------------------------------------|
| <code>SYMDEF (&amp;PAGTOTL=' (10,5) ')</code> | <code>PAGTOTL=&amp;PAGTOTL.,</code> |

The following notation is not valid because the parentheses are split between the system symbol definition and the hard-coded definition in IEASYSxx:

| IEASYMxx                                     | IEASYSxx                              |
|----------------------------------------------|---------------------------------------|
| <code>SYMDEF (&amp;PAGTOTL=' 10,5) ')</code> | <code>PAGTOTL= (&amp;PAGTOTL.,</code> |

## Example of using system symbols in IEASYSxx

Suppose the following system symbols have the values indicated:

```
SC68 = SYSA  
&SYSCLOCK = SA  
&SYSPLEX = PX01
```

Then assume that you want to do the following in IEASYSxx:

1. Specify the LNKSTxx member identified by the last two letters in the sysplex name and also LNKSTxx members 03 and 00.
2. Specify the CLOCKxx member identified by &SYSCLOCK.
3. Specify the PROGxx member identified by the first two letters in the system name.
4. Specify a data set name for error recording that has the system name as a high-level qualifier.

Code IEASYSxx as follows:

```
LNK=(&SYSPLEX(-2:2) .,03,00,L),  
CLOCK=&SYSCLOCK .,  
PROG=SC68(1:2),  
LOGREC=SC68.LOGREC
```

The values of the parameters resolve to:

```
LNK=(01,03,00,L),  
CLOCK=SA,  
PROG=SY,  
LOGREC=SYSA.LOGREC
```

## 1.12 IEASYMxx

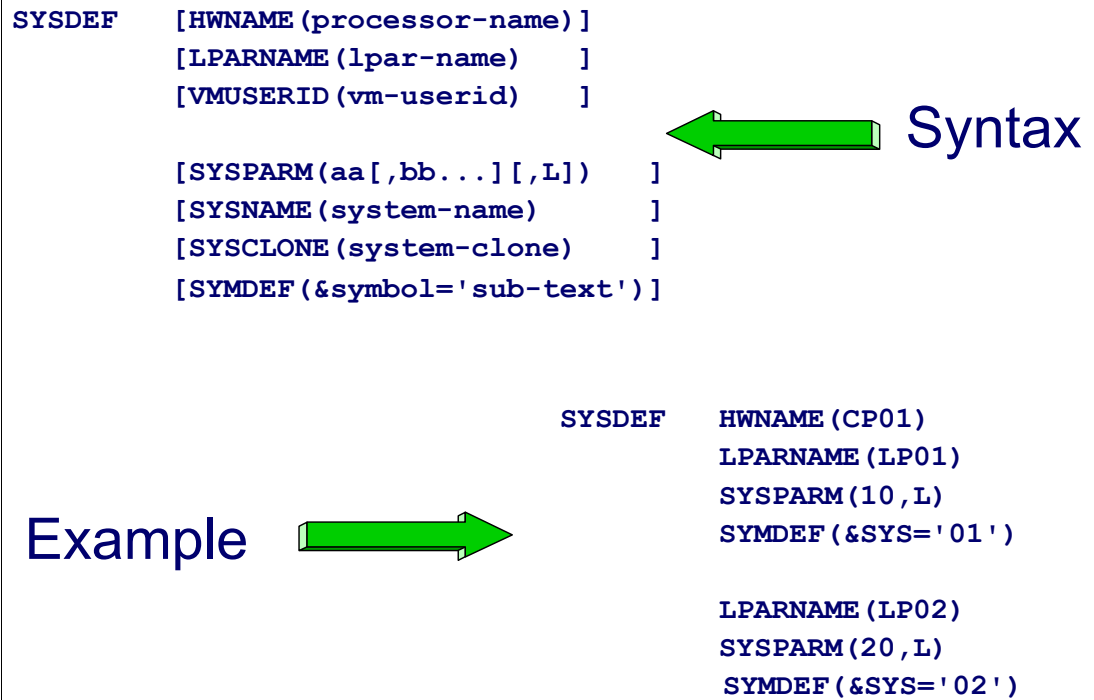


Figure 1-12 Specifying an IEASYMxx parmlib member

### Create an IEASYMxx parmlib member

The main purpose of IEASYMxx is to provide a single place to specify system parameters for each system in a multisystem environment. The IEASYMxx parmlib member contains statements that do the following:

- Define static system symbols
- Specify IEASYSxx parmlib members that contain system parameters

You can apply the statements in IEASYMxx to any system in your environment. Therefore, only one IEASYMxx member is required to define static system symbols and specify system parameters for all systems.

In IEASYMxx, you can define up to 99 additional static system symbols for each system in a multisystem environment. In other words, you can define as many additional static system symbols in IEASYMxx as you like, so long as no more than 99 of those system symbols apply to a particular system at any time during an IPL.

The LOADxx parmlib member specifies the IEASYMxx member that the system is to use. For information about how to specify the suffix of the IEASYMxx member in LOADxx, see the next figure.

## 1.13 LOADxx parmlib member

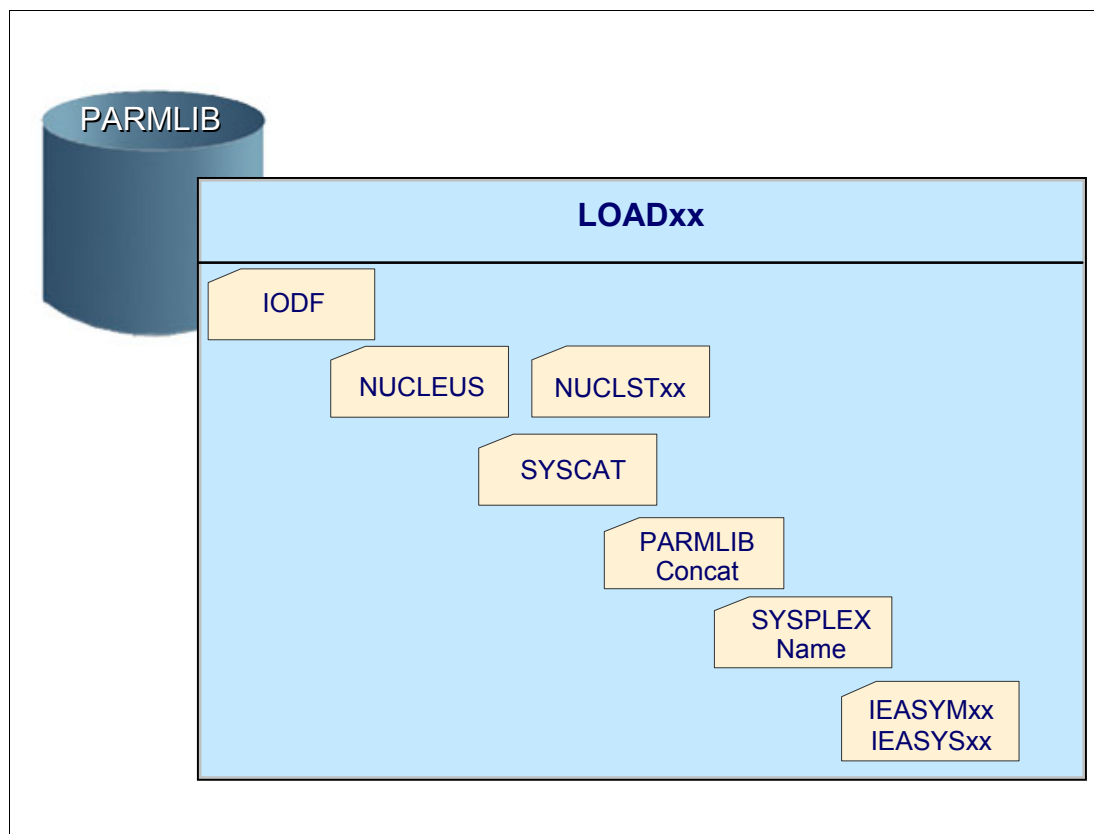


Figure 1-13 Specifying the LOADxx parmlib member data sets

### LOADxx (system configuration data sets)

The system must have access to a LOADxx member which specifies:

- ▶ Information about your I/O configuration as specified by the IODFxx suffix to be used.
- ▶ An alternate nucleus ID.
- ▶ The NUCLSTxx member that you use to add and delete modules from the nucleus region at IPL-time.
- ▶ The name of the master catalog.
- ▶ Information about the parmlib concatenation.
- ▶ The name of the sysplex (systems complex) that a system is participating in; it is also the substitution text for the &SYSPLEX system symbol.
- ▶ The IEASYMxx and IEASYSxx parmlib members that the system is to use.
- ▶ Additional parmlib data sets that the system will use to IPL. These data sets are concatenated ahead of SYS1.PARMLIB to make up the parmlib concatenation.
- ▶ Filtering keywords so you can use a single LOADxx member to define IPL parameters for multiple systems. The initial values of the filter keywords (HWNAME, LPARNAME, and VMUSERID) are set at IPL to match the actual values of the system that is being IPLed. The LOADxx member can be segmented by these keywords.

The LOADxx member is selected through the use of the LOAD parameter on the *system control* (SYSCTL) frame of the system console. For information about specifying the LOAD parameter, see *z/OS MVS System Commands*, SA22-7627. If the operator does not select a LOADxx member on the system console, the system uses LOAD00.

## 1.14 Placement of LOADxx member

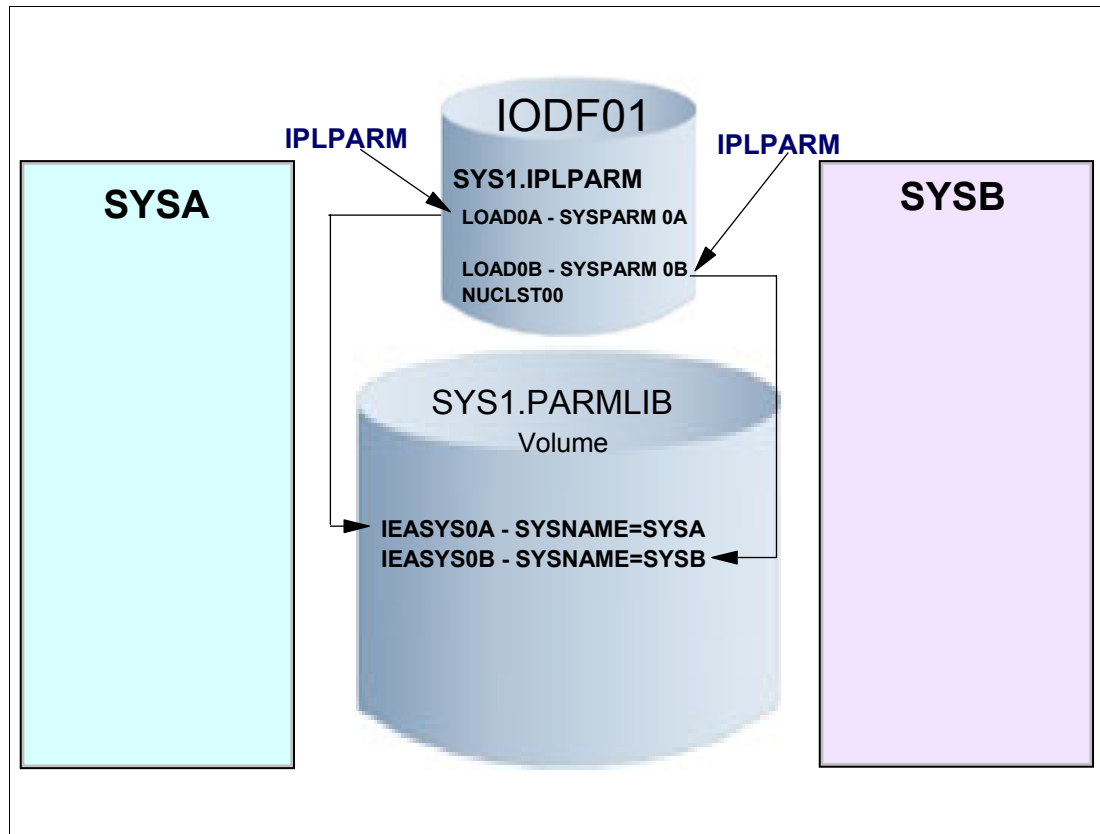


Figure 1-14 Placing the LOADxx member into a data set

### Placement of LOADxx

You can place the LOADxx member in one of the system data sets:

**SYSn.IPLPARM** SYSn.IPLPARM is an optional partitioned data set that contains LOADxx members that point to I/O definition files (IODFs) that reside on the same volume as SYSn.IPLPARM. SYSn.IPLPARM is particularly useful in a multisystem environment, as shown in Figure 1-14.

You can use one SYSn.IPLPARM data set for each system. On each system, the SYSn.IPLPARM data set must be on the volume where the production IODF for that system resides.

For more information about the relationships between a SYSn.IPLPARM data set and its associated IODFs, see *z/OS MVS System Data Set Definition*, SA22-7629.

**Note:** This system data set must reside on a direct access volume. This volume should contain only one SYSn.IPLPARM data set and its associated IODFs. (The character *n* is a single numeral, 0 through 9.)

## **SYS1.PARMLIB**

Consider placing LOADxx in the SYSn.IPLPARM data set. During IPL, the system looks for LOADxx in the following order:

1. SYS0.IPLPARM through SYS9.IPLPARM on the IODF volume
2. SYS1.PARMLIB on the IODF volume
3. SYS1.PARMLIB on the SYSRES volume

Do not create a SYSn.IPLPARM data set unless it contains the LOADxx member that is used to configure your system. When the system finds either SYSn.IPLPARM or SYS1.PARMLIB, it expects to find a LOADxx member in the data set. If the LOADxx member specified on the LOAD parameter is not in the data set, the system loads a wait state.

The NUCLSTxx member must reside in the same data set as the LOADxx member. This member can reside in either SYS1.PARMLIB or SYSn.IPLPARM, depending on how the installation defined its I/O configuration.

Member SYSCATLG of SYS1.NUCLEUS can contain a pointer to the master catalog. However, IBM recommends that you use the SYSCAT statement of the LOADxx member of SYS1.PARMLIB or SYSn.IPLPARM to identify the master catalog.



## 1.15 Controlling parmlib

- ☐ Make full use of up to 10 installation-defined parmlib
- ☐ Include changed members in one of the 10 parmlib
- ☐ Use installation-defined parmlib for testing
- ☐ Delete unsupported parameters and members
- ☐ Use parmlib members for appropriate functions
- ☐ Update parmlib for changes
- ☐ Keep track of parameters in parmlib members
- ☐ Allocate sufficient space for parmlib
- ☐ Ensure EXITxx and GTFPARM reside in SYS1.PARMLIB
- ☐ Decide where parmlib resides
- ☐ Protect the parmlib

Figure 1-15 Ways to control parmlib members

### Controlling parmlib

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

- ▶ Use the ability to have up to 10 installation-defined parmlib data sets to separate your parmlib members along organization or function lines and use appropriate RACF security for each data set.
- ▶ Include members with installation changes in one of the 10 installation-defined parmlib data sets to avoid having the member overlaid by IBM maintenance on SYS1.PARMLIB.
- ▶ Use an installation-defined parmlib data set to contain any parmlib members to be used on test systems. They can be included in front of your *standard* parmlib concatenation without forcing changes to the *standard* parmlib concatenation.
- ▶ Delete unsupported parameters and members. Because most components treat unsupported parameters from previous releases as syntax errors, you should probably remove the old parameters or build parmlib from scratch. This action will minimize the need for operator responses during an IPL. Then, you can save space by removing unsupported members.
- ▶ Use the parmlib members for the appropriate functions. For example, use COMMNDxx to contain commands useful at system initialization. Use IEACMDxx for IBM-supplied commands. Use IEASLPxx for SLIP commands. See each member for further information.

- ▶ Update parmlib with new or replacement members as you increase your experience with new releases.
- ▶ Keep track of which parameters are included in particular parmlib members. This bookkeeping is necessary for two reasons:
  - a. The system doesn't keep track of parmlib members and their parameters.
  - b. The default general parameter list IEASYS00 is always read by the system and 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 IEALPxx).

The installation should keep records of which parameters and which values are in particular members, and which general members point to which particular specialized members (COMMNDxx, IEALPxx, and so forth). A grid or matrix for such bookkeeping is very helpful.

- ▶ 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 blocksize of the data set. Then add a suitable growth factor (for example, 100 to 300 percent) to allow for future growth of alternate members. To recapture space occupied by deleted members, use the **compress** function of IEBCOPY. However, should the data set run out of space, you may 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.
- ▶ Ensure EXITxx and GTFPARM reside in SYS1.PARMLIB since they can only be accessed from SYS1.PARMLIB.
- ▶ Decide which volume(s) and device(s) should 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 could be placed on a slow or moderate speed device. For information about the placement of parmlib data sets and the IODF data set, see *z/OS MVS System Data Set Definition*, SA22-7629.
- ▶ Use a security product (like 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 (IEAAPFxx and PROGxx) from user tampering.

## General syntax rules for the creation of members

The following general syntax rules apply to the creation of most parmlib members. Exceptions to these rules are described under specific members. The general rules are:

- ▶ Logical record size is 80 bytes.
- ▶ Blocksize must be a multiple of 80.
- ▶ Any columns between 1 and 71 may contain data.
- ▶ Statements are entered in uppercase characters.
- ▶ Suffix member identifiers can be any combination of A to Z and 0 to 9, though some member identifiers may allow other characters.
- ▶ Columns 72 through 80 are ignored.
- ▶ Continuation is indicated by a comma followed by one or more blanks after the last entry on a record.
- ▶ Leading blanks are suppressed. A record therefore need not start at a particular column.
- ▶ Suffix member identifiers (such as LNK=A2) can be any alphanumeric combination.
- ▶ Comments are most often indicated by using /\* and \*/ as the delimiters in columns 1–71, for example, /\*comment\*/

## 1.16 Initialization of z/OS

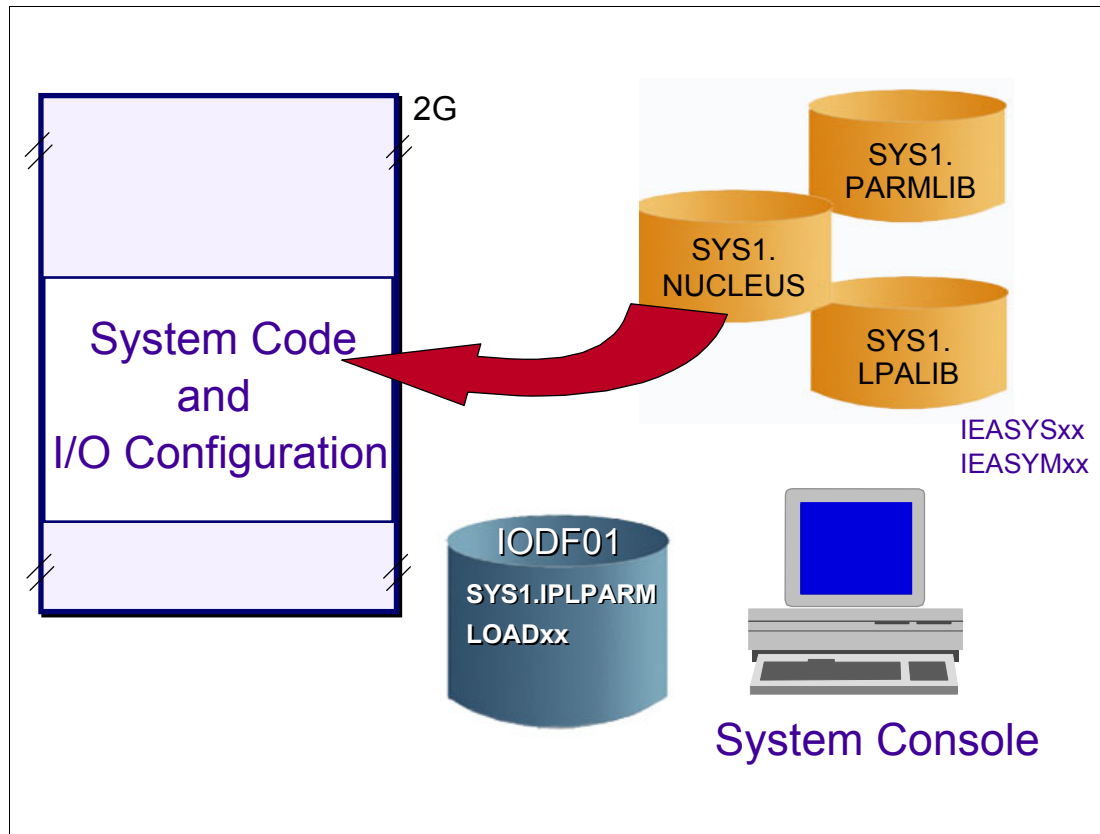


Figure 1-16 Initialization process for z/OS

### IPL of z/OS

Initial program loading (IPL) provides a manual means for causing a program to be read from a designated device and for initiating execution of that program. When the system hardware is ready, you can use the system console to load the system software.

During initialization of a z/OS system, the operator uses the system console, which is connected to the processor controller or support element. From the system console, the operator initializes the system control program during the nucleus initialization program (NIP) stage.

During the NIP stage, the system might prompt the operator to provide system parameters that control the operation of MVS. The system also issues informational messages that inform the operator about the stages of the initialization process.

The LOADxx parmlib member allows your installation to control the initialization process. For example, in LOADxx, you specify 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.

The definition of these parameters are discussed later in this chapter.

## 1.17 Types of IPLs

- ❑ Cold start
  - Loads PLPA - (CLPA)
- ❑ Quick start
  - No reload of PLPA - (CVIO)
- ❑ Warm start
  - No reload of PLPA

### Message to operator and a possible response

IEA101A SPECIFY SYSTEM PARAMETERS

R 00,SYSP=xx,CLPA

R 00,SYSP=xx,CVIO

Figure 1-17 The three types of IPLs and the operator response

### Types of IPL

It is possible that in this discussion we will use terms that are unfamiliar to you (like LPA, SQA, VIO, and so forth). The objective is to condense the IPL types and process (shown in the next figure) in order to give you an overview of the IPL and its interactions with the operator. Further, we discuss the main aspects of the architecture as well as the mechanisms of z/OS mentioned here.

Depending on the level of customization, a system IPL can bring up many different configurations, but there are only three basic types of IPL:

- Cold Start** Any IPL that loads or reloads the Pageable Link Pack Area (PLPA) and does not preserve 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 will automatically be loaded from the LPALST concatenation. The page data sets for this IPL will be those specified in IEASYSxx, plus any additional ones specified by the operator. Subsequent IPLs need only be cold starts if the PLPA has to be reloaded either to alter its contents or to restore it 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 modules in the LPALST concatenation.
  - After the PLPA page data set has been damaged and is therefore unusable, so its contents must be restored. The PLPA can be reloaded

by responding CLPA (Create Link Pack Area) 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. You would normally perform a Quick Start IPL after a power up. The PLPA from the previous session can be used without reloading it from the LPALST concatenation. A Quick Start can be initiated by replying CVIO (Clear VIO) 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.

### System parameters prompt

If the LOADPARM, shown in Figure 1-18 on page 30, has been set to indicate the operator should be prompted for system parameters (A, P, S or T), NIP issues the following prompt on the NIP console:

```
IEA101A SPECIFY SYSTEM PARAMETERS
```

At this point, the operator can respond in one of a number of ways:

- ▶ Pressing Enter will cause the system to use the parameters coded in SYS1.PARMLIB(IEASYS00).
- ▶ Entering **SYSP=xx** will cause the parameters in SYS1.PARMLIB(IEASYSxx) to be used.
- ▶ Entering multiple IEASYSxx members, **SYSP=(xx,yy,...)**, for multiple concatenations to be used.

Specifying any of the individual parameters that appear in IEASYSxx will cause these parameters to be used instead of the ones coded in IEASYSxx except PAGE parameters. If multiple IEASYSxx members are searched, the system *always* reads IEASYS00 first, followed by the additional ones you specify. If the same parameters are found in one or more IEASYSxx members, the latest overriding parameter is used.

### PAGE parameters

The PAGE parameter allows the installation to name page data sets as additions to existing page data sets. The maximum number of page data sets is 256, which includes the optional page data set specified by the DUPLEX parameter. The system determines which page data sets to use by merging information from three sources: IEASYS00, IEASYSxx, and the PAGE parameter.

During system initialization, the system first uses the list of page data sets specified on the PAGE parameter of the IEASYS00 parmlib member. It then uses any other IEASYSxx parmlib member (identified via the SYSP=xx parameter). The IEASYSxx PAGE data set name list overrides the one in IEASYS00.

## 1.18 Initial Program Load (IPL)

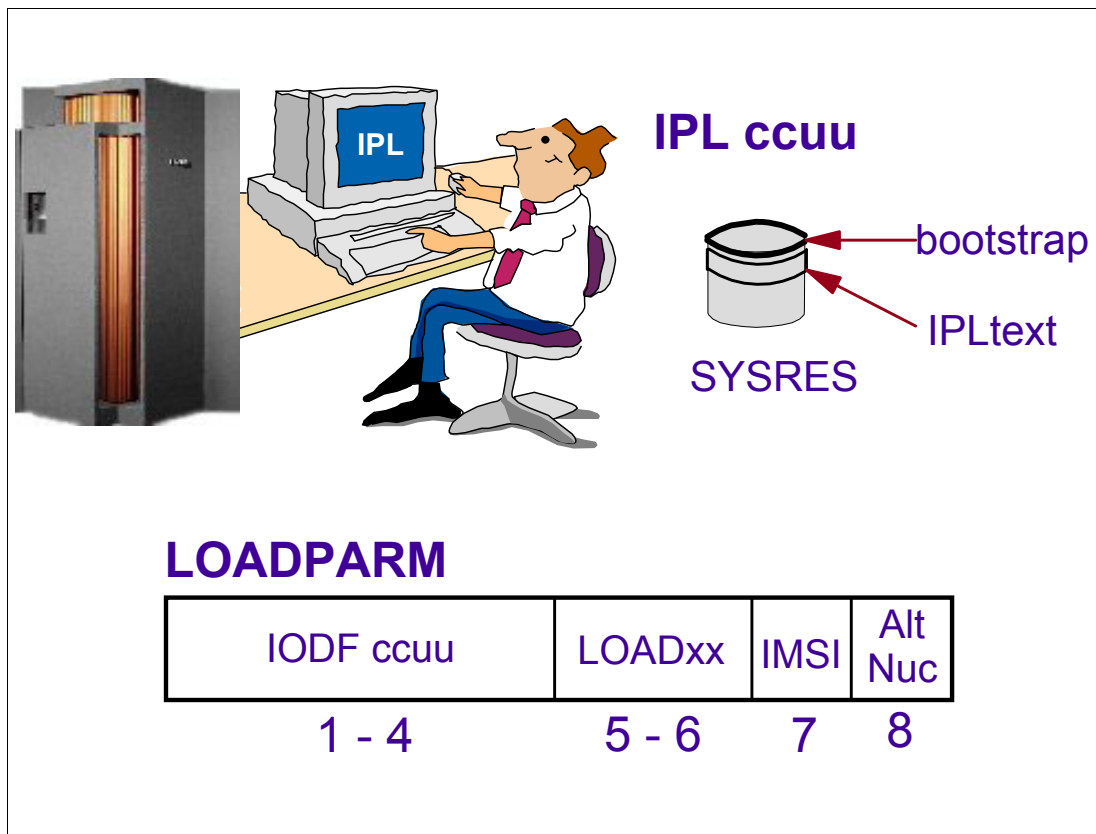


Figure 1-18 IPLing the system

### The IPL process

An Initial Program Load (IPL) is the act of loading a copy of the operating system from disk into the CPU's central storage and executing it. Not all disks attached to a CPU will have loadable code on them. A disk that does is generally referred to as an "IPLable" disk, and more specifically as the SYSRES volume.

IPLable disks will contain a bootstrap module at cylinder 0 track 0. At IPL, this bootstrap is loaded into storage at real address zero and control is passed to it. The bootstrap then reads the IPL control program IEAIPLO0 (also known as IPL text) and passes control to it. This in turn starts the more complex task of loading the operating system and executing it.

Attempts to IPL from a disk that does not contain IPL text results in an error condition.

### IEAIPLO0 parmlib member

IEAIPLO0 prepares an environment suitable for starting the programs and modules that make up the operating system. First, it clears central storage to zeros before defining storage areas for the master scheduler. It then locates the 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 will then start to construct the normal operating system environment of control blocks and subsystems that make up a z/OS system. Some of the more significant tasks performed by the IRIMs are to:

- Read the LOADPARM information entered on the hardware console at the time the IPL command was executed. LOADPARM is discussed in “The load parameter (LOADPARM)” on page 31.
- Search the volume specified in the LOADPARM as containing the IODF data set for the LOADxx member—the value of xx is also taken from the LOADPARM. IRIM will first attempt to locate LOADxx in SYS0.IPLPARM. If this is unsuccessful, it will look for SYS1.IPLPARM, and so on, up to and including SYS9.IPLPARM. If, at this point, it still has not been located, the search will continue in SYS1.PARMLIB.

When a LOADxx member has been successfully located, it will be opened and information including the nucleus suffix (unless overridden in LOADPARM), the master catalog name, and the suffix of the IEASYSxx member to be used, will be read from it.

- Load the MVS nucleus.
- Initialize virtual storage in the master scheduler address space for the System Queue Area (SQA), the Extended SQA (ESQA), the Local SQA (LSQA), and the Prefixed Save Area (PSA). At the end of the IPL sequence, the PSA will replace IEAIPL00 at real storage location zero, where it will then stay.
- Initialize real storage management, including the segment table for 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 Nucleus Initialization Program (NIP), which then invokes the Resource Initialization Modules (RIMs), one of the earliest of which starts up communications with the NIP console defined in SYS1.NUCLEUS.

## The load parameter (LOADPARM)

The most basic level of control is through the Load Parameter (LOADPARM). This is an eight-character value made up of four fields that the operating system will refer to as it IPLs, causing it to take the specified actions. The LOADPARM is made up of the following components:

|                  |                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                             |
|------------------|---------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| <b>IODF ccuu</b> | The IODF device address. This 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 specify the hexadecimal device address for the device that contains the I/O Definition File (IODF) VSAM data set. This is also the device on which the search for the LOADxx member of SYSn.IPLPARM or SYS1.PARMLIB begins. This device address can be in the range X'0000' to X'FFFF'. If the address is less than four digits, pad it with leading zeros—for example, a device address of <i>c1</i> should be entered as <i>00c1</i> . If you do not specify the device address, the system uses the device address of the system residence (SYSRES) volume. |
| <b>LOADxx</b>    | <p>The xx suffix of the LOADxx parmlib member. The LOADxx member contains information about the name of the IODF data set, which master catalog to use, and which IEASYSxx members of SYS1.PARMLIB to use.</p> <p>The default for the LOADxx suffix is zeroes. The system reads the LOADxx and NUCLSTxx members from SYSn.IPLPARM or SYS1.PARMLIB on the volume specified on the LOAD parameter (or the SYSRES volume, if no volume is specified). After the system opens the master catalog, the system reads all other members from the SYS1.PARMLIB data set that is pointed to by the master catalog. This SYS1.PARMLIB might be different from the SYS1.PARMLIB to which the LOAD parameter points.</p>                                |
| <b>IMSI x</b>    | The prompt feature character specifies the prompting and message suppression characteristics that the system is to use at IPL. 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 cause you to miss some critical messages, so you should always review the hardcopy log after initialization is complete.

**Alt Nuc x**

The last character specifies the alternate nucleus identifier (0–9). Use this character at the system programmer's direction. 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.



## 1.19 PARMLIB commands

### ❑ D PARMLIB

```
IEE251I 09.32.31 PARMLIB DISPLAY 452
PARMLIB DATA SETS SPECIFIED
AT IPL
ENTRY  FLAGS  VOLUME  DATA SET
      1      D      TOSY1  SYS1.PARMLIB
MASTER PROCESSING USING THE FOLLOWING PARMLIBS
ENTRY  FLAGS  VOLUME  DATA SET
      1      S      TOSY1  SYS1.PARMLIB
```

### ❑ D IPLINFO

```
IEE254I 13.06.10 IPLINFO DISPLAY 025
SYSTEM IPLED AT 08.25.41 ON 08/05/1996
RELEASE SP6.0.2
USED LOADR2 IN SYS0.IPLPARM ON OCD0
IEASYM LIST = XX
IEASYS LIST = (R2,XX) (OP)
```

Figure 1-19 Commands to display parmlib information after IPL

### Parmlib commands

The use of commands enables installations to display the current logical parmlib, display general IPL information, and change the current logical parmlib settings.

The commands are:

#### ► DISPLAY PARMLIB

This command displays the logical parmlib setup for the IPLed system. The output of this command includes the parmlib data set name(s) and volser(s) that were defined by LOADxx PARMLIB statement(s), and if used, MASTER JCL IEFPARM DD statements. When the errors option on the command is used, the display shows any parmlibs that were defined in LOADxx but were not found. This command is only valid before a **SETLOAD** command is issued. A sample output is shown in the visual.

#### ► DISPLAY IPLINFO

This command displays the general IPL information used by the system. The output includes the date and time of the IPL, release level, LOADxx information, and what IEASYSxx and IEASYMxx parmlib members were used. A sample output of the command is shown in Figure 1-19.

## 1.20 Parmlib commands (continued)

### SETLOAD xx,PARMLIB

```
SETLOAD R2,PARMLIB,DSN=SYS0.IPLPARM
IEF196I IEF237I OCD0 ALLOCATED TO SYS00006
IEE252I MEMBER   LOADR2 FOUND IN SYS0.IPLPARM
IEF196I IEF237I OCD0 ALLOCATED TO SYS00007
IEF196I IEF237I OFC1 ALLOCATED TO SYS00008
IEF196I IEF237I OFC1 ALLOCATED TO SYS00009
IEF196I IEF285I   SYS1.PARMLIB                      KEPT
IEF196I IEF285I   VOL SER NOS= TOTSYS1.
IEF196I IEF285I   SYS0.IPLPARM                      KEPT
IEF196I IEF285I   VOL SER NOS= IODFPK.
IEF107I PARMLIB CONCATENATION WAS UPDATED FROM LOADR2
```

Figure 1-20 Changing parmlib concatenations with the setload command

### Parmlib commands

The following command (which is also shown in the figure) allows the installation to dynamically change a parmlib concatenation without having to IPL. The **SETLOAD** command specifies the LOADxx member that contains the PARMLIB statements to use for the switch. The syntax of the command is as follows:

```
SETLOAD xx,PARMLIB[, {DSNAME|DSN}=dsn] [, {VOLUME|VOL|VOLSER}=vo1]
```

Where:

- ▶ **xx** - Specifies the one or two character suffix used to identify the LOADxx member that you want to process.
- ▶ **PARMLIB** - Specifies that the system is to process the PARMLIB statements in the LOADxx member according to the filter parameters (HWNAME, LPARNAME, VMUSERID). For more information on filter parameters, see the LOADxx member in *z/OS MVS Initialization and Tuning Reference*, SA22-7592.
- ▶ **DSNAME** or **DSN =dsn** - Specifies the 1 to 44 character name of the data set where the LOADxx member resides. The default is to locate the LOADxx member specified in a data set within the existing parmlib concatenation.
- ▶ **VOLUME** or **VOL** or **VOLSER =vo1** - Specifies the 1 to 6 character serial number identifier of the volume where the specified data set resides.

A sample output of the **SETLOAD** command is shown in Figure 1-20.

```
SETLOAD R2,PARMLIB,DSN=SYS0.IPLPARM
```

## 1.21 Use of SETLOAD command

### ❑ After SETLOAD issued

#### D PARMLIB

```
IEE251I 09.36.52 PARMLIB DISPLAY 470
PARMLIB DATA SETS SPECIFIED
AT 09.34.11 ON 08/03/1996
LOADR2 DATA SET=SYS0.IPLPARM
      VOLUME=CATALOG
ENTRY  FLAGS  VOLUME  DATA SET
   1      S    CATALOG  SYS0.IPLPARM
   2      S    CATALOG  SYS1.OS390R2.PARMLIB
   3      S    CATALOG  SYS1.PARMLIB
```

### ❑ After SETLOAD issued and an IPL

#### D PARMLIB

```
IEE251I 13.10.57 PARMLIB DISPLAY 027
PARMLIB DATA SETS SPECIFIED
AT IPL
ENTRY  FLAGS  VOLUME  DATA SET
   1      S    IODFPK   SYS0.IPLPARM
   2      S    TOTSYS1  SYS1.OS390R2.PARMLIB
   3      D    TOTSYS1  SYS1.PARMLIB
```

Figure 1-21 Displaying the parmlib members in use after a setload command

## Use of SETLOAD command

The sample output shown in Figure 1-21 is a result after the **SETLOAD** command was issued.

The second display shows the output of a **DISPLAY PARMLIB** command after an IPL had taken place using **LOADR2**. **PARMLIB** statements were specified in **LOADR2**. Note the difference in the **FLAGS** column shown in the figure.

The **FLAGS** describe how the parmlibs were specified:

- ▶ S denotes the **LOADxx PARMLIB** statement.
- ▶ D denotes the default (**SYS1.PARMLIB**).

## 1.22 Catalogs

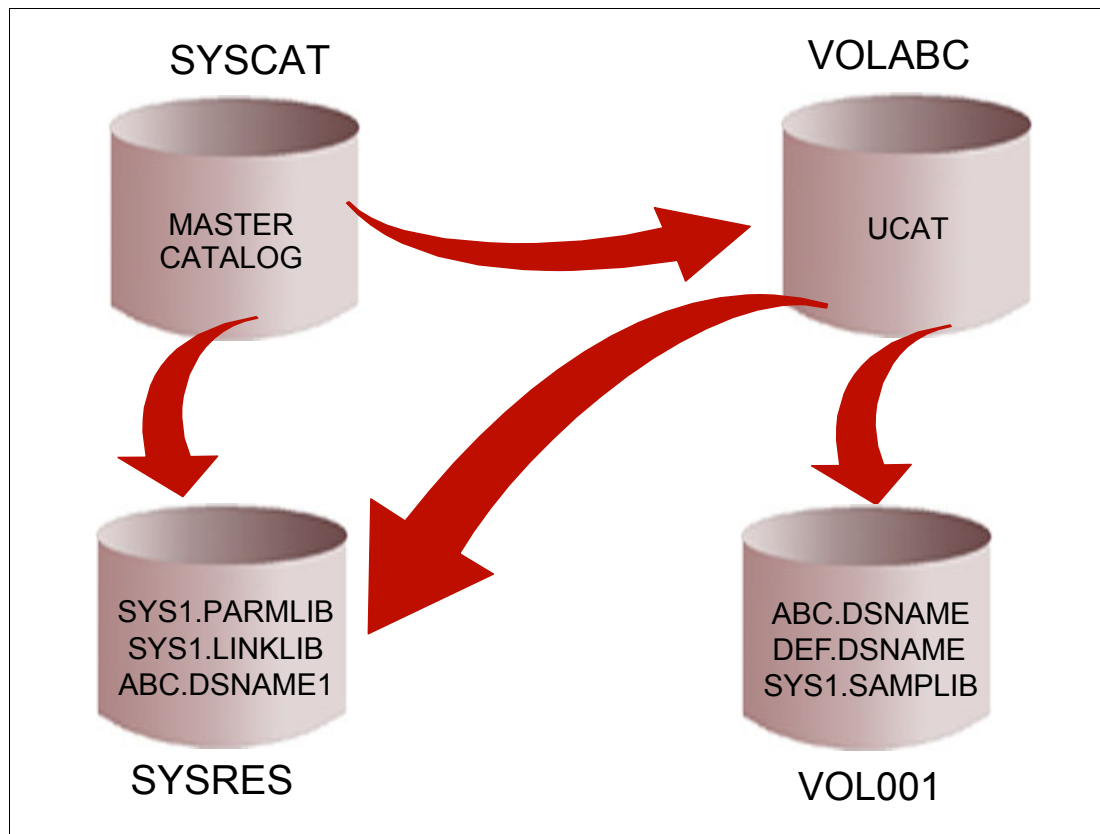


Figure 1-22 Defining catalog data sets

### Catalogs

A catalog is a data set that contains information about other data sets. It provides users with the ability to locate a data set by name, without knowing where the data set resides. By cataloging data sets, your users will need to know less about your storage setup. Thus, data can be moved from one device to another, without requiring a change in JCL DD statements which refer to an existing data set.

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 must be cataloged.

### Using indirect catalog entries

Indirect cataloging, also known as 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 system residence (IPL) volume when accessed through the catalog. This allows you to change the volume serial number or the device type of the system residence volume without also having to recatalog the non-VSAM data sets on that volume.

The extended indirect volume serial support that was introduced in OS/390 V2R3 allows catalog entries to be resolved using system symbols defined in the IEASYMxx parmlib member, so that indirect references can be made to one or more logical extensions to the system residence volume. Therefore, you can have multiple levels of z/OS data sets residing

on multiple sets of volumes with different names and device types, and use them with the same master catalog.

Using indirect catalog entries, together with the indirect volume serial enhancements, allows you to share the master catalog among multiple images that use different volumes with different names for the system residence volumes and their extensions.

For more information on indirect volume serial support, see *z/OS MVS Initialization and Tuning Reference*, SA22-7592.

## **Catalog management**

Ensuring that your catalogs are effectively managed is therefore a crucial aspect of DASD management. Some of the most common catalog management tasks that a system programmer may have are:

- ▶ Defining and maintaining the master catalog
- ▶ Defining the alias and user catalog
- ▶ Protecting the catalogs
- ▶ Cleaning up catalogs
- ▶ Backing up catalogs and performing catalog recovery

## 1.23 Separating data from software

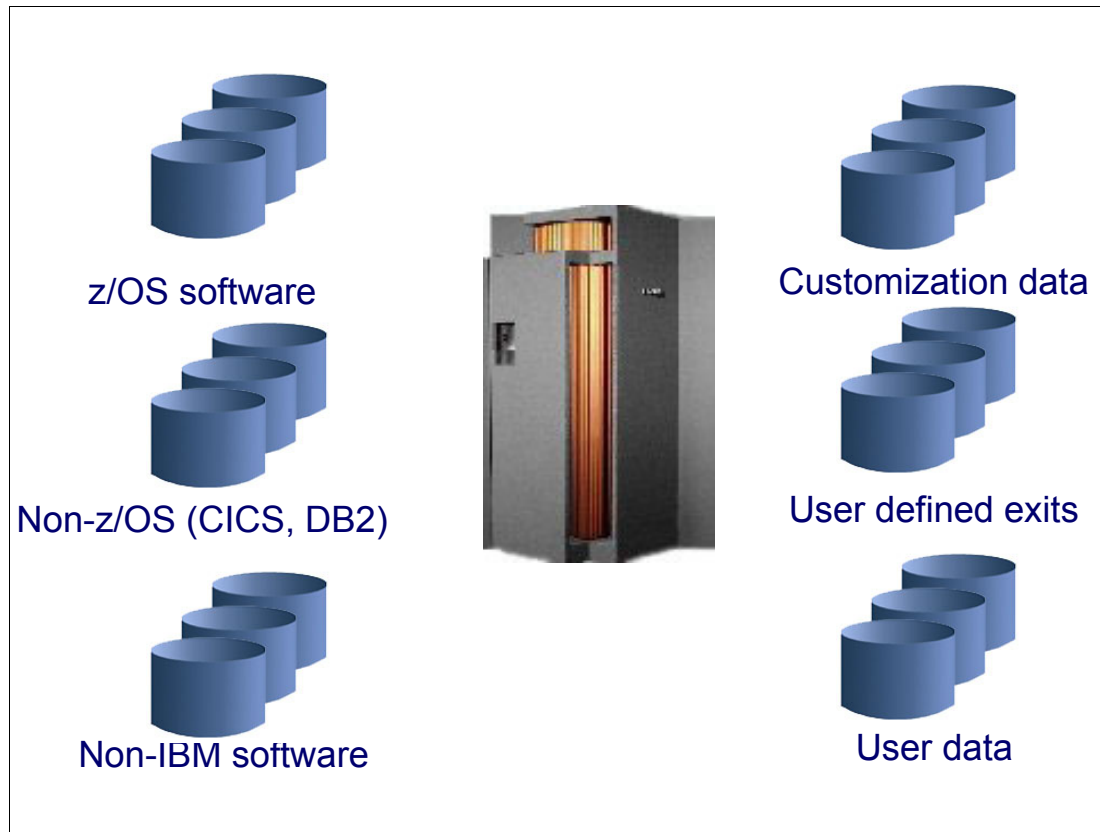


Figure 1-23 Placing the software on separate volumes from user data

### Separating data from software

When you separate your data from your system software, you eliminate many tasks that would otherwise need to be performed each time you upgrade or replace your system software. One effective way to achieve this is to use dedicated pools of DASD volumes for each.

The kinds of data you should separate from z/OS software are:

- ▶ Customization data, including most system control files
- ▶ Non-IBM software
- ▶ IBM non-z/OS products, for example CICS and DB2®
- ▶ User-defined exits
- ▶ User data

Your goal is to make it easier to replace the volumes that contain z/OS software, which allows you to keep the other software and data you will need to use with z/OS across migrations.

## 1.24 Placing data sets on specific volumes

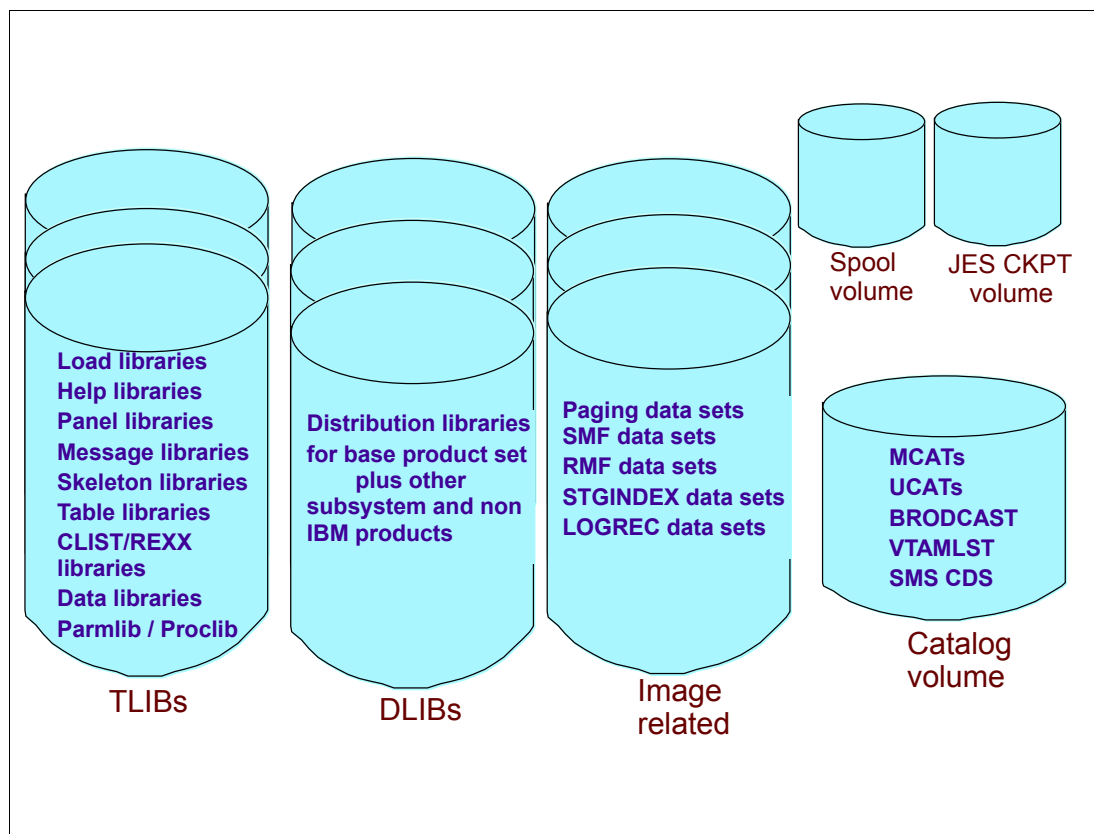


Figure 1-24 Where to place data sets

### Placing data sets on specific volumes

Some SYSRES volume types, such as the 3390-3, are not big enough to hold all the target libraries for a z/OS system. Therefore, you have to move some data sets to SYSRES logical extension volumes (or overflow volumes). The considerations you should take into account for placing data sets on specific volumes are:

- ▶ Your ability to use a system (or subsystem) replacement
- ▶ Data set and system availability
- ▶ System performance
- ▶ System cloning and servicing techniques
- ▶ Sysplex/multisystem operations
- ▶ Sharing data sets
- ▶ Backup and recovery
- ▶ Disaster recovery

With these considerations in mind, you should determine which data sets to place on each volume based on data set type, not based on element, feature, or product. There are basically five types of data sets, each of which can be placed on a separate (logical) volume:

- ▶ SMP/E global-shared data sets
- ▶ Target libraries (TLIBs) for product sets

- ▶ Distribution libraries (DLIBs) for product sets
- ▶ Image-related data sets
- ▶ Cluster-related data sets

### **SMP/E global-shared data sets**

These data sets should be placed on a volume shared by all systems in the complex that need SMP/E global information. Sharing these data sets will provide for easy backup and recovery. The recommended types of data sets for this volume are:

- ▶ SMP/E global CSI
- ▶ SMPPTS
- ▶ SMP/E global logs (SMPLOG and SMPLOGA)



## 1.25 The TLIBs volumes

- ❑ TVOL1
- ❑ TVOL2 - n
- ❑ HFS Target Volume
- ❑ Licensed Product Target Volume
- ❑ Vendor Product Target Volume
- ❑ Subsystem Target Volume

Figure 1-25 Target libraries

### Target libraries (TLIBs) for product sets

These target libraries spread across several volumes, from primary to secondary volumes:

- ▶ **Primary target volume (TVOL1)** - The TVOL1 is the first target library volume and the system residence (IPL) volume; it contains many of the z/OS target libraries. Make sure you leave enough free space to allow for future growth. The recommended types of TVOL1 data sets are:
  - *Load libraries* - data sets containing load modules.
  - *Change migration libraries* - used during migration from one level of software to another.
  - *Help libraries* - data sets that contain help information.
  - *Panel libraries* - data sets that contain ISPF panels.
  - *Message libraries* - z/OS libraries that contain ISPF messages and MMS source messages.
  - *Skeleton libraries* - z/OS libraries that contain ISPF skeletons.
  - *Table libraries* - z/OS libraries that contain ISPF tables.
  - *Fixed-block CLIST and EXEC libraries* - data sets that contain CLIST and REXX EXECs.
  - *Data libraries* - z/OS libraries that contains data parts which include workstation information, header files, or other various types of data.
  - *SMP/E-managed PARMLIB* - the data set that is pointed to by the PARMLIB DDDEF, which will be used to store parmlib members supplied by products you install.

- *SMP/E-managed PROCLIB* - the data set that is pointed to by the PROCLIB DDDEF, which will be used to store JCL procedures supplied by products you install.
- ▶ **Secondary target volumes (TVOL2–TVOLn)** - The TVOL2 through TVOLn are volumes used for data sets that do not fit on TVOL1. They are for the z/OS product set and the recommended types of TVOL2 through TVOLn data sets are:
  - *Fixed-block CLIST and EXEC libraries* - used only if variable-block CLIST and EXEC libraries were used on TVOL1.
  - *Sample and JCL libraries* - z/OS libraries that contain samples, header files, and JCL jobs.
  - *Source libraries* - z/OS libraries that contain source code.
  - *Macro libraries* - z/OS libraries that contain assembler macros, header files, and other information identified in SMP/E as the element type MACRO.
  - *Workstation libraries* - can be combined with the data libraries.
  - *Softcopy libraries into which SMP/E installs* - z/OS libraries that contain books, bookshelves, and book indexes.
  - *Font and printing libraries* - z/OS libraries that contain fonts and data sets required for printing.
  - *Flat files that SMP/E cannot manage* - interface repositories and so forth, excluding books.
  - SMP/E target CSI
  - *SMP/E target data sets* - including SMPLTS, SMPMTS, SMPSTS, and SMPSCDS.
  - User catalog for the SMP/E target CSI and MVS-supplied data sets
- ▶ **HFS target volume** - Recommended types of data sets (filesystems) for this volume are:
  - HFS data sets for z/OS elements or features that install into an HFS.
  - Any non-z/OS HFS data set, except those containing customization data.
- ▶ **Licensed product target volume** - The libraries on this volume consist of the licensed product set that you might not have in a system-replacement order and you want to keep separate. The recommended types of data sets for this volume are:
  - Licensed program target libraries.
  - SMP/E target CSI.
  - SMP/E target data sets: SMPLTS, SMPMTS, SMPSTS, and SMPSCDS.
  - User catalog where all the licensed program libraries and the SMP/E target CSI are cataloged.
- ▶ **Vendor product target volume** - The libraries on this volume consist of the vendor product set that you might not have in a system-replacement order and you want to keep separate. The recommended types of data sets for this volume are:
  - Vendor target libraries.
  - SMP/E target CSI.
  - SMP/E target data sets: SMPLTS, SMPMTS, SMPSTS, and SMPSCDS,
  - User catalog where all the vendor product target libraries including the SMP/E target CSI are cataloged.
- ▶ **Subsystem target volume** - The libraries on this volume consist of the subsystem product sets (for example, CISCs, DB2, IMS™, or NCP). The recommended types of data sets for this volume are:
  - Subsystem target libraries.
  - Alternate subsystem SMP/E global CSI, if applicable.
  - SMP/E target CSI.
  - SMP/E target data sets: SMPLTS, SMPMTS, SMPSTS, and SMPSCDS.
  - User catalog where the subsystem targets libraries including SMP/E CSIs are cataloged.

## 1.26 The DLIB volumes

- ❑ DLIB Vol for TVOL1, TVOL2 - TVOLn, HFS
- ❑ DLIB Vol for Licensed Products
- ❑ DLIB Vol for Vendor Products
- ❑ DLIB Vol for Subsystems

*Figure 1-26 The distribution library volumes*

### **Distribution libraries (DLIBs) for product sets**

You should place data sets on the DLIB volumes wherever they fit. However, keep in mind how other systems will use the distribution libraries when you are deciding where to place them. There are cases where you don't want or need a set of distribution libraries available on certain packs, for example having multiple target zones connect to a DLIB zone. The DLIB volumes, like the target volumes, can be divided into primary and secondary volumes:

- ▶ DLIB volumes for TVOL1, TVOL2 through TVOLn, and HFS  
These distribution libraries are the ones that are placed by ServerPac for your z/OS product sets. By keeping the distribution libraries on the same volumes it will be easier to avoid overlaying data sets.
- ▶ DLIB volumes for licensed products  
These are the distribution libraries that correspond to the target libraries for the licensed product set. These are data sets that would not be overlaid in a system replacement.
- ▶ DLIB volume for vendor products  
These are the distribution libraries that correspond to the target libraries for the vendor product set. These are data sets that would not be overlaid in a system replacement.
- ▶ DLIB volumes for subsystems  
These distribution libraries are the ones that are placed by ServerPac for subsystem product sets. Keeping the distribution libraries on the same volumes makes it easier to avoid overlaying data sets.

## 1.27 The image-related volumes

- ❑ Page data set volume 1
- ❑ Page data set volume 2 through n
- ❑ HFS customization volume

Figure 1-27 Volumes containing non-shareable system information

### Image-related data sets

These data sets contain non-shareable system image information. Although the recommendation is that they be put on separate volumes, if DASD space is scarce you can combine them at the expense of performance or availability, or both. The image-related data sets are placed in the following recommended volumes:

- ▶ **Page data sets volume 1** - The recommended types of data sets for this volume are:
  - PLPA (one-cylinder allocation)
  - COMMON

**Note:** Unless your system is central-storage constrained, and has significant PLPA paging activity, there is little or no performance impact to combining the PLPA and COMMON page data sets. The PLPA data set should be allocated first, as a one-cylinder data set, with the COMMON data set allocated second, immediately following the PLPA data set on the same volume. The size of the COMMON data set should be large enough to contain both PLPA and COMMOND pages.

This causes the vast majority of PLPA pages to be written to the COMMOND page data set during IPL. This allows the operating system to use the chained CCWs within a single data set and improves performance when both data sets are on the same volume.

The message (ILR005E PLPA PAGE DATA SET FULL, OVERFLOWING TO COMMON DATA SET) during IPL can be ignored when the PLPA and COMMON page data sets are on the same volume.

- ▶ **Page data set volumes 2 to n** - The recommended types of data sets for these volumes are:
  - Local
  - SMF
  - RMF™ reporting
  - STGINDEX data set (if used)
  - Image-related LOGREC data set (if used)
- ▶ **HFS customization volume** - This is an installation-maintained volume that contains data sets that will not be overlaid by system replacement. This volume is separate from the HFS target volume because it contains unshareable HFS files that will generally need to be mounted MODE(RDWR). The recommended types of data sets for this volume are:
  - HFS data sets that must be in write mode (for instance, /tmp, /etc, /dev, /u) and that contain customized information.
  - User catalog where the HFS data sets are cataloged.

## 1.28 The cluster-related volumes

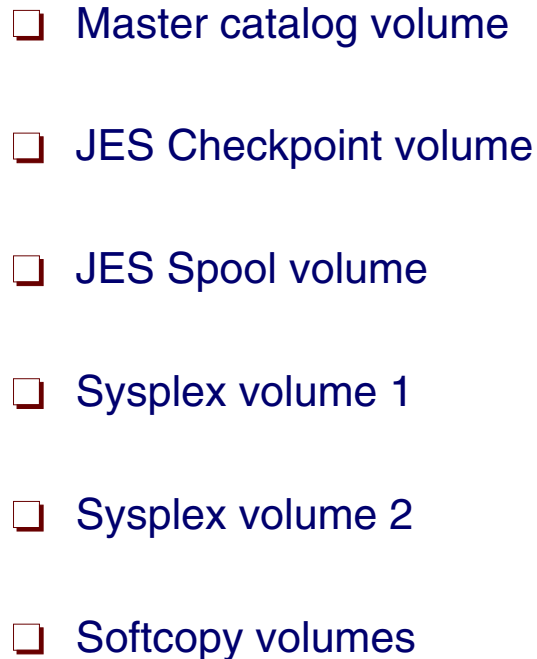
- 
- ❑ Master catalog volume
  - ❑ JES Checkpoint volume
  - ❑ JES Spool volume
  - ❑ Sysplex volume 1
  - ❑ Sysplex volume 2
  - ❑ Softcopy volumes

Figure 1-28 Volumes containing shareable data sets

### Cluster-related data sets

These are shareable data sets used in a multisystem environment. Cluster-related data sets should use system symbolics in their names for easier maintainability. While all cluster-related data sets can be combined on the same volume, it is usually preferable to separate certain data sets from others for performance or availability reasons. You can group the cluster-related data sets into the following volumes:

- ▶ **Master catalog volume** - The recommended types of data sets for this volume are:
  - Master Catalog
  - BROADCAST data set
  - Customer parmlib concatenation (not the SMP/E DDDEFed PARMLIB)
  - Customer proclib concatenation (not the SMP/E DDDEFed PROCLIB)
  - UADS data set (if used)
  - VTAMLST data set
  - SMS control data sets (ACDS, SCDS, and COMMDS), HSM, RMM, and so forth
  - APPC VSAM data set
  - System control files (TCPI/P configuration and so forth)
  - Primary RACF database
  - IODF data set
  - SYS0.IPLPARM
  - UCATs
  - SYS1.DDIR sysplex dump directory data set
  - DAE data set

- ▶ **JES2 checkpoint volume** - For maximum performance and reduced contention, place the primary JES2 checkpoint data set on its own dedicated volume. The JES2 checkpoint primary data set may be on a Coupling Facility.
- ▶ **Sysplex-related volume 1** - The recommended types of data sets for this volume are:
  - Sysplex primary
  - CFRM alternate
  - ARM primary
  - WLM primary
  - LOGR primary
  - OMVS primary

**Note:** The CFRM primary and SYSPLEX primary should be on different volumes attached to different control units. All other primary couple data sets can reside on the same volume, and all other alternate couple data sets can reside on a different volume.

- ▶ **Sysplex-related volume 2** - The recommended types of data sets for this volume are:
  - Sysplex alternate
  - CFRM alternate
  - ARM alternate
  - WLM alternate
  - LOGR alternate
  - OMVS alternate
  - Secondary RACF database
- ▶ **Softcopy volume** - This volume holds softcopy books and related data sets. The recommended types of data sets for this volume are:
  - Books
  - Bookshelves
  - Book indexes

Many volumes on your system will contain data sets that are not supplied by ServerPac. Keeping such volumes separate from those that ServerPac will replace, or that you will replace when migrating the new system to other system images, makes it easier to prevent overlaying data sets that you want to keep.

For more information on data set placement, see *z/OS and z/OS.e Planning for Installation*, GA22-7504.

## 1.29 Naming conventions for data sets

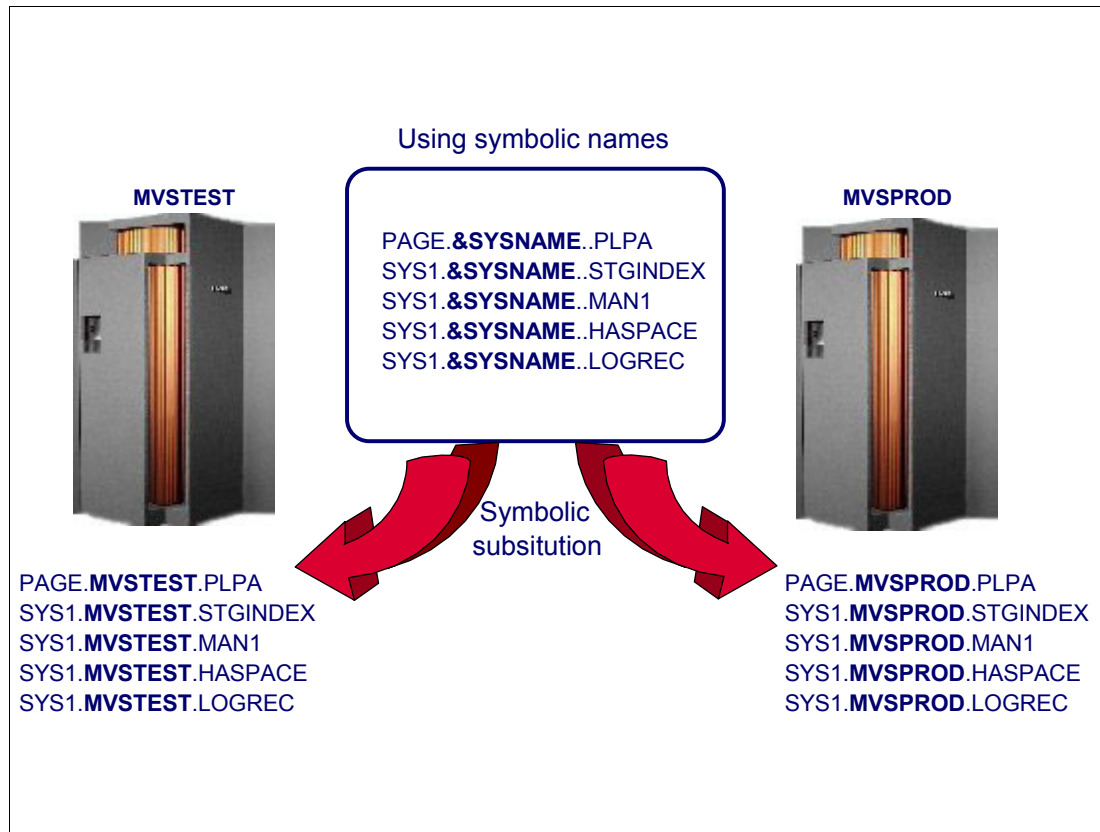


Figure 1-29 Choosing names for z/OS data sets

### Choosing a naming convention for data sets

Choosing the right naming conventions for system software data sets can save you considerable time during installation and migration.

Some data sets are associated with only one system in a multisystem environment. Choose names for these data sets that reflect the name of the specific system. Names of system operational data sets, such as page and swap data sets, should include the system name. You can accomplish this using the IBM-supplied system symbol `SC68`.

Using symbolic substitution involves carefully establishing naming conventions for data sets such as parmlib and proclib members, system images, HCD definition, and so forth. Remember that once your system goes into production with a set of naming conventions, you cannot easily change them.

IBM has been removing the level-dependent qualifier (such as `V1R1M0`) from default data set names for z/OS elements in order to ease your migration to a new z/OS release. This preserves much of your existing JCL, system procedures, parmlib and other system control file contents, and security system definitions across upgrades, saving you time and reducing the opportunity for error because updates will be limited to just the data sets in which the low-level qualifiers were changed.



## 1.30 DASD space utilization and performance

- ❑ Non-RECFM U Data Sets
  - Use System Determine Block Size, BLKSIZE=0
- ❑ RECFM U Data Sets
  - Use BLOCKSIZE=32760
- ❑ UADS Data Sets
  - Same block size as the currently used
- ❑ Font Libraries
  - Use BLOCKSIZE=12288

Figure 1-30 Determining space utilization for data sets

### DASD space utilization and performance

The space required by system software data sets, except for PDSE data sets, is affected by the block sizes you choose for those data sets. Generally, data sets with larger block sizes use less space to store the same data than those with smaller block sizes.

The exception to this general rule is fixed block (FB) record format data sets. They should not be allocated with block sizes larger than half the track length of the DASD they are allocated on. Doing so will cause considerable DASD space to be wasted because current DASD track lengths are less than twice the maximum block size of 32760 bytes.

### System-determined block size

Generally, system-determined block sizes (SDB) are the best choice for block size for fixed block (FB), variable blocked (VB), and variable block spanned (VBS) record format data sets. You should use SDB for all system software data sets with these record formats except those for which IBM specifically recommends other block sizes. One way to do this is by specifying BLKSIZE=0 in the DCB parameter of a DD statement when allocating new data sets using JCL. For more information, see *z/OS MVS JCL Reference*, SA22-7597.

Data sets with undefined (U) record formats do not follow the same rules as those with other record formats. In particular, most load libraries in partitioned data sets (not PDSEs) will require less space and offer better performance at increasing block sizes right up to the block size limit of 32760 bytes. This is because the program management binder, linkage editor,

and IEBCOPY's **COPYMOD** command use the data set block size only to set the maximum block length they will use.

### **Load libraries**

Allocate all load libraries using a block size of 32760 bytes unless you plan to move your system software data sets from the device types on which they were originally allocated to device types with shorter track lengths, or plan to move them between device types having different track lengths without using IEBCOPY **COPYMOD**.

For most efficient use of DASD, it is recommended that you allocate z/OS data sets using the following block sizes:

- ▶ Use the SDB for most non-RECFM U data sets.
- ▶ For RECFM U data sets, use BLKSIZE=32760.
- ▶ For UADS data sets for TSO/E, use the same block size you currently use to allocate a new one. Do not use SDB, as this will result in very poor DASD space utilization.
- ▶ You should not use the SDB for font libraries too. The correct block size for the font libraries is 12288.

For more information on the usage of SDB, see *z/OS DFSMS: Using Data Sets*, SC26-7410.

## 1.31 System data sets

|                                         |                                        |
|-----------------------------------------|----------------------------------------|
| <input type="checkbox"/> Master Catalog | <input type="checkbox"/> SYS1.LPALIB   |
| <input type="checkbox"/> IODF           | <input type="checkbox"/> SYS1.MACLIB   |
| <input type="checkbox"/> SYSn.IPLPARM   | <input type="checkbox"/> SYS1.MIGLIB   |
| <input type="checkbox"/> SYS1.BROADCAST | <input type="checkbox"/> SYS1.MODGEN   |
| <input type="checkbox"/> SYS1.CMDLIB    | <input type="checkbox"/> SYS1.NUCLEUS  |
| <input type="checkbox"/> SYS1.CSSLIB    | <input type="checkbox"/> SYS1.PARMLIB  |
| <input type="checkbox"/> SYS1.DUMPnn    | <input type="checkbox"/> SYS1.PROCLIB  |
| <input type="checkbox"/> SYS1.HELP      | <input type="checkbox"/> SYS1.SAMPLIB  |
| <input type="checkbox"/> SYS1.IMAGELIB  | <input type="checkbox"/> SYS1.SVCLIB   |
| <input type="checkbox"/> SYS1.LINKLIB   | <input type="checkbox"/> SYS1.UADS     |
| <input type="checkbox"/> LOGREC         | <input type="checkbox"/> SYS1.STGINDEX |
|                                         | <input type="checkbox"/> SYS1.VTAMLIB  |

Figure 1-31 System data sets

### System data sets

Before you install z/OS, you must select and define the system data sets that you need.

Before you include components from the distribution libraries (DLIBs) and user-defined data sets in the system, use JCL and access method services to define the system data sets.

Figure 1-31 lists some of the common system data sets. For a complete list of system data sets, see *z/OS MVS System Data Set Definition*, SA22-7629.

## 1.32 System administration

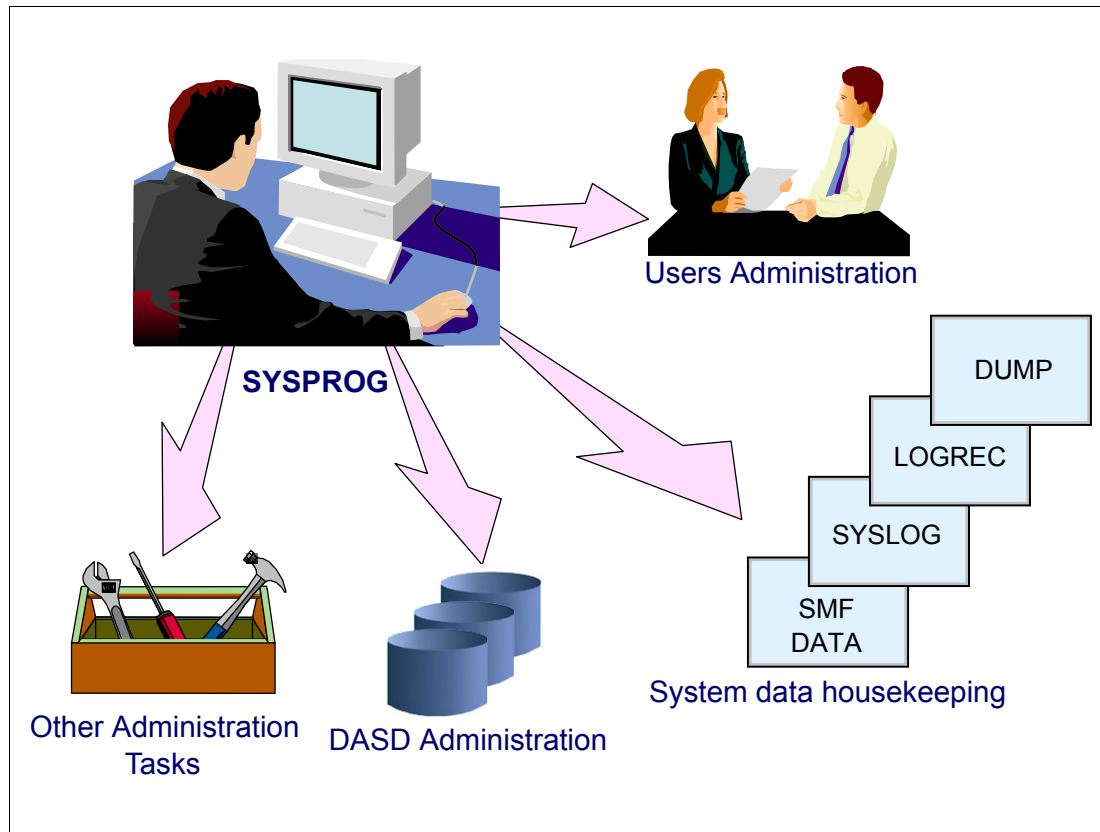


Figure 1-32 System administration tasks

### System administration tasks

Once you have the system up and running, most of your time is spent maintaining the system and providing technical support to the users. The remainder of this chapter describes some of the administrative tasks performed by a system programmer, specifically:

- ▶ User administration
- ▶ DASD administration
  - Adding a new DASD volume
  - Implementing DFSMS
  - Handling DASD problems
- ▶ System data housekeeping
  - SMF recording
    - SMFPRMxx parmlib member - Dumping SMF data
  - LOGREC data - SYSLOG data
- ▶ Other administration tasks
  - Working with MIH
  - Adding page data sets
  - Changing TSO timeout
  - Adding spool volumes
  - Deleting spool volumes
  - Verifying system configuration
  - Viewing SYSOUT using ISPF
  - Changing your TSO profile
  - Backing up and restoring z/OS

## 1.33 User administration

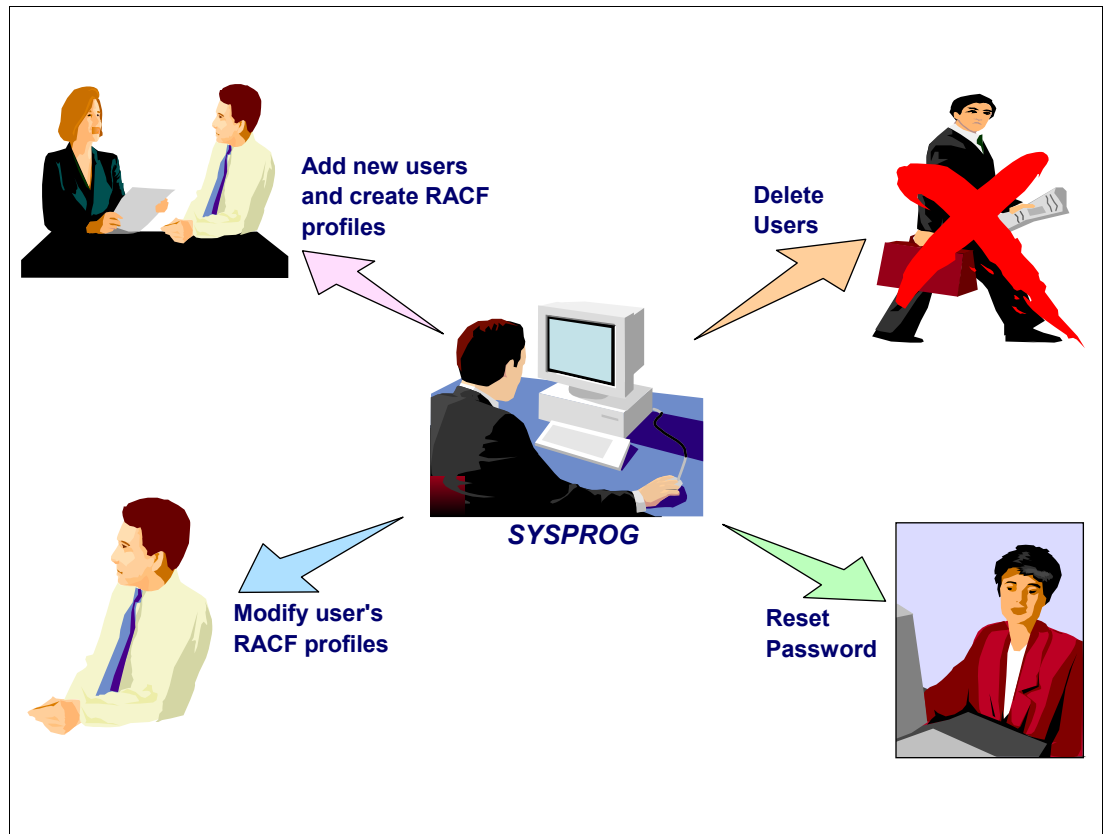


Figure 1-33 User administration tasks

### User administration

A system is never complete without users using it. Most of the time, you will find yourself having to perform certain user administration tasks. The next few sections provide examples of typical processes used when adding, deleting, or administering users.

Some of the common user administrative tasks are:

- ▶ Adding new users
- ▶ Creating RACF profiles for new users
- ▶ Modifying RACF profiles for existing users
- ▶ Resetting passwords
- ▶ Deleting users and their RACF profiles

## 1.34 DASD administration

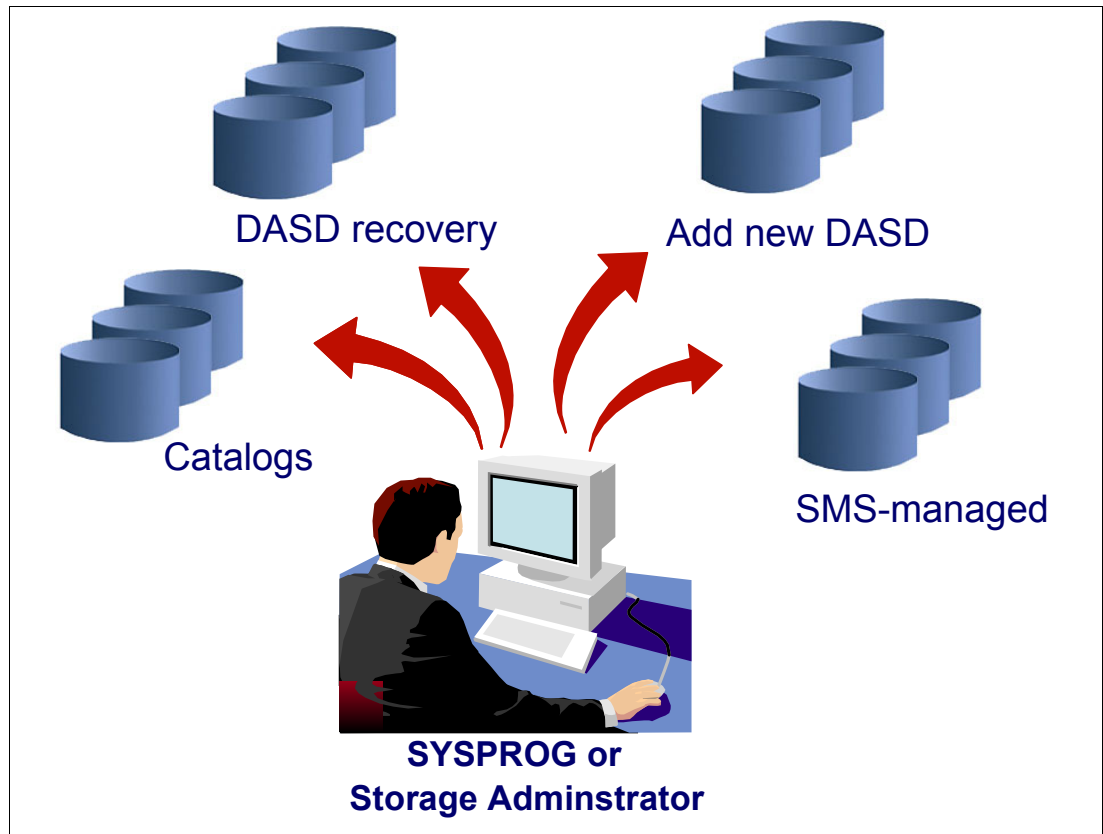


Figure 1-34 DASD administration tasks

### DASD administration

On a large installation, you might have a team of storage administrators to take care of all DASD administration tasks, and you as a system programmer may only be required to help with obscure problems and exit routines. In that environment, you may never have to deal with the hardware side of DASD management, other than modifying IODF and IOCDS using HCD. On the other hand, your installation may be of a smaller scale that doesn't warrant having a dedicated team of storage administrators, so you may have to take on far more of these roles.

### DASD management

As a system programmer taking on the additional responsibility of a storage administrator, you might have to handle these aspects of DASD management:

- ▶ Adding new DASD devices and volumes to your system
- ▶ Implementing SMS
- ▶ Effectively managing catalogs
- ▶ Dealing with DASD problems

## 1.35 Adding a new DASD volume

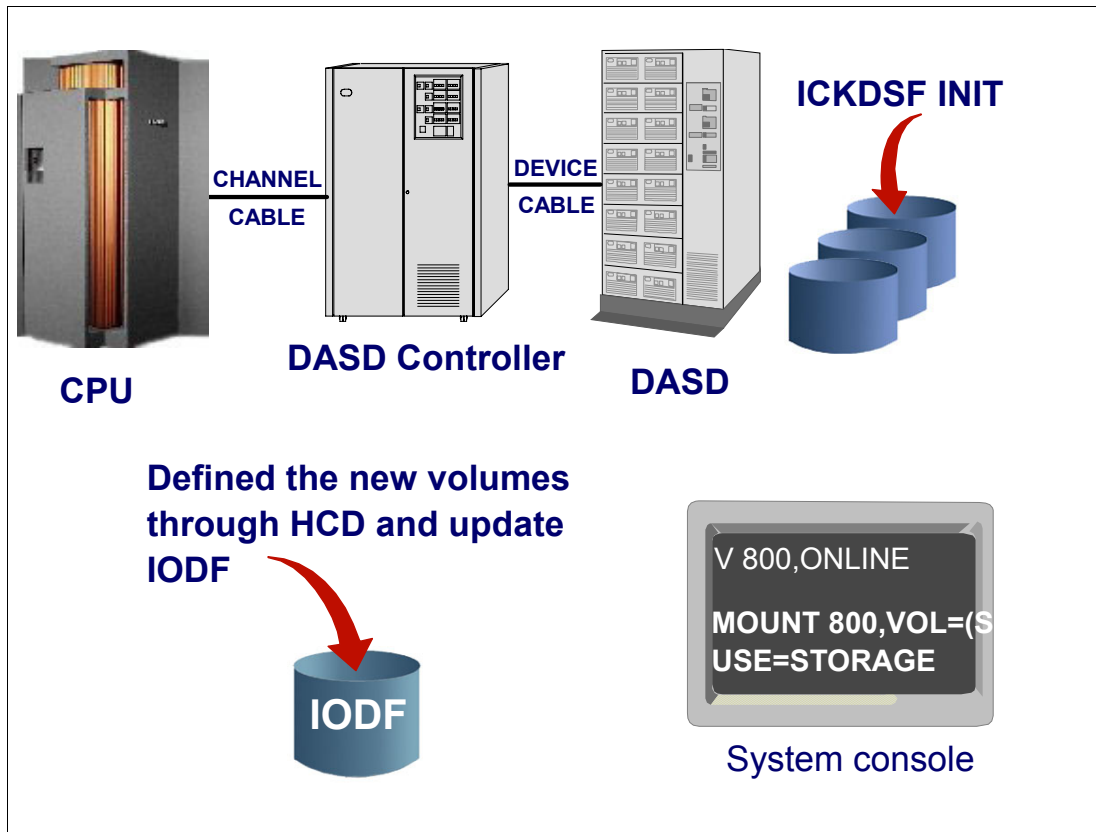


Figure 1-35 Adding a new DASD volume

### How to add additional DASD volumes

The following list outlines the steps for making a new DASD volume available to your system:

1. Physically connect the new device to the storage controller, which is connected to the system through an available channel. Normally, this task is performed by an IBM Customer Engineer who takes care of all the hardware installation.
2. Using HCD, update the IODF and IOCDS to include the new devices. You might want to define the device as belonging to one or more eligible device tables defined in your current IODF, so that data sets can be allocated on it using the UNIT parameter with the associated esoteric device name. For more information on how to use HCD, see *z/OS Hardware Configuration Definition User's Guide*, SC33-7988.
3. Using ICKDSF, initialize the new volume with a volume serial label, a volume table of contents (VTOC), and a VTOC index. For more information on ICKDSF, see *Device Support Facilities User's Guide and Reference Release 17*, GC35-0033.
4. Vary the device online, and issue an appropriate **MOUNT** command, or re-IPL MVS to cause the new volume to be mounted according to the VATLST entries.

A disk volume (except a DFSMS-managed volume) is mounted with one of the following use attributes:

– PRIVATE

New data sets will be created on this volume only if the user (by using JCL or the TSO **ALLOCATE** command or an ISPF menu specification) specifies the volume serial number of this disk volume.

– PUBLIC

MVS may place a temporary data set on this volume if the user (by using JCL or otherwise) did not specify a volume serial number for the temporary data set. Data sets may also be placed on this volume by specifying the volume serial number, as with the PRIVATE volumes. A temporary data set is one with a DSNNAME beginning with an ampersand or with a disposition equivalent to NEW,DELETE. For more information about the DSNNAME parameter in JCL, see *z/OS MVS JCL Reference*, SA22-7597.

– STORAGE

MVS places permanent data sets on this volume if the user did not supply a volume serial number for the data sets. In addition, temporary data sets, and data sets placed by volume serial number, may also be placed on this volume.

You can set the volume use attributes by:

- Using the **MOUNT** operator command
- Using the VATLSTxx parmlib member

For more information on the **MOUNT** command, see *z/OS MVS System Commands*, SA22-7627.

The VATLSTxx parmlib member defines the default mount and use attributes for disk volumes found during IPL. The mount attribute determines the conditions under which a volume can be unmounted, while the use attribute controls the type of request for which a volume can be allocated.

**Example of VATLSTxx: VATDEF IPLUSE(PRIVATE),SYSUSE(PRIVATE)**

MPCAT1,1,0,3390,Y

MPRES1,1,2,3390,Y

MPRES2,1,2,3390,Y

This example specifies the use attribute. STORAGE is denoted by 0, PUBLIC by 1, and PRIVATE by 2.

For more information on the VATLSTxx parmlib member, see *z/OS MVS Initialization and Tuning Reference*, SA22-7592.



## 1.36 Implementing DFSMS

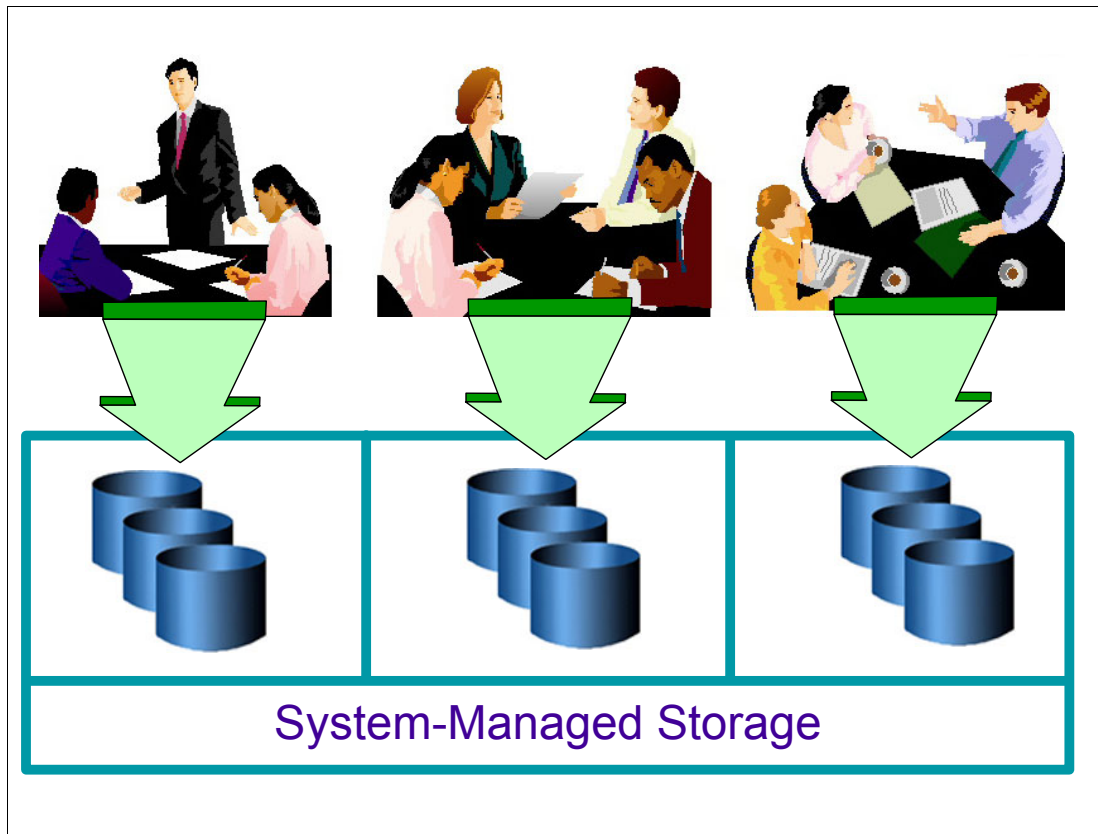


Figure 1-36 Implementing DFSMS - system-managed storage

### Implementing System Managed Storage (SMS)

After you have added the new volumes to the system, you should not make all the volumes available to the users and allow them to use them as they wish. With no restriction on which volumes they can use or how much space they can allocate, your new disk volumes will be filled up fairly quickly. You could spend most of your time providing a new supply of disk volumes for other essential data, tracking down the owner of each data set to establish their importance, and trying to negotiate for space reduction. One way to solve this problem is to allocate individual volumes to each application, usage type, or group of users. Each of these volumes are mounted PRIVATE and allocation can be done by specifying the volume serial number. However, this method of control has some disadvantages:

- ▶ The requirement of specifying a volume serial number in JCL during allocation can lead to inflexibility; for example, a space allocation on one volume might fail even though there is a lot of space on other volumes available to the user.
- ▶ It is time-consuming and difficult to change the volumes available to the user because this requires changing the hard-coded volume serial number in the JCL.

To avoid these problems, implement DFSMS. It uses software to help automate the management of storage, data security, data set placement, data set migration and recall, data set backup, and data set recovery and deletion. This ensures that current data is available when needed and obsolete data is removed from storage, providing the following benefits:

- ▶ Simplified data allocation
- ▶ Ensured data integrity on new allocation
- ▶ Improved allocation control

## 1.37 Handling DASD problems

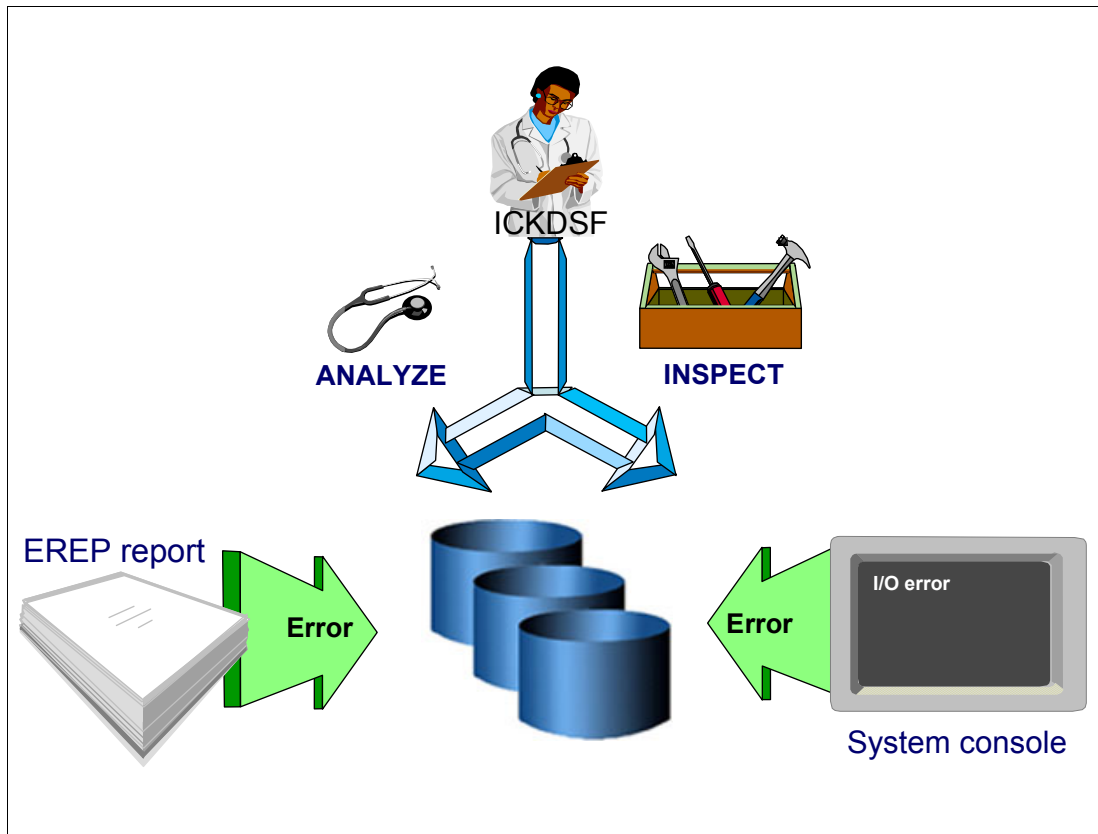


Figure 1-37 Handling DASD problems

### How to deal with DASD problems

The most common DASD problem you will encounter in your role as a system programmer is an I/O error.

The normal error handling processing by the storage subsystem (that is, the storage control and its attached storage devices) and by the operating system, z/OS, includes various functions that recognize errors and recover from them whenever possible. The storage subsystem performs the following functions:

- ▶ Adds ECC (error checking and correction) information to each field of a record when written.
- ▶ Detects errors in reading data, in performing control operations, in functioning of the hardware, and in programming.
- ▶ Retries I/O operations for certain error conditions.
- ▶ Assembles usage and error information in the form of sense information.
- ▶ Maintains counts of disk storage errors.
- ▶ Does ECC correction activity or sends the ECC data to the operating system for correction.

The operating system, MVS provides standard error recovery procedures to handle errors detected by the storage subsystem. For example, MVS:

- ▶ **Implements recovery actions** - The specific recovery actions depend on the particular error condition that was defined in the sense information sent from the storage subsystem;

for example, retrying an operation when an equipment check is reported in the sense information.

- **Logs usage and error information records** - The error data that is sent to the system is stored in the Error Recording Data set (ERDS), that is SYS1.LOGREC. The system processes the sense information to produce and supplement data records describing the conditions under which the error occurred. The EREP program formats error reports and may also perform error analysis based on information it obtains from the SYS1.LOGREC.
- **Issues system messages at the operator console** - System console messages may be the initial notification of hardware or media problem. Most information messages contain data on the type and location of an error, and give sense information in hexadecimal format.

## EREP reports

The errors shown on the EREP report should be followed up, particularly as temporary errors tend to lead to permanent errors. Normally, you will call an IBM Customer Engineer to look into the problem, or your hardware engineer if it turns out to be a hardware error. However, when the problem is caused by a disk media failure, you may have to perform some form of media recovery actions using ICKDSF.

## ANALYZE command

You can use the **ANALYZE** command to detect and differentiate recording surface and drive-related problems on a volume. It can also scan data to help detect possible media problems. Sample JCL for the **ANALYZE** command follows:

```
//ICKDSF1 JOB (),'MVSSP',NOTIFY=&SYSUID,CLASS=A,MSGLEVEL=(1,1),
//          MSGCLASS=X
//ANALYZE EXEC PGM=ICKDSF
//VOLUME DD UNIT=3390,VOL=SER=MPDLB2,DISP=SHR
//SYSPRINT DD SYSOUT=*
//SYSIN DD *
          ANALYZE DDNAME(VOLUME) SCAN DRIVETEST
```

**Note:** When diagnosing media problems, you should always use the **ANALYZE** command with the **DRIVETEST** operand. This will ensure that the device hardware can perform basic operations, such as seek, reads, and writes.

When the **ANALYZE** report shows that there are potential media errors, you can use the **INSPECT** command to check the surface of a track to determine if there is a defect, then flag the defective track and assign an alternate track. The **PRESERVE** operand allows you to save the data on the inspected tracks to the assigned alternate tracks. Sample JCL for the **INSPECT** command follows:

```
//ICKDSF2 JOB (),'MVSSP',NOTIFY=&SYSUID,CLASS=A,MSGLEVEL=(1,1),
//          MSGCLASS=X
//INSPECT EXEC PGM=ICKDSF
//VOLUME DD UNIT=3390,VOL=SER=MPDLB2,DISP=SHR
//SYSPRINT DD SYSOUT=*
//SYSIN DD *
          INSPECT DDNAME(VOLUME) ASSIGN CHECK PRESERVE
```

For more information on **ANALYZE** and **INSPECT**, and other ICKDSF commands, see *Device Support Facilities User's Guide and Reference Release 17*, GC35-0033.

## 1.38 System data housekeeping

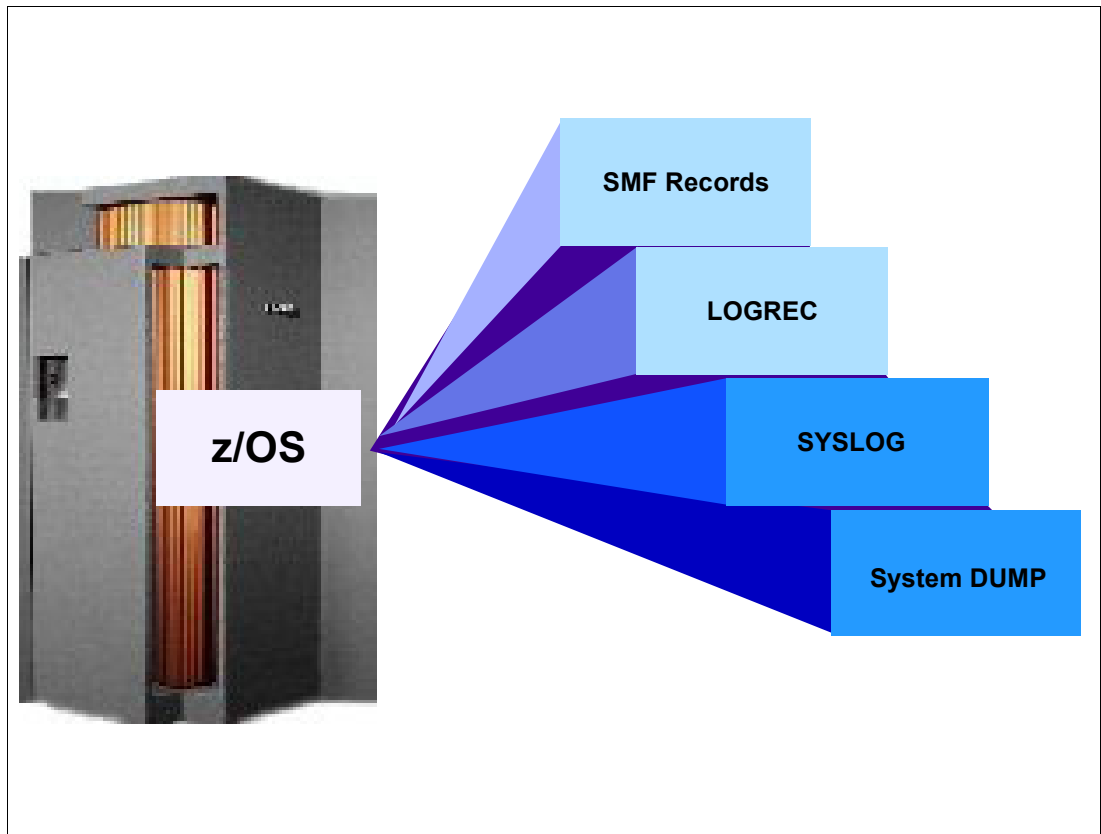


Figure 1-38 System data housekeeping

### System data housekeeping

To help you manage your system properly, MVS generates lots of useful data of various types and stores them in system data sets. There are four types of system data which you will have to deal with:

- ▶ **SMF records** - The SMF records contain a variety of information that enables you to produce many types of analysis reports and summary reports so that you can evaluate changes in configuration, 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.
- ▶ **LOGREC data** - The LOGREC data set contains statistical data about machine failures (processor failures, I/O device errors, channel errors). It also contains records for program error recording, missing interrupt information, and dynamic device reconfiguration (DDR) routines.
- ▶ **SYSLOG data** - This data set resides in JES2's spool space. It can be used by application and system programmers to record communications about problem programs and system functions. It also contains a record of console messages and operator commands for audit and diagnosis purposes.
- ▶ **System DUMP data sets** - These are sequential data sets which contain system dumps that record areas of virtual storage in case of system task failures.

## 1.39 SMF data

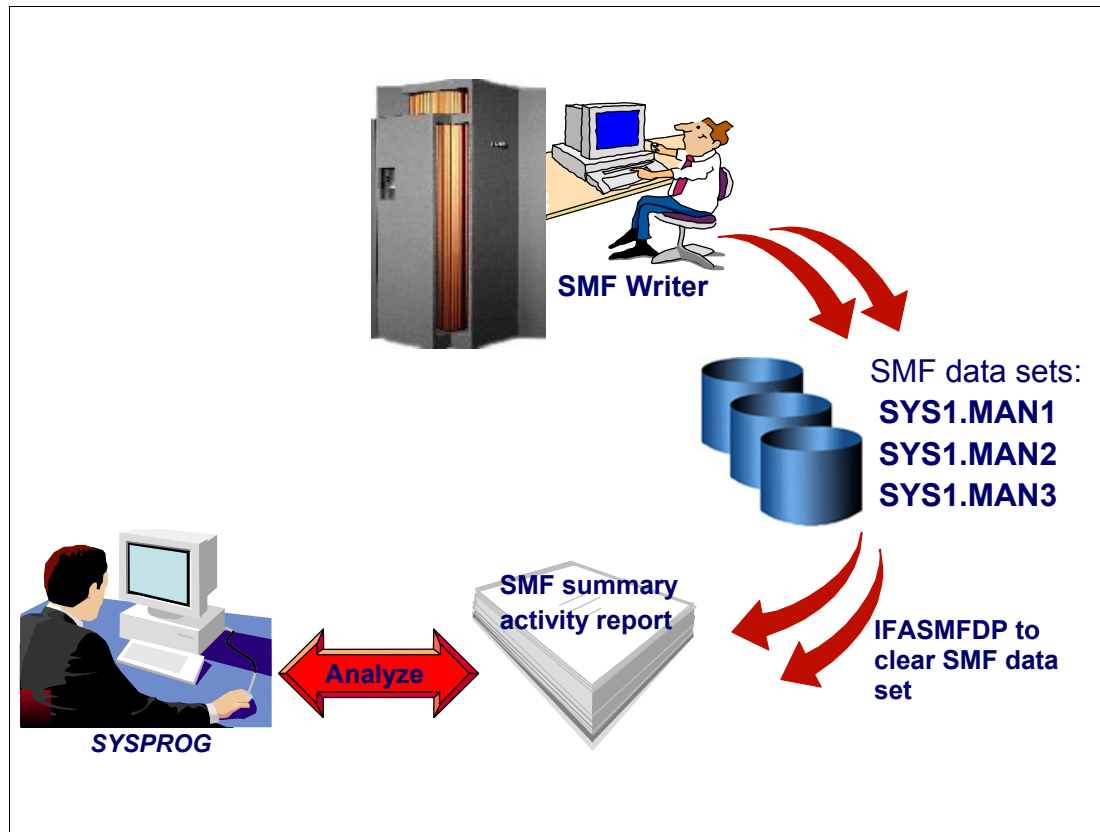


Figure 1-39 Collecting SMF data

### How to collect SMF data

To record SMF data, you have to allocate direct access space and catalog the SMF data sets. You should allocate at least two data sets for SMF use (with at least one on a high-performance device), as follows:

1. A high-performance device is needed because, if the I/O rate is too slow, data will have to be buffered. The buffers will eventually fill up, which could result in lost data.
2. Consider the following factors when determining which device type to specify when allocating SMF data sets:
  - Your system configuration
  - The amount of SMF data to be written
  - The size of SMF buffers (the control interval size)
  - Your installation's report program requirement

The naming convention used for SMF data sets is SYS1.MANx, such as SYS1.MAN1, SYS1.MAN2, SYS1.MAN3, and so forth. You can use the sample JCL, as follows, to allocate new SMF data sets.

```

//DEFINE JOB (),'MVSSP',NOTIFY=&SYSUID,CLASS=A,MSGLEVEL=(1,1),
//      MSGCLASS=X
//STEP1 EXEC PGM=IDCAMS
//SYSPRINT DD SYSOUT=*
//SYSIN DD *
DEFINE CLUSTER( -
    CONTROLINTERVALSIZE(4096) -
    CYLINDERS(40) -
    NAME(SYS1.MAN1) -
    NONINDEXED -
    RECORDSIZE(4086,32767) -
    REUSE -
    SHAREOPTIONS(2) -
    SPANNED -
    SPEED -
    VOLUME(MTCAT1) ) -
DATA( -
    NAME(SYS1.MAN1.DATA) )
/*

```

## 1.40 SMFPRMxx parmlib member

```
ACTIVE                                /*ACTIVE SMF RECORDING*/
DSNAME (SYS1.MAN1 ,                  /*SMF DATA SET NAMES*/
      SYS1.MAN2 ,                    /*SMF DATA SET NAMES*/
      SYS1.MAN3)                     /*SMF DATA SET NAMES*/
NOPROMPT                             /*DON'T PROMPT THE OPERATOR*/
REC (PERM)                           /*TYPE 17 PERM RECORDS ONLY*/
INTVAL (10)                          /* INTERVAL RECORDING EVERY 10 MIN */
MAXDORM (3000)                       /* WRITE AN IDLE BUFFER AFTER 30 MIN*/
STATUS (010000)                     /* WRITE SMF STATS AFTER 1 HOUR*/
JWT (2400)                           /* 522 AFTER 24 HOURS */
SID (&SYSNAME (1:4))                /* SYSTEM ID IS &SYSNAME */
LISTDSN                             /* LIST DATA SET STATUS AT IPL*/
LASTDS (MSG)                        /*DEFAULT TO MESSAGE */
NOBUFFS (MSG)                       /*DEFAULT TO MESSAGE */
SYS (TYPE (0,6,7,15,17,21,26,30,43,45,61,64,65,66,70:79,80,81,110) ,
     EXITS (IEFU83,IEFU84,IEFACTRT,IEFUJV,
           IEFUSI,IEFUJI,IEFUTL,IEFU29) , NOINTERVAL, NODETAIL)
```

Figure 1-40 Example of an SMFPRMxx parmlib member

### SMFPRMxx parmlib member

Once you have allocated all the required SMF data sets, you have to update the SMFPRMxx in SYS1.PARMLIB to reflect the new SMF data sets. In order to enable SMF recording, make sure that the first parameter in SMFPRMxx is set to ACTIVE, as shown in Figure 1-40.

To activate the new changes to SMF parameters, you can use the **SET** command to make them effective (assuming that you have added SYS1.MAN3 to SMFPRM00) as shown in the following example of issuing the **SET** command. (The notes referenced in this code are found on the following page.)

```
SET SMF=00
IEF196I IEF237I 0800 ALLOCATED TO IEFPARM
IEE252I MEMBER SMFPRM00 FOUND IN SYS1.PARMLIB
IEF196I IEF237I 0802 ALLOCATED TO SYS00004
IEE966I SYS1.MAN3 IS BEING FORMATTED 1
IEF196I IEF285I   SYS1.PARMLIB                                KEPT
IEF196I IEF285I   VOL SER NOS= MPRES1.
IEE949I 03.01.03 SMF DATA SETS 097
          NAME      VOLSER SIZE(BLKS) %FULL  STATUS
 2 P-SYS1.MAN1 MPCAT1      7200    9  ACTIVE
    S-SYS1.MAN2 MPCAT1      1800    0  ALTERNATE
    S-SYS1.MAN3 MPCAT1      1800    0  ALTERNATE
IEE536I SMF      VALUE 00 NOW IN EFFECT
```

**1** SMF found that SYS1.MAN3 has not been formatted. The formatting is now taking place.

**2** The name of the SMF data set preceded by a *P* indicates that it is the primary SMF data set. If the name is preceded by an *S*, the data set is a secondary SMF data set.

## Types of SMF records

You can control which SMF record types are recorded by using the SYS parameter in the SMFPRMxx parmlib member. You can specify SYS(TYPE(0:255)) to record all SMF records. However, this can create considerable overhead and fill up the SMF data sets very quickly. The default SYS parameter is *SYS(NOTYPE(14:19,62:69,99))*. This collects all SMF records except the following:

- ▶ Record type 14 is written for input non-VSAM direct access, tape data sets, or VIO data sets.
- ▶ Record type 15 is written for output non-VSAM direct access, or VIO data sets.
- ▶ Record type 16 is written to record information about events and operations of the sorting program.
- ▶ Record type 17 is written when a non-temporary data set or temporary data set is scratch.
- ▶ Record type 18 is written when a non-VSAM data set is renamed.
- ▶ Record type 19 is written for a list of online devices with specific IPL time frame.
- ▶ Record types 62 to 69 are written for various VSAM data sets and catalog activities.
- ▶ Record type 99 is written by the System Resource Manager (SRM) component when running in goal mode.

For more information about the rest of SMF record types, see *z/OS MVS System Management Facilities (SMF)*, SA22-7630.



## 1.41 Dumping SMF data

```
//SMFCLR JOB (),'MVSSP',NOTIFY=&SYSUID,CLASS=A,MSGLEVEL=(1,1),
//      MSGCLASS=X
//STEP1 EXEC PGM=IFASMFDP,REGION=4M
//DUMPIN DD DSN=SYS1.MAN1,DISP=SHR
//DUMPOUT DD DISP=(NEW,CATLG),DSN=SMF.ALLRECS,
//          UNIT=SYSALLDA,VOL=SER=MPDLB2,SPACE=(CYL,(30,10),RLSE),
//          DCB=(RECFM=VBS,LRECL=32760,BLKSIZE=4096)
//SYSPRINT DD SYSOUT=*
//SYSIN DD *
//          INDD(DUMPIN,OPTIONS(ALL))
/*
```

Figure 1-41 Dumping SMF data JCL

### Dumping SMF data

When the current recording data set cannot accommodate any more records, the SMF writer routine automatically switches recording from the active SMF data set to an empty SMF data set and issues the following message regarding the full data set:

```
*IEE362A SMF ENTER DUMP FOR SYS1.MAN1 ON MPCAT1
```

You can use the sample JCL shown in the visual to dump the SMF data to a user data set and clear the SMF data set for reuse.

**OPTIONS(ALL)** indicates that the input data set specified in the DUMPIN DD statement is to be copied, then reset and preformatted.

**Note:** The output data set in the DUMPOUT DD statement should be large enough to hold one SMF data set; the minimum requirement is 30 cylinders.

Alternatively, you may code an IEFU29 exit to submit the job. The IEFU29 exit is invoked (if activated in SMFPRMxx) whenever an SMF data set switch occurs. For more information about the IEFU29 dump exit, see *z/OS MVS Installation Exits*, SA22-7593.

## 1.42 LOGREC data

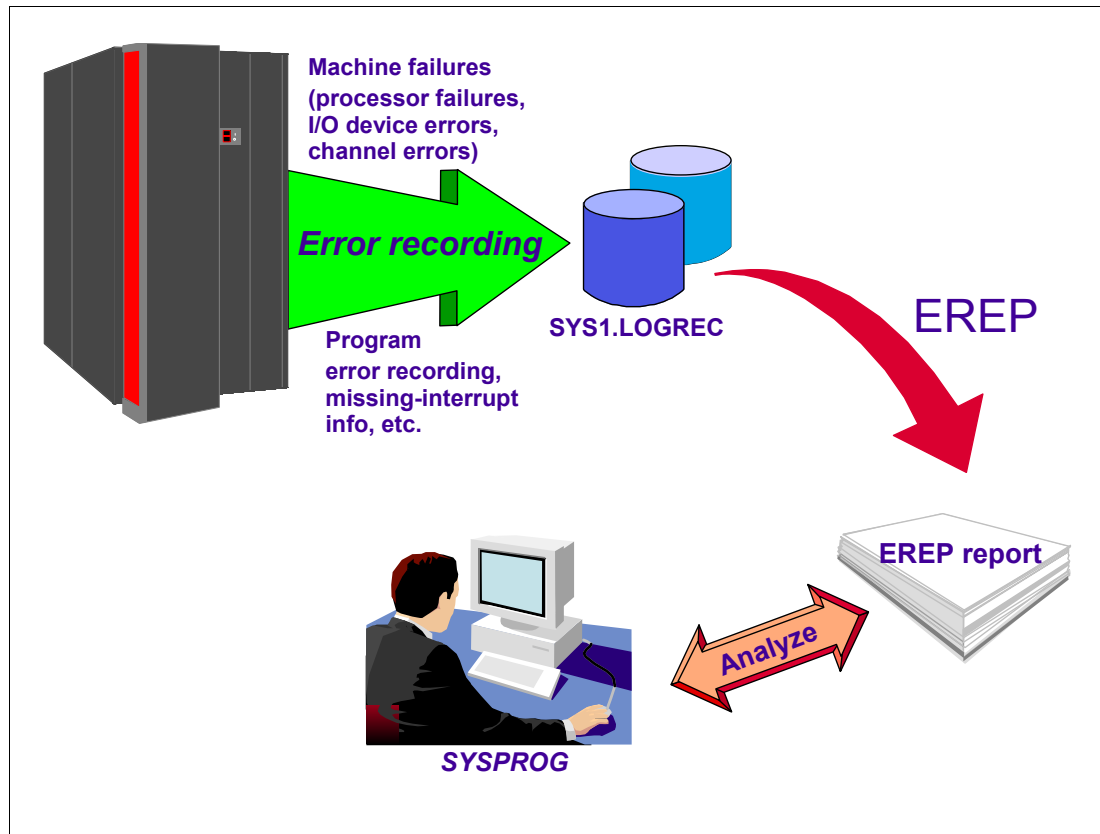


Figure 1-42 Overview of logrec processing

### How to create SYS1.LOGREC

When an error occurs, the system records information about the error in the LOGREC data set. The information provides you with a history of all hardware failures, selected software errors, and selected system conditions. You can use the Environment Record, Editing, and Printing (EREP) program to:

- ▶ 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.

**Note:** Whenever you allocate or reallocate the logrec data set, the newly allocated data set will not be used until you initialize it and IPL the system on which it is to be used

You can use the following sample JCL to create and initialize a new logrec data set:

```
//LOGREC JOB (),'MVSSP',NOTIFY=&SYSUID,CLASS=A,MSGLEVEL=(1,1),
//      MSGCLASS=X
//*****
//* CREATE A NEW LOGREC DATASET AND INITIALIZE IT
//*****
//STEP1 EXEC PGM=IFCDIP00,COND=(4000,LT)
//SERERDS DD UNIT=SYSDA,VOL=SER=MPRES1,SPACE=(CYL,3,,CONTIG),
//      DSN=SYS1.LOGREC,DISP=(,CATLG)
```

**Note:** The IBM-supplied default name for the logrec data set is SYS1.LOGREC. To change the logrec data set name known to the system, use the LOGREC parameter of the IEASYSxx parmlib member.

If you use the default name SYS1.LOGREC in a multisystem environment, take care to ensure that the logrec data set is uniquely specified with the volume-serial. This will prevent the initialization of another system's logrec data set in the event that IFCDIP00 is run on a different system.

## How to clear SYS1.LOGREC

The logrec data set will eventually become full, the same as the SMF data set, and console messages appear requesting the operator to take action. Therefore, this data set needs to be cleared so that it can be reused. The process is a little different because there is only one logrec data set.

As a system programmer, you have to set up a process to clear the logrec data set when the following message appears:

```
IFB080E LOGREC DATA SET NEW FULL, DSN=SYS1.LOGREC
```

This is issued when the logrec data set becomes 90 percent full; if action is not taken and logrec fills up completely, the following message, which requires prompt action, is issued:

```
IFB081I LOGREC DATA SET IS FULL, hh.mm.ss, DSN=SYS1.LOGREC
```

At this stage, if you do not take any action, the system continues processing, but further error records will be lost. You can use the following sample JCL to clear the logrec data set:

```
//CLRLOG JOB (),'MVSSP',NOTIFY=&SYSUID,CLASS=A,MSGLEVEL=(1,1),
//      MSGCLASS=X
//STEP EXEC PGM=IFCEREPI,PARM='CARD'
//SERLOG DD DSN=SYS1.LOGREC,DISP=OLD
//ACCDEV DD DSN=SYS1.LOGREC.TEMPSTOR,DISP=MOD
//TOURIST DD SYSOUT=*
//EREPT DD SYSOUT=*
//SYSIN DD DUMMY
        SYSUM
        ACC=Y
        ZERO=Y
```

The job will copy the logrec data set to the data set specified in the DDNAME ACCDEV, print a brief summary of logrec data, and then clear the data set. For more information about the rest of the parameters, see *Environmental Record Editing and Printing Program (EREP) Reference*, GC35-0152.

## 1.43 SYSLOG data

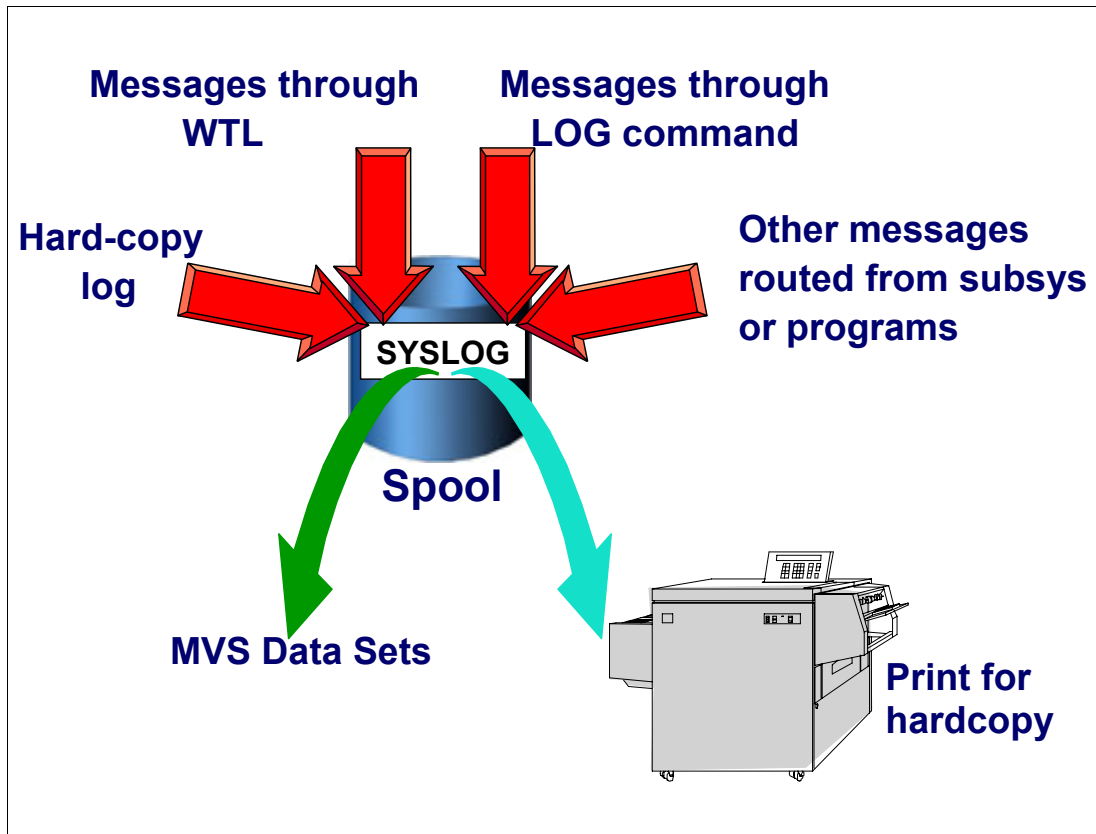


Figure 1-43 SYSLOG data

### The SYSLOG

The SYSLOG on the MVS system actually consists of two separate logs: the system log and the hardcopy log. The hardcopy log is a record of all system message traffic, specifically:

- Messages to and from all consoles
- Commands and replies entered by the operator

In a JES3 system, the hardcopy log is always written to the SYSLOG. While in a JES2 system, you can choose to have the hardcopy log written to the SYSLOG or to a console printer. You can do this by specifying the **HARDCOPY** statement in the **CONSOLxx** parmlib member as shown in the **HARDCOPY** statement in the **CONSOLxx** member of parmlib in the following example:

```
HARDCOPY DEVNUM(SYSLOG,OPERLOG) 1
ROUTCODE(ALL)
CMDLEVEL(CMDS)
UD(Y)
```

**1** SYSLOG indicates that the system log is to be the hardcopy medium, while OPERLOG indicates that the operations log will be activated and will receive the hardcopy message set.

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

**Note:** The system defaults to SYSLOG as the hardcopy medium in any of the following cases:

- ▶ You do not code a HARDCOPY statement.
- ▶ You specify an unusable hardcopy console.
- ▶ You specify OPERLOG alone but the operations log is unusable.

The SYSLOG is a SYSOUT data set provided by the job entry subsystem (either JES2 or JES3). SYSOUT data sets are output spool data sets on direct access storage devices (DASD). An installation should print the SYSLOG periodically to check for problems. The SYSLOG consists of the following:

- ▶ All messages issued through Write To Log (WTL) macros.
- ▶ All messages entered by the LOG operator commands.
- ▶ Usually, the hardcopy log.
- ▶ Any messages routed to the SYSLOG from any system component or program.

You can limit the maximum number of WTLs (messages) 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 processing by JES. When this value is reached, the log processing:

- ▶ 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.

**Example of LOGLMT and LOGCLS in IEASYSxx:** LOGCLS=L, LOGLMT=99999,

In the example, the SYSLOG is scheduled for SYSOUT processing on output class L when 99,999 WTLs have been issued. For more information on LOGLMT and LOGCLS, see *z/OS MVS Initialization and Tuning Reference*, SA22-7592.

Remember that the SYSLOG 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, you should set up a procedure to organize the SYSLOG data set depending on your installation's requirement.

You can perform your housekeeping as follows:

1. Set up a procedure to have the system operator issue the **WRITELOG** command to close and allocate a new SYSLOG at the end of each processing day. For example, **WRITELOG H**.

**Note:** Class H is not a default printer class, so the SYSLOG data set is kept in the spool.

2. At the end of the week, you will have seven SYSLOG data sets in the spool. You can now decide to offload the SYSLOG to an MVS data set or route it to a printer for a hardcopy printout.
3. It's really not necessary to keep SYSLOG data sets for more than a week, so if you keep a week's worth of SYSLOG data sets, you should not have a problem.

## 1.44 Other administration tasks

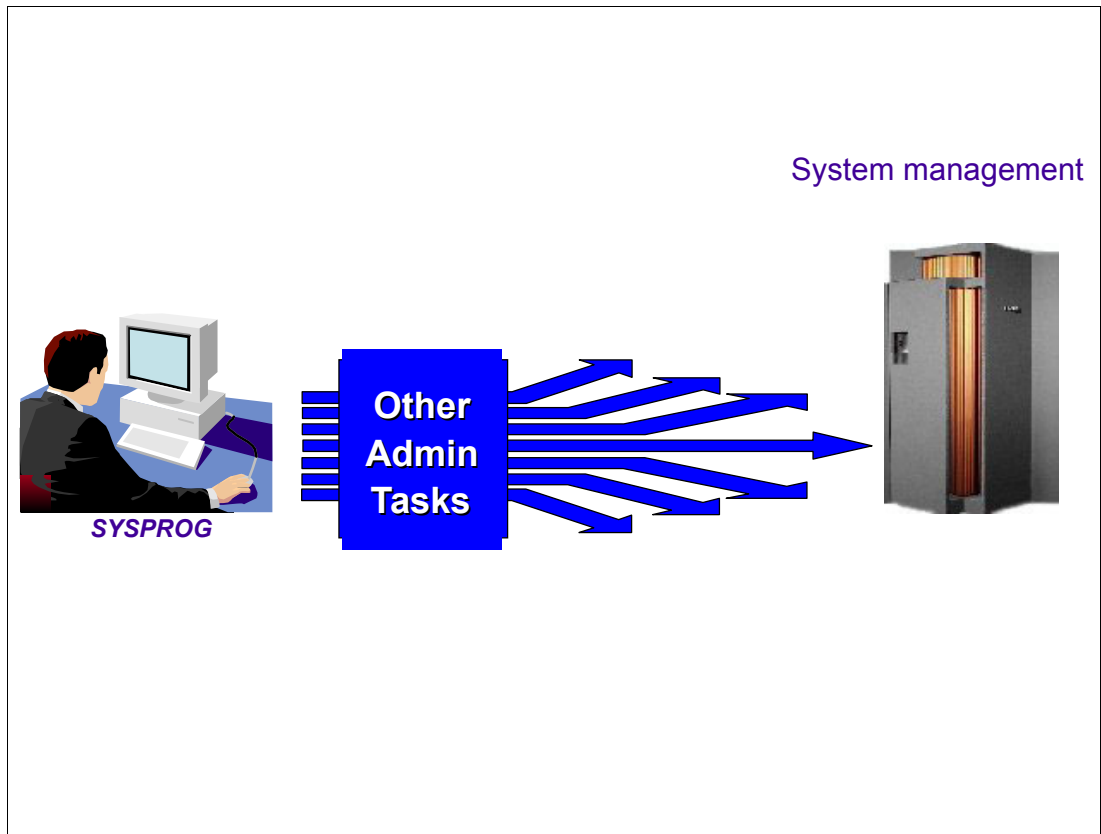


Figure 1-44 Other administration tasks for the sysprog

### Other administration tasks

In the remainder of this chapter we discuss some other administrative tasks and techniques. Specifically, we show you how to:

- ▶ Work with MIH
- ▶ Add page data sets
- ▶ Change TSO timeout
- ▶ Add spool volumes
- ▶ Delete spool volumes
- ▶ Verify system configuration
- ▶ View SYSOUT using ISPF
- ▶ Change your TSO profile
- ▶ Back up and restore z/OS

## 1.45 Working with missing interrupt handler

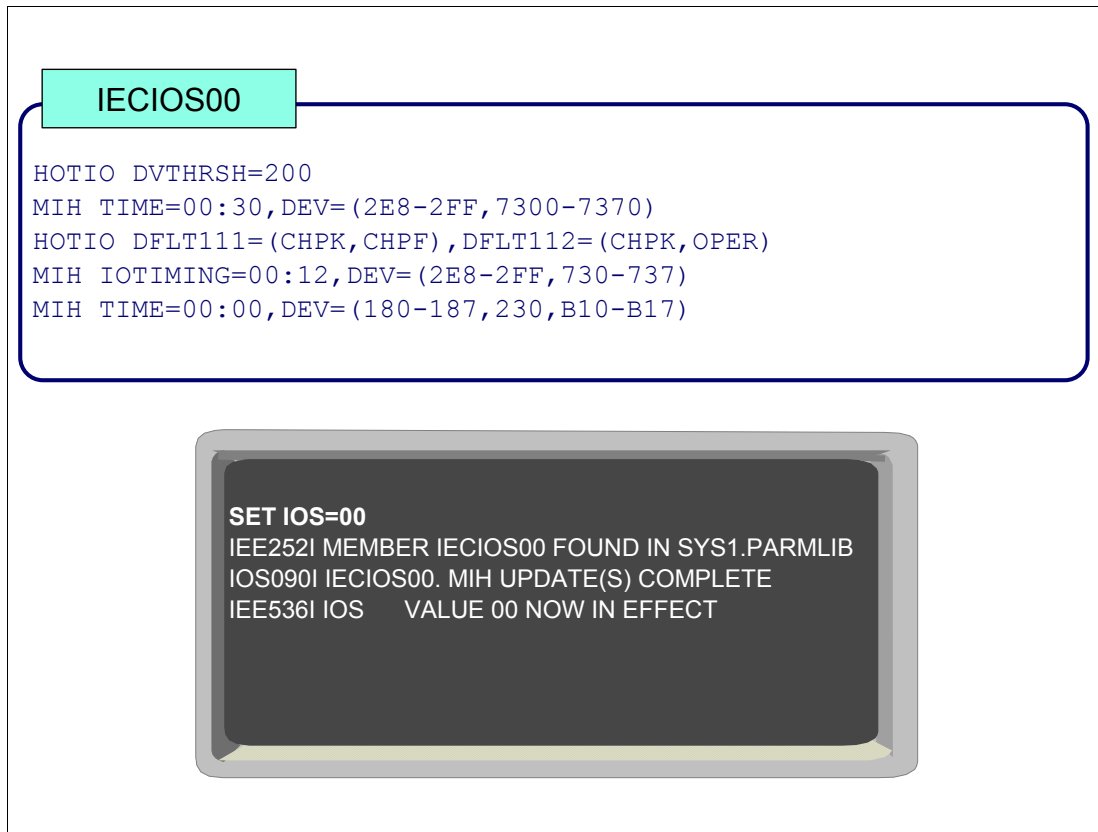


Figure 1-45 Specifying MIH parameters in parmlib

### How to work with missing interrupt handler (MIH)

When z/OS sends channel commands to a control unit, it waits for a response. If no response is received after a certain amount of time, a missing interrupt situation exists. The following conditions qualify as missing interrupts if the specified time interval has elapsed:

- ▶ Primary status interrupt pending
- ▶ Secondary status interrupt pending
- ▶ Start pending condition
- ▶ Idle with work queued
- ▶ Mount pending

You can use the IECIOSxx parmlib member to specify the timing limits for devices in your system. When the value that you specify for the I/O timing limit is greater than the value that you specify for MIH, normal MIH recovery will be in effect until the I/O timing limit is reached. Once this limit is reached, the system abnormally ends the I/O request.

Hot I/O refers to a hardware malfunction that causes repeated, unsolicited I/O interrupts. If those I/O interrupts are not deleted, the system can loop or the System Queue Area (SQA) can be depleted. The I/O subsystem (IOS) tries to detect the hot I/O condition and performs recovery actions before the system requires an IPL. Recovery actions taken for a hot device depend on the type of device and its reserve status. You can use the HOTIO statement in IECIOSxx, as shown in the following example, to modify the Hot I/O Detection Table (HIDT),

and to eliminate operator intervention where recovery actions defined in HIDT (by default) require the operator to respond to a message.

```
HOTIO DVTHRS=200
MIH TIME=00:30,DEV=(2E8-2FF,7300-7370)
HOTIO DFLT111=(CHPK,CHPF),DFLT112=(CHPK,OPER)
MIH IOTIMING=00:12,DEV=(2E8-2FF,730-737)
MIH TIME=00:00,DEV=(180-187,230,B10-B17)
```

Once you have updated the IECIOSxx parmlib member, you can activate the changes using the **SET** command, as shown in the following example:

```
SET IOS=00
IEE252I MEMBER IECIOS00 FOUND IN SYS1.PARMLIB
IOS090I IECIOS00. MIH UPDATE(S) COMPLETE
IEE536I IOS      VALUE 00 NOW IN EFFECT
```

For more information on the IECIOSxx parmlib member, see *z/OS MVS Initialization and Tuning Reference*, SA22-7592.



## 1.46 Adding a page data set

```
PAGEADD PAGE=PAGE.MHQPROD.LOCAL3
IEE783I PAGEADD COMMAND- 531
PAGE.MHQPROD.LOCAL3 PAGE DATA SET
NOW AVAILABLE FOR SYTEM USE

D ASM
IEE200I 02.28.37 DISPLAY ASM 539
TYPE      FULL  STAT  DEV  DATASET NAME
PLPA      67%   OK    0802  PAGE.MHQPROD.PLPA
COMMON   13%   OK    0802  PAGE.MHQPROD.COMMON
LOCAL    25%   OK    0802  PAGE.MHQPROD.LOCAL1
LOCAL    23%   OK    0802  PAGE.MHQPROD.LOCAL2
LOCAL     3%   OK    0802  PAGE.MHQPROD.LOCAL3
NO SWAP DATASETS ARE IN USE
PAGEDEL COMMAND IS NOT ACTIVE
```

Figure 1-46 Command to add a page data set

### How to add additional page data sets

z/OS can support a large amount of virtual storage by using paging data sets. Data sets named page spaces are used to contain this virtual storage. Another term often used to collectively describe these page spaces is auxiliary storage, which is managed by Auxiliary Storage Manager (ASM).

As your workload increases, you may run out of available auxiliary storage. When the system detects a shortage of available slots in the auxiliary storage paging space, meaning that 70 percent of available slots are already in use, it issues the following message:

```
IRA200E AUXILIARY STORAGE SHORTAGE
```

The system rejects **LOGON**, **MOUNT**, and **START** commands until the shortage is relieved.

If no action is taken, the system issues the following message when 90 percent of all available auxiliary storage is in use:

```
IRA201E CRITICAL AUXILIARY STORAGE SHORTAGE
```

The solution to this problem is to increase the auxiliary storage available for z/OS use by defining more page space. You can use the sample JCL that follows to define a new page data set.

```
//DEFPAGE JOB (),'MVSSP',NOTIFY=&SYSUID,CLASS=A,MSGLEVEL=(1,1),
//          MSGCLASS=X
//STEP1 EXEC PGM=IDCAMS,COND=(4000,LT)
//SYSPRINT DD SYSOUT=*
//PAGELOC DD UNIT=3390,VOL=SER=MPCAT1,DISP=OLD
//SYSIN DD *
        DEFINE PAGESPACE( -
                FILE(PAGELOC) -
                NAME(PAGE.MHQPROD.LOCAL3) -
                CYLINDERS(200) -
                VOLUME(MPCAT1) )
```

After you have successfully defined a new page data set, you can relieve the auxiliary shortage by adding the new page data set to the system as shown in the following example:

```
PAGEADD PAGE=PAGE.MHQPROD.LOCAL3
IEE783I PAGEADD COMMAND- 531
PAGE.MHQPROD.LOCAL3 PAGE DATA SET
NOW AVAILABLE FOR SYSTEM USE
```

After you have added the new page data set to the system, the system issues the following message:

```
IRA202I AUXILIARY STORAGE SHORTAGE RELIEVED
```

You can check the status of the auxiliary storage by issuing the **D ASM** command as follows:

```
D ASM
IEE200I 02.28.37 DISPLAY ASM 539
TYPE      FULL STAT  DEV  DATASET NAME
PLPA      67%   OK   0802  PAGE.MHQPROD.PLPA
COMMON    13%   OK   0802  PAGE.MHQPROD.COMMON
LOCAL     25%   OK   0802  PAGE.MHQPROD.LOCAL1
LOCAL     23%   OK   0802  PAGE.MHQPROD.LOCAL2
LOCAL      3%   OK   0802  PAGE.MHQPROD.LOCAL3
NO SWAP DATASETS ARE IN USE
```

To ensure this new page data set remains a permanent part of your z/OS system, you should update the **PAGE=** parameter in your IEASYSxx parmlib member to include the new page data set, as shown in the following example:

```
PAGE=(PAGE.MHQPROD.PLPA,
        PAGE.MHQPROD.COMMON,
        PAGE.MHQPROD.LOCAL1,
        PAGE.MHQPROD.LOCAL2,
        PAGE.MHQPROD.LOCAL3,L),
```

## 1.47 Changing TSO timeout

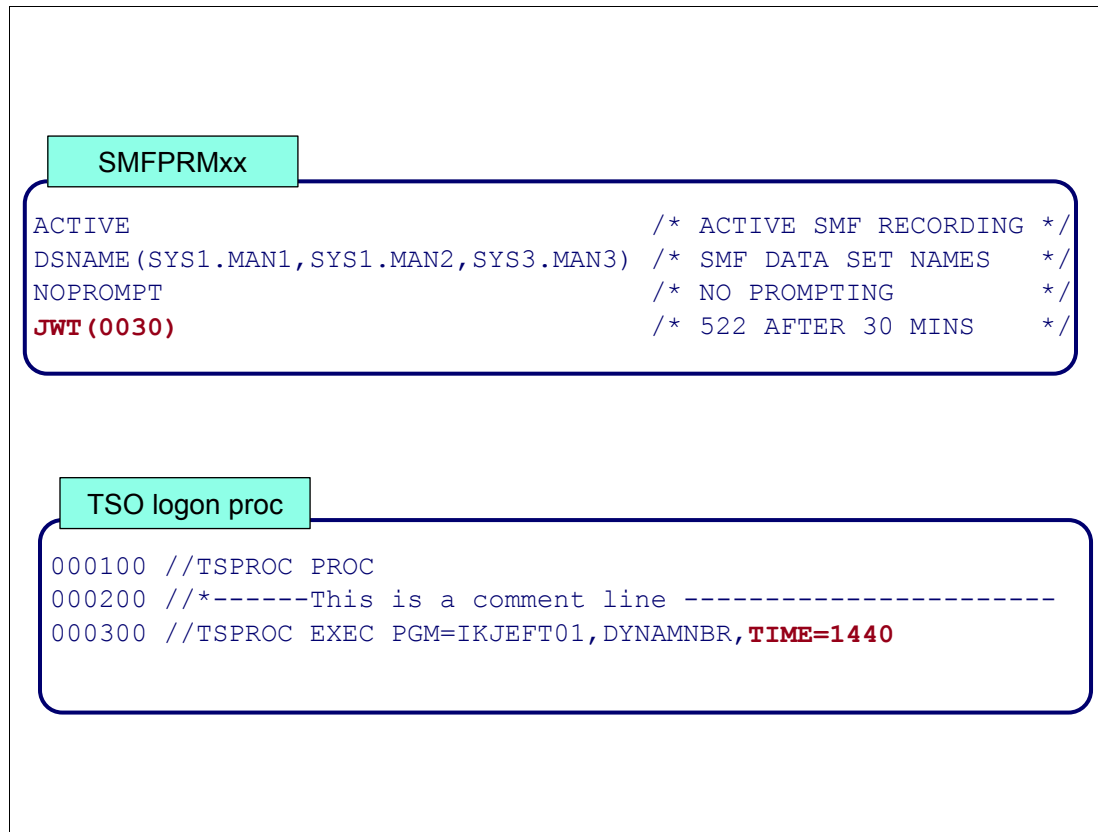
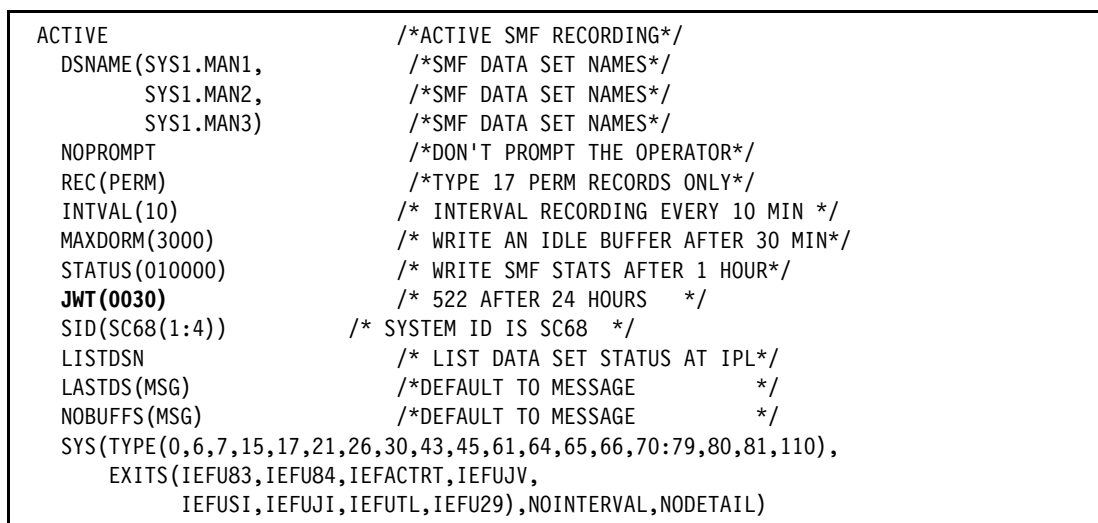


Figure 1-47 Changing the TSO timeout value

### How to change the TSO timeout value

After a period of inactivity, TSO will automatically log a user off. This elapsed time is set in the SMFPRMxx parmlib member using the parameter JWT(hhmm), as shown in the following example where *hh* is the amount of real time in hours and *mm* is in minutes.



In the example, a user is forced off after 30 minutes of no activity. You can edit this PARMLIB member and change this value. You can set this new value by using the **SET** operator command:

```
SET SMF=00
```

**Note:** There is a possibility that TSO users are not being timed out when the value specified in the JWT parameter has expired. This usually happens when TIME=1440 is specified in the TSO logon procedure that the users used to log on to the system.

```
//TSPROC PROC  
//TSPROC EXEC PGM=IKJEFT01,DYNAMNBR=256,TIME=1440
```

The TIME=1440 is a special value that will:

- ▶ Allow the particular job to run for an unlimited amount of time
- ▶ Ignore the JWT parameter in SMFPRMxx parmlib member for the job

## 1.48 Adding spool volumes in JES2

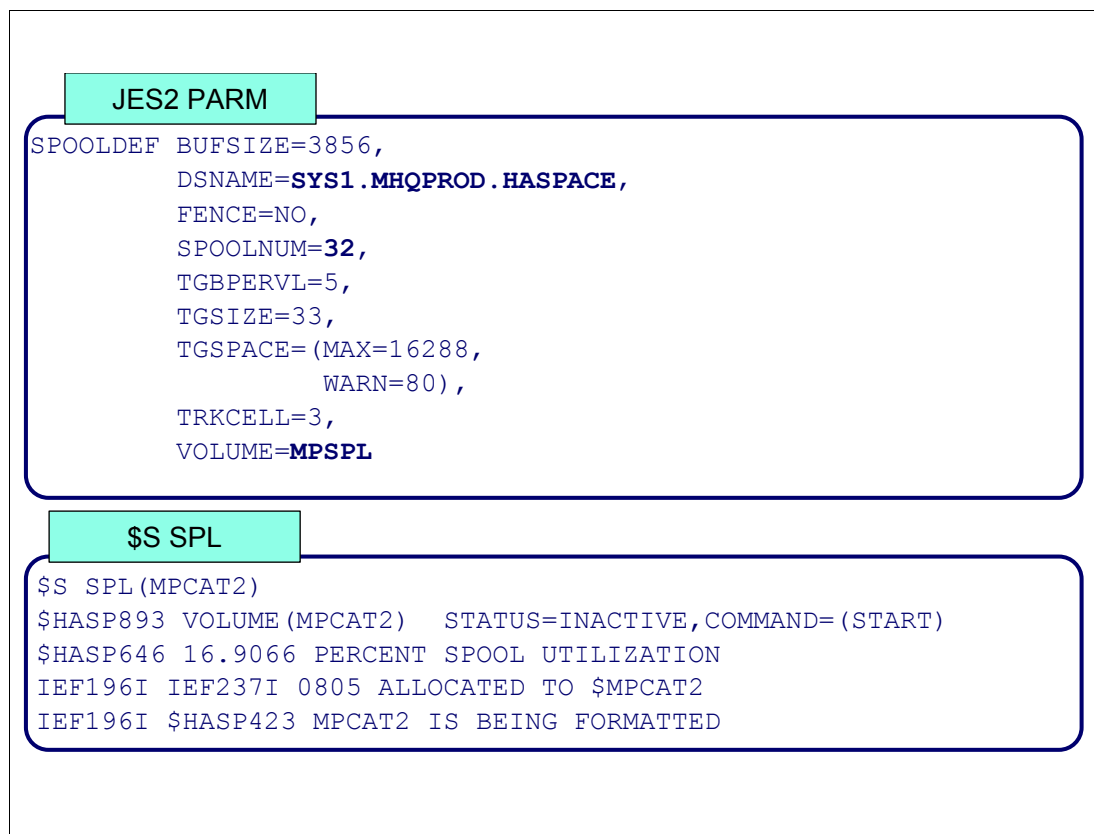


Figure 1-48 Adding spool volumes in a JES2 system

### How to add spool volumes

The spool is the repository for all input jobs and most system out (SYSOUT) that JES2 manages. Due to fluctuating workload requirements, such as I/O errors, hardware failures, or utilization needs, you might need to add spool packs.

Spool volumes are added either at cold start or dynamically through the use of the operator command **\$\$ SPL**. You can define a maximum of 253 spool volumes to JES2 at any one time.

To define the spool volumes during JES2 startup, use the **SPOOLDEF** statement in the **JES2 PARM**. The **SPOOLDEF** statement that follows is an example of **SPOOLDEF** statement in the **JES2 PARM**. (Notes referenced in the code are on the following page.)

```
SPOOLDEF BUFSIZE=3856, /* MAXIMUM BUFFER SIZE &BUFSIZE c*/
          DSNAME=SYS1.MHQPROD.HASPACE, 1
          FENCE=NO, /* Don't Force to Min.Vol. &FENCE oc*/
          SPOOLNUM=32 2, /* Max. Num. Spool Vols c*/
          TGBPERVL=5, /* Track Groups per volume in BLOB ownc*/
          TGSPACE=(MAX=16288, /* Fits TGMs into 4K Page &TGSPACE wnc*/
                  WARN=80), /* &NUMTG=(,% onc*/
          TRKCELL=3, /* 3 Buffers/Track-cell &TCELSIZ c*/
          VOLUME=MPSPL 3 /* SPOOL VOLUME SERIAL &SPOOL c*/
```

**1** The JES2 spool data set name.

**2** This specifies the maximum number of spool volumes which can be defined at any one time.

**3** Specifies the four- to five-character prefix assigned to JES2's spool volumes. The first four to five characters of the volume serial must be identical to the character specified by this parameter. Those volumes beginning with this prefix and a data set named the same as the DSNNAME specification are considered JES2 volumes.

**Note:** If the maximum of 253 spool volumes is exceeded during a cold start, JES2 issues a message informing the operator that more spool volumes were found than expected from the SPOOLNUM parameter on the SPOOLDEF initialization statement.

As mentioned, you can also add spool volumes dynamically by using operator command, **\$S SPL** as shown in the following example:

```
$S SPL(MPCAT2)  
$HASP893 VOLUME(MPCAT2) STATUS=INACTIVE,COMMAND=(START)  
$HASP646 16.9066 PERCENT SPOOL UTILIZATION  
IEF196I IEF237I 0805 ALLOCATED TO $MPCAT2  
IEF196I $HASP423 MPCAT2 IS BEING FORMATTED  
$HASP423 MPCAT2 IS BEING FORMATTED  
$HASP850 3750 TRACK GROUPS ON MPCAT1  
$HASP850 586 TRACK GROUPS ON MPCAT2  
$HASP851 28232 TOTAL TRACK GROUPS MAY BE ADDED  
$HASP630 VOLUME MPCAT2 ACTIVE 0 PERCENT UTILIZATION
```

**Note:** If the dynamic addition of another spool volume is attempted (thereby exceeding the maximum of 253 spool volumes), the command will be ignored, the volume will not be added, and the message \$HASP411 will be issued to the operator.

You can now display the status and the percentage utilization of all spool volumes using the operator command, **\$D SPL** as shown in the following example:

```
$DSPL  
$HASP893 VOLUME(MPCAT1) STATUS=ACTIVE,PERCENT=16  
$HASP893 VOLUME(MPCAT2) STATUS=ACTIVE,PERCENT=0  
$HASP646 14.7370 PERCENT SPOOL UTILIZATION
```

## 1.49 Deletion of spool volumes in JES2

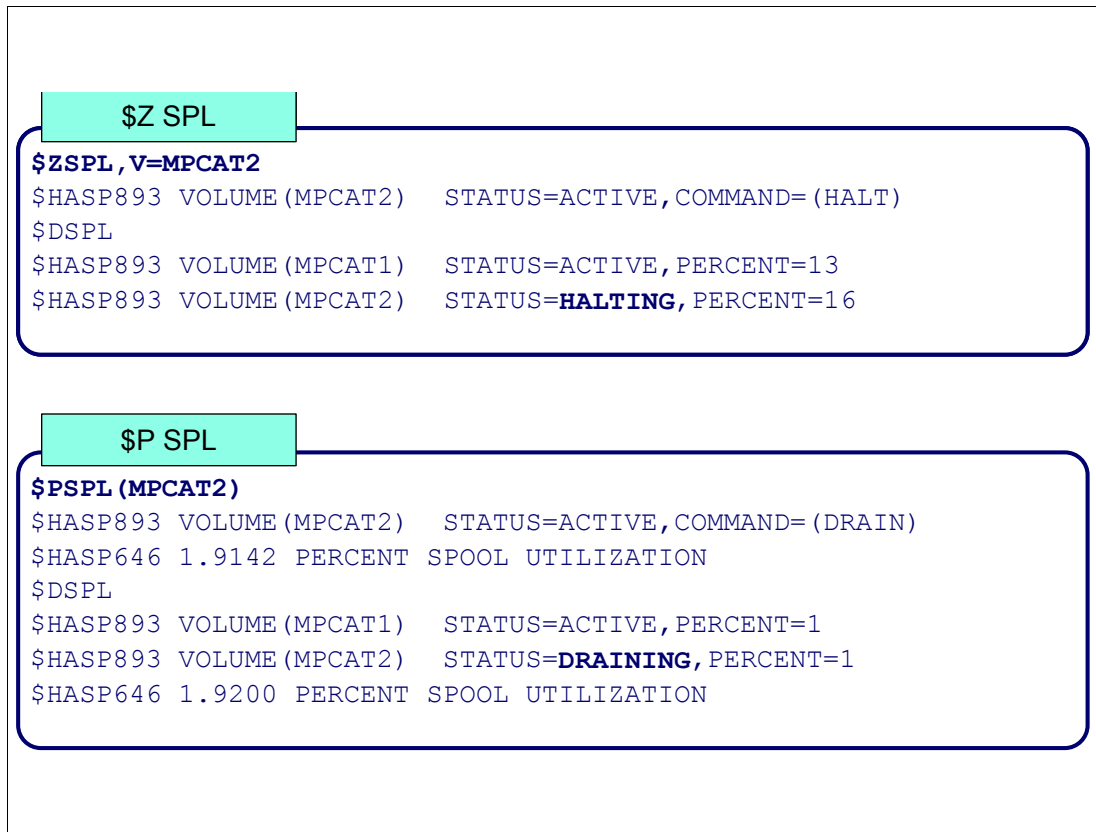


Figure 1-49 Deleting spool volumes in a JES2 system

### How to delete spool volumes

Spool volumes can be deleted dynamically as needed to meet your installation's requirements, using the operator commands:

- ▶ **\$Z SPL**
- ▶ **\$P SPL**

Some considerations must be recognized and precautions must be taken to prevent improper usage of the spool volume deletion commands, specifically:

- ▶ If the volume being deleted (halted or drained) contains spooled or remote messages or the JESNEWS data set, these items are automatically moved to another volume.

**Note:** This process may take some time because spooled remote messages and SYSOUT are only moved from the draining volume when they become the lowest-numbered job on the volume.

- ▶ If there are no active volumes available for these data sets, the deletion process will not complete until an active volume is available to accept them.

You use the **\$Z SPL** command to halt an active spool volume. When you issue the **\$Z** command, no new work is allowed to start and no new space on the volume is allocated. Any currently allocated space remains intact and the volume can be removed without the loss of work.

The volume status is HALTING until all currently active jobs which have space allocated on the volume are completed. After that the status of the volume changes to INACTIVE. System response to the **\$Z** command is shown in the following example:

```
$ZSPL,V=MPCAT2
$HASP893 VOLUME(MPCAT2) STATUS=ACTIVE 1,COMMAND=(HALT)
$DSPL 2
$HASP893 VOLUME(MPCAT1) STATUS=ACTIVE,PERCENT=13
$HASP893 VOLUME(MPCAT2) STATUS=HALTING 3,PERCENT=16
$HASP646 13.7066 PERCENT SPOOL UTILIZATION
.
.
.
$HASP395 LSTDDF ENDED
$HASP309 INIT B INACTIVE ***** C=GA
IEF196I IEF285I SYS1.MHQPROD.HASPACE KEPT
IEF196I IEF285I VOL SER NOS= MPCAT2.
$HASP630 VOLUME MPCAT2 INACTIVE 4 16 PERCENT UTILIZATION
```

- 1 The initial status of the spool volume, MPCAT2 was ACTIVE.
- 2 To display the status of spool volumes, you used the **\$DSPL** command.
- 3 After the **\$Z** command was issued, the status changed to HALTING.
- 4 When the job (LSTDDF) ends, the status of the spool volume (MPCAT2) changed to INACTIVE.

To drain and delete an entire spool volume, you can use the **\$P SPL** operator command. The command prevents any available tracks on the draining spool volume from being selected for allocation. Until all jobs currently allocated to the volume have completed all phases of job processing, the volume is considered to have a status of draining. The spool volume is drained when no allocated spool space remains on that volume, and the volume is unallocated to all systems in the multi-access spool complex.

**Note:** Once the spool volume is drained, JES2 does not retain any information concerning that volume. Therefore, you cannot display any information about the volume.

The following example shows the system response when you issue the **\$P SPL** command when an active job is running:

```
$PSPL(MPCAT2)
$HASP893 VOLUME(MPCAT2) STATUS=ACTIVE 1,COMMAND=(DRAIN)
$HASP646 1.9142 PERCENT SPOOL UTILIZATION
$DSPL
$HASP893 VOLUME(MPCAT1) STATUS=ACTIVE,PERCENT=1
$HASP893 VOLUME(MPCAT2) STATUS=DRAINING 2,PERCENT=1
$HASP646 1.9200 PERCENT SPOOL UTILIZATION
```

- 1 The initial status of the spool volume to be drained.
- 2 After the **\$PSPL(MPCAT2)** command was issued, the status of the spool volume (MPCAT2) changed to DRAINING.

If you want to cancel and deallocate all cancellable jobs on the drained spool volume, you can use the CANCEL operand together with the **\$P SPL** command. All jobs and SYSOUT will be



deleted from the volume; any spooled remote messages and the JESNEWS data set (if present) are automatically moved to another active spool volume.

**Note:** The **\$P SPL,CANCEL** command can also be used to drain an inactive volume, purging all jobs allocated to it. However, this can result in lost track groups on other volumes. You can recover any lost space by doing an all-member warm start, or the JES2 automatic spool reclamation function will also recover them within one week, whichever occurs first. You can prevent the loss of spool space by using the **\$S SPL,P,CANCEL** command only if the spool volume can be remounted.

An example of the **\$SPL,P,CANCEL** command is shown, as follows:

```
$$$SPL(MPCAT2),P,CANCEL 1
$HASP893 VOLUME(MPCAT2) STATUS=DRAINING,COMMAND=(START,DRAIN)
$HASP646 7.1200 PERCENT SPOOL UTILIZATION
      .
      .
      .
IEF196I IEF285I   SYS1.MHQPROD.HASPACE                                KEPT
IEF196I IEF285I   VOL SER NOS= MPCAT2.
$HASP806 VOLUME MPCAT2 DRAINED 2
```

**1** The command will fail with the message: REQUEST INVALID DUE TO ACTIVE STATUS if the spool volume is not in the DRAINING status.

**2** At the successful completion of the command the status of the spool changed to DRAINED, which indicates you can no longer display any information about the volume.

**Note:** If all the spool volumes are deleted, JES2 will not be capable of performing any work. Hence, be very careful when you delete any spool volumes.

For more information on the use and syntax of the spool configuration commands, see *z/OS JES2 Commands*, SA22-7526.

## 1.50 Verifying the system configuration

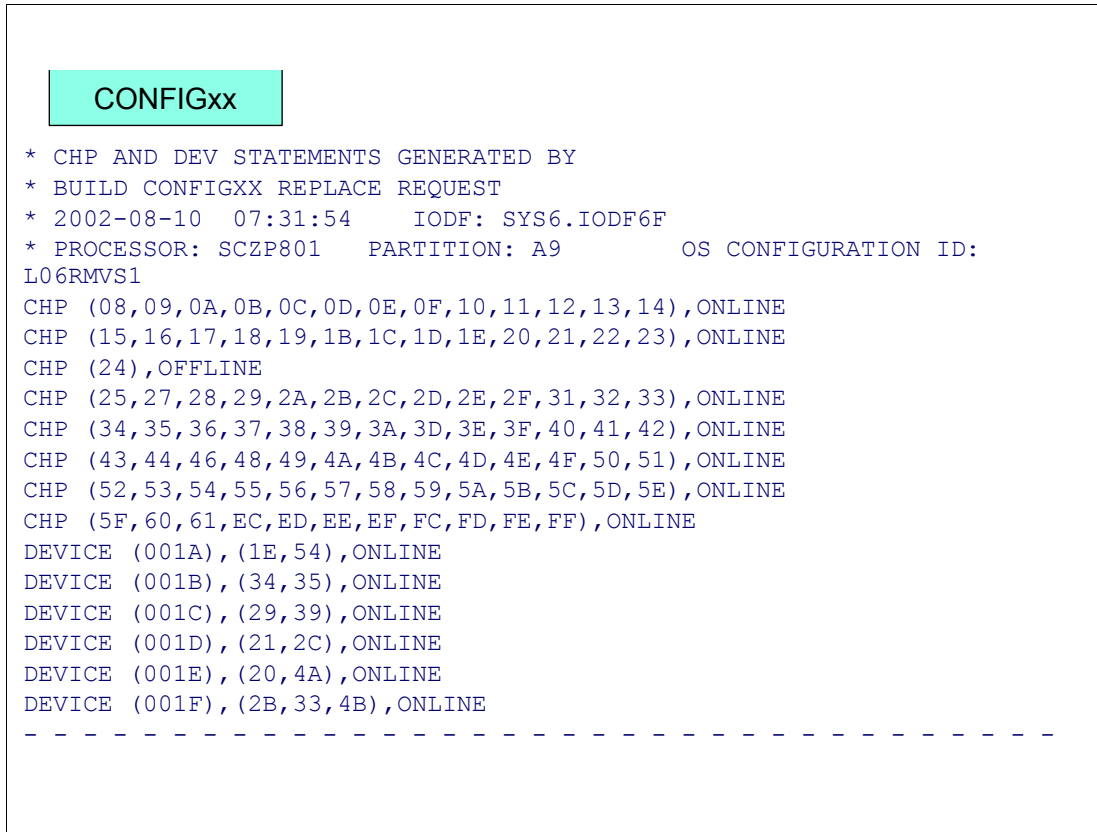


Figure 1-50 Verifying the system configuration

### How to verify the system configuration

When you start an HCD session, you need to specify the IODF that HCD is to use. Before you can activate your configuration, you must build a production IODF. The IODF data sets must be cataloged so that you can use them with HCD.

The CONFIGxx parmlib member is a list of records that an installation can use to define a standard configuration of system elements. The system elements include the processors, the expanded storage, vector facilities, storage, channel paths, devices, and volumes. You can use the configuration defined in CONFIGxx in two ways:

- ▶ To compare the differences between the current configuration and the standard configuration as defined in a CONFIGxx member. To do that, use the **DISPLAY M=CONFIG(xx)** operator command.
- ▶ To reconfigure some of the system elements by using the **CONFIG** operator command with the **MEMBER** option.

You can create a CONFIGxx parmlib member manually, as described in the *z/OS MVS Initialization and Tuning Reference*, SA22-7592; or through the *Build CONFIGxx member* option using Hardware Configuration Definition (HCD).

## 1.51 HCD primary panel

```
z/OS V1.4 HCD
Command ==> _____
(C) Copyright IBM Corp. 1990, 2003. All rights reserved.
      Hardware Configuration

Select one of the following.

2_  1. Define, modify, or view configuration data
    2. Activate or process configuration data
    3. Print or compare configuration data
    4. Create or view graphical configuration report
    5. Migrate configuration data
    6. Maintain I/O definition files
    7. Query supported hardware and installed UIMs
    8. Getting started with this dialog
    9. What's new in this release

For options 1 to 5, specify the name of the IODF to be used.

I/O definition file . . . 'SYS1.IODF00.WORK' +
```

Figure 1-51 Select Option 2 to activate or process configuration data

### HCD primary panel

To build the CONFIGxx using HCD, follow these steps:

1. From the Hardware Configuration panel, select Option 2, Activate or process configuration data, as shown in Figure 1-51.

Remember to specify your production I/O definition file (IODF).

## 1.52 HCD activate or process configuration data

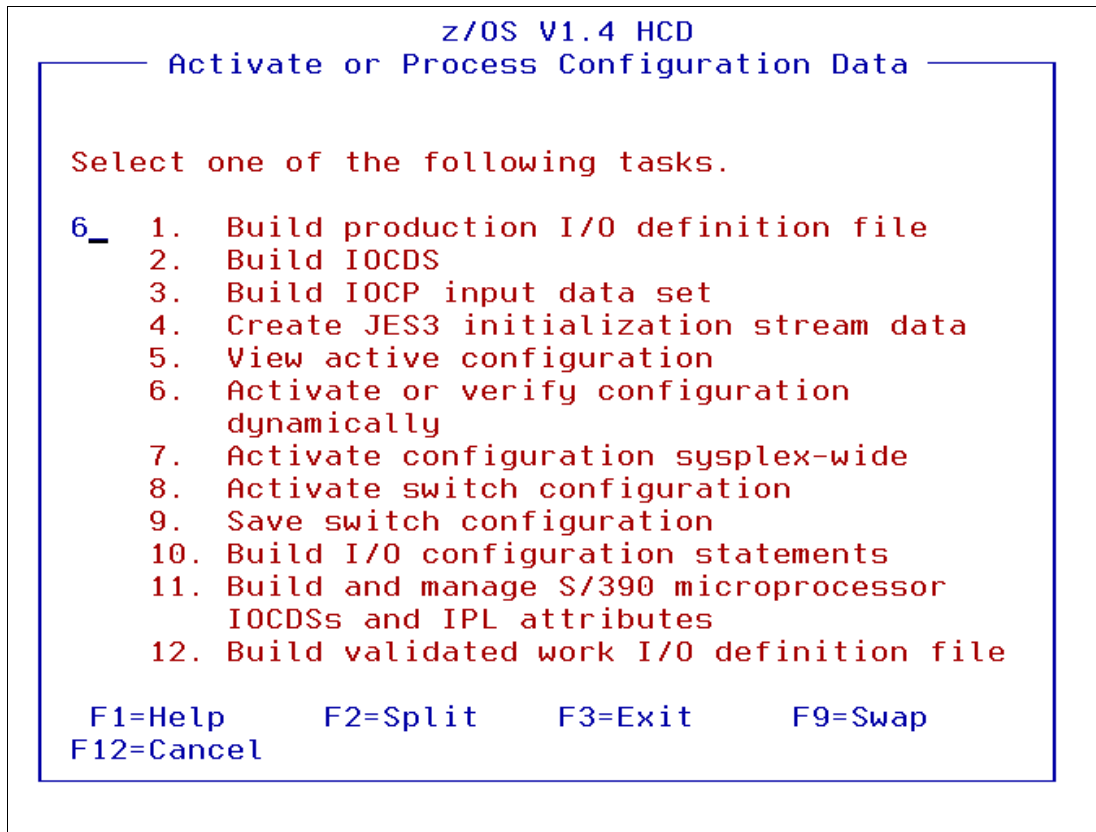


Figure 1-52 Select Option 6 to activate or verify configuration dynamically

### Activate a configuration

2. Select Option 6, Activate or verify configuration dynamically.

For the verify function on the Activate or Verify Configuration panel to be available, the processor configuration from which the active IOCDS was built must match the configuration in the IODF used for IPL (token match).

The system programmer (or other authorized persons) can use the option Activate or verify configuration dynamically, or the `ACTIVATE` operator command, to make changes to a running configuration. That is, the possibility is offered to change from a currently active configuration to some other configuration that is to be made active without the need to POR or IPL the system again.

When activating a configuration dynamically, HCD compares the currently active IODF with the IODF that is to be activated and then processes the difference.

For the IODF that is to be activated, HCD uses the production IODF that is currently in use with the dialog. Use the same high-level qualifier for the currently active IODF and the IODF to be activated.

*z/OS Hardware Configuration Definition Planning*, GA22-7525 gives a detailed description of how to dynamically activate a configuration. It describes the prerequisites for a dynamic activation, explains when hardware and software changes or software-only changes are allowed, and describes the actions necessary to change your I/O configuration dynamically.

## 1.53 Activate or verify configuration

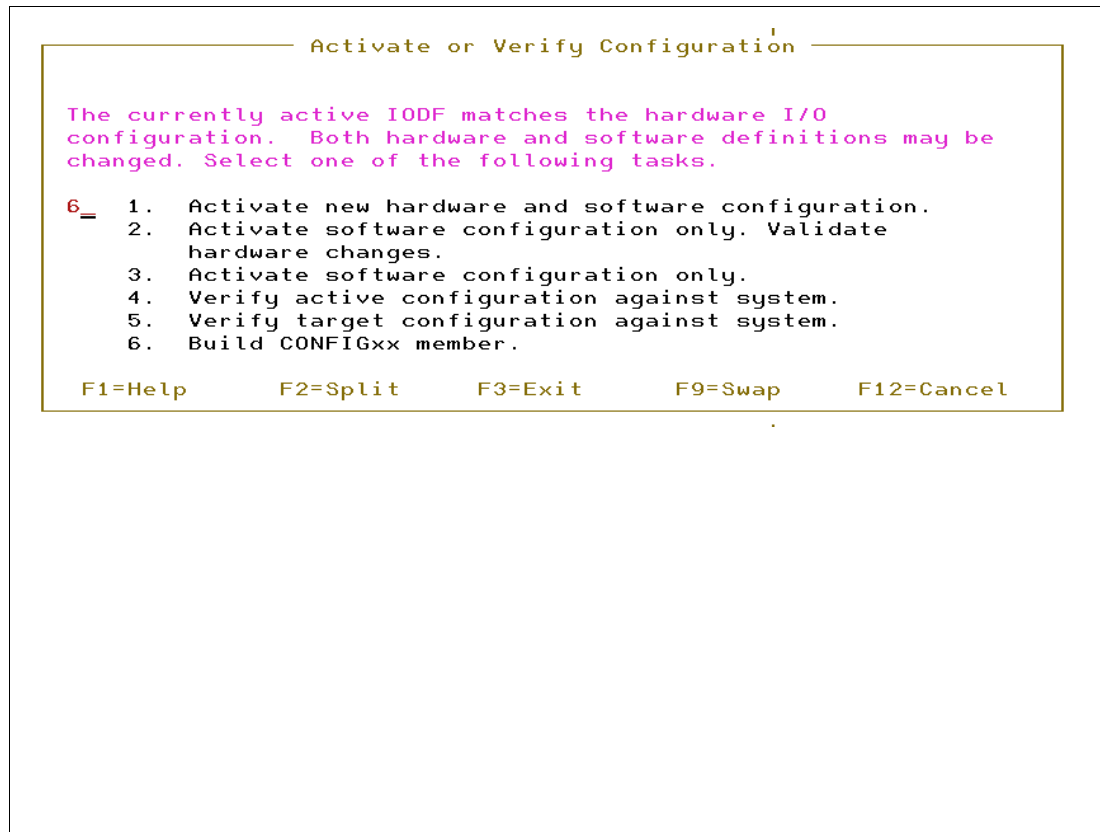


Figure 1-53 Activate or verify configuration panel

### Build a CONFIGxx member

3. Specify the system that you want your CONFIGxx member to build from. A CONFIGxx member can be built by:
  - a. Selecting the Build CONFIGxx member action from the Activate or Verify Configuration panel (for the local system)
  - b. Selecting the Build CONFIGxx member action from the Active Sysplex Member panel (for a system in a sysplex)
  - c. Using a batch utility

## 1.54 Identify system I/O configuration

Identify System I/O Configuration

Specify or revise the following values. Press ENTER to continue.

IODF to be used . . . . : SYS6.IODF71  
Processor ID . . . . . SCZP801 +  
Partition name . . . . . A9 +  
OS configuration ID . . L06RMVS1 +

I/O Cluster name . . . . . \_\_\_\_\_ + (only for Build CONFIGxx)  
F1=Help      F2=Split      F3=Exit      F4=Prompt      F5=Reset  
F9=Swap      F12=Cancel

Figure 1-54 Identify system I/O configuration panel

### Identify system configuration panel

After selecting Build CONFIG member, the Identify System I/O Configuration panel is displayed (Figure 1-54). After selecting a system, and an I/O cluster name for managed channel paths, the Restrict Ports Eligible for Dynamic CHPID Management panel is displayed if the configuration contains managed channel paths for the selected I/O cluster. This panel shows all control units known by the selected system and manageable by DCM, and their switch ports set to eligible for DCM (indicated by a Y). You can specify ports as ineligible for DCM by overtyping Y with N.

4. Complete the process by specifying the partitioned data set that will contain CONFIGxx and the suffix of the CONFIGxx.

## 1.55 Build CONFIGxx Member panel

```
Build CONFIGxx Member

Specify or revise the values for the CONFIGxx member. Press ENTER to
continue.

Partitioned data set name . _____
Suffix of CONFIGxx member . _____

Volume serial number . . . . . _____ + (if data set not cataloged)
F1=Help      F2=Split    F3=Exit    F4=Prompt    F5=Reset    F9=Swap
F12=Cancel
```

Figure 1-55 Build CONFIGxx Member panel

### Build CONFIGxx Member panel

The initial value for the partitioned data set name is SYS1.PARMLIB.

- Specify the CONFIGxx member suffix.

If the specified CONFIGxx member already exists, the Confirm Update CONFIGxx Member panel is displayed.

**Note:** It is recommended that you update the corresponding CONFIGxx member to reflect changes that you have made, especially after any dynamic changes to the system elements.

## 1.56 Creating CONFIGxx member in batch

```
//BCONFG    JOB (),'MVSSP', NOTIFY=&SYSUID,CLASS=A,  
//          MSGLEVEL=(1,1), MSGCLASS=X  
//BUILD     EXEC  PGM=CBDMGHCP,  
//          PARM='CONFIG,XX,00,SCZP601,A1,L06RMVS1,R,BACK00' 1  
//HCDIODFS  DD   DSN=SYS6.IODF71,DISP=OLD 2  
//HCDDECK   DD   DSN=SYS1.PARMLIB,DISP=SHR 3  
//HCDMLOG   DD   DSN=TAYYF.HCD.MSGLOG,DISP=SHR  
//HCDTRACE  DD   DSN=TAYFF.HCD.TRACE,DISP=SHR
```

D M=CONFIG

Figure 1-56 Creating a CONFIGxx member using a batch job

### Creating CONFIGxx member in batch

You can also use a batch job to create the CONFIGxx member. The JCL in Figure 1-56 is a sample to accomplish the task. The following notes apply to this code:

1. This parameter creates a CONFIG00 member with the Processor ID (SCZP601), Partition ID (A1) and OS configuration ID (L06RMVS1).
2. This specifies the Source IODF.
3. This specifies the output partitioned data set for CONFIGxx.

For more information, see *z/OS Hardware Configuration Definition User's Guide*, SC33-7988.

When you issue the **DISPLAY M=CONFIG(xx)** operator command, the system displays the differences between the current configuration and the configuration described in the CONFIGxx parmlib member. The default suffix of the CONFIGxx is 00.

If the existing configuration matches the one specified in the CONFIGxx, the following message is shown:



```
D M=CONFIG
IEF196I IEF237I OFCO ALLOCATED TO SYS00118
IEE252I MEMBER CONFIG00 FOUND IN SYS1.PARMLIB
IEE097I 11.07.34 DEVIATION STATUS 633
          FROM MEMBER CONFIG00
          NO DEVIATION FROM REQUESTED CONFIGURATION
```

On the other hand, if there is any mismatch between the existing configuration and that defined in the CONFIGxx, the following message is shown (note that this is only an example, you might receive something different depending on what is deviated from in the configuration):

```
D M=CONFIG
IEE252I MEMBER CONFIG00 FOUND IN SYS1.PARMLIB
IEE097I 23.15.15 DEVIATION STATUS 663
          FROM MEMBER CONFIG00
CHP          DESIRED  ACTUAL
20           ONLINE   NOT AVAILABLE
DEVICE       DESIRED  ACTUAL
0790,95      ONLINE   OFFLINE
```

## 1.57 View sysout using ISPF

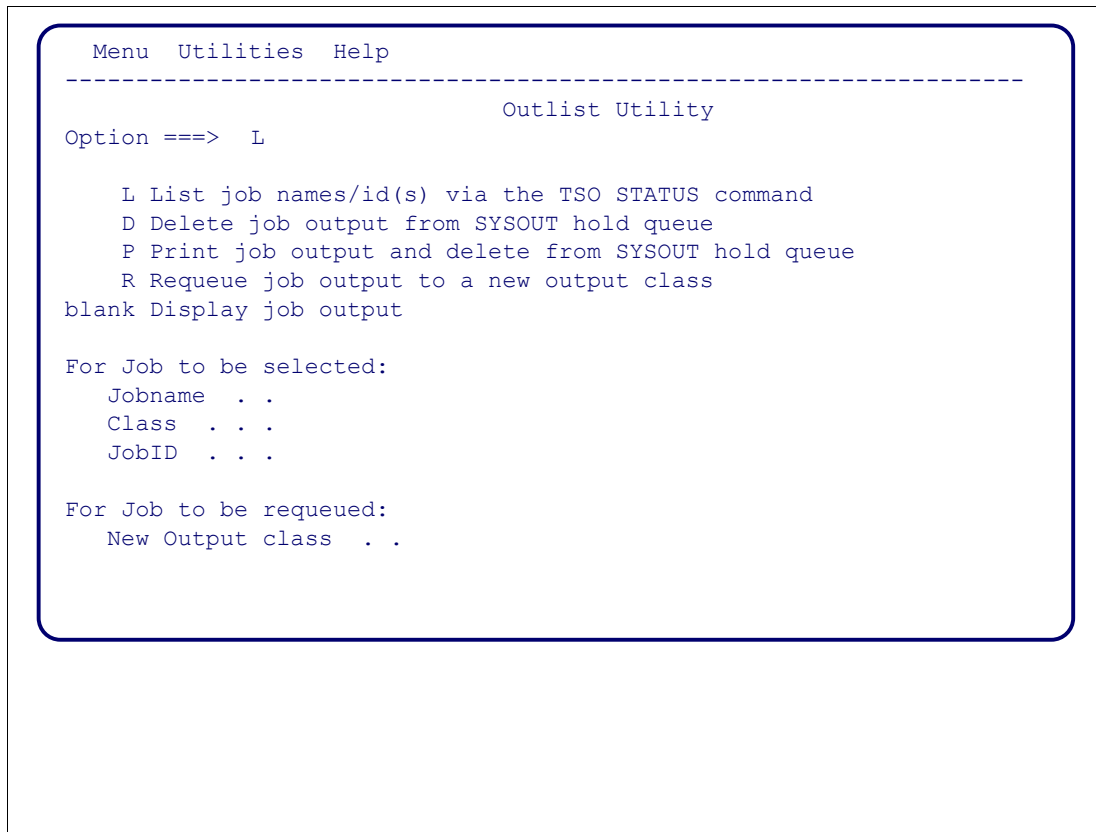


Figure 1-57 View the sysout using ISPF

### How to view sysout using ISPF

You can use Option 3.8 from the ISPF Primary Option Menu to view the SYSOUT of your jobs, instead of SDSF.

The list jobs function is used to obtain status information on jobs in your system. To list job status, fill in the following fields of the Outlist Utility panel:

- ▶ Enter *L* in the option field.
- ▶ Optionally, enter the jobname parameter as follows:
  - If jobname is blank or consists of your user ID plus one character, status is listed for all jobs having jobnames consisting of your user ID followed by one character.

**Note:** The jobname must be your user ID or start with your user ID.

- If jobname is anything else, status is listed for the specified jobname only.

The job status information will be displayed on the lower part of the screen, as shown in Job status information in the following display. If the list is too long to fit on the screen, \*\*\* will be written as the last line. Press Enter to display the next page of data.

```
For Job to be requeued:  
New Output class . .
```

```
JOB TAYYFB(JOB20684) ON OUTPUT QUEUE  
JOB TAYYFA(JOB20683) ON OUTPUT QUEUE  
JOB TAYYFC(JOB20685) ON OUTPUT QUEUE  
***
```

After you have the list of your jobs in the output queue, you can display the output for a particular job by entering the Jobname or the Job ID in the appropriate place and pressing Enter. The next panel displayed will be similar to that shown in Status display for a particular job. The Job ID is useful if there are several jobs with the same job name, as follows:

```
BROWSE      TAYYF.SPF103.OUTLIST                Line 00000000 Col 001 080  
Command ==>                                     Scroll ==> CSR  
***** Top of Data *****  
              J E S 2  J O B  L O G  --  S Y S T E M  S C 5 2  --  
  
13.54.38 JOB20683 ---- THURSDAY, 06 MAY 1999 ----  
13.54.38 JOB20683 IRR010I USERID TAYYF  IS ASSIGNED TO THIS JOB.  
13.54.38 JOB20683 ICH70001I TAYYF    LAST ACCESS AT 13:38:05 ON THURSDA  
13.54.38 JOB20683 $HASP373 TAYYFA  STARTED - INIT A    - CLASS A - SYS  
13.54.38 JOB20683 IEF403I TAYYFA - STARTED - TIME=13.54.38  
13.54.45 JOB20683 -                               --TIMINGS (  
13.54.45 JOB20683 -JOBNAME STEPNAME PROCSTEP   RC  EXCP  CPU   SR  
13.54.45 JOB20683 -TAYYFA          APPRSU      00  3047   .02   .0  
13.54.45 JOB20683 IEF404I TAYYFA - ENDED - TIME=13.54.45  
13.54.45 JOB20683 -TAYYFA  ENDED.  NAME-MVSSP                TOTAL CPU  
13.54.45 JOB20683 $HASP395 TAYYFA  ENDED
```

You can also use the requeue job output function to change output from a held SYSOUT queue to some other output class. It is typically used to print the output by directing it to a non-held output class. (Use the list job option if you wish to obtain the names and IDs of currently held jobs.) To requeue job output, fill in the following fields of the Outlist Utility panel:

- ▶ Enter R in the option field.
- ▶ Enter the job name.
- ▶ Enter the class to specify the held SYSOUT queue (which currently contains the job output).
- ▶ Enter the job ID if duplicate jobnames exist in the held SYSOUT queue (otherwise, omit).
- ▶ Enter the new output class to specify the output class into which the held output will be requeued.

## 1.58 Changing your TSO profile

```
READY
profile
CHAR(0) LINE(0) PROMPT INTERCOM NOPAUSE NOMSGID MODE WTPMSG
NORECOVER PREFIX(TAYYF) PLANGUAGE(ENU) SLANGUAGE(ENU)
DEFAULT LINE/CHARACTER DELETE CHARACTERS IN EFFECT FOR THIS TERMINAL
READY
profile noprefix
READY
profile
CHAR(0) LINE(0) PROMPT INTERCOM NOPAUSE NOMSGID MODE WTPMSG
NORECOVER NOPREFIX PLANGUAGE(ENU) SLANGUAGE(ENU)
DEFAULT LINE/CHARACTER DELETE CHARACTERS IN EFFECT FOR THIS TERMINAL
READY
```

Figure 1-58 Changing your TSO profile

### How to change your TSO user profile

Your user profile defines how you want the system to respond to information sent to or from your terminal. The typical user profile is defined by default values of the **PROFILE** command's operands. You can use the **PROFILE** command to display or change your user profile

You can enter the **PROFILE** command from a TSO prompt (either a READY prompt, or Option 6, ISPF Command Shell in ISPF). You can also enter the **PROFILE** command in the command line of any ISPF panel by prefixing the command with TSO:

```
TSO PROFILE
```

An example of the profile is:

```
CHAR(0) LINE(0) PROMPT INTERCOM NOPAUSE NOMSGID MODE WTPMSG NORECOVER
PREFIX(TAYYF) PLANGUAGE(ENU) SLANGUAGE(ENU)
DEFAULT LINE/CHARACTER DELETE CHARACTERS IN EFFECT FOR THIS TERMINAL
```

Using the **PROFILE** command, you can specify:

- ▶ On terminals without delete keys, the characters that delete a single character or the remainder of a line (CHAR operand, LIST operand).
- ▶ Whether you want the system to prompt you for information (PROMPT/NOPROMPT operand).
- ▶ Whether you want to receive messages from other users (INTERCOM/NOINTERCOM operand).

- ▶ Whether you want to be able to obtain information about messages issued to your terminal while you execute a CLIST (PAUSE/NOPAUSE operand).
- ▶ Whether you want to see the message numbers of messages displayed at your terminal (MSGID/NOMSGID operand).
- ▶ Whether you want to receive mode messages at your terminal (MODE/NOMODE operand).
- ▶ Whether you want to see at your terminal messages issued to you by a program (write-to-programmer messages) (WTPMSG/NOWTPMSG operand).
- ▶ Whether you want the EDIT recovery function in effect (RECOVER/NORECOVER operand).
- ▶ A user ID or character string to be used as the first qualifier of all non-fully-qualified data set names (PREFIX operand).
- ▶ Primary and secondary languages (PLANGUAGE and SLANGUAGE operands) to be used in displaying translated messages, help information, and the TRANSMIT full-screen panel.

One of the most common settings in your profile that you will change is the TSO PREFIX. The TSO PREFIX determines the first qualifier to be added to all data sets that are not fully qualified.

**Note:** To specify a fully qualified name for a data set, enclose it in quotes. For example, 'SYS1.PARMLIB'.

To change the TSO PREFIX to, for example, TAYYF; enter the following command:

```
PROFILE PREFIX(TAYYF)
```

When you specify a data set without quotes, such as TEST.JCLLIB, the system recognizes it as *TAYYF.TEST.JCLLIB*. You can set the prefix to any value you want, but normally it is your user ID.

If you do not want to append a prefix to non-fully-qualified data sets, enter:

```
PROFILE NOPREFIX
```

For more information on the rest of the profile settings, see *z/OS TSO/E User's Guide*, SA22-77942.

## 1.59 Backup and restore of z/OS

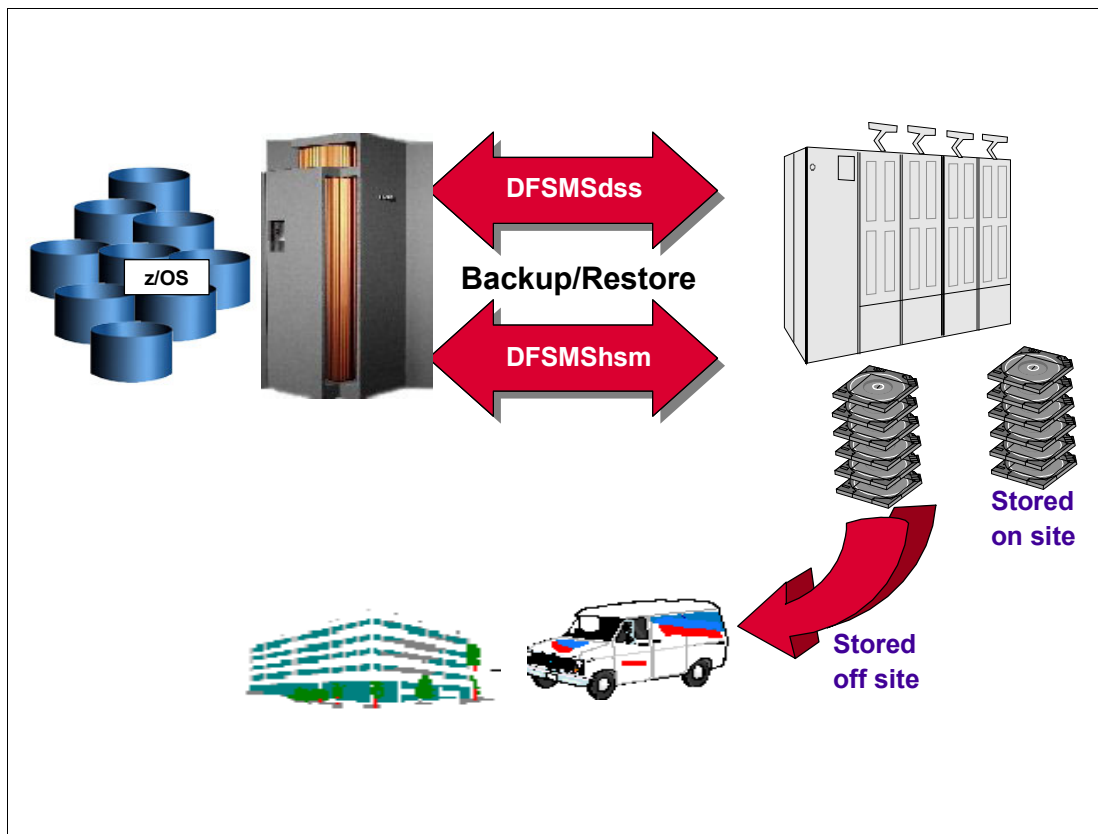


Figure 1-59 Backup and restore of z/OS

### How to back up and restore z/OS

There are times when you have made a change to some system parameters and at the next scheduled IPL, the system simply refuses to start up. You realize that you have made a mistake in the change, and try to go back to the way things were before the change. That's why you need to back up the system.

There are several other situations in which you need to perform recovery actions, for instance:

- ▶ When a hardware error occurs; for example, a DASD crash that took out a critical DASD volume.
- ▶ An I/O error during IPL due to a change to a system library, for example SYS1.NUCLEUS, or perhaps the master catalog, or any one of the critical system data sets that you need for a successful IPL.
- ▶ A corrupted system data set.

Before you start backing up your system, you have to set up a backup/restore strategy if one is not in place in your environment. There are a number of aspects you should consider in establishing your policy, including:

- ▶ The frequency of the backup

Normally, a full system backup is done weekly. This is sufficient if you do not have frequent system changes. For any minor changes before the scheduled backup, you might want to consider backup by data sets instead of a full system backup.

- The resource needed for backup

The tape or cartridge drive (for example, IBM or compatible 3420, 3480, or 3490) is considered one of the most commonly used backup/restore devices. The disadvantage of using these devices is that they are relatively slow as compared to a direct access device, like the IBM 3390.

- How many backup sets you should keep

You probably want to keep two sets of backup, if your resource permits. One set is kept at your installation, and the other is kept offsite.

You can use the DFSMSdss™ program ADRDSSU to back up your system. Using ADRASSU, you can plan to have the following typical backup:

- Full or complete

All data in the system is backed up regardless of type, content, or access pattern. Sample JCL for a full volume dump show is shown in the following:

```
//DSSDUMP JOB (),'MVSSP',NOTIFY=&SYSUID,CLASS=A,MSGLEVEL=(1,1),
//          MSGCLASS=X
//STEP1    EXEC  PGM=ADRDSSU
//SYSPRINT DD   SYSOUT=A
//DASD     DD    UNIT=3390,VOL=SER=MPDLB1,DISP=OLD
//TAPE     DD    UNIT=(3480,2),VOL=SER=(TAPE01,TAPE02),
// LABEL=(1,SL),DISP=(NEW,CATLG),DSNAME=MPDLB1.BACKUP
//SYSIN    DD    *
            DUMP INDDNAME(DASD1) -
                OUTDDNAME(BACKUP) -
                FULL OPTIMIZE(4) -
                COMPRESS
/*
```

- Incremental

An incremental backup only backs up data that has been changed since a previous backup. Managing data this way requires the operating system to maintain some form of change control for individual data sets. Sample JCL for an incremental dump shows Sample JCL for doing an incremental dump, on only data sets changed since the last backup, is as follows:

```
//DSSDUMP JOB (),'MVSSP',NOTIFY=&SYSUID,CLASS=A,MSGLEVEL=(1,1),
//          MSGCLASS=X
//STEP1    EXEC  PGM=ADRDSSU
//SYSPRINT DD   SYSOUT=A
//TAPE     DD    UNIT=(3480,2),VOL=SER=(TAPE01,TAPE02),
// LABEL=(1,SL),DISP=(NEW,CATLG),DSNAME=MPDLB1.BACKUP
//SYSIN    DD    *
            DUMP OUTDD(TAPE) -
                DS(INCL(**) -
                BY((DSCHA EQ 1)))
/*
```

► Selective

Data backed up under control of the user. Specific volumes and data sets are selected for backup based on importance or access pattern. Sample JCL for backing up selective data sets, specifically data sets that have a high level qualifier of ISP, is shown in the following:

```
//DSSDUMP JOB (),'MVSSP',NOTIFY=&SYSUID,CLASS=A,MSGLEVEL=(1,1),
//          MSGCLASS=X
//STEP1    EXEC  PGM=ADRDSSU
//SYSPRINT DD  SYSOUT=A
//TAPE     DD    UNIT=(3480,2),VOL=SER=(TAPE01,TAPE02),
// LABEL=(1,SL),DISP=(NEW,CATLG),DSNAME=MPRES2.BACKUP
//SYSIN    DD    *
            DUMP OUTDD(TAPE) -
              DS(INCL(ISP.**)) -
              TOL(ENQF) OPTIMIZE(4) COMPRESS
/*
```

For more information on the use of ADRDSSU, see *z/OS DFSMSdss Storage Administration Reference*, SC35-0424.





## Subsystems and subsystem interface (SSI)

This chapter describes basic concepts that you need to understand if you want to write your own subsystem or want to use services provided by IBM subsystems. It also describes planning considerations for setting up and writing your own subsystem.

When you want to write your own subsystem, you must:

- ▶ Provide the routines to support the request for a function. These function routines get control from the SSI. They may actually perform the function or may pass control to other routines that you provide.
- ▶ Provide a subsystem address space (if required).
- ▶ Let MVS know that the subsystem exists (define the subsystem).
- ▶ Provide the information to the SSI that it will need to find your function routines (initialize the subsystem).
- ▶ Provide accounting information parameters to your subsystem (if required).

**Note:** When writing your own subsystem you must also provide any control blocks or resources that the subsystem requires for its own operation, which MVS does not provide.

## 2.1 Defining subsystems and subsystem interface

- ❑ What is a subsystem?
- ❑ Types of subsystems
  - Master
  - Job entry or primary
  - Secondary
  - Functional
- ❑ What is the SSI
- ❑ Unique attributes of the SSI

*Figure 2-1 Defining subsystems and subsystem interface*

### Defining subsystems

A subsystem is a service provider that performs one function or many functions, but does nothing until it is requested. Although the term “subsystem” is used in other ways, in this book a subsystem must be the master subsystem or be defined to MVS in one of the following ways:

- ▶ Processing the IEFSSNxx parmlib member during IPL
  - You can use either the keyword format or positional format of the IEFSSNxx parmlib member. We recommend that you use the keyword format, which allows you to define and dynamically manage your subsystems.
- ▶ Issuing the IEFSSI macro
- ▶ Issuing the SETSSI system command

**Note:** The master subsystem (MSTR) is a part of MVS and is not defined in any of these ways.

Some examples of IBM-supplied subsystems that use these interfaces are the following:

- ▶ JES2
- ▶ JES3
- ▶ IMS

- ▶ NetView®
- ▶ OPC

## Types of subsystems

There are four types of subsystems:

- ▶ The *master scheduler subsystem* is used to establish communication between the operating system and the primary job entry subsystem, which can be JES2 or JES3. It is also used to initialize system services such as the system log and communications task.
- ▶ The *primary subsystem* is the job entry subsystem that MVS uses to do work. It can be either JES2 or JES3.
- ▶ *Secondary subsystems* provide functions as needed by IBM products, vendor products, or the installation.
- ▶ *Functional subsystems*: JES2/JES3 allows certain functions to operate outside the JES2/JES3 address space. JES3 does this using:
  - The functional subsystem address space (FSS)
  - The functional subsystem interface (FSI)
  - The functional subsystem application (FSA)

The JES3 FSS that deals with output services is one type of FSS. This particular FSS address space may be created automatically or established by a CALL command for a printer device which is capable of running under the control of an FSS address space. The operator CALL command designates a printer as a “hot writer,” while a writer invoked automatically when output is queued is a “dynamic writer.”

MVS communicates with subsystems through the SSI.

## Subsystem interface (SSI)

The SSI is the interface used by routines (IBM, vendor, or installation-written) to request services of, or to pass information to, subsystems. An installation can design its own subsystem and use the SSI to monitor subsystem requests. An installation 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.

## Unique attributes of the SSI

The SSI is a way for one routine to call another routine. There are a number of other ways that a routine can call another routine, such as:

- ▶ Branch and link register (BALR) 14,15
- ▶ LINK or LINKX macro
- ▶ Program call (PC)
- ▶ SVC

The SSI is different from these linkage interfaces, however, in that:

- ▶ The called routine does not have to be there. That is, when a routine calls the subsystem, the SSI checks to see if the subsystem either is not interested in the request or does not exist. The caller then receives an appropriate return code.
- ▶ A caller's request can be routed to multiple subsystem routines.

## 2.2 Subsystem initialization

- ❑ Primary Subsystem
  - JES2
  - JES3
- ❑ Subsystem Initialization

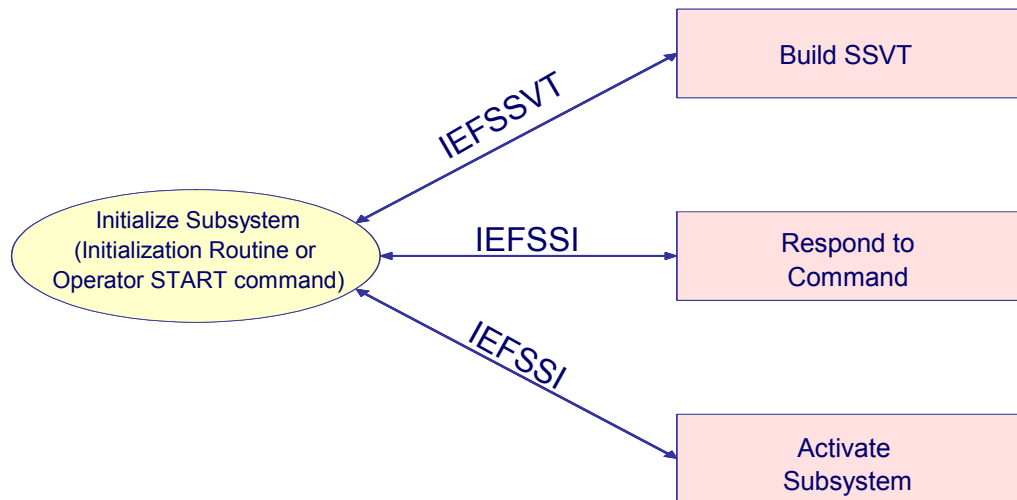


Figure 2-2 Subsystem initialization

### The primary subsystem

For work to be done, MVS requires that at least one subsystem be defined as a job entry subsystem (JES) to bring jobs into the system. The JES, in fact, is called the primary subsystem. You can select either JES2 or JES3. If you do not specify an IEFSSNxx member in SYS1.PARMLIB, MVS attempts to use the system default member, IEFSSN00. IEFSSN00, as supplied by IBM, contains the definition for the default primary job entry subsystem, JES2.

If you attempt to IPL without specifying an IEFSSNxx member, and IEFSSN00 is not present or does not identify the primary subsystem, the system issues message IEFJ005I and prompts the operator for the primary subsystem.

For an IPL, do not define a subsystem more than once in a combination of IEFSSNxx members that can be used together or within a single member. The same subsystem can appear in two different IEFSSNxx members when the members will not be used together. In general, if MVS detects a duplicate name, both of the following are true:

- ▶ MVS does not define the duplicate subsystem
- ▶ MVS does not give control to the initialization routine.

The system issues the following message:

```
IEFJ003I: DUPLICATE SUBSYSTEM subname NOT INITIALIZED
```

## How to initialize your subsystem

There are two ways to initialize your subsystem:

- ▶ Specifying an initialization routine
- ▶ Using the START command

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

### Specifying an initialization routine

You can optionally specify the name of your subsystem initialization routine when you define your subsystem. If the functions 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 the subsystem is active.

### Using the START command

If the subsystem functions are not needed until a later time, you can use the START command to initialize your subsystem. See *z/OS MVS System Commands*, SA22-7627-04 and *z/OS MVS JCL Reference*, SA22-7597-04 for more information on the START command.

Figure 2-2 on page 100 shows how you can initialize your subsystem either by specifying an initialization routine or by using the START command.

## 2.3 Types of subsystem requests

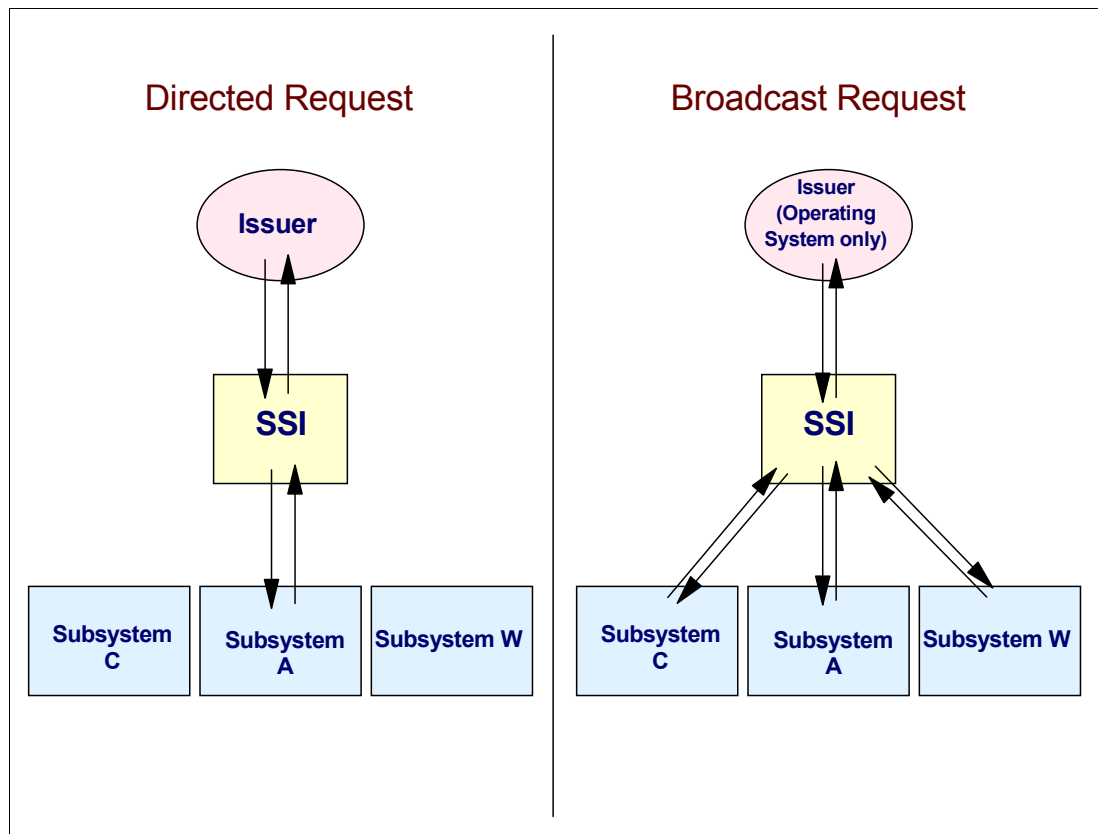


Figure 2-3 Types of subsystem request

### Subsystem requests

The SSI handles two types of requests: directed requests and broadcast requests. Directed requests, which can be defined by the installation, are made 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.

The left side of Figure 2-3 shows the processing for a directed request.

Broadcast requests, which are defined by MVS, 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, your subsystem can be informed when a WTOR message is issued in order to automate a response to the WTOR. The right side of Figure 2-3 shows the processing for a broadcast request.

The IEFJFRQ installation exit provides a way to examine and modify subsystem function requests. See *z/OS MVS Installation Exits*, SA22-7593-04 for more information on the capabilities and use of the IEFJFRQ exit.

## 2.4 IEFSSNxx parmlib member

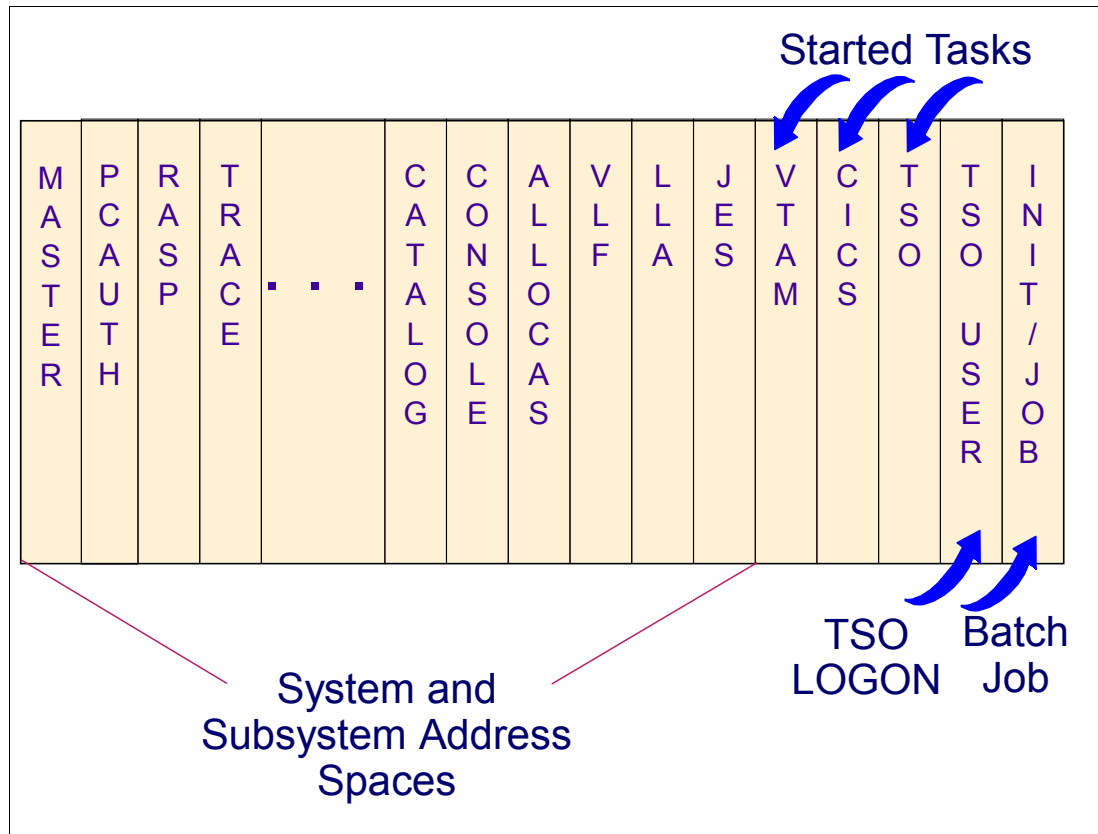


Figure 2-4 IEFSSNxx parmlib member

### IEFSSNxx (subsystem definitions) - keyword parameter form

A subsystem is a service provider that performs one or many functions, but does nothing until it is requested. Subsystems are defined to MVS by the processing of the IEFSSNxx parmlib member during IPL. You can use either the keyword format or positional format of the IEFSSNxx parmlib member. We recommend that you use the keyword format, which allows you to define and dynamically manage your subsystems. You do this by:

- ▶ Issuing the IEFSSI macro
- ▶ Issuing the **SETSSI** system command

IEFSSNxx contains parameters that define the primary subsystem and the various secondary subsystems that are to be initialized during system initialization.

IEFSSNxx allows you to:

- ▶ Name the subsystem initialization routine to be given control during master scheduler initialization.
- ▶ Specify the input parameter string to be passed to the subsystem initialization routine.
- ▶ Specify a primary subsystem name and whether you want it started automatically.

For information about writing subsystems, see *z/OS MVS Using the Subsystem Interface*, SA22-7642-01.

The order in which the subsystems are initialized depends on the order in which they are defined in the IEFSSNxx parmlib member on the SSN parameter. Unless you are starting the Storage Management Subsystem (SMS), start the primary subsystem (JES) first. Some subsystems require the services of the primary subsystem in their initialization routines. Problems can occur if subsystems that use the subsystem affinity service in their initialization routines are initialized before the primary subsystem. If you are starting SMS, specify its record before you specify the primary subsystem record.

**Note:** In general, it is a good idea to make the subsystem name the same as the name of the member of the SYS1.PROCLIB used to start the subsystem. If the name does not match, you may receive error messages when you start the subsystem.

## Restrictions for IEFSSNxx

The following restrictions apply for IEFSSNxx:

- ▶ All subsystem definitions in a single IEFSSNxx member must use the same format. A single member cannot contain both positional and keyword definitions.
  - If a member begins with the positional format and switches to the keyword format, processing of the member stops, and the IEFJ002I message is issued. The last subsystem definition processed for the member is the last positional format definition before the switch. The system does not process another definition of either format from the member, but continues processing with the next member, if any.
  - If a member begins with the keyword format and a positional format definition is found, the system issues a syntax error message, and processing continues with the next definition of the keyword format.

Only subsystems that have been defined using the keyword format IEFSSNxx parmlib member, the IEFSSI REQUEST=ADD macro, or the **SETSSI ADD** system command can use the following dynamic SSI services:

- ▶ Macros
  - IEFSSI REQUEST=ACTIVATE
  - IEFSSI REQUEST=DEACTIVATE
  - IEFSSI REQUEST=OPTIONS
  - IEFSSI REQUEST=SWAP
  - IEFSSI REQUEST=GET
  - IEFSSI REQUEST=PUT
  - IEFSSVT
- ▶ System commands
  - SETSSI ACTIVATE
  - SETSSI DEACTIVATE

You cannot use dynamic SSI services for subsystems defined with the positional form of this member.



## 2.5 Subsystem definitions

### SYS1.PARMLIB: IEFSSNxx

```
SUBSYS      SUBNAME (subname)
             [CONSNAME (consname) ]
             [INITRTN (initrtn)
             [INITPARM (initparm) ] ]
             [PRIMARY ( {NO | YES} )
             [START ( {YES | NO} ) ] ]
```

```
SUBSYS SUBNAME(SMS) INITRTN(IGDSSIIN)
INITPARM('ID=60,PROMPT=YES')
SUBSYS SUBNAME(JES2) PRIMARY(YES)
```

### SYS1.PARMLIB: IEASYSxx SSN=xx

Figure 2-5 Subsystem definitions

#### Statements and parameters for IEFSSNxx

The storage management subsystem (SMS) is the only subsystem that can be defined before the primary subsystem. Refer to the description of parmlib member IEFSSNxx in *z/OS V1R4.0 MVS Initialization and Tuning Reference*, SA22-7592-03, for SMS considerations. The statements and parameters for IEFSSNxx are:

|                           |                                                                                                                                                                                                                                                                                                                                                  |
|---------------------------|--------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| <b>SUBSYS</b>             | The statement that defines a subsystem that is to be added to the system. If more than one SUBSYS statement appears for the same subsystem name, the first statement will be the one used to define the subsystem. The duplicate statements will be rejected with a failure message that is sent to the console.                                 |
| <b>SUBNAME(subname)</b>   | The subsystem name. The name can be up to four characters long; it must begin with an alphabetic or special character (#, @, or \$), and the remaining characters (if any) can be alphanumeric or special.                                                                                                                                       |
| <b>CONSNAME(consname)</b> | The name of the console to which any messages that the SSI issues as part of initialization processing are to be routed. This name is optional and can be two to eight characters long. This console name is also passed to the routine named on the INITRTN keyword if it is specified. The default is to issue messages to the master console. |
| <b>INITRTN(initrtn)</b>   | The name of the subsystem initialization routine. This name is optional and can be one to eight characters long. The first                                                                                                                                                                                                                       |

character can be either alphabetic or special. The remaining characters can be either alphanumeric or special. The routine receives control in supervisor state key 0. It must be the name of a program accessible through LINKLIB.

#### **INITPARM(initparm)**

Input parameters to be passed to the subsystem initialization routine. The input parameters are optional and are variable in length for a maximum of 60 characters. If blanks, commas, single quotes, or parentheses are included in the input parameters, the entire parm field must be enclosed in single quotes. If the parm field is enclosed in single quotes, a single quote within the field must be specified as two single quotes. The INITPARM keyword can only be specified if the INITRTN keyword is specified.

#### **PRIMARY({NOIYES})**

This parameter indicates whether this is the primary subsystem. The primary subsystem is typically a job entry subsystem (either JES2 or JES3).

This parameter is optional. Initialize the primary subsystem before any secondary subsystem(s) except SMS. If you specify PRIMARY on more than one statement, the system issues message IEFJ008I and defines the second subsystem but ignores the PRIMARY specification.

The IEFSSNxx parmlib member is the only place you can define the primary subsystem. It cannot be defined using the dynamic SSI services IEFSSI REQUEST=ADD macro or the **SETSSI ADD** command. The default is NO.

#### **START({YESINO})**

This parameter indicates whether an automatic **START** command should be issued for the primary subsystem.

If the parmlib entry for the primary subsystem is START(NO), the operator must start it later with a **START** command. If the parmlib entry for the primary subsystem does not specify the START parameter, it defaults to START(YES).

The START parameter cannot be specified for a secondary subsystem. If you specify the PRIMARY(NO) parameter, there is no default for the START parameter.

### **Subsystem example**

Define subsystem JES2 as a primary subsystem and the **START** command to be issued by the system. No initialization routine is required because subsystem JES2 builds the SSVT when the **START** command is issued.

```
SUBSYS SUBNAME(JES2) PRIMARY(YES)
```

### **Parameter in IEASYSxx (or specified by the operator)**

The SSN parameter in IEASYSxx identifies the IEFSSNxx member that the system is to use to initialize the subsystems, as follows:

```
SSN=      { aa      }
          { (aa,bb,...) }
```

The two-character identifier, represented by aa (or bb, and so forth) is appended to IEFSSN to identify IEFSSNxx members of parmlib. If the SSN parameter is not specified, the system uses the IEFSSN00 parmlib member.

The order in which the subsystems are defined on the SSN parameter is the order in which they are initialized. For example, a specification of SSN=(13,Z5) would cause those subsystems defined in the IEFSSN13 parmlib member to be initialized first, followed by those subsystems defined in the IEFSSNZ5 parmlib member. If you specify duplicate subsystem names in IEFSSNxx parmlib members, the system issues message IEFJ003I to the SYSLOG, the master console, and consoles that monitor routing code 10 messages.

Some exits that use system services may run before other system address spaces are active. You must ensure that any address spaces required by the system services are available prior to invoking the service.

For more information, see the section on handling errors in defining your subsystem in *z/OS MVS Using the Subsystem Interface*, SA22-7642.

### **Syntax rules for IEFSSNxx**

The following rules apply to the creation of IEFSSNxx:

- ▶ Each SUBSYS statement in IEFSSNxx defines one subsystem to be initialized.
- ▶ Use columns 1 through 71. Do not use columns 72 through 80 because the system ignores these columns.
- ▶ Comments may appear in columns 1 through 71 and must begin with /\* and end with \*/.
- ▶ A statement must begin with a valid statement type followed by at least one blank or end-of-line.
- ▶ A statement ends with the beginning of the next valid statement type or end-of-file.
- ▶ A statement can be continued even though there is no explicit continuation character.
- ▶ Operands must be separated by valid delimiters. Valid delimiters are a blank, or column 71. If the operand contains parentheses, then the right parenthesis is accepted as a valid delimiter.
- ▶ Multiple occurrences of a delimiter (except for parentheses) are accepted, but treated as one.

### **IBM-supplied default for IEFSSNxx**

If you do not specify the SSN system parameter, the system uses the IEFSSN00 parmlib member. IEFSSN00 specifies JES2 as the primary subsystem.

If you specify a set of IEFSSNxx members that do not identify a primary subsystem, the system issues a message that prompts the operator to specify the primary subsystem.

## 2.6 Subsystem interface (SSI)

- ❑ Means of communication between:
  - System and subsystem
- ❑ Consists of:
  - Control blocks - macros - routines

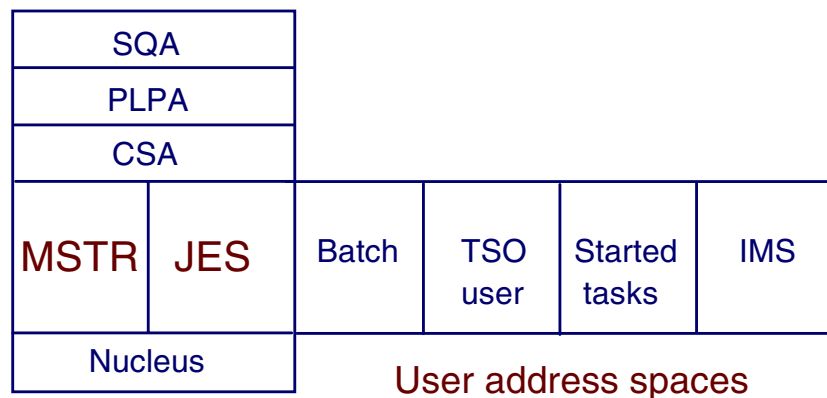


Figure 2-6 Subsystem interface - SSI

### Subsystem Interface (SSI)

The SSI is an interface that provides communication between MVS and JES through the IEFSSREQ macro. The SSI is the interface used by routines (IBM, vendor, or installation-written) to request services of, or to pass information to, subsystems. An installation can design its own subsystem and use the SSI to monitor subsystem requests. An installation 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.

### IEFSSREQ macro requests

MVS functions issue the IEFSSREQ macro to invoke JES or any subsystem. The calling routine uses the subsystem option block (SSOB) and subsystem identification block (SSIB) to identify the required processing to JES. The calling routine uses the IEFSSREQ macro to pass the SSIB and SSOB to JES.

## 2.7 SSI control blocks and routines

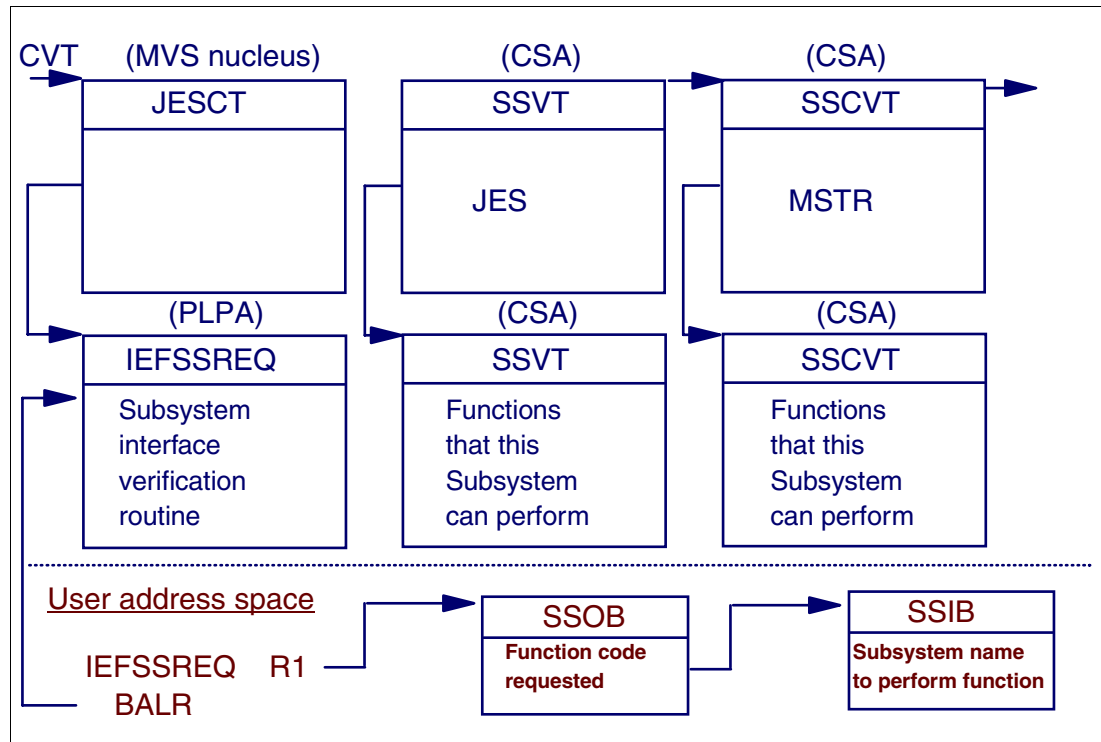


Figure 2-7 SSI control blocks and routines

### SSI control blocks

The MVS nucleus contains a control block named the JES control table (JESCT). This block contains a pointer to the primary subsystem (JES2/JES3) communication vector table (SSCVT), which is located in CSA. There will always be at least two SSCVTs, one of which represents the MVS master subsystem.

The macro that provides communication between MVS and JES2/JES3 is the IEFSSREQ macro. MVS functions issue this macro to invoke JES3. The calling routine uses the subsystem option block (SSOB) and subsystem identification block (SSIB) to identify the required processing to JES3. The calling routine uses the IEFSSREQ macro to pass the SSIB and SSOB to JES. The control blocks involved are:

- JESCT** JES control table (JESCT). A control block in the MVS nucleus that contains information used by subsystem interface routines.
- SSCVT** Primary subsystem communication vector table (SSCVT). This control block is the common storage area that contains information used by the subsystem interface routines.
- SSVT** The subsystem vector table (SSVT). The SSVT resides in the JES common service area (CSA) and contains the following information:
  - SSOB** Subsystem options block (SSOB). The control block into which MVS places a function code when communicating with JES over the subsystem interface. The function code identifies a requested service.
  - SSIB** Subsystem identification block (SSIB). The control block into which MVS places the name of the subsystem to which it is directing a request over the subsystem interface.

## 2.8 SSI request to master subsystem

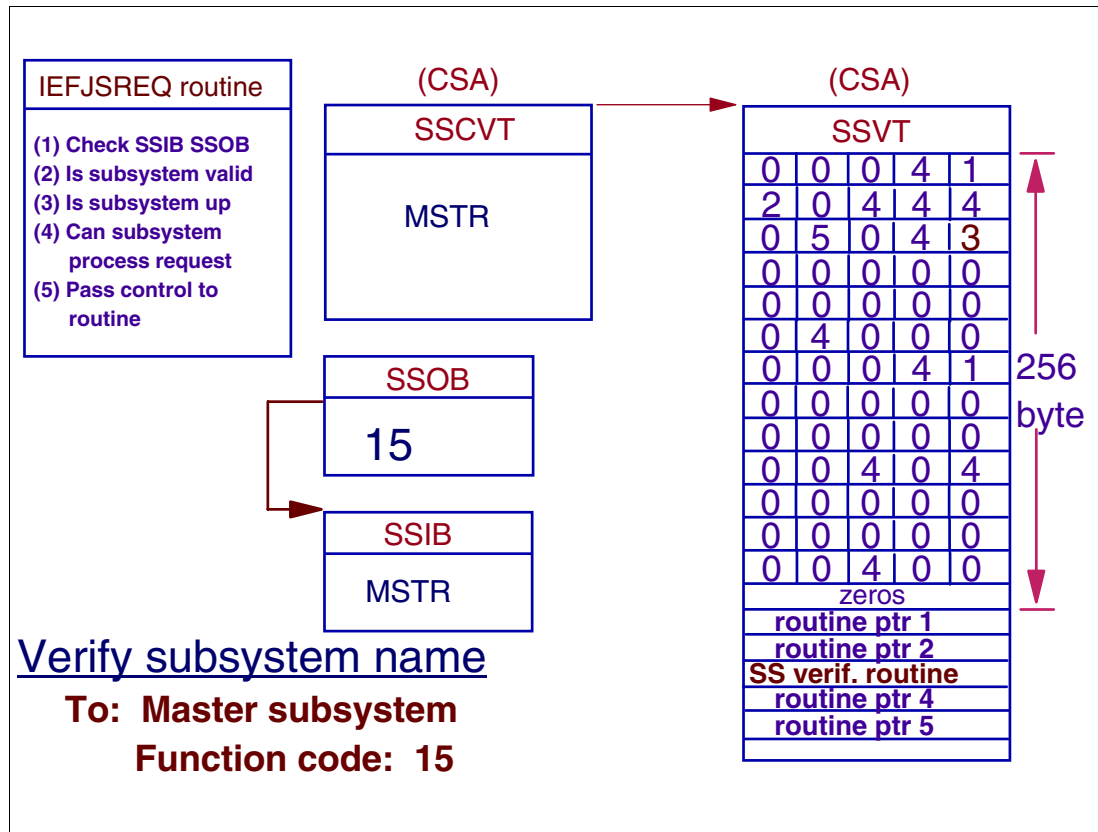


Figure 2-8 SSI request to master subsystem

### Master subsystem requests

The IEFJSREQ routine checks the validity of the SSOB and SSIB; the request routine determines that the target subsystem exists and is started. It next uses the function code to determine if the subsystem performs the requested function and to derive the address of a routine to which the request is to be passed. The SSOB contains the function code requested by the caller and the SSIB contains the name of the subsystem to perform the function requested.

There are 15 SSI functions supported by the master subsystem, but several of the routines perform more than one function.

The SSVT contains a matrix of the function codes and in the visual the first function code supported by the MSTR subsystem is function code 4. The number 4 shown in the fourth position indicates that the routine that gets control is the fourth routine pointer (ptr) shown after the function code matrix in the SSVT.

Function code 15 is used by the master subsystem (MSTR) to verify the subsystem (JES) name when JES initializes, and is the third routine pointer, as function code 15 has a 3 in the matrix.

## 2.9 JES2 supported SSI functions

|    |                        |    |                        |
|----|------------------------|----|------------------------|
| 1  | SYSOUT                 | 53 | FSS/FSA CONNECT/DISCON |
| 2  | CANCEL                 | 54 | SUBSYSTEM VERSION      |
| 3  | JOB STATUS             | 64 | TRANSACTION PROCESSING |
| 4  | EOT                    | 70 | SJF SPOOL SERVICES     |
| 5  | JOB SELECTION          | 71 | JOB INFORMATION        |
| 6  | ALLOCATION             | 75 | NOTIFY-USER MSG ROUTER |
| 7  | UNALLOCATION           | 77 | PERSISTENT JCL         |
| 8  | EOM                    | 79 | SYSOUT API             |
| 9  | WTO/WTOR               | 80 | ENHANCED STATUS        |
| 10 | CMD PROCESSING (SVC34) |    |                        |
| 11 | REMOT DEST VALIDITY CK |    |                        |
| 12 | JOB DELETION           |    |                        |
| 13 | JOB RE-ENQUEUE         |    |                        |
| 16 | OPEN                   |    |                        |
| 17 | CLOSE                  |    |                        |
| 18 | CHECKPOINT             |    |                        |
| 19 | RESTART                |    |                        |
| 20 | REQUEST JOB ID         |    |                        |
| 21 | RETURN JOB ID          |    |                        |

Figure 2-9 JES2 supported SSI functions

### SSI functions supported by JES2

Figure 2-9 lists the functions supported by JES2 as a subsystem.

Following is a list of directed function codes a user program can request. IBM subsystems provide these function codes.

| Code | Requested Function  | Subsystem*       | Type of Request                   |
|------|---------------------|------------------|-----------------------------------|
| 1    | Process SYSOUT data | JES2/JES3        | Directed sets                     |
| 15   | Verify subsystem    | Master           | Directed function                 |
| 20   | Request job ID      | JES2/JES3        | Directed                          |
| 21   | Return job ID       | JES2/JES3        | Directed                          |
| 54   | Request subsystem   | JES2/JES3/Master | Directed version information      |
| 71   | JES JOB information | JES2             | Directed                          |
| 75   | Notify user message | JES2/JES3        | Directed service                  |
| 79   | SYSOUT Application  | JES2/JES3        | Directed Program Interface (SAPI) |

## 2.10 JES3 supported SSI functions

|    |                        |    |                           |
|----|------------------------|----|---------------------------|
| 1  | SYSOUT                 | 24 | COMMON ALLOCATION         |
| 2  | CANCEL                 | 25 | COMMON UNALLOCATION       |
| 3  | JOB STATUS             | 26 | CHANGE DDNAME             |
| 4  | EOT                    | 27 | CHANGE ENQ USE ATTRIBUTE  |
| 5  | JOB SELECTION          | 28 | DDR CANDIDATE SELECT      |
| 6  | ALLOCATION             | 29 | DDR CANDIDATE VERIFY      |
| 7  | UNALLOCATION           | 30 | DDR SWAP NOTIFICATION     |
| 8  | EOM                    | 31 | DDR SWAP COMPLETE         |
| 9  | WTO/WTOR               | 32 | SVC34 COMMAND FAIL        |
| 10 | CMD PROCESSING (SVC34) | 34 | WRITE TO LOG              |
| 11 | REMOT DEST VALIDITY CK | 40 | EARLY VOLUME RELEASE      |
| 12 | JOB DELETION           | 53 | FSS/FSA CONNECT/DISCON    |
| 13 | JOB RE-ENQUEUE         | 54 | SUBSYSTEM VERSION         |
| 16 | OPEN                   | 56 | SMS TO JES3 COMMUNICATION |
| 17 | CLOSE                  | 62 | BDT SUBSYSTEM             |
| 18 | CHECKPOINT             | 64 | TRANSACTION PROCESSING    |
| 19 | RESTART                | 72 | VARY PATH CALL            |
| 20 | REQUEST JOB ID         | 75 | NOTIFY-USER MSG ROUTER    |
| 21 | RETURN JOB ID          | 77 | PERSISTENT JCL            |
| 22 | STEP INITIATION        | 79 | SYSOUT API                |
| 23 | DYNAMIC ALLOCATION     | 80 | ENHANCED STATUS           |

Figure 2-10 JES3 supported SSI functions

### SSI function codes supported by JES3

Figure 2-10 lists the function codes supported by JES3 as a subsystem.

Following is a list of directed function codes, along with their purposes, that a user program can request. IBM subsystems provide these function codes.

| Code | Requested Function  | Subsystem*       | Type of Request                   |
|------|---------------------|------------------|-----------------------------------|
| 1    | Process SYSOUT data | JES2/JES3        | Directed sets                     |
| 15   | Verify subsystem    | Master           | Directed function                 |
| 20   | Request job ID      | JES2/JES3        | Directed                          |
| 21   | Return job ID       | JES2/JES3        | Directed                          |
| 54   | Request subsystem   | JES2/JES3/Master | Directed version information      |
| 71   | JES JOB information | JES2             | Directed                          |
| 75   | Notify user message | JES2/JES3        | Directed service                  |
| 79   | SYSOUT Application  | JES2/JES3        | Directed Program Interface (SAPI) |





## 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. JES provides supplementary job management, data management, and task management functions such as: scheduling, control of job flow, and spooling. 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 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 job entry subsystem (JES2 or JES3) and DFSMSdfp™ 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:

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

## 3.1 z/OS and job management

- ❑ Understanding JCL
- ❑ Job control statements
- ❑ Required JCL statements
- ❑ Why JES?

Figure 3-1 z/OS and job management

### z/OS and job management

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:

- ▶ 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 used by the completed jobs, making the resources available to other jobs

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 MVS system divides the management of jobs and resources with a JES. JES does job management and resource management before and after job execution, while MVS does it during job execution. The JES receives jobs into MVS, schedules them for processing by MVS, and controls their output processing.

## 3.2 Job management

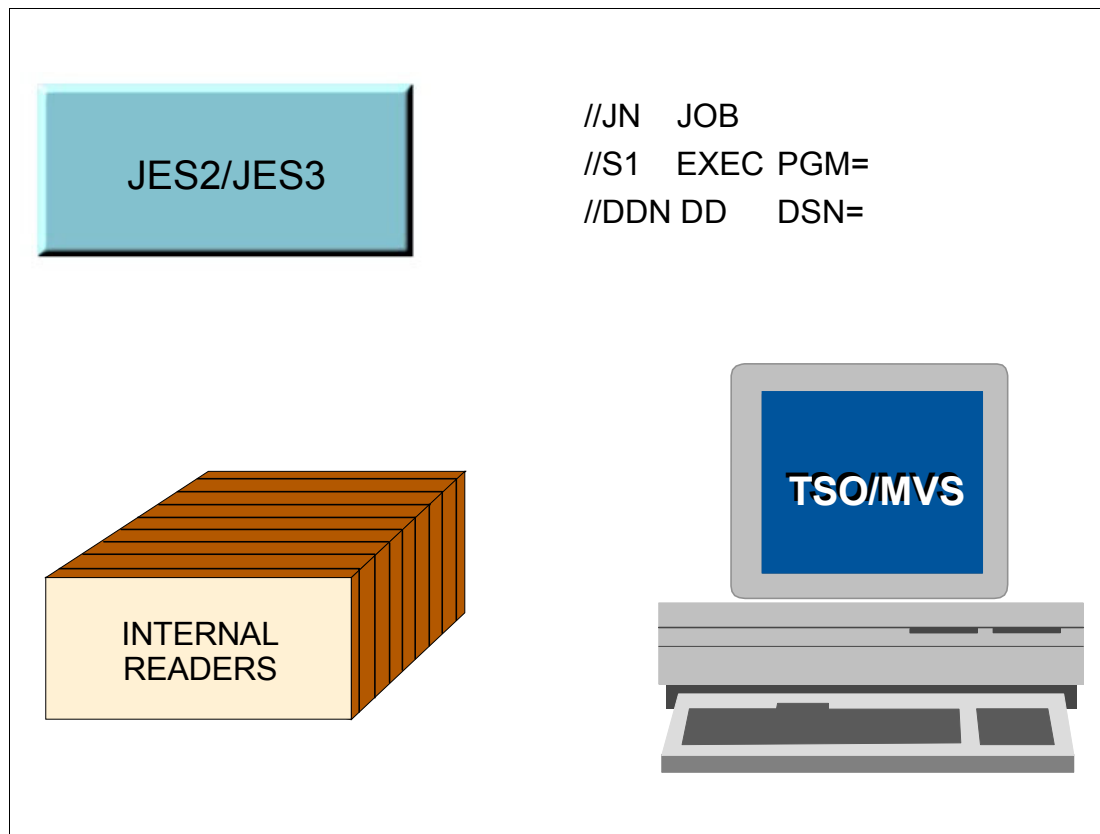


Figure 3-2 Job management

### Understanding JCL

To get your MVS 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 MVS. Refer to volume one, chapter three for a description of JCL and how it works.

As shown in Figure 3-2 and described in volume 1, you can submit a job from a TSO/E terminal to JES2 or JES3 using internal readers. Jobs are submitted to different classes of execution according to the CLASS parameter.

Figure 3-2 shows an overview of the job-submission process. The user performs the parts on the left side of the figure, and the system performs the parts on the right. In this figure, z/OS and JES comprise the “system.”

### 3.3 JCL-related actions

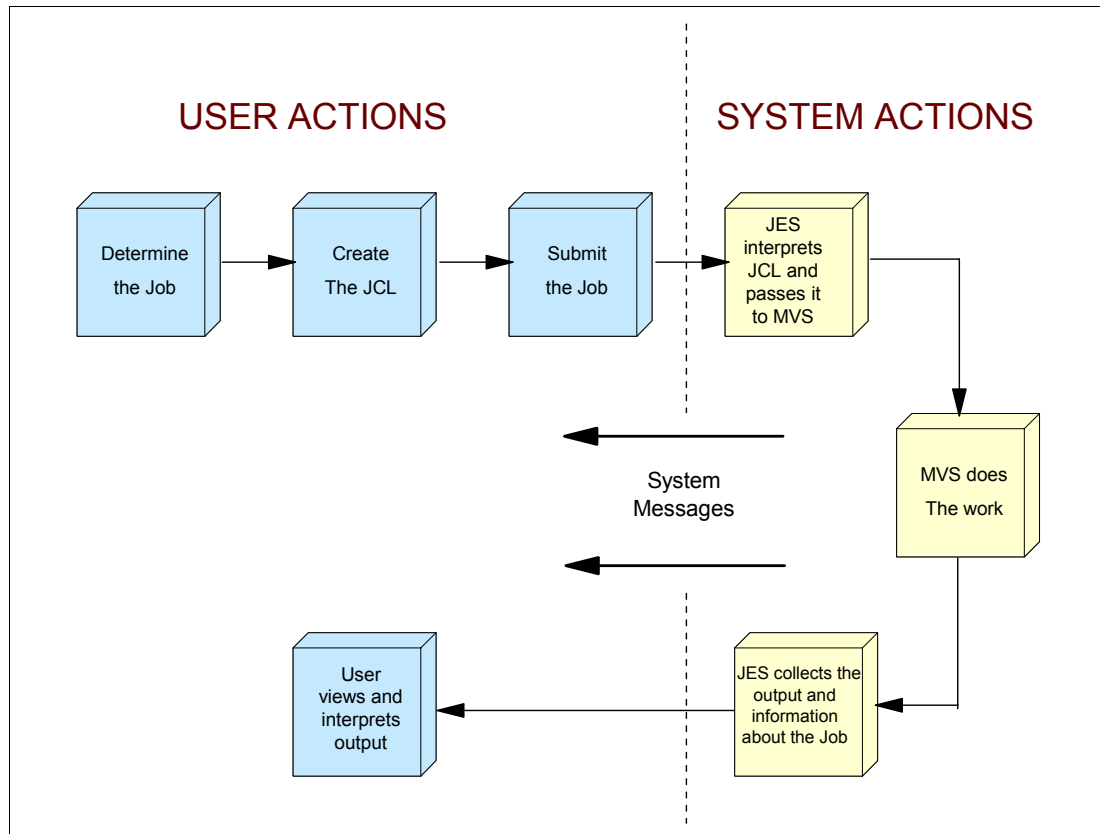


Figure 3-3 JCL-related actions

#### Job control statements

For your program to execute on the computer and perform the work you designed it to do, your program must be processed by your operating system. Your operating system consists of an MVS base control program (BCP) with a job entry subsystem (JES2 or JES3) and DFSMSdfp 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, which consist of:

- ▶ JCL statements
- ▶ JES2 control statements
- ▶ JES3 control statements

In general, statements in the MVS Job Control Language (JCL) and in the Job Entry Control Language (JECL) for JES2 and JES3 subsystems are used by the users to define a job's execution steps and resource requirements. The MVS system creates an internal format of the job's JCL statements before it can process it. MVS converter/interpreter transforms the external (JCL) format job description 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 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.

## 3.4 JES2 and JES3 main differences

### ❑ JES2

- Each JES2 is independent
- Device allocation by MVS

### ❑ JES3

- Global JES3 and Local JES3
- Global JES3 controls all:
  - Job selecting
  - Job scheduling
  - Job output
- Installation can choose device allocation
  - Global JES3 or MVS
- Workload balancing according to resource requirements of jobs

Figure 3-4 JES2 and JES3 main differences

### JES differences

You can choose between two options for your primary JES: JES2 and JES3.

If your installation has only one MVS image, then both JES2 and JES3 perform similar functions, with JES3 providing additional functions related to job scheduling. The common functions executed by both JESes are reading jobs into the system, converting them to internal machine-readable form, select them for processing, processing their output, and purging them from the system.

If your installation has more than one MVS image in a configuration, then the scheduling functions of JES3 are a big advantage. If you have more than one MVS image and you are running JES2, then JES2 in each MVS image exercises independent control over its job processing functions. That is, within the configuration, each JES2 processor controls its own job input, job scheduling, and job output processing.

In a JES3 environment, however, one MVS image hosts a JES3 that performs centralized control over its and the other MVS images' functions. This JES3 is called *JES3 global processor*; the JES3 instances in the other MVS images are called *JES3 local processors*.

It is from the global processor that JES3 manages jobs and resources for the entire complex, matching jobs with available resources. JES3 ensures that they are available before selecting the job for processing.

To better understand the functional differences between JES2 and JES3, refer to *z/OS JES2 Initialization and Tuning Guide*, SA22-7532, and *z/OS JES3 Initialization and Tuning Guide*, SA22-7549.

## 3.5 JES2 primary job entry subsystem

- ❑ JES2 functions
- ❑ JES2 job flow
- ❑ JES2 spool data set
- ❑ JES2 checkpointing
- ❑ JES2 configuration
- ❑ JES2 customization
- ❑ JES2 exits
- ❑ How to start JES2
- ❑ How to stop JES2
- ❑ JES2 operation

*Figure 3-5 JES2 primary job entry subsystem*

### What is JES2?

MVS uses a JES to receive jobs into the operating system, schedule them for processing by MVS, and control their output processing. JES2 is descended from HASP (Houston automatic spooling priority). HASP is defined as a computer program that provides supplementary job management, data management, and task management functions such as scheduling, control of job flow, and spooling. HASP remains within JES2 as the prefix of most module names and the prefix of all messages sent by JES2 to the operator.

JES2 is a functional extension of the HASP II program that receives jobs into the system and processes all output data produced by the job. So, what does all that mean? Simply stated, JES2 is that component of MVS that provides the necessary functions to get jobs into, and output out of, the MVS system. It is designed to provide efficient spooling, scheduling, and management facilities for the MVS operating system.

But, none of this explains why MVS needs a JES. Basically, by separating job processing into a number of tasks, MVS operates more efficiently. At any point in time, the computer system resources are busy processing the tasks for individual jobs, while other tasks are waiting for those resources to become available. In its most simple view, MVS divides the management of jobs and resources between the JES and the base control program of MVS. In this manner, JES2 manages jobs before and after running the program; the base control program manages them during processing.

The following sections describe each JES2 function identified in Figure 3-5.

## 3.6 JES2 functions

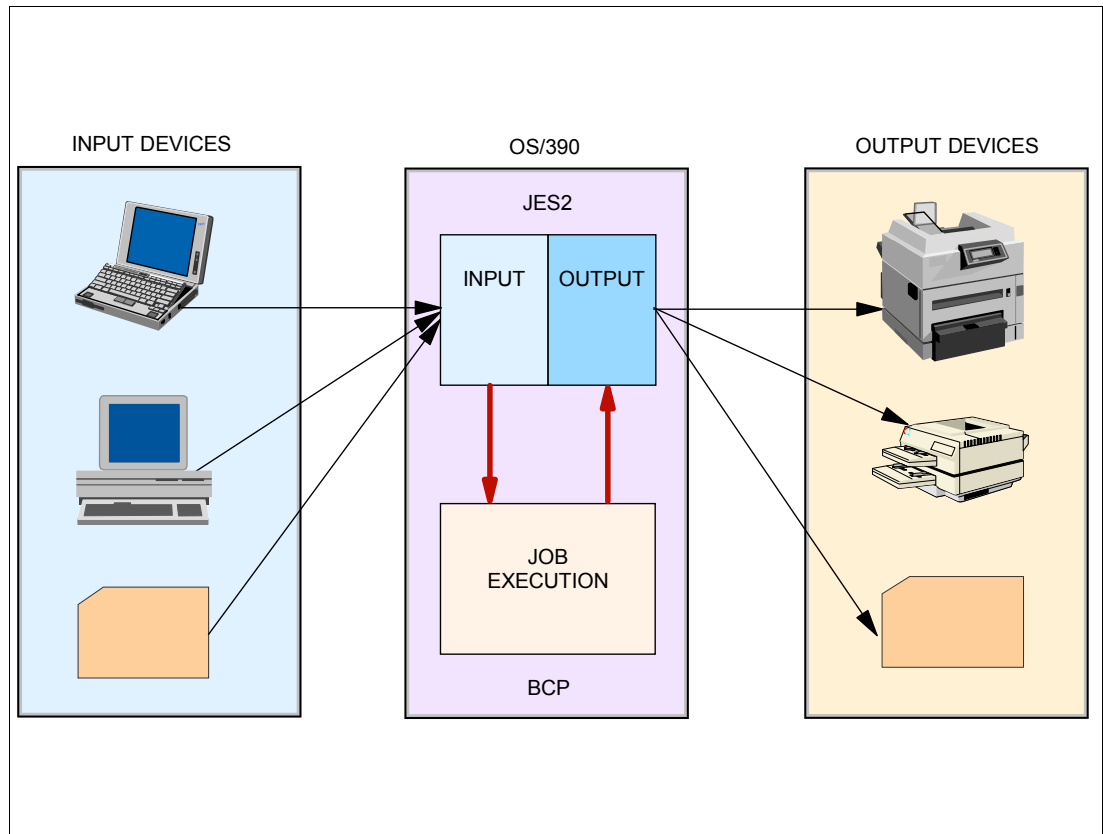


Figure 3-6 JES2 functions

### 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. Generally speaking, a job goes through the following phases:

- ▶ Input
- ▶ Conversion
- ▶ Processing
- ▶ Output
- ▶ Print/punch (hard copy)
- ▶ Purge

Except for processing, all the job phases are controlled by JES2.

### JES2 functional areas

To manage job input/output, JES2 controls a number of functional areas, all of which you can customize according to your installation's need. Some of the major functional areas and processing capabilities are:

- ▶ Getting work into and out of MVS (input/output 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 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. As shown in Figure 3-6, you can also enter jobs to JES2 using 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. It can be specific for each output device (local and remote printers and punches and offload devices). The work selection criteria are defined in the device initialization statements and can be changed dynamically using JES2 commands. You can define:

- Specification of job and output characteristics that JES2 considers when selecting work for an output device
- Priority of the selection characteristics
- 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 may 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.

► Offloading work and backing up system workload:

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

- Preserve jobs and SYSOUT across a cold start
- Migrate your installation to another release of JES2
- Convert to another DASD type for 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 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 available for this purpose. A more complete security policy can be implemented with System Security Facility (SAF) and a security product such as RACF.

► Supporting advanced function printers (AFPs)



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

Figure 3-7 on page 122 presents an overview of the job phases.

## 3.7 JES2 job flow

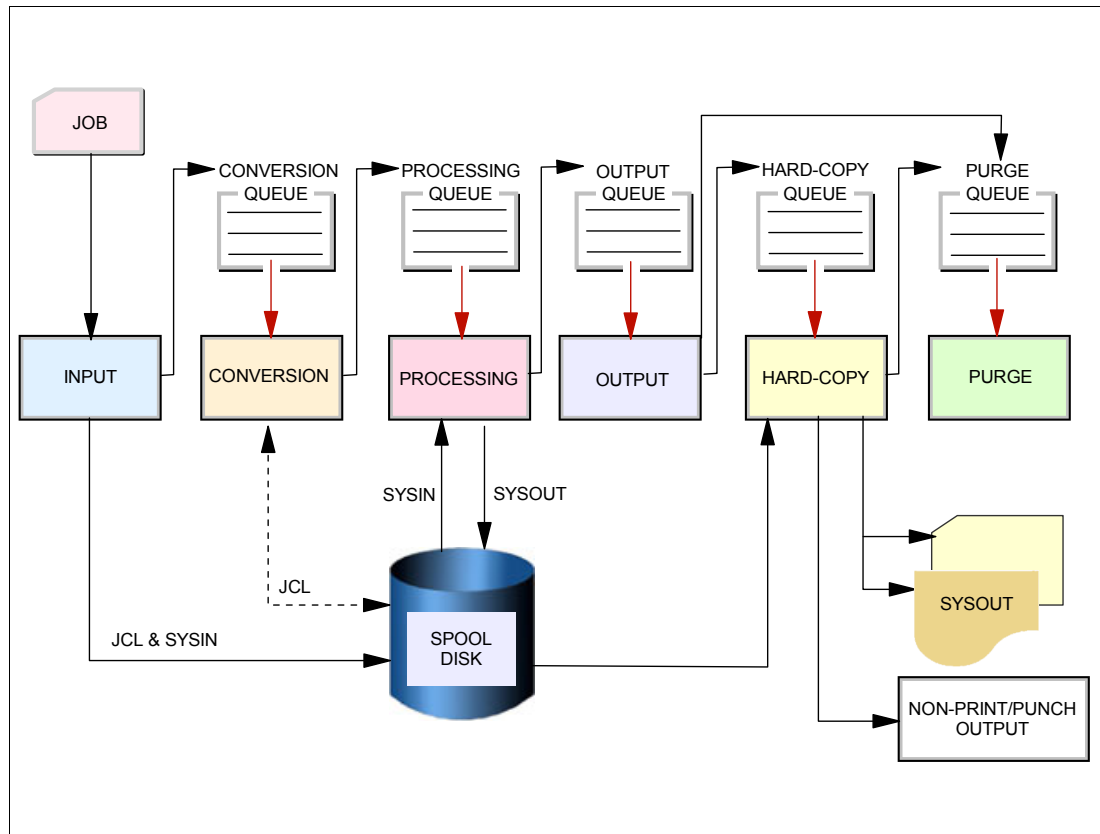


Figure 3-7 JES2 job flow

### JES2 job queue

The job queues contain jobs that are:

- ▶ Waiting to run - conversion queue
- ▶ Currently running - execution queue
- ▶ Waiting for their output to be produced - output queue
- ▶ Having their output produced - hard-copy (print/punch) queue
- ▶ Waiting to be purged from the system - purge queue

### Job processing phases

The six phases of job processing are as follows:

#### 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 MVS can use an internal reader to pass an input stream to JES2. JES2 can receive multiple jobs simultaneously through multiple internal readers. MVS uses internal readers, allocated during system initialization, to pass to 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/user procedure libraries can be defined in the PROCxx DD statement of the JES2 startup procedure. Then, JES2 converts the composite JCL into converter/interpreter text that both JES2 and the job scheduler functions of MVS can recognize. Next, JES2 stores the converter/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 MVS initiators. An initiator is a system program that is controlled by JES or by WLM (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 in order 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 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 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/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 MVS Basic Control Program (BCP) communicate constantly to control system processing. The communication mechanism, known as the subsystem interface, allows MVS to request services of JES2. For example, a requestor can ask JES2 to find a job, message or command processing, or open (access) a SYSIN or SYSOUT data set. Further, the base control program 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. MVS directs to sysout system messages 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 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.

## 3.8 JES2 spool data sets

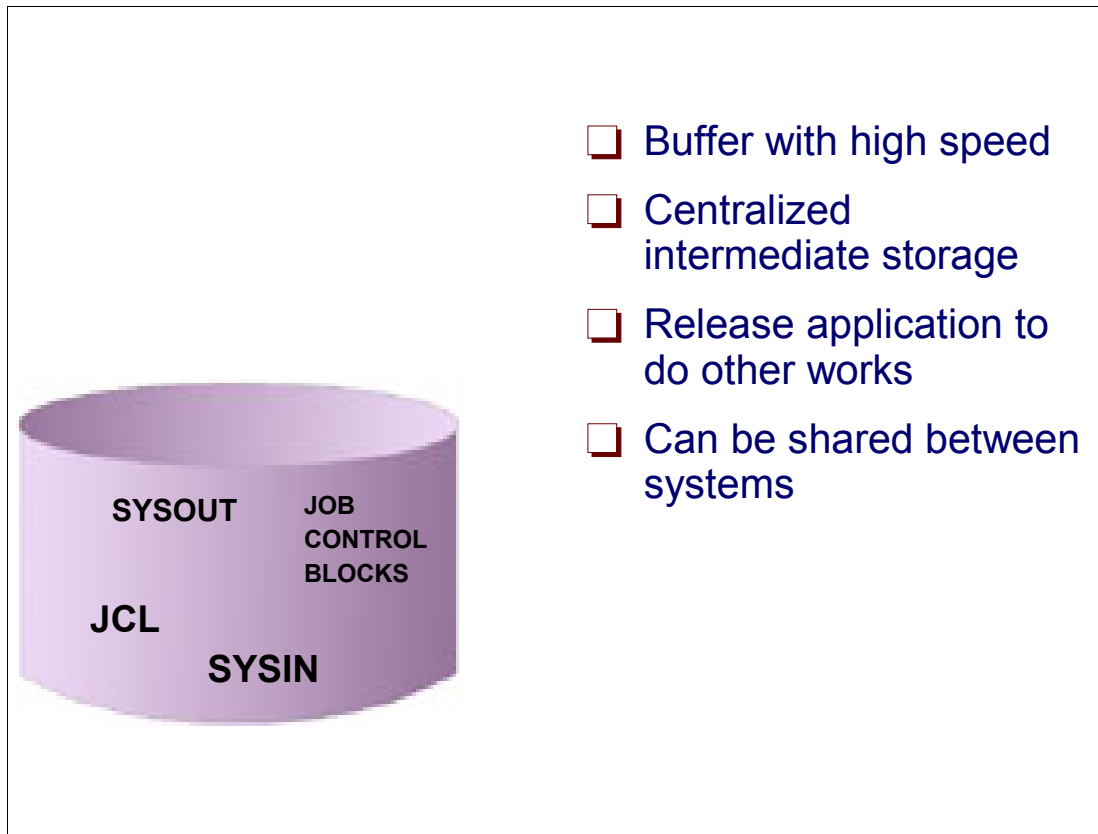


Figure 3-8 JES2 spool data sets

### JES2 spool data sets

JES2 maintains copies of its data sets that contain job and output queues (that is, lists of jobs to be processed by MVS) on direct access storage devices (DASD). Future work is added to these queues and JES2 selects work for processing from them. These data sets and queues must remain current and accurate to maintain system integrity and performance. The following is a discussion of the JES2 spool and checkpoint data sets and the processing JES2 uses to maintain them.

### Spool data sets and spooling

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

- ▶ Using storage on direct access storage devices (DASD) as a buffer storage to reduce processing delays when transferring data between peripheral equipment and a program to be run.
- ▶ Reading and writing input and output streams on an intermediate device for later processing or output.
- ▶ Performing an operation such as printing while the computer is busy with other work.

Spool also refers to the direct access device that contains the spool data sets. This definition is generally apparent from the context of its use and should not cause any misunderstanding in the following sections of this book or other JES2 documentation.

Spooling is critical to maintain performance in the MVS-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. Once 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 MVS-JES2 environment.

## 3.9 JES2 checkpoint data set

- ❑ Backup of the jobs and output queues
- ❑ Accessible for all member of the MAS complex
- ❑ Communication among all members in the MAS
- ❑ Maintain system integrity

*Figure 3-9 JES2 checkpoint data set*

### JES2 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. The checkpoint data set, checkpointing, and the checkpoint reconfiguration dialog all help to make this possible.

The checkpoint is the data set that JES2 maintains on either DASD or a Coupling Facility that permits JES2 to perform two separate functions:

1. Job and output queue backup to ensure ease of JES2 restart, since it contains information about what work is yet to be processed and how far along that work has progressed.
2. In a multi-access spool (MAS) environment, member-to-member workload communication to ensure efficient independent JES2 operation.

JES2 periodically updates the checkpoint data set by copying the changed information from the in-storage copy to the checkpoint data set copy. Information in the checkpoint data set is critical to JES2 for normal processing and also if a JES2 or system failure occurs.

### Taking checkpoints

Taking checkpoints is the periodic copying of a member's in-storage job and output queues to the checkpoint data set. In a MAS environment, 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 maintained by JES2.

## Defining checkpoint configuration mode

There are three ways of specifying the checkpoint configuration mode. The type of mode you select depends upon your JES2 configuration:

- ▶ **DUPLEX-mode processing with backup:**

JES2 can operate its checkpoint configuration in DUPLEX mode if you have defined two checkpoint data sets, a primary and a duplex. They can be either in DASD or a Coupling Facility. The duplex data set is updated once for every write to the primary. If the primary data set suffers an error, the duplex can provide either an exact duplicate of the primary or a very similar, almost current (depending on when it was last updated) copy of the checkpoint data.

It is recommended that all members in the JES2 configuration operate, if possible, with DUPLEX=ON in the CKPTDEF. This provides the greatest protection from checkpoint error.

- ▶ **DUPLEX-mode processing without backup:**

Processing with a single checkpoint data set is not recommended in most cases. If that data set becomes damaged, lost, or suffers an I/O error, and your member experiences a concurrent failure, you will not have a checkpoint data set available to restart JES2. But you have the benefit that JES2 will not need to read/write to two data sets. However, depending on your specific volume or structure use and configuration, it might not be practical for particular members of your JES2 configuration to maintain a backup checkpoint (for example, in systems that manage basically online transactions).

- ▶ **DUAL-mode processing:**

This configuration also uses two data sets, but in a “flip-flop” scheme; that is, individual members do not always read and write to the same CKPTn data set. However, the member performing a read always uses the checkpoint data set last updated. This is not required in a single-member environment. However, it is recommended in a MAS environment, if the installation is suffering from degraded system performance due to high checkpoint data set I/O. The use of the change provides reduced I/O to the checkpoint data set during most, if not all, update accesses by a member of the multi-access spool configuration.

**Note:** This mode cannot be used if any checkpoint data set resides on a Coupling Facility structure.

## Checkpoint reconfiguration dialog

JES2 provides a dynamic means by which the current checkpoint configuration can be changed. It is the checkpoint reconfiguration dialog, which can be initiated by the operator system or by JES2. JES2 will enter a checkpoint reconfiguration for any of these reasons:

- ▶ To complete initialization processing:
  - Either JES2 could not 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 move the checkpoint off a volatile Coupling Facility structure



## 3.10 JES2 configurations

- ☐ Single-processor
- ☐ Multiple-system (multi-access spool)
- ☐ Poly-JES
- ☐ Remote job entry (RJE)
- ☐ Network job entry (NJE)

*Figure 3-10 JES2 configurations*

### JES2 configurations

The following are possible JES2 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 multi-access spool (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 z/Series 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 utilizes synchronous data link control (SDLC) or binary synchronous communication (BSC) protocols for communicating between JES2 and remote devices. The remote device will be either a system network architecture (SNA) remote, which uses SDLC, or a BSC remote, which uses BSC. Figure 3-11 on page 131 shows a simple RJE configuration.

- ▶ 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.

### 3.11 JES2 example of an RJE configuration

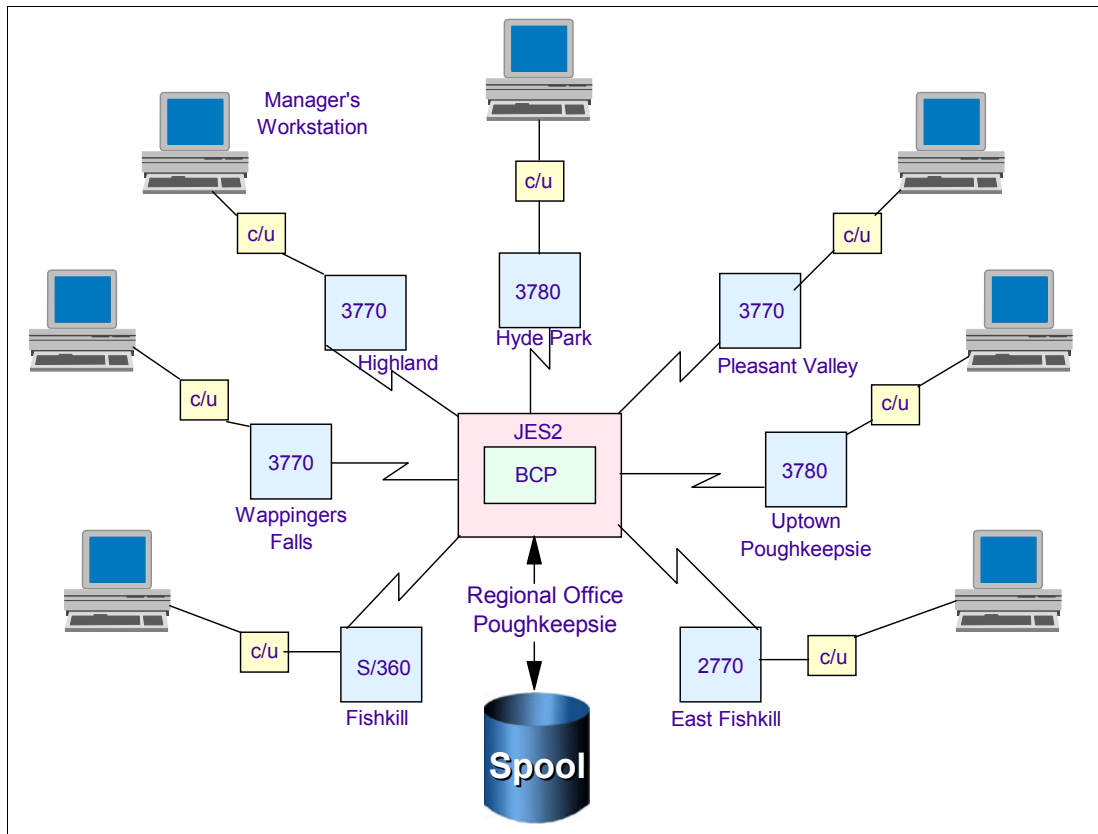


Figure 3-11 JES2 example of a RJE configuration

#### JES2 RJE configuration

An RJE workstation is an extension of the local processing facility, except that work is sent across teleprocessing lines. Sending work across teleprocessing lines is convenient for an installation that needs to provide many data entry points at remote locations or route output to many diverse locations. The following examples illustrate the use of RJE in two familiar daily business scenarios:

- ▶ In a large department store, many sales clerks need to print output on printers located at the many customer check-out desks throughout the store. Because the location of the building prevents direct connection to the MVS system located in the central office located several hundred miles away, each printer can be defined to JES2 as a remote printer.
- ▶ Consider a clothing store chain in which the store managers at seven different stores all need to place orders, access inventory, and provide billing information, all the information for which is maintained on storage devices located in the computer center at the main office. An individual store's workstation is defined to JES2 as an SNA-attached remote to which the individual terminals are defined and as such become an extension of the JES2 configuration located at the main office.

As depicted in Figure 3-11, each of the clothing stores within a localized region of New York (the Hudson Valley towns of Highland, Hyde Park, Pleasant Valley, and so forth) is connected to the single processor located at the regional office in Poughkeepsie. One MVS/JES2 system conducts the business of inventory control, shipping, and billing for all of the stores in the region.

JES2 processes remote jobs in the same manner as those received from a local reader. (Local devices are printers, punches, card readers, and lines directly attached to the system without the need of transmission facilities.) The terminals and printers located in the Poughkeepsie office are locally attached; all other I/O devices (terminals and printers) in all the other branch stores are defined to JES2 as remote terminals and printers.

To provide RJE processing, the RJE workstation must be defined to the local processor. There are two protocols available by which JES2 can communicate with the RJE workstations: synchronous data link control (SDLC) and binary synchronous communication (BSC).

## **RJE workstations**

An RJE workstation can have a processor, like the System/370™, that runs a JES2-generated program. The JES2-generated program allows the processor to send jobs to, and receive data from, JES2. Such RJE workstations have generally been replaced by either a programmable workstation (such as a personal computer) or a network job entry configuration, and they are rarely used in today's processing environment. Some RJE workstations do not have a processor. These workstations use a remote terminal, for example, a 2780 or 2770, to enter jobs into, and receive data from, JES2.

Refer to *z/OS JES2 Initialization and Tuning Guide*, SA22-7532 for a more complete discussion of RJE concepts.

### 3.12 JES2 example of an NJE configuration

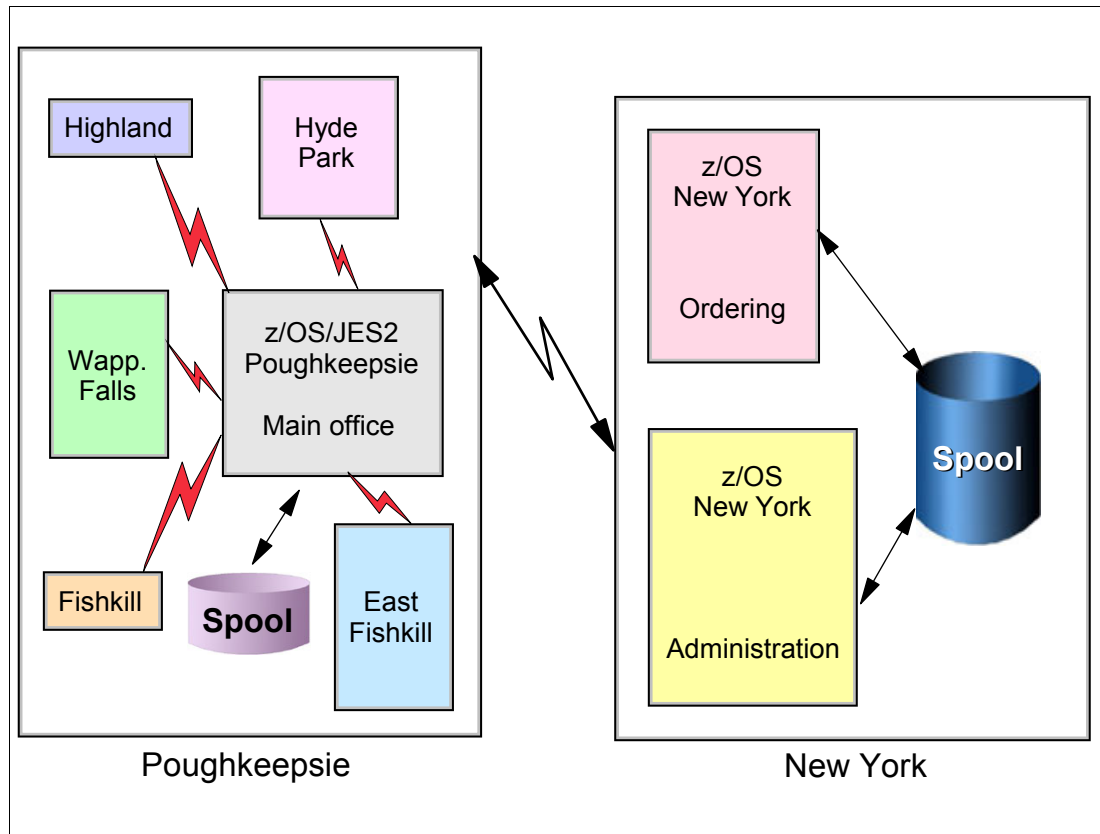


Figure 3-12 JES2 example of an NJE configuration

#### NJE configuration example

The JES2 network job entry (NJE) facility is similar to remote job entry (RJE) in that they both provide extensions to a computer system.

NJE network nodes communicate (send and receive data) using various teleprocessing facilities. Nodes on the same physical processor use the Virtual Telecommunications Access Method (ACF/VTAM) program product to communicate; no hardware is required. Nodes located in the same room or building can utilize channel-to-channel adapters (CTCA) or telecommunication links. Nodes that are geographically dispersed utilize SNA or BSC telecommunication links. The following example further explains this concept.

If we return to the previous example of the clothing store chain, the Poughkeepsie regional office is the location of the processor (MVS and JES2); each of the sites (Highland, Hyde Park, Pleasant Valley, Fishkill, and so forth) is attached to the Poughkeepsie office as a remote site. Together, all the locations shown in Figure 3-11 comprise the Poughkeepsie node. For a locally owned, relatively small clothing store chain such as the one depicted, this complex may suffice. However, for a national clothier or one dealing in the import/export business, one processor would likely prove to be inadequate. Such a company would likely elect to establish many nodes throughout the world, each connected to all others in an NJE complex. Refer to Figure 3-12 for a diagrammatic view of such a network. Note that the different groups of the ordering/billing department and the business administration/payroll department can be separate members of a MAS making up the New York node.

Also note that the New York node is a multi-access spool configuration; Poughkeepsie is the configuration illustrated in Figure 3-11 on page 131, and all the other nodes located around the world (which are not depicted) are simple single-system configurations. In the network, a store manager in Pleasant Valley can order an item, the Poughkeepsie office (the main office for the Hudson Valley region) tallies the orders from this store and its other six stores and submits its order to the clothier chain headquarters located in New York City. The request and subsequent confirmation of the order is not instantaneous due to the distance and the traffic in the system but is a faster and a more efficient method of doing business than a telephone conversation, particularly if you further consider time zone differences. *Traffic* refers to the number of users, requests, jobs, and data currently being routed across available teleprocessing lines.

## 3.13 JES2 customization

- ❑ Meet installation's needs
  - Initialization data set
- ❑ IBM-defined exits
- ❑ Installation-defined exits
- ❑ Table pairs

Figure 3-13 JES2 customization

### JES2 customization

JES2 is designed to be tailored to meet an installation's particular processing needs. You can address your 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.

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 implemented through direct source code modification are error prone, counter-productive during migration to future releases, and can prove to be very time-consuming when debugging, diagnosing, and applying IBM-written fixes (program temporary fixes (PTFs) and authorized program analysis report (APARs) fixes) to code. 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.

Recommended methods for tailoring JES2 processing include JES2 table pairs, IBM-defined exits, and installation-defined exits. A general discussion of each is provided later in this chapter. Refer to *z/OS JES2 Installation Exits*, SA22-7534-03 for a complete description of each IBM-defined exit.

### JES2 initialization data set

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 tailorable, 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.AHASAMP 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.

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.

## 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
- ▶ 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.VnRnMn.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 would experience if users were allowed to define their own job classes and priorities. Very 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 3-1 identifies the available JES2 initialization statements.

*Table 3-1 JES2 initialization statements*

| Initialization statement | Function                                                                                        |
|--------------------------|-------------------------------------------------------------------------------------------------|
| APPL(avvvvvvv)           | 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(s) 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.                                     |



| Initialization statement | Function                                                                                                                                                                                                   |
|--------------------------|------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| 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(jxxxxxxx)         | Defines a destination name (mostly for end-user use, as on a JCL statement) for a remote terminal, another NJE node, or a local device.                                                                    |
| D MODULE(jxxxxxxx)       | 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 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 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 taken.                                   |
| EXIT(nnn)                | Associates the exit points defined in JES2 with installation exit routines.                                                                                                                                |
| FSS(cccccccc)            | Specifies the functional subsystem for printers that are supported by an FSS (for example Print Services Facility).                                                                                        |
| INCLUDE                  | Allows new initialization data sets to be processed.                                                                                                                                                       |
| INIT(nnn)                | Specifies the characteristics of a JES2 logical initiator.                                                                                                                                                 |
| INITDEF                  | Specifies the number of JES2 logical initiators to be defined.                                                                                                                                             |
| INTRDR                   | Specifies the characteristics of all JES2 internal readers.                                                                                                                                                |
| 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.                                                                                                                        |
| JOBPRTY(n)               | Specifies the relationship between job scheduling priorities and job execution time.                                                                                                                       |
| LINE(nnnn)               | Specifies the characteristics of one teleprocessing line or logical line (for SNA) to be used during remote job entry or network job entry.                                                                |
| L(nnnn).JT(m)            | Specifies the characteristics for a job transmitter on an NJE line.                                                                                                                                        |
| L(nnnn).ST(m)            | Specifies the characteristics for a SYSOUT transmitter on a line defined for network job entry.                                                                                                            |
| LOADMOD(jxxxxxxx)        | 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.                                                                                                                                                         |

| Initialization statement | Function                                                                                                            |
|--------------------------|---------------------------------------------------------------------------------------------------------------------|
| 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(nnnn)               | 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.                                                          |
| 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(nnnn)                | Specifies the characteristics of a local printer.                                                                   |
| PUNCHDEF                 | Defines the JES2 punch environment.                                                                                 |
| PUN(nn)                  | Specifies the characteristics of a local card punch.                                                                |
| R(nnnn).PR(m)            | Specifies the characteristics of a remote printer.                                                                  |
| R(nnnn).PU(m)            | Specifies the characteristics of a remote punch.                                                                    |
| R(nnnn).RD(m)            | Specifies the characteristics of a remote card reader.                                                              |
| RDR(nn)                  | Specifies the characteristics of a local card reader.                                                               |
| RECVOPTS(type)           | Specifies the error rate below which the operator will not be involved in the recovery process.                     |
| REDIRect(vvvvvvv)        | 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.                                                |

| Initialization statement | Function                                                                                        |
|--------------------------|-------------------------------------------------------------------------------------------------|
| RMT(nnnn)                | 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(nnn)                 | 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(s) 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.

## JES2 table pairs

Table pairs provide a facility to change, delete, or add to JES2 processing and/or function. Changes made to JES2 processing using table pairs are generally less prone to error than are changes made through installation exits. This is true because JES2 macros generate the tables and generally require you to write less code to be run.

A number of JES2 functions (such as initialization statement processing, command processing, and message building) use tables. You can customize these JES2 functions, and others, by extending their associated tables. JES2 examines two tables, known as a table pair. The first table (the JES2 table) provides the default processing specifications; the second table (the user table) is used to extend, change, or delete the default processing specifications. For example, you can add your own JES2 commands and messages, add a new initialization statement or parameter, abbreviate the length of a JES2 command, or delete an unnecessary command to prevent its accidental misuse.

To simplify the use of this facility you can use the JES2 default tables as templates for construction of installation-written tables. Depending on the table(s) from which you choose to add, change, or delete, using table pairs generally takes less detailed knowledge of JES2 internal structure than does writing an exit.

Table pairs do not replace the need for exits. Table pairs and exit points can provide added capability either independently or in conjunction with one another.

## 3.14 JES2 exits

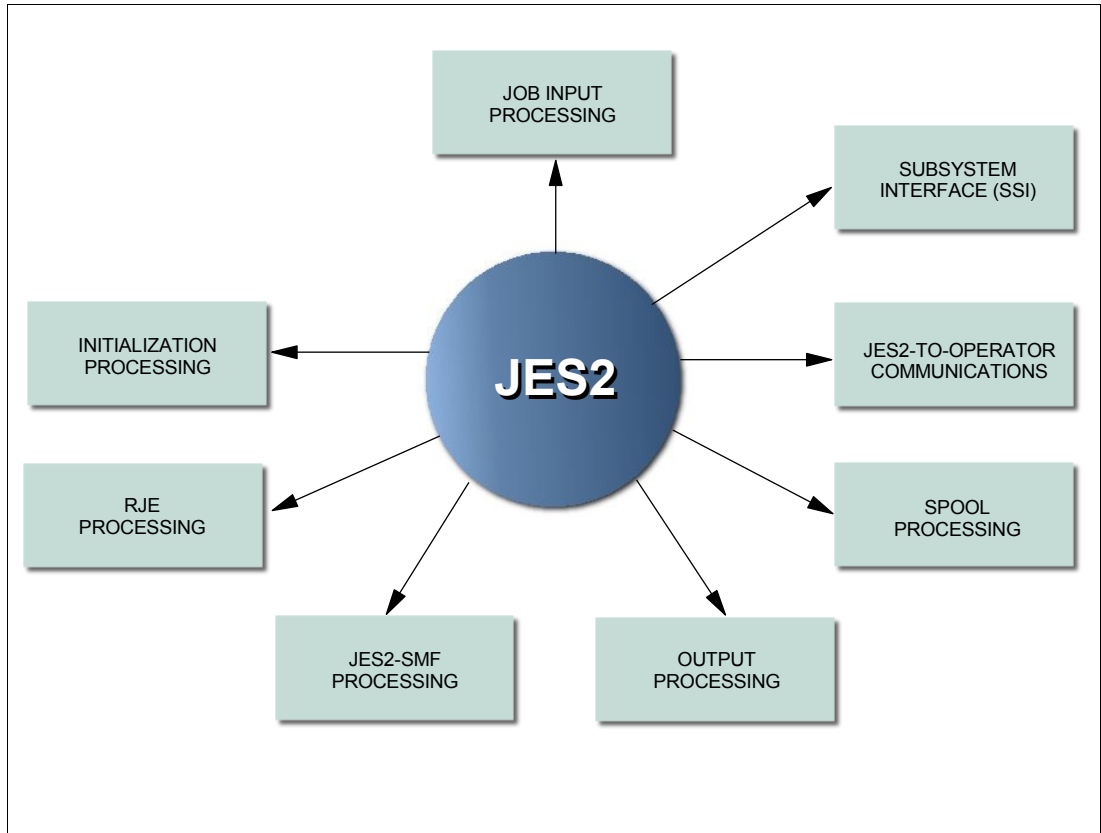


Figure 3-14 JES2 exits

### JES2 exits

JES2 may not satisfy all installation-specific needs at a given installation. When you modify JES2 code to accomplish your 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.

- ▶ 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 your 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(*nnn*) 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.

The IBM-defined exits can be classified into two 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.

## 3.15 JES2 input-related exits

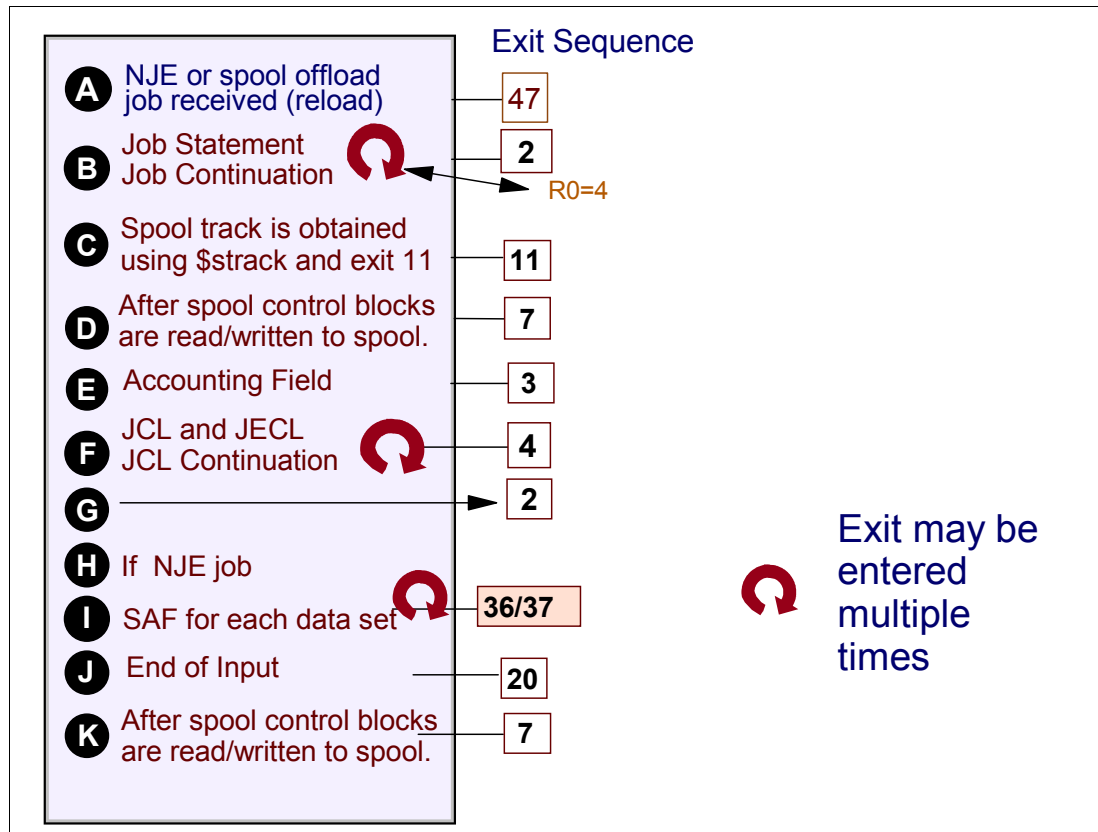


Figure 3-15 JES2 input-related exits

### JES2 input exits

Jobs that come in through the input phase go through the following exits:

- **Specific purpose exits** provide specific functions. These exits do not occur at predictable intervals during the life of a job. For that reason, they are not appropriate for general-purpose use. Examples of specific purpose exits are job output overflow (Exit 9) and spool partitioning exits (Exits 11 and 12).
- **General purpose exits** are usually considered when there is a user requirement to control installation standards, job resources, security, output processing, and other job-related functions.

There are two major considerations when selecting an exit to satisfy user requirements:

1. The environment of the exit

This determines the addressable data areas, the facilities available when the exit is taken, and so forth.

2. The sequence of the exits

Which exits precede and which exits follow each exit? What processing has preceded and followed the exit? It is very important to avoid some exit processing in order to avoid being overridden by another exit's processing.

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 3-2 on

page 143 and Table 3-3 on page 145 list the job-related exits in order 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 3-2 JES2 job-related exits

| Exit | Exit title                                             | Comments and some specific uses                                                                                                                                                                                               |
|------|--------------------------------------------------------|-------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| 1    | Print/punch job separator                              | Taken when a job's data sets have been 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 has been 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.                                                                                                                                                       |
| 31   | SSI Data set Allocation                                | Taken for SYSIN, SYSOUT, or internal reader Allocation processing.<br>Uses:<br>Fail an allocation.<br>Affect how JES2 processes data set characteristics.                                                                     |
| 32   | SSI Job Selection                                      | Taken after all job selection processing is complete.<br>Uses:<br>Suppress job selection-related messages.<br>Perform job-related processing such as allocation of resources and I/O for installation-defined control blocks. |

| Exit | Exit title                         | Comments and some specific uses                                                                                                                                                                                                                |
|------|------------------------------------|------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| 33   | SSI Data Set Close                 | Taken for SYSIN, SYSOUT, or internal reader Close processing.<br>Uses:<br>Free resources obtained at OPEN.                                                                                                                                     |
| 34   | SSI Data Set Unallocation - Early  | Taken for SYSIN, SYSOUT, or internal reader early in allocate processing.<br>Uses:<br>Free resources obtained by Exit 30.                                                                                                                      |
| 35   | SSI End-of-Task                    | Taken at end of each task during job execution.<br>Uses:<br>Free task-related resources.                                                                                                                                                       |
| 36   | Pre-SAF                            | Taken just prior to JES2 call to SAF.<br>Uses:<br>Provide/change additional information to SAF.<br>Eliminate call to SAF.                                                                                                                      |
| 37   | Post-SAF                           | Taken just after the return from the JES call to SAF.<br>Uses:<br>Change the result of SAF verification.<br>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.<br>Uses:<br>Change the destination of a SYSOUT data set.<br>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.<br>Uses:<br>Change fields in the \$JQE and \$JCT.                                                                                                       |
| 46   | NJE Transmission                   | Taken for NJE header, trailer, and data set header during NJE job transmission.<br>Uses:<br>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.<br>Uses:<br>Remove/change installation-defined sections that were previously added to an NJE data area.                                                           |
| 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.<br>Uses:<br>Provide an algorithm to accept or not accept a JES2-selected job.<br>Control WLM initiator job selection.                                                        |



Table 3-3 Some SMF job-related exits

| Exit     | Exit Title          | Comments and some specific uses                                                                                                                                                                                                                        |
|----------|---------------------|--------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| IEFUJV   | SMF Job Validation  | Receives control:<br>Before each JCL statement is interpreted, and<br>After all the JCL is converted, and again<br>After all the JCL is interpreted.                                                                                                   |
| IEFUJI   | SMF Job Initiation  | Receives control before a job on the input queue is selected for initiation.<br>Uses:<br>Selectively cancel the job.                                                                                                                                   |
| IEFUSI   | SMF Step Initiation | Receives control before each job step is started (before allocation).<br>Uses:<br>Limit the user region size.                                                                                                                                          |
| IEFACTRT | SMF Job Termination | Receives control on the termination of each step or job.<br>Uses:<br>Decide whether the system is to continue the job (for step job).<br>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.<br>Uses:<br>Selectively decide whether the SMF job purge record (type 26) is to be written to the SMF data set. |

## 3.16 JES2 exits in conversion phase

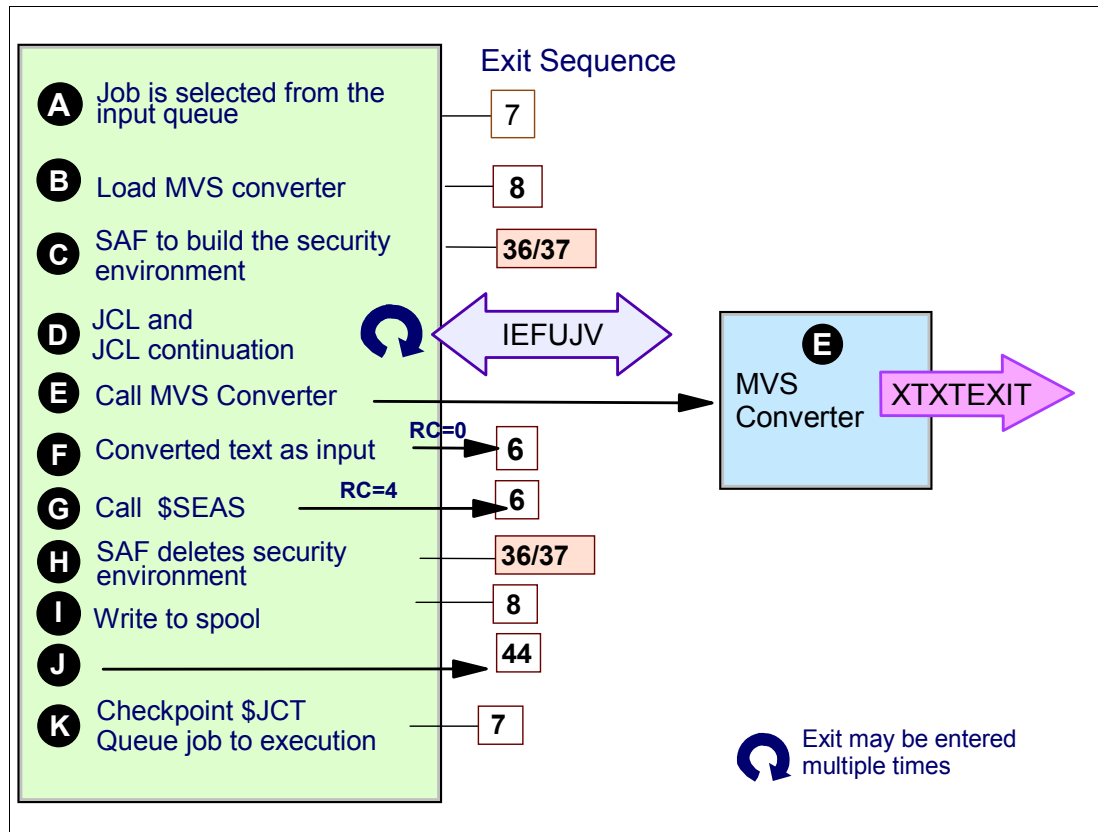


Figure 3-16 JES2 exits in conversion phase

### JES2 conversion exits

The conversion phase of JES2 processing is accomplished in two environments. First the Converter Processor Control Element (\$PCE) is dispatched in the JES2 maintask environment to select a job from the input queue. Secondly, the Converter subtask, after being posted by the Converter maintask, calls the MVS Converter to do the actual conversion (JCL to C/I text). The reason for the subtask environment is that the conversion process requires the reading of the JCL data set from spool, reading JCL from PROCLIB, writing JCL images to spool, and the writing of C/I text to spool. These I/O operations cannot be accomplished in the maintask environment.

It's important to understand the difference in these two environments when considering exit usage. Exit 7 executes in the maintask environment, and Exit 6, and the SMF IEFUJV exit, execute in the subtask environment. If maintask functions are needed for a subtask exit, it may be necessary to use two exits, for example Exits 6 and 44 in conjunction, to provide a specific function.

Another important consideration is that there can be (and usually are) more than one converter processor (and subtask) and therefore, any exits taken in the subtask (Exits 6 and exit, IEFUJV) must be MVS reentrant. The scenario illustrated in Figure 3-16 is described by the conversion processing presented in Table 3-4.

Table 3-4 Conversion Phase Processing

| Step | Processing                                                                                                                                                                                                                                                                                                                                                                             | Exit used          |
|------|----------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|--------------------|
| A    | A job is selected from the input queue, and the job's \$JCT is read from spool. Exit 7 is invoked with a value of zero in general register zero (R0=0). The Daughter Task Element (DTE) is initialized and the Converter subtask is POSTed.                                                                                                                                            | 7                  |
| B    | The JES2 conversion subtask locates the job's \$PDDBs (JES2 Peripheral Data Definition Blocks) and Fake Opens the ACBs (Access Control Blocks) for internal text, job log, system messages, JCL, and JCL images data sets. The Converter subtask LOADs the MVS Converter, if the Converter has not already been loaded. Exit 8 is taken for reading the \$IOTs from spool.             | 8                  |
| C    | The Security Access Service (\$SEAS) macro calls the Security Authorization Facility (SAF) to build the security environment in case the job stream contains MVS commands which if present, would be issued by the Converter using the Command SVC. The user ID associated with the command would be the user's, not JES2. As a result of the \$SEAS call, Exits 36 and 37 are called. | 36<br>37           |
| D    | For each JCL image, SMF exit IEFUJV (entry codes 0, 4, 8, and 64) is taken. This includes continuation statements. IEFUJV is called once more with an entry code of 16.                                                                                                                                                                                                                | SMF exit<br>IEFUJV |
| E    | After the statement and all continuation statements have been converted into C/I text, the Converter exit, XTXTXIT is called to provide spool data set names for SYSIN and SYSOUT JCL statements. If the statement represents a SYSIN data set, a \$SEAS call is made to audit the creation.                                                                                           | XTXTXIT            |
| F    | After the spool data set names have been generated (if SYSIN or SYSOUT), Exit 6 is invoked (R0=0) with the completed C/I text statement as input to the exit.                                                                                                                                                                                                                          | 6                  |
| G    | At the completion of conversion and after the Converter returns to the JES2 converter processor module, a \$SEAS call is issued to delete the security environment. Exit 6 (R0=4) is taken again to allow final processing.                                                                                                                                                            | 6                  |
| H    | As a result of the \$SEAS call, Exits 36 and 37 are called.                                                                                                                                                                                                                                                                                                                            | 36<br>37           |
| I    | Exit 8 is taken to write the \$IOTs. The JES2 converter processor module subtask POSTs its maintask and WAITs for the next job.                                                                                                                                                                                                                                                        | 8                  |
| J    | Exit 44 is taken to allow user modifications that require the maintask environment. Using the \$DOGJQE macro you can access and optionally update fields in the JQE.                                                                                                                                                                                                                   | 44                 |
| K    | The JES2 converter processor module maintask checkpoints the \$JCT, invokes Exit 7, and queues the \$JQE to the execution job queue.                                                                                                                                                                                                                                                   | 7                  |

The conversion phase offers the only chance to have exit control over all of a job's JCL. Although SMF exit, IEFUJV is taken for each JCL and JCL continuation statement, JES2 Exit 6 offers some advantages.

First, the format of the C/I text is more structured. It is in parsed form and all major syntax errors have been removed. This has all been done by the converter before the exit gets control.

Another advantage of Exit 6 over IEFUJV is that once JCL statements have been converted into C/I text, there are no continuation statements. That is, the entire JCL statement, along with all continuation statements, are represented by a single C/I text statement.

A SAF security environment exists within the subtask and can be used with the RACF FACILITY class to control the specification of options within JCL. Exit 6 messages can be returned to the Converter to be issued by the Converter.

### 3.17 JES2 exits in execution phase

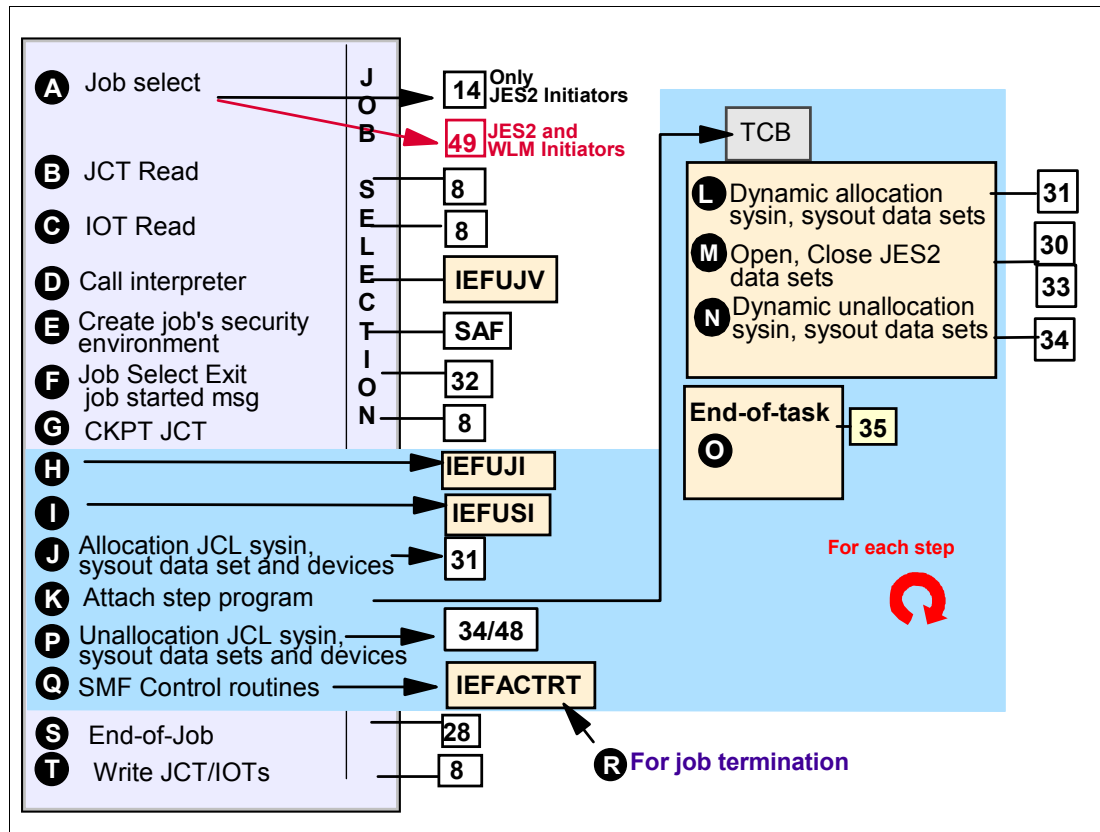


Figure 3-17 JES2 exits in execution phase

#### JES2 execution exits

This section attempts to merge those functions provided by a section of JES2 code in the JES2 Job Select/Termination module known as "Job Selection" and the pieces of MVS code known in the broad sense as the "Initiator." The MVS Initiator consists of many modules which perform job selection, allocation, and initiator attach services (and others). JES2 Job Select also includes end-of-job functions.

For the purpose of this discussion, job selection is defined as the period, starting with the initiator's Subsystem Interface (SSI) call for job selection by class and ending with the JES2 message, \$HASP373 JOB STARTED. Table 3-5 describes the processing that occurs during the Execution Phase, as depicted in Table 3-17.

Table 3-5 Execution phase exits

| Step | Processing                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                      | Exit used           |
|------|-------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|---------------------|
| A    | <p>The MVS Job Selection module issues an SSI call specifying function code 5, which identifies the call to JES2 as a request to select a job by class.</p> <p>SSI calls with a function code of 5 are processed by the JES2 Job Select/Termination module. JBSELECT POSTs JES2 execution processing and WAITs for a job to be selected.</p> <p>If a JES2 initiator is selecting work, JES2 calls Exit 14 to allow the your installation to provide its own queue selection routine or to tailor the selection request. Exit 14 is not a job-related exit, that is, JES2 has not selected a job at this time. Exit 14 can select a job or it can tell JES2 to select a job. If a WLM initiator is selecting work, JES2 does not call Exit 14.</p> <p>After JES2 selects a job from the execution queue, it calls Exit 49, which can accept or reject the job. If Exit 49 rejects the job, JES2 searches for another job. JES2 does not call Exit 49 if Exit 14 selects a job.</p> <p>If JES2 execution processing finds a job that matches the Initiator's defined job classes, it POSTs the waiting initiator and provides the job's \$JCT spool address in the \$\$SJB. If a job has been found, control is given to the JBFOUND routine.</p> | <p>14</p> <p>49</p> |
| B    | <p>The JBFOUND routine reads the job's JES2 \$JCT using the spool address passed in the \$\$SJB. Exit 8 is the first exit taken out of the user's (or job's) address space after a job is selected. This first entry to Exit 8 is taken after the job's \$JCT has been read. The job name and job-id are available as well as all other information in the \$JCT.</p> <p>If later SMF exits for this job need addressability to the JES2 \$JCT, store the JES2 \$JCT address (as contained in Exit 8 parameter list) into the JCTUCOM field that later becomes the JMRUCOM.</p>                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                 | 8                   |
| C    | <p>Exit 8 is again taken to read the primary allocation \$IOT. There also may be additional calls to Exit 8 to read secondary allocation \$IOTs and/or \$PDDDB-only \$IOTs based on the job's JCL. Exit 8 is called for all spool control block reads and writes.</p> <p>JES2 allows installations to create extensions to the \$JCT where job-related accounting data can be stored and transmitted through the network. Using the \$JCTX macro extension service, you can add, expand, locate, and delete these extensions.</p>                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                               | 8                   |
| D    | <p>The JBFOUND routine calls the MVS SWA Create Control module to obtains storage for and initialize the Interpreter Entry List. The Interpreter Entry List contains information from JES2, such as user ID and security information, and is used for linking to the MVS Interpreter.</p> <p>Both JES2 and MVS have a data area named JCT. The two JCTs are not similar and one is not a copy, or partial copy, of the other. The Interpreter Entry List contains a pointer to the in-storage copy of the beginning of the \$JCT JMR area which is used to create the CEPA/JMR.</p> <p>The MVS Interpreter Initialization routine calls the MVS Interpreter Router routine and after the internal text has been interpreted, the MVS Enqueue routine issues the call to SMF exit IEFUJV (entry code of 32). This is the first SMF exit for a job during the execution phase. The Scheduler Work Area (SWA) job and step tables have been created. The JMR pointer, called the CEPA in SMF documentation, is provided in the exit parameter list.</p>                                                                                                                                                                                            | IEFUJV              |

| Step | Processing                                                                                                                                                                                                                                                                                                                                                                                                          | Exit used            |
|------|---------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|----------------------|
| E    | After the Interpreter returns control to the MVS SWA Create Control module, a RACROUTE REQUEST=VERIFY,ENV=CREATE is then issued to create the job's security environment. The SAF Router exit is invoked if it exists and Message ICH70001I is issued by RACF identifying the user. If an error occurred during Job Select processing, for example a JCL error, then the job's security environment is not created. | SAF Router exit      |
| F    | Exit 32 is called. The \$JCT, all \$IOTs, the JMR, and the ACEE have been created and are available.<br><br>The JBSELECT routine then issues the \$HASP373 JOB STARTED message.                                                                                                                                                                                                                                     | 32                   |
| G    | Before job select processing is complete and control returns to the Initiator, JES2 checkpoints (writes to spool) the \$JCT. Exit 8 is called.                                                                                                                                                                                                                                                                      | 8                    |
| H    | Job initiation calls SMF exit, IEFUJI. MVS job initiation is a series of calls to step initiation based on the number of steps in a job.                                                                                                                                                                                                                                                                            | IEFUJI               |
| I    | MVS step initiation consists of a call to SMF exit, IEFUSI, step allocation for those data sets and devices defined in the job's JCL, and a call to the MVS Initiator Attach routine.                                                                                                                                                                                                                               | IEFUSI               |
| J    | Allocation of JCL defined SYSIN, SYSOUT, and internal readers initiates a call to Exit 31.                                                                                                                                                                                                                                                                                                                          | 31                   |
| K/L  | The MVS Initiator Attach routine attaches a subtask with an entry point of the program name specified on the EXEC JCL statement for the job step. The job step could dynamically allocate JES2 SYSIN, SYSOUT, or internal readers and therefore Exit 31 can be called.                                                                                                                                              | 31                   |
| M    | The OPEN and CLOSE of JES2 data sets and internal readers call Exits 30 and 33.                                                                                                                                                                                                                                                                                                                                     | 30<br>33             |
| N    | Dynamic Unallocation of JES2 data sets and internal readers initiate a call to Exit 34. Exit 48 can be used in preference to Exit 34. Exit 34 may be too early to affect some fields in the \$PDDB because unallocation processing takes place after Exit 34. Use Exit 48 when altering fields in the \$PDDB, this exit can also be used to control Spin processing.                                                | 34<br>48             |
| O    | At End-of-Task (EOT) processing an SSI call is made to JES2 and Exit 35 is called.                                                                                                                                                                                                                                                                                                                                  | 35                   |
| P    | Control is passed (return from Attach) to the MVS Initiator Attach routine and subsequently MVS Step Delete calls Step Unallocation which unallocates those data sets and devices defined in the job's JCL on a step basis. Exit 34 is called for JCL defined SYSIN, SYSOUT, and internal readers. Exit 48 is also taken, as mentioned previously.                                                                  | 34<br>48             |
| Q    | The MVS Unallocation routine calls the MVS SMF Control routine which calls SMF exit IEFACTRT with entry codes 20 and 12. If additional job steps are to be processed, control is passed back to step 8. Otherwise, control is passed to Job Termination at step 17.                                                                                                                                                 | SMF exit<br>IEFACTRT |
| R    | Job Termination (actually this is Step Termination for the last step) again calls SMF exit IEFACTRT with entry codes 20 and 16. Control is then passed to MVS Step Delete where an SSI call (12) is made for Job Termination.                                                                                                                                                                                       | IEFACTRT             |
| S    | End-of-job processing calls Exit 28. This exit can clean up resources obtained over the life of job execution.                                                                                                                                                                                                                                                                                                      | 28                   |
| T    | Spool control blocks are checkpointed. Exit 8 is taken for the \$JCT write. The \$JQE is placed on the OUTPUT queue to await output processing.                                                                                                                                                                                                                                                                     | 8                    |

## 3.18 JES2 start procedure

```
//JES2      PROC M=JES2PARM
//IEFPROC   EXEC PGM=HASJES20,TIME=1440,
//HASPPARM  DD DSN=SYS1.PARMLIB(MEMBER1)
//          DD DSN=SYS1.PARMLIB(COMMON)
//          DD DSN=SYS1.PARMLIB(NJEDEFS)
//          DD DSN=SYS1.PARMLIB(PRINTERS)
//PROC00    DD DISP=SHR,DSN=SYS1.PROCLIB
//PROC01    DD DISP=SHR,DSN=SYS1.PROCLIB
```

Figure 3-18 JES2 start procedure

### How to start JES2

JES2 can be started after the z/OS system has been initialized. The z/OS system automatically starts JES2 if your installation provides this capability. Otherwise, you must issue the START command to invoke a JCL procedure in SYS1.PROCLIB that starts JES2. JES2 initialization is performed after JES2 has been started. Initiators will not accept work (process jobs) until JES2 initialization is complete.

### JES2 start procedure

To start the JES2 component, an installation must provide a subsystem cataloged JCL procedure (PROC). The PROC name can be a maximum of four characters. The PROC must be defined in the IEFSSNxx member of SYS1.PARMLIB.

The basic JCL procedure is shown in Figure 3-18. It contains an EXEC statement and five DD statements. The EXEC statement could specify the following parameters:

- PGM=** Specify the name of the JES2 load module that resides in an authorized library.
- TIME=** Specify 'NOLIMIT' or 1440 to prevent system ABEND code 322.
- PARM=** Specify JES2 start options. It is suggested that you specify *NOREQ* for the start parameters. *WARM* is the default. Table 3-6 on page 159 lists the JES2 start options available.



It is recommended that you do not specify the REGION parameter to prevent any virtual storage limitations in the JES2 address space. The DD statement parameters specify as follows:

|                       |                                                                                                                                                                                                                                          |
|-----------------------|------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| <b>PROC00</b>         | Defines a default procedure library to be used for converting the JCL of batch jobs, time-sharing logons, and system tasks.                                                                                                              |
| <b>PROCxx (01-99)</b> | Can be used to define other user-cataloged procedure libraries that are associated with job classes by the JOBCLASS initialization statement or by the /*JOBPARM JES2 control statement.                                                 |
| <b>HASPPARM</b>       | Specifies the data set containing the initialization statements that will be used for JES2 initialization. With this statement, you can control all JES2 functions. The majority of them can be changed dynamically using JES2 commands. |

## 3.19 HASPPARM using INCLUDE statement

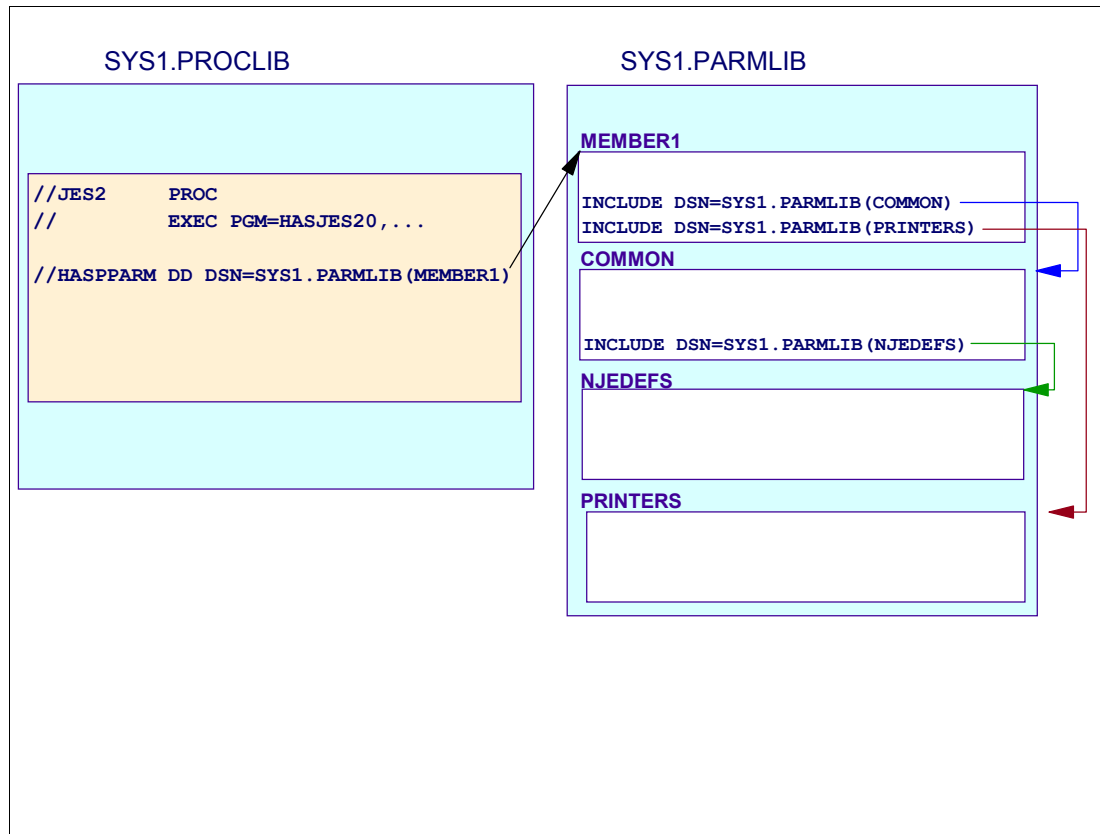


Figure 3-19 HASPPARM using the INCLUDE statement

### INCLUDE initialization statement

With JES2 V1R2, the syntax of the INCLUDE initialization statement specifies a required data set name, and a volser and unit (required only if needed for allocation). The data set name can have a member name. The statements in the included data set are processed immediately. When the end of the included data set is reached, processing continues with the statement after the include of the original data set. Includes can be nested. There is loop detection to prevent a nesting loop.

The INCLUDE statement has the following considerations:

- ▶ DSNNAME can include a member name.
- ▶ VOLSER and UNIT are optional (if data set is cataloged).
- ▶ Statements in the data sets are processed immediately.
- ▶ An INCLUDE data set can have INCLUDE statements.

The following example shows four members that make up the JES2 initialization definitions in the JES2 procedure. Using the INCLUDE statement, the JES2 procedure is simplified, as shown in Figure 3-19.

```
//JES2      PROC
//          EXEC PGM=HASJES20,...
//HASPPARM DD DSN=SYS1.PARMLIB(MEMBER1)
//          DD DSN=SYS1.PARMLIB(COMMON)
//          DD DSN=SYS1.PARMLIB(NJEDEFS)
//          DD DSN=SYS1.PARMLIB(PRINTERS)
```

## 3.20 Simplified procedure using the default parmlib member

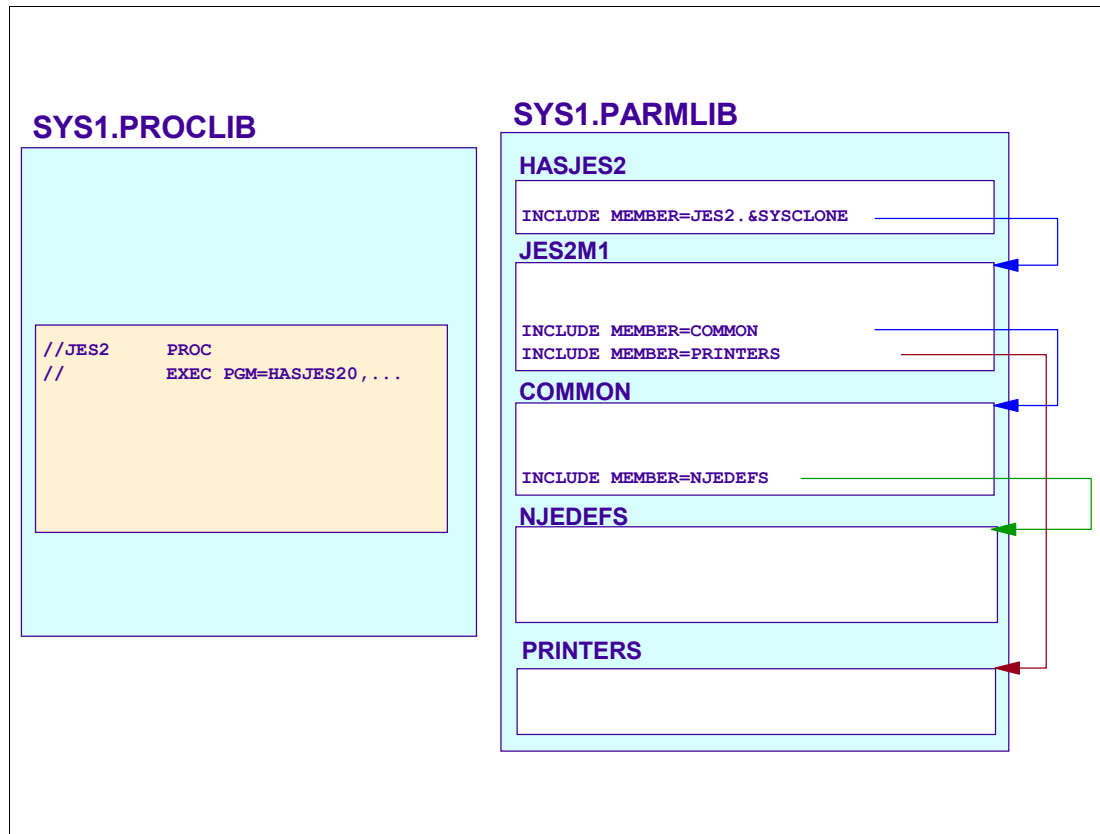


Figure 3-20 Simplified JES2 procedure using the default parmlib member

### Enhancement to INCLUDE statement

Improvements to the INCLUDE initialization statement enable the reading of an additional member from the current data sets for the initialization statements. The new syntax added to the INCLUDE statement in JES2 V1R4 is as follows:

- ▶ **INCLUDE MEMBER=member**  
member\_a should be in the current parmlib data set being processed.
- ▶ **INCLUDE PARMLIB\_MEMBER=member**  
member\_b should be in the default logical parmlib.

**Note:** The new keywords MEMBER and PARMLIB\_MEMBER are mutually exclusive with the keywords DSNAME or VOLSER or UNIT.

### Default parmlib member

This JES2 V1R4 support allows for a default PARMLIB member. This member is used only if there is no HASPPARM DD and the operator does not use a HASPPARM= start option. The default is HASJES2 where JES2 is the actual subsystem name. The default member comes from the logical PARMLIB concatenation.

There is also the ability to specify the "default" member as a start option, (MEMBER=). If MEMBER= is specified, it will be used. HASPPARM = and MEMBER= are mutually exclusive and can be specified as follows:

|                    |                                                                                              |
|--------------------|----------------------------------------------------------------------------------------------|
| <b>MEMBER=</b>     | If specified by the operator as a start option, then use that member as the default PARMLIB. |
| <b>HASPPARM=</b>   | If specified by the operator as a start option, then use that DD name for the PARMLIB.       |
| <b>HASPPARM DD</b> | If this DD exists in the JES2 procedure, then use that DD name as the PARMLIB.               |

**Note:** If none of the above are specified, then use HASJES2 as the default PARMLIB concatenation.

If there are any problems with any of these, message HASP450 is issued and then the operator can respond if JES2 should continue, just as is done previously if the OPEN of the DD for PARMLIB fails. If the operator replies 'Y', JES2 initialization continues and console mode is entered, which again is the same as previously.

### PARMLIB search order

Following is the search order to determine which PARMLIB to use:

1. If HASPPARM=ddname parameter is specified, use that DD.
2. If MEMBER=PARMLIB\_MEMBER= parameter is specified, use that member from logical PARMLIB.
3. If neither is specified, try to open DD with ddname of HASPPARM.
4. If no HASPPARM DD is found, use HASJesx member from the logical parmlib, which is the default PARMLIB concatenation. This is HASJES2 in the examples.

### Using default PARMLIB

With the new enhancements to the INCLUDE processing, it is now possible to remove the HASPPARM DD from the JES2 procedure, as shown in Figure 3-20. In this example, all the members must be in the logical PARMLIB concatenation and they are in SYS1.PARMLIB.

The HASJES2 member contains an INCLUDE statement that points to PARMLIB member JES2M1, as the value of &SYSCONE is M1.

## 3.21 JES2 start parameters

- |                                                           |                                       |
|-----------------------------------------------------------|---------------------------------------|
| <input type="checkbox"/> COLD/ <u>WARM</u>                | <input type="checkbox"/> CONSOLE      |
| <input type="checkbox"/> CKPT1/CKPT2                      | <input type="checkbox"/> REC ONFIG    |
| <input type="checkbox"/> NOREQ/ <u>REQ</u>                | <input type="checkbox"/> NONE / U / N |
| <input type="checkbox"/> NOLIST/ <u>LIST</u>              |                                       |
| <input type="checkbox"/> NOLOG/ <u>LOG</u>                |                                       |
| <input type="checkbox"/> FORMAT/ <u>NOFMT</u>             |                                       |
| <input type="checkbox"/> HASPPARM=ddname                  |                                       |
| <input type="checkbox"/> <u>SPOOL=VALIDATE/NOVALIDATE</u> |                                       |

Figure 3-21 JES2 start parameters

### JES2 start options

JES2 uses start options to determine how it will perform the current initialization. You can specify start options in either of two 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,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 have been added to the JES2 initialization data set or an initialization statement is in error.

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

```
*id $HASP426 SPECIFY OPTIONS - jesname
```

You should respond using the z/OS **REPLY** command to specify the JES2 options determined by your installation procedures.

**REPLY id,options**

Table 3-6 explains the JES2 start options.

If you respond to message \$HASP426 with the **\$PJES2** command, JES2 will terminate.

**Note:** If you specify the NOREQ option, JES2 will automatically start processing following initialization. Otherwise, you must enter the **\$S** command in response to the \$HASP400 ENTER REQUESTS message to start JES2 processing.

Table 3-6 JES2 start parameters

| Option                         | Explanation                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                               |
|--------------------------------|---------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| FORMAT<br>NOFMT                | <p>FORMAT specifies that JES2 is to format all existing spool volumes. If you add unformatted spool volumes, JES2 automatically formats them whether FORMAT is specified or not. With FORMAT, 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<br>WARM                   | <p>COLD specifies that JES2 is to be cold-started. All jobs in the system will be purged and all job data on the spool volumes will be scratched.</p> <p>Default: WARM specifies that JES2 is to continue processing jobs from where they were stopped. If the FORMAT option was also coded, then JES2 will ignore the WARM specification and perform a cold start.</p>                                                                                                                                                                                                                                                                                                                                                                                                   |
| SPOOL = VALIDATE<br>NOVALIDATE | <p>VALIDATE specifies that the track group map is validated on a JES2 all-member warm start.</p> <p>Default: SPOOL=NOVALIDATE specifies that the track group map is not validated when JES2 restarts.</p>                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                 |
| NOREQ<br>REQ                   | <p>NOREQ specifies that the \$HASP400 (ENTER REQUESTS) message is to be suppressed and JES2 is to automatically start processing when initialization is complete.</p> <p>Default: REQ specifies that the \$HASP400 (ENTER REQUESTS) message is to be written at the console. This message allows you to start JES2 processing with the \$S command.</p>                                                                                                                                                                                                                                                                                                                                                                                                                   |
| NOLIST<br>LIST                 | <p>NOLIST 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> presents an example of an initialization data set listing produced by using the list option.</p> <p>Default: LIST 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.) LIST will not print any statements that follow a NOLIST control statement in the initialization data set.</p> |
| NOLOG<br>LOG                   | <p>NOLOG specifies that JES2 is not to copy initialization statements or initialization errors to the HARDCPY console. If you specify NOLOG, JES2 ignores LOG control statements in the initialization data set.</p> <p>Default: LOG specifies that JES2 is to honor any LOG statements in the initialization data set.</p>                                                                                                                                                                                                                                                                                                                                                                                                                                               |

| Option          | Explanation                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                              |
|-----------------|------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| CKPT1<br>CKPT2  | Specifies what checkpoint data set JES2 must use for building the JES2 work queues.<br><br>Default: If you do not specify, JES2 automatically determines which checkpoint data set to use.                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                               |
| RECONFIG        | RECONFIG specifies that JES2 will use 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 RECONFIG will also cause JES2 to enter the reconfiguration dialog during initialization and issue message \$HASP289 CKPT1 AND/OR CKPT2 SPECIFICATIONS ARE REQUIRED.<br><br>If you previously reconfigured your checkpoint configuration through the checkpoint reconfiguration dialog, the CKPTDEF statement definition may not contain the most current checkpoint definition. Changes made through the checkpoint reconfiguration dialog are not saved in the input stream data set. |
| HASPPARM=ddname | HASPPARM=ddname specifies the name of the data definition (DD) statement that defines the data set containing the initialization statements that JES2 is to use for this initialization.<br><br>Default: HASPPARM=HASPPARM specifies that JES2 is to be initialized using the initialization statements in the data set defined by the HASPPARM DD statement in the JES2 procedure.                                                                                                                                                                                                                                                                                                                                                                                                                                                      |
| CONSOLE         | Causes JES2 to simulate receiving a CONSOLE initialization statement after all initialization statements are processed. That is, if CONSOLE is specified, JES2 will divert to the operator console for further parameter information after the input stream data set has been exhausted.                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                 |
| NONE<br>U<br>N  | NONE, U, or N character specifies that JES2 is to use all of the default start options. There is no difference between these three options; they are equivalent. When NONE, U, or N is specified, JES2 uses the default start options, which are:<br>NOFMT<br>WARM<br>REQ<br>LIST<br>LOG                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                 |

### New start option with default PARMLIB

You can now start JES2 with the new start option of reading the initialization stream from the default PARMLIB specification, as follows:

- S JES2,PARM=('MEMBER=member')
- S JES2,PARM=('PARMLIB\_MEMBER=member')

The operator should reply to the \$HASP467 message as follows:

r xx, MEMBER=member

where member is the member of logical parmlib in each of the above specifications.

The following conditions should be considered:

- HASPPARM=ddname and MEMBER= are mutually exclusive.



- ▶ If neither HASPPARM= nor MEMBER= is specified, then it will process from the default HASjesx member of logical parmlib (where jesx is the JES2 subsystem name).
- ▶ IBM does not ship a default parmlib member.

### **Starting JES2 without a JES2 PROC**

In an emergency, you can start JES2 without a JES2 procedure because of the elimination of the need to specify the HASPPARM data set and the PROCLIB data sets, a change that was introduced in JES2 V1R2. Start JES2 as follows:

- ▶ **S IEESYSAS,PROG=HASJES20,JOBNAME=JES2**

This start command assumes that HASJES20 is in the LINKLIST (no STEPLIB). During JES2 initialization when the OPEN of HASPPARM fails, the logical parmlib member HASJES2 will be used. If HASJES2 is not found, message \$HASP469 is issued.

- ▶ **S IEESYSAS,PROG=HASJES20,JOBNAME=JES2,PARM='MEMBER=MEMBER2'**

This start command uses the logical parmlib member MEMBER2 and no OPEN of the HASPPARM DD is attempted. This option is new with JES2 V1R4 because of the new MEMBER= capabilities. If MEMBER2 is not found, message \$HASP469 is issued.

## 3.22 Restarting JES2

- ❑ Cold start
  - All job data in spool is lost
- ❑ Warm start
  - All-member warm start
  - Single-member warm start
  - Quick start
  - Hot start

Figure 3-22 Restarting JES2

### Restarting JES2 with cold start option

Very few definitions, or redefinitions, of some JES2 facilities and resources require that the JES2 system be totally shut down. JES2 must be restarted with a cold start to allow all component systems to be aware of the changed facilities and resources. The time to restart JES2 in this manner is based on the work in the system and, if not scheduled, causes a disruption in data processing services. All job data previously on the spool volumes is lost with cold start. You can avoid data loss by scheduling a spool offload of the JES2 queues. In a MAS configuration, no other member can be active during a cold start. Use of the FORMAT option causes a cold start.

An IPL 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:

- ▶ 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, then 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.

There are four ways in which a warm start can be performed in a multi-access spool configuration, as follows:

1. 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.

2. Single-member warm start

This 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.

3. Quick start

This is performed when you specified a warm start and JES2 determined 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. This occurs:

- After **\$P JES2** has been 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 has been 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.

4. 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 had 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.

## 3.23 Stopping JES2

- ❑ \$P command to stop all JES2 processing
- ❑ Log off all TSO/E users
- ❑ \$HASP099
- ❑ Stop all started tasks
- ❑ \$P JES2
  - \$P JES2,QUICK (poly-JES environment)
  - \$P JES2,ABEND
  - \$P JES2 ABEND,FORCE
- ❑ HALT EOD

Figure 3-23 Stopping JES2

### Stopping JES2

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. This 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 for most parameters. Before stopping JES2 for the purpose of changing initialization statement parameters, refer to *z/OS JES2 Initialization and Tuning Reference*, SA22-7533-03.

To stop JES2, do the following:

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 has completed.

If the \$P JES2,ABEND command does not successfully terminate JES2, you can also specify the FORCE parameter. The \$PJES2,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, you should send a message to the user asking them to end their cross-system activity.

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

## 3.24 JES2 operations

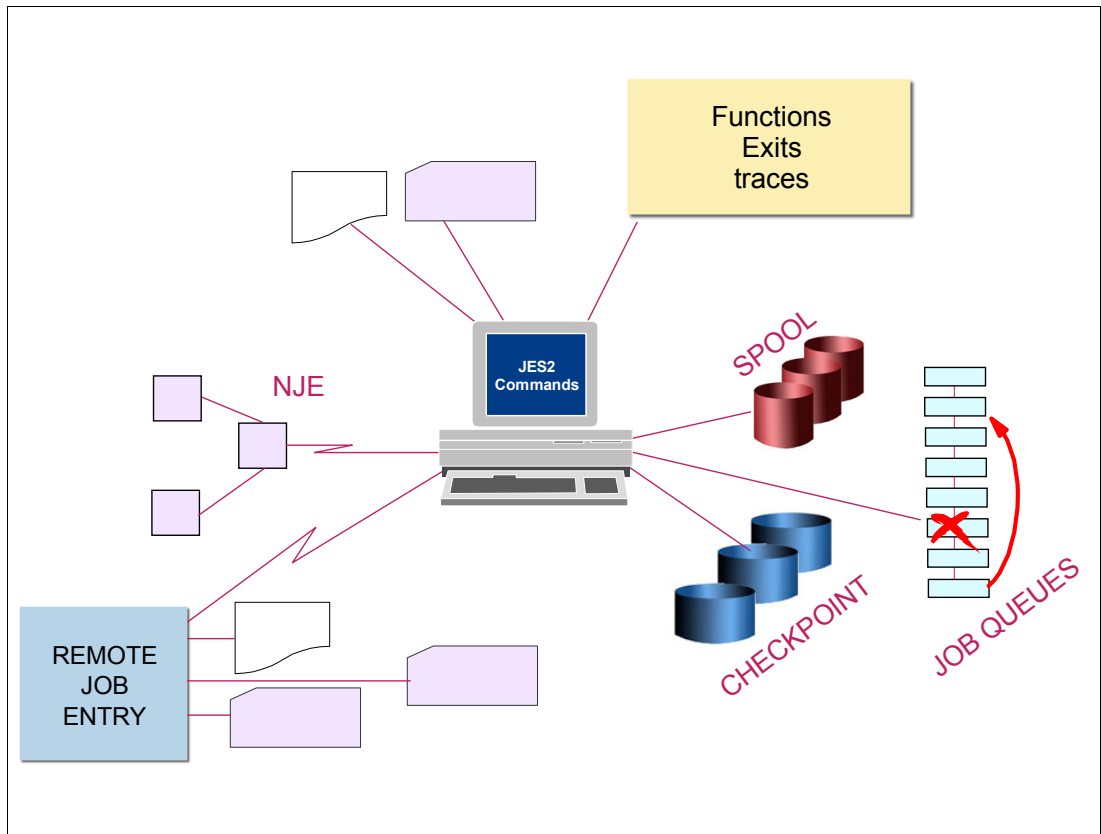


Figure 3-24 JES2 operations

### JES2 operations

No large data processing system or subsystem can continuously operate independently of system programmer or operator intervention. To help you maintain your overall work environment, JES2 provides a set of commands that permits you to control most of the JES2 functions and devices under its control.

As your JES2 complex becomes more sophisticated, you might connect your system to others to form a network of systems. You can use operator commands to control the lines that connect the separate systems, as well as to define the separate systems to yours. This is typically a very dynamic environment since different systems are added or deleted from the network due to maintenance, hardware reconfiguration requirements, workload balancing, or the need to access a database at a different location. JES2 commands permit you to alter most of your original network definition, as required. Almost all JES2 initialization statements can be changed dynamically by JES2 operator commands.

Operator commands can be used to:

- ▶ Display status information and device definition
- ▶ Start, stop, and halt devices under JES2's control
- ▶ Assign units to local printers, punches, card readers, and lines, or reassign units to these devices

- ▶ Modify processing, such as: output definition, the dynamic alteration of the checkpoint definition, enabling installation-defined exits, offload devices, printer and punch characteristics, and job characteristics
- ▶ Add function and functional subsystems
- ▶ Delete function, network systems, exits, and diagnostic traces

Your installation can require that only certain operators can issue certain JES2 commands and/or restrict commands coming in from RJE workstations, NJE nodes, or any other device. You can use RACF and customization techniques to limit the degree of control an individual or group can have over JES2 resources.

All JES2 commands are entered using standard z/OS command interfaces such as a z/OS console, or within the JES2 initialization data set. As the default, JES2 prefixes its commands and messages with a dollar sign (\$). For commands, the prefix character defines the scope of the command as being JES2 only; for messages, the prefix character is informational in that it designates that the message was issued by the JES2 component. You can change this symbol through the CONDEF initialization statement.

## 3.25 Controlling the JES2 environment

- ☐ **\$S**
- ☐ **\$P**
- ☐ **\$P JES2**
- ☐ **\$T BUFDEF**
- ☐ **\$T SMFDEF**

*Figure 3-25 Controlling the JES2 environment*

### Controlling JES2

The following commands are useful to control the JES2 environment:

- ▶ Start JES2 processing, **\$S**  
To start system activity.
- ▶ Stop JES2 processing, **\$P**  
To stop all system initiators, printers, punches, job transmitters, and SYSOUT transmitters after they complete their current activity.
- ▶ Withdraw JES2 from the system, **\$P JES2**  
To withdraw JES2 from the system to which the entering console is attached.
- ▶ Monitor local buffers, **\$T BUFDEF**  
To specify the percentage of local buffers use at which JES2 alerts the operator of a local buffer shortage. Local buffers can reside below 16 megabytes of virtual storage or above 16 megabytes of virtual storage.
- ▶ Monitor SMF buffers, **\$T SMFDEF**  
To set the SMF buffer warning level.



## 3.26 Controlling a MAS environment

- ☐ **\$D MASDEF**
- ☐ **\$D MEMBER**
- ☐ **\$T MASDEF**
- ☐ **\$D CKPTDEF**
- ☐ **\$T CKPTDEF**

Figure 3-26 Controlling a MAS environment

### Controlling the MAS environment

The following commands are useful to control your multi-access spool (MAS) environment:

- ▶ Display characteristics of the MAS, **\$D MASDEF**  
To display multi-access spool definition and tuning parameters.
- ▶ Display the status of MAS members, **\$D MEMBER**  
The information displayed for each member includes the member name and any of the following information:
  - The date and time the member was started.
  - The member that is restarting this member's work.
  - The status of the member.
  - The name of the MVS system image of the member.
  - The time the member last accessed the checkpoint.
  - The version of JES2 running on the member.

This command has parameters you can use as a filtering technique to limit the type of information to search, or the amount of information to display.
- ▶ Control the MAS environment, **\$T MASDEF**  
To specify multi-access spool definition and tuning parameters.
- ▶ Display the checkpoint definition, **\$D CKPTDEF**  
This command has parameters you can use as a filtering technique to limit the type of information to search, or the amount of information to display.
- ▶ Control the checkpoint definition, **\$T CKPTDEF**
  - To modify the checkpoint definition
  - To initiate a checkpoint reconfiguration dialog with JES2

## 3.27 Controlling JES2 spooling

- ☐ **\$D SPOOL**
- ☐ **\$S SPOOL**
- ☐ **\$T SPOOLDEF**
- ☐ **\$P SPOOL**
- ☐ **\$Z SPOOL**

Figure 3-27 Controlling Controlling JES2 spooling

### Controlling the JES2 spool

The following commands are useful to control the JES2 spool:

- ▶ Display spool volume usage, **\$D SPOOL**  
To display the status and percent utilization of a specified spool volume or all spool volumes
- ▶ Start a spool volume, **\$S SPOOL**  
To add or reactivate a spool volume to the spool configuration
- ▶ Control the JES2 spooling environment, **\$T SPOOLDEF**  
To specify the JES2 spooling environment characteristics
- ▶ Drain a spool volume, **\$P SPOOL**  
To drain and delete an entire spool volume by processing all work on the volume and preventing any available space on the volume from being allocated
- ▶ Halt a spool volume, **\$Z SPOOL**  
To deallocate a spool volume after active work completes its current phase of processing (executing, printing, punching)

## 3.28 Controlling JES2 jobs

|                                   |                                   |
|-----------------------------------|-----------------------------------|
| <input type="checkbox"/> \$A Job  | <input type="checkbox"/> \$O Job  |
| <input type="checkbox"/> \$C Job  | <input type="checkbox"/> \$P Job  |
| <input type="checkbox"/> \$CO Job | <input type="checkbox"/> \$PO Job |
| <input type="checkbox"/> \$D Job  | <input type="checkbox"/> \$T Job  |
| <input type="checkbox"/> \$H Job  | <input type="checkbox"/> \$T O    |
| <input type="checkbox"/> \$L Job  |                                   |

Figure 3-28 Controlling JES2 jobs

### Controlling JES2 jobs

The following commands are useful to control JES2 jobs:

- ▶ Release specified jobs, **\$A Job**
- ▶ Cancel a job, **\$C Job**
- ▶ Cancel output groups, **\$CO Job**
- ▶ Display information about a specified job, **\$D Job**
- ▶ Hold a specified job, **\$H Job**
- ▶ List job output information, **\$L Job**
- ▶ Release or cancel held output groups, **\$O Job**
- ▶ Purge a job, **\$P Job**
- ▶ Purge an output job, **\$PO Job**
- ▶ Change a job's class, scheduling priority, or member affinity, **\$T Job**
- ▶ Set output characteristics, **\$T O**

## 3.29 Controlling JES2 printers

- ☐ **\$D PRT**
- ☐ **\$E PRTnnnn**
- ☐ **\$P PRTnnnn**
- ☐ **\$S PRTnnnn**
- ☐ **\$Z PRTnnnn**

Figure 3-29 Controlling JES2 printers

### Controlling JES2 printers

The following commands are useful to control JES2 printers:

- ▶ Display printers, **\$D PRT**  
To display printer work selection and processing characteristics. This command has parameters you can use as a filtering technique to limit the type of information to search, or the amount of information to display.
- ▶ Restart printer activity, **\$E PRTnnnn**  
To stop the printing of the current output group and requeue the output according to its job priority for later processing.
- ▶ Stop a printer, **\$P PRTnnnn**  
To stop a printer after it completes processing the current output group and then free the associated system resources.
- ▶ Start a printer, **\$S PRTnnnn**  
To start printer activity. The \$HASP190 message asks whether the requested setup has been completed for the specified device. Either verify that the setup is complete or issue the **\$T PRTnnnn** command to override specific setup specifications. Then reissue the **\$S** command to physically start the printer.
- ▶ Halt printer activity, **\$Z PRTnnnn**  
To temporarily halt local or remote printer activity.

## **JES2 diagnosis communication mechanisms**

The following diagnostic tools are available for detecting and diagnosing problems occurring in the JES2 environment:

- ▶ **Messages**

JES2 provides a set of messages to alert the JES2 operator and system programmer of processing errors.

- ▶ **Traces**

Optionally, your installation can use the JES2 tracing facility, a function that records events associated with specific functions, such as each time JES2 is initialized or terminated or each time an exit routine is taken.

- ▶ **IPCS**

JES2 exploits the interactive problem control system (IPCS) facility to allow you to view the formatted contents of JES2 control blocks and dumps of system data necessary when diagnosing and recovering from processing errors.

## 3.30 JES3 processing

- ❑ JES3 configuration
- ❑ JES3 complex
- ❑ JES3 single-system image
- ❑ JES3 multi-system image
- ❑ Availability
- ❑ Workload balancing
- ❑ Spooling
- ❑ Control flexibility
- ❑ JES3 phases of job processing
- ❑ JES3 operator control
- ❑ How to start and stop JES3

Figure 3-30 JES3 processing

### JES3 processing

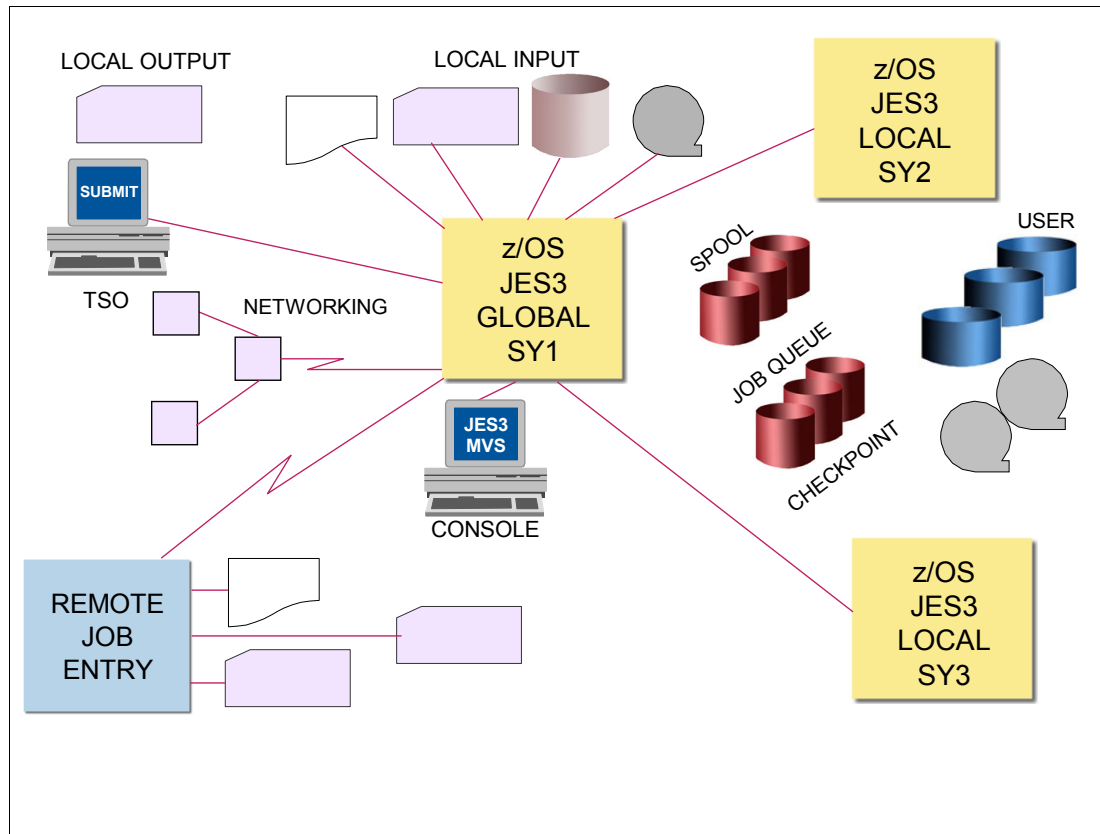
With the z/OS MVS JES3 system, resource management and workflow management are shared between MVS and its JES3 component. Generally speaking, JES3 does resource management and workflow management *before* and *after* job execution, while MVS does resource and workflow management *during* job execution.

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 3-30 shows the JES3 topics discussed in this section.

### 3.31 JES3 configuration



*Figure 3-31 JES3 configuration*

## JES3 configuration

JES3 can run on a single processor, or on multiple processors, up to thirty-two 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 hardware components used for connecting the systems are:

- ▶ 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 may 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 input/output (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.



### 3.32 JES3 complex

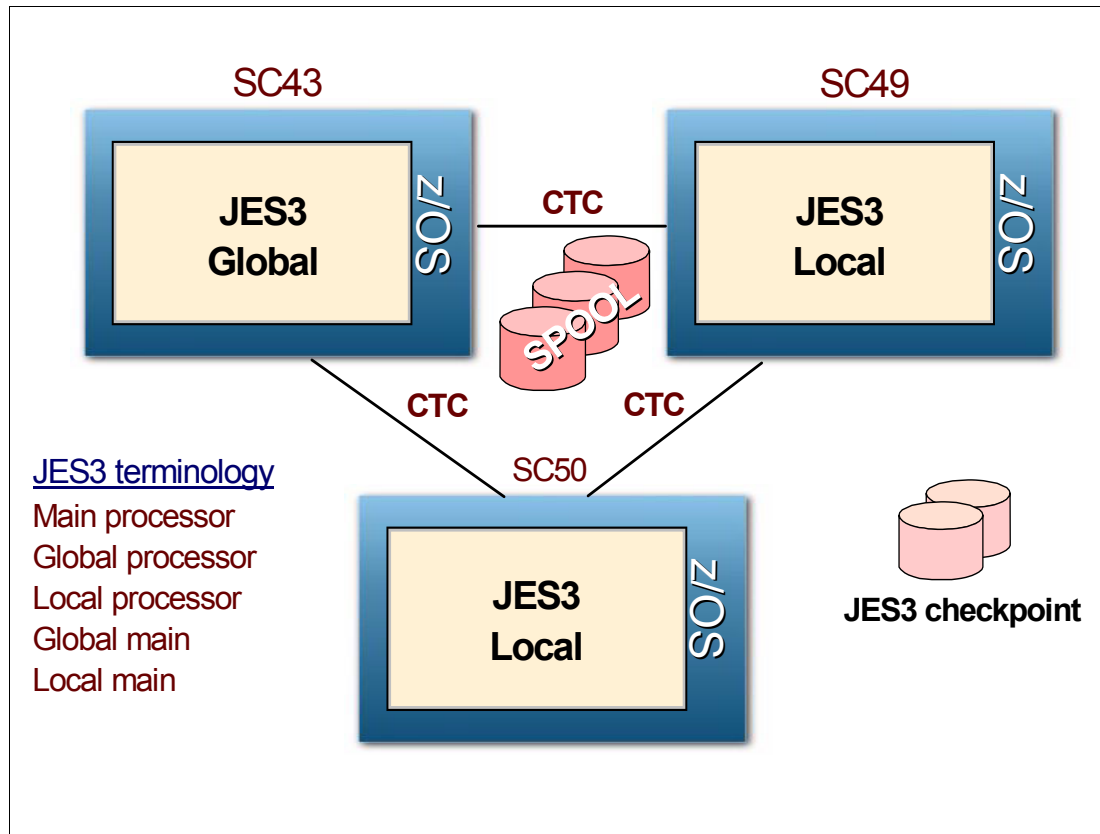


Figure 3-32 JES3 complex

#### JES3 complex

z/OS uses JES3 to control the input, processing, and output of jobs. JES3 services the job processing requirements of from 1 to 32 physically connected z/OS processors called *mains*. Viewed as a whole, the 1- to 32-main environment serviced by JES3 is called a *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 3-32 illustrates the basic structure of a JES3 complex.

### 3.33 MVS and JES3 environment

- ❑ **Primary subsystem - JES3**
  - System input and output - (JES3 Global)
  - Manage jobs and spool - (JES3 Global)
- ❑ **Functional subsystems - FSS**
  - Offload function to locals and global address space
    - Converter/Interpreter and printers
  - JES3 global controls
    - FSS start and stop
    - Selection of work
    - Operator communication

*Figure 3-33 MVS subsystems in JES3 environment*

#### **JES3 as a primary subsystem**

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.
- ▶ Performs 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.

#### **Functional subsystem**

JES3 allows certain functions to operate outside the JES3 address space. JES3 does this using:

- ▶ The functional subsystem address space (FSS)

- ▶ The functional subsystem interface (FSI)
- ▶ The functional subsystem application (FSA)

The JES3 FSS that deals with output services is one type of FSS. This particular FSS address space may be created automatically or established by a **CALL** command for a printer device which is capable of running under the control of an FSS address space. The operator **CALL** command designates a printer as a “hot writer,” while a writer invoked automatically when output is queued is called a “dynamic writer.”

Another FSS deals with converter/interpreter services similar to those that occur in the JES3 global.

### 3.34 JES3 global

- ❑ Global processor manages jobs and resources
  - For the entire complex
  - Matches jobs with available resources
- ❑ JES3 global manages
  - Processors - I/O devices - Volumes - Data sets
- ❑ When resources are not available
  - JES3 ensures that they are before selecting the job for processing to an initiator
- ❑ Operations aid
  - Operator commands
  - Diagnostic tools

Figure 3-34 JES3 global

#### JES3 global processor

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 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 duplication of job management function becomes unnecessary, and an awareness of the status of all jobs entering or leaving the system can be easily maintained. This awareness is particularly useful in recovery situations, where the scope of recovery is largely a function of the quality of the tracking performed prior to the failure.

#### 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 cataloged data sets. When SMS is activated, JES3 will not manage devices and volumes for SMS-managed data sets.

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 input/output (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.

## **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, it is relatively simple to provide control over the entire system. 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 less need for operator intervention and decision-making.

Operators have special JES3 commands. Some commands activate programs to handle I/O devices, while others obtain or change the status of jobs being processed. With multiple processing-unit systems, JES3 operators have less to do than for an equal number of individual systems, because they can control the entire complex from a central point, and because JES3 decides where and when jobs will be processed.

For diagnostic purposes, JES3 provides traces and dumps. Traces are copies of data made during normal processing to help monitor activity in the system, and dumps are copies of data made at times of failure.

### 3.35 JES3 sysplex components

- ❑ Sysplex - 1 to 32 coupled systems
  - Using hardware and software elements
- ❑ XCF - Cross-system coupling facility
  - Provides communication services for a sysplex
- ❑ CTC - Channel to channel adapter
  - Provides direct connection between MVS systems
- ❑ GRS - Global resource serialization
  - Used to serialize resources - Uses XCF

Figure 3-35 JES3 sysplex components

#### JES3 sysplex components

The sysplex components and terminology are as follows:

- |                |                                                                                                                                                                                                                                                                                                                                        |
|----------------|----------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| <b>Sysplex</b> | Up to 32 systems or MVS images may be defined in a sysplex coupled together by hardware and software elements.                                                                                                                                                                                                                         |
| <b>XCF</b>     | The cross-system coupling facility (XCF) is the operating system component that controls members and groups, provides inter-member communications (exchange data, programs, and so on), and monitoring services for members.                                                                                                           |
| <b>CTC</b>     | A CTC is used for channel-to-channel connectivity. It is a form of direct connection between processors or between channels of the same processor. In the context of this document, this type of connection refers to an ESCON® channel operating in CTC mode. CTC links may be used when a sysplex is made up of two or more systems. |
| <b>GRS</b>     | Whenever one or more systems in a GRS complex are not in a sysplex, GRS uses its own dedicated CTCs to communicate between systems not in the sysplex, and uses XCF services between systems in the sysplex.                                                                                                                           |

### 3.36 JES3 multisystem sysplex

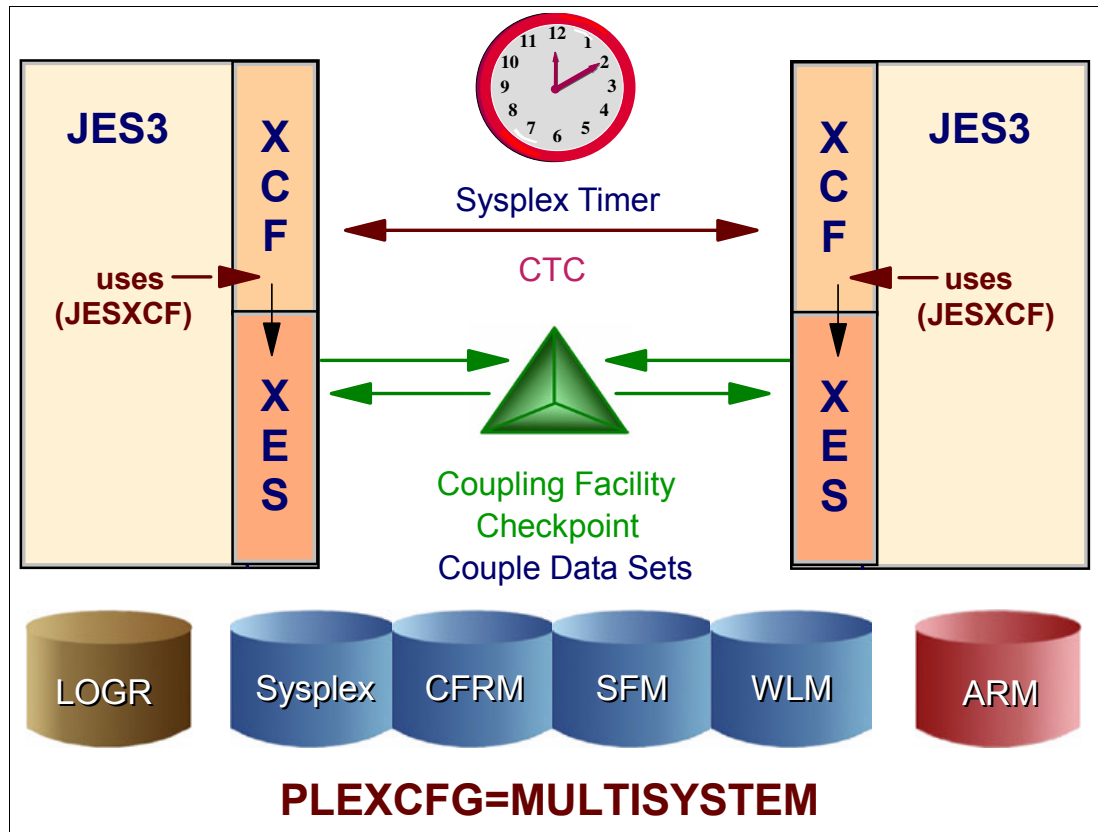


Figure 3-36 JES3 multisystem sysplex

#### JES3 multisystem sysplex

With multiple MVS images in multiple CPCs, PLEXCFG=MULTISYSTEM is the XCF mode required. A Sysplex Timer® is required, as shown in the Figure 3-36. The two JES3 systems communicate using XCF, and the signalling services can be through either CTCs or the coupling facility.

Multiprocessing is a way of doing work with two or more connected processors. Some of the advantages of running JES3 in this environment are:

- ▶ Elimination of much of the overhead of scheduling work for and operating separate processors
- ▶ Sharing devices by processors, which means that the devices can be used more efficiently
- ▶ Movement of work to other processors, should one processor become overworked, need maintenance, or need to be removed from the complex for any reason

The global processor and the local processors communicate using the MVS cross-system coupling facility (XCF). JES3 devices (the devices from which JES3 reads jobs, and on which JES3 stores jobs awaiting processing, stores job output, and writes job output) are connected to the global processor. Some JES3 devices that write job output can also be connected to the local processors.

The global processor runs most of JES3, for it is from the global processor that JES3 allocates resources to jobs and sends jobs to local processors for processing. In addition, JES3 can also pass jobs to MVS in the global processor for processing.

Each local processor has a complete MVS system and JES3 routines for spooling and for communication with the global processor. When MVS in a local processor needs a job, JES3 in that local processor sends the job request to JES3 in the global processor via XCF signalling. JES3 in the global processor returns identifying information about a job, and JES3 in the local processor uses that information to retrieve the job from the spool device.

If a problem arises on the global processor while JES3 is running, you can transfer global functions to a local processor. This transfer is called dynamic system interchange (DSI). The processor you choose must be capable of assuming the same workload as the previous global.



## 3.37 JES3 global functions

### ❑ JES3 Global functions

#### ➤ Workflow manager

- Schedule jobs
- Manage job queue
- System coupler

#### ➤ Resource manager

- Processors - Devices - Volumes - Data sets

#### ➤ Operations aid

- Operator commands
- Diagnostic tools

*Figure 3-37 JES3 global functions*

### JES3 global functions

The JES3 global controls all work in a JES3 complex, as follows:

- ▶ **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 cataloged 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, it is relatively simple to provide control over the entire system. 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.

## 3.38 JES3 terminology

- ❑ JES3 naming conventions
  - Module names: IAT ff xx
    - FF: functional area
    - XX: function performed
  - Module examples
    - IATISDV - IATOSDR - IATSICA
    - IATISDR - IATOSSC - IATSI34
  - Coding macros - IATX...
  - Data macros - IATY...
  - JES3 nucleus - IATNUC

Figure 3-38 JES3 terminology

### JES3 terminology and naming conventions

Most JES3 module names are seven characters long and begin with the characters IAT. The fourth and fifth characters are a mnemonic that represents a specific function. The sixth and seventh characters indicate a service performed by that module within the function. For example IATISDV:

- ▶ IAT signifies that this is a JES3 module.
- ▶ IS signifies that this is an input service module.
- ▶ DV signifies that this is the device driver module.

The mnemonic in the fourth and fifth character indicates any of the following module functions:

|       |                                |
|-------|--------------------------------|
| AB    | abnormal termination           |
| BD    | MVS/Bulk Data Transfer         |
| CN    | console services               |
| DC    | dependent job control          |
| DJ    | dump job                       |
| DL    | deadline scheduling            |
| DM    | spool data management          |
| DS    | dynamic system interchange     |
| DY    | dynamic allocation fastpath    |
| FC/FP | functional subsystem interface |
| FS    | failsoft                       |

|       |                                                        |
|-------|--------------------------------------------------------|
| GR    | general routines                                       |
| GS    | general service                                        |
| II    | converter/interpreter service                          |
| IN    | initialization                                         |
| IP    | interactive problem control system                     |
| IQ™   | operator inquiry commands                              |
| IS    | input service                                          |
| JV    | job validation                                         |
| LV    | locate/verify                                          |
| MD    | main device scheduler                                  |
| MF    | JES3 monitoring facility                               |
| MO    | operator modify commands                               |
| MS    | main service and generalized main scheduling           |
| NT    | JES3 networking                                        |
| OS    | output service                                         |
| PU    | purge                                                  |
| RJ    | binary synchronous communication remote job processing |
| SA    | mass storage system table create                       |
| SN    | systems network architecture remote job processing     |
| SS/SI | subsystem interface                                    |
| UT    | operator utilities                                     |
| UX    | user exit                                              |

The sixth and seventh characters restrict any of the above mnemonics to further classification of a specific task, and indicate (but are not limited to) any of the following:

|          |                                    |
|----------|------------------------------------|
| XM       | cross memory processing            |
| DV or DR | driver module for a particular DSP |
| DT or DA | data CSECT for a particular DSP    |
| CR       | card reader processing             |
| MN       | monitor                            |

Due to the limited number of combinations provided by 2 characters, there are exceptions to these conventions.

## JES3 internal naming conventions

Additional conventions are:

- ▶ The fourth character of most JES3 executable macros is an X (but there are executable macros that do not follow this convention):  
IATX
- ▶ The fourth character of a JES3 data area mapping macro is a Y:  
IATY
- ▶ The fourth character of a JES3 macro that expands within other JES3 macros is a Z:  
IATZ

The JES3 nucleus load modules are named:

IATNUC - JES3 address space  
IATNUCF - C/I FSS address space  
AITNUCI - Initialization stream checker

### 3.39 JES3 single LPAR

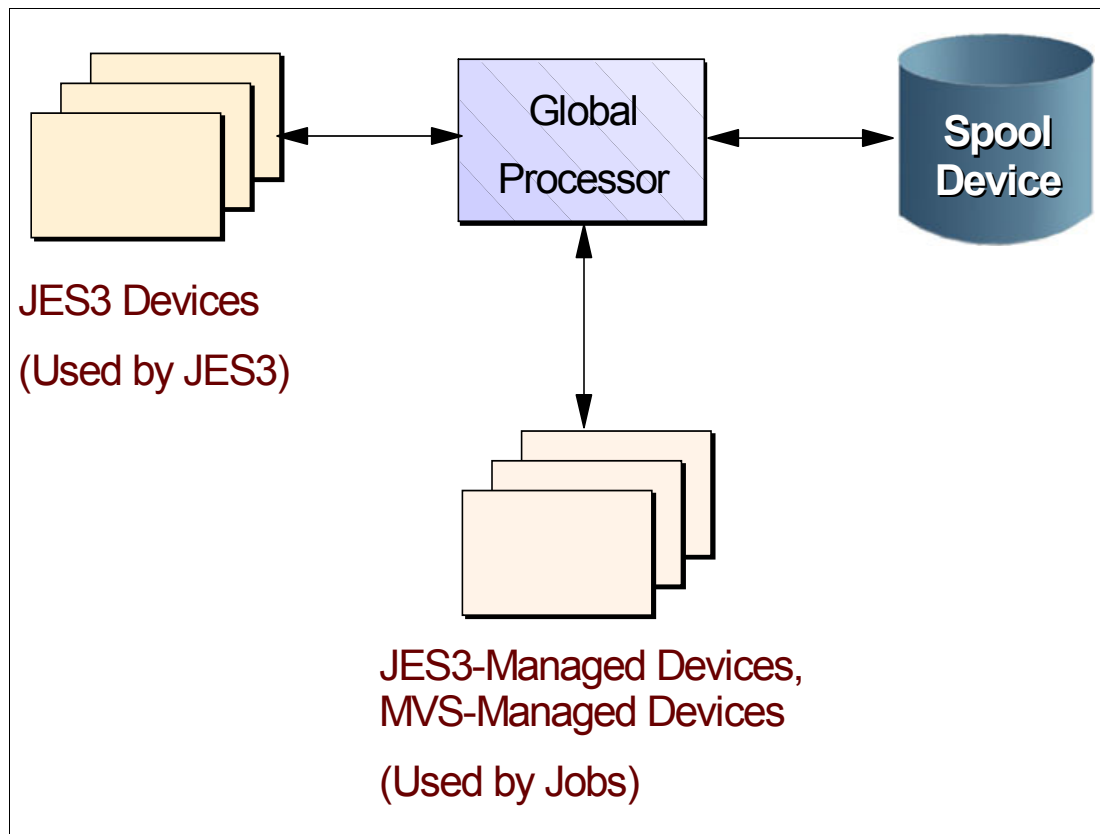


Figure 3-39 JES3 single LPAR

#### JES3 single LPAR

Figure 3-39 shows JES3 in a single-processor environment, also known as a single-system sysplex. Besides the processor, two categories of I/O devices are shown:

- ▶ JES3 devices (those used by JES3)
- ▶ JES3- and MVS-managed devices (those used by jobs).

The spool device is a direct-access storage device (DASD) that is treated in a special way by JES3, so it is shown and explained separately.

In installations with one processor, the global processor, JES3 coexists in that processor with MVS and with running jobs. In many respects, JES3 is treated like a job by MVS. (That is, MVS handles JES3 in much the same way it handles the jobs it receives from JES3 for processing.) JES3 becomes a processable “job” during initialization, while jobs submitted to the system become processable by the actions of JES3.

## 3.40 JES3 multiprocessing

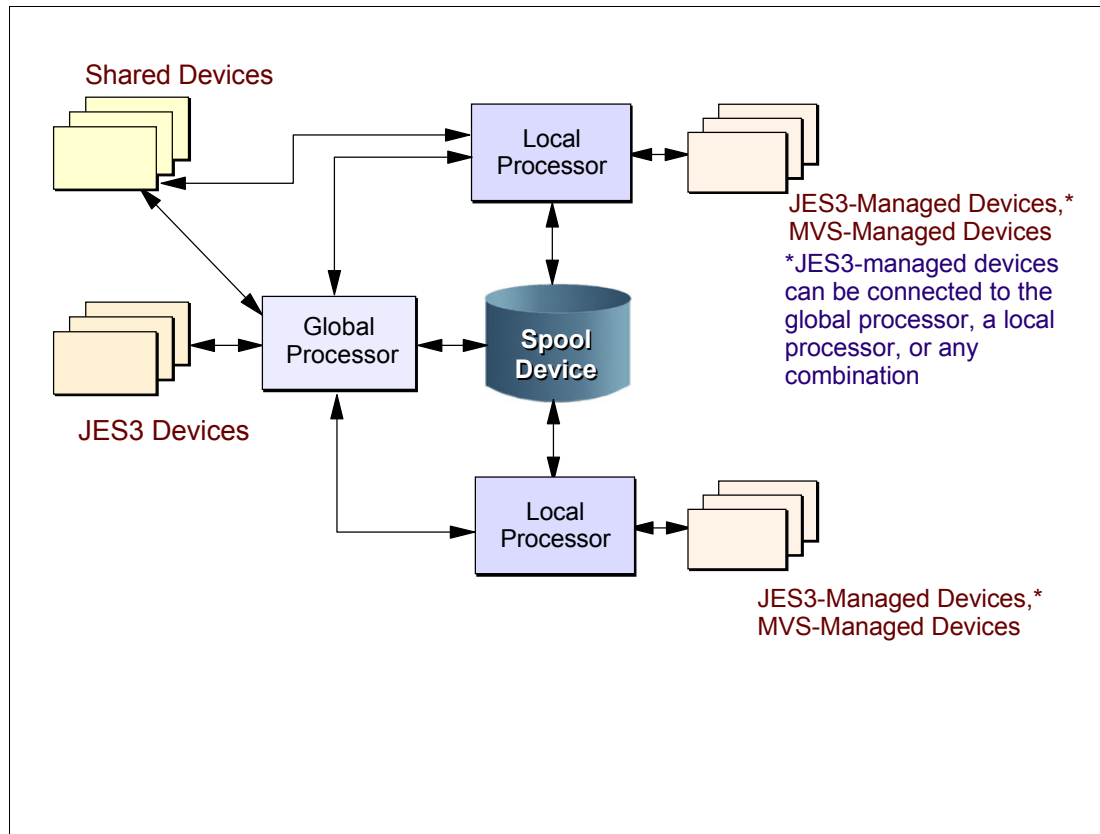


Figure 3-40 JES multiprocessing

### Multi-system image

As previously discussed, *multiprocessing* is a way of doing work with two or more connected processors. In a multiprocessing environment, also known as a multisystem sysplex, JES3 allows up to 32 processors, also known as mains, to be configured into the complex. JES3 uses one processor (called the global) to do work and also to distribute work to up to 31 other processors (called locals).

Figure 3-40 shows a JES3 multiprocessing system that has three processors, that communicate via XCF signalling. The global processor runs most of JES3, for it is from the global processor that JES3 allocates resources to jobs and sends jobs to local processors for processing. In addition, JES3 can also pass jobs to z/OS in the global processor for processing.

Each local processor has a complete z/OS system and JES3 routines for spooling and for communication with the global processor. When z/OS in a local processor needs a job, JES3 in that local processor sends the job request to JES3 in the global processor via XCF signalling. JES3 in the global processor returns identifying information about a job, and JES3 in the local processor uses that information to retrieve the job from the spool device.

If a problem arises on the global processor while JES3 is running, you can transfer global functions to a local processor. This transfer is called dynamic system interchange (DSI). The processor you choose must be capable of assuming the same workload as the previous global.

## **Availability**

If a problem develops with the global processor, you can have one of the other processors assume the global functions (operators have JES3 commands to cause the switch). Jobs running on other processors, including the one to become the new global processor, will usually be unaffected (except for a waiting period until global activities are resumed).

If part of JES3 fails, JES3 collects failure symptoms, records error data, and attempts recovery. All major JES3 components are protected by at least one special JES3 recovery routine. If recovery is unsuccessful, the failing component gets insulated from the rest of JES3, resources are freed, and the failing component will not be used again. If the component is not critical to overall JES3 processing, complete JES3 failure may be avoided.

## **Workload balancing**

JES3 balances workload among processors by considering the resource requirements of jobs. The method JES3 uses is the same whether one or several processors make up the configuration. Thus, addition of another processor does not mean a new operational and scheduling environment.

### 3.41 JES3 spooling

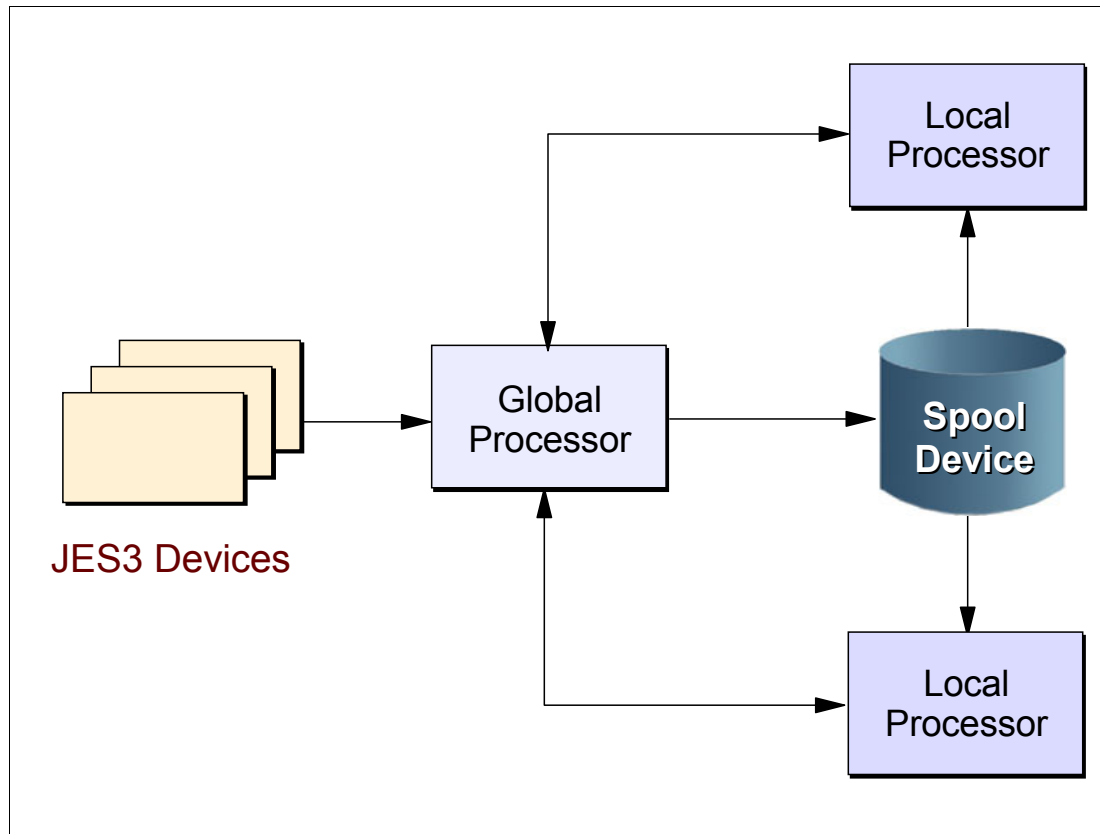


Figure 3-41 JES3 spooling

#### Spool devices

Most multiprocessing systems use shared data storage. In JES3 the spool device becomes the shared data storage. Thus, besides being a buffer (as for single-processor systems), in multiprocessing systems the spool device becomes a collection point for data to be distributed to individual processors. Also, the spool device becomes a collection point for data coming from local processors routed to JES3 output devices connected to the global processor.

#### Control flexibility

Operating systems must be easy to control. Internal complexity must be offset by features that make the systems easy to operate, to monitor, and to change. JES3 has designed-in features for operators, application programmers, and system programmers, including:

- Operators have special JES3 commands. Some commands activate programs to handle I/O devices, while others obtain or change the status of jobs being processed. With multiple processing-unit systems, JES3 operators have less to do than for an equal number of individual systems because they can control the entire complex from a central point, and because JES3 decides where and when jobs will be processed.

JES3 applies installation policies, as defined in the JES3 initialization stream, to perform job scheduling, thus freeing the operator from this task.

Even though JES3 handles job flow control, there are operational controls in JES3 for the operator to use at his or her discretion to override JES3 decisions, and to take control in unusual situations.

- ▶ Application programmers have special JES3 control statements (similar to JCL statements). There are control statements to make some jobs run only after successful (or unsuccessful) processing of other jobs, and for specifying the time of day, week, month, or even the year when jobs should run.
- ▶ System programmers have special JES3 initialization statements to define the way JES3 is to manage devices and jobs. Operators or application programmers can override many of these initialization options on a job-by-job basis. JES3 gives system programmers a unique way of setting installation policy for device and job management. They define separate groups of processing rules. Application programmers select and apply the groups of rules in various ways to individual jobs.

Where JES3 does not provide the exact function that your complex requires, such as special printing requirements, the system programmers can write their own routines and make them part of the JES3 program. This is done through installation exits provided with JES3 and through dynamic support programs that can be added to JES3. For diagnostic purposes, JES3 provides traces and dumps. Traces are copies of data made during normal processing to help monitor activity in the system, and dumps are copies of data made at times of failure.



## 3.42 JES3 job flow

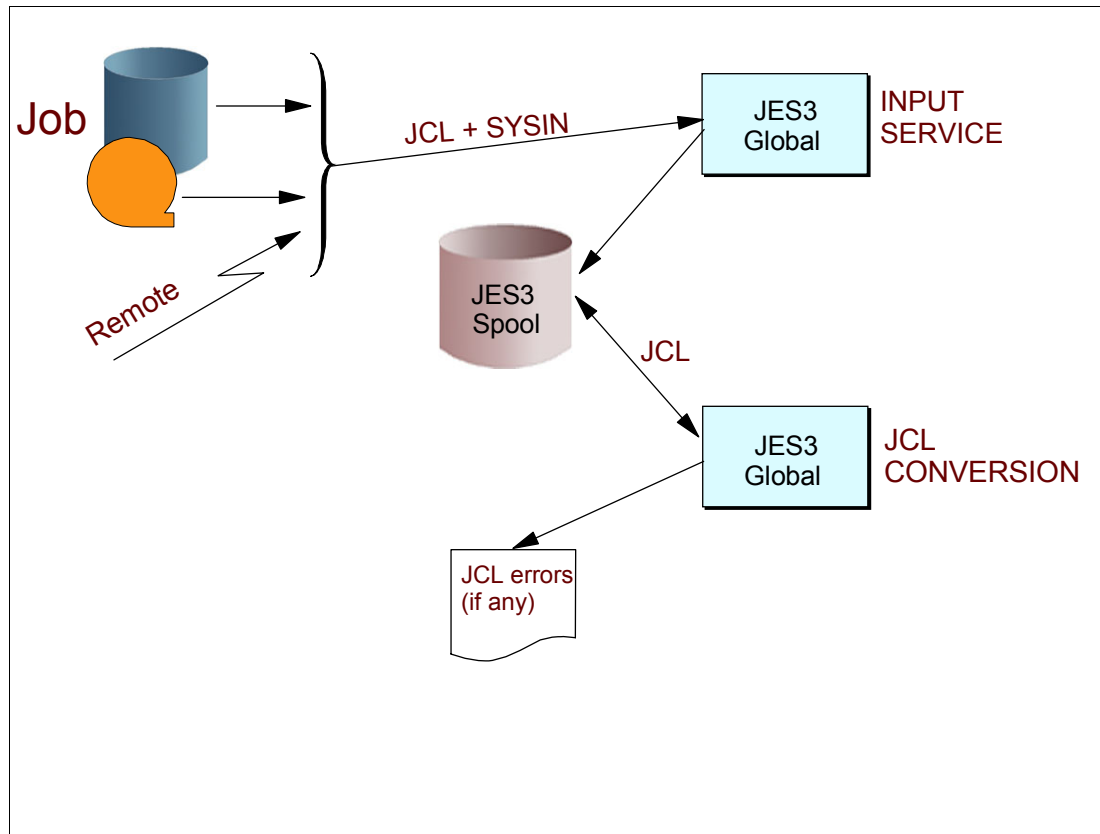


Figure 3-42 JES3 job flow

### JES3 phases of job processing

Following are the first two phases of processing for a job in a JES3 system:

- Input service

JES3 initially reads all jobs into the global and assigns to each job a unique JES3 job number from the available job number pool. Jobs can be submitted from a locally-attached tape, disk, or card reader. In addition, jobs can be submitted from remote job processing (RJP) workstations, time-sharing option (TSO/E) terminals, other systems in a job entry network, or by the internal reader.

START and MOUNT commands and TSO/E LOGONs cause jobs to be started from predefined procedures. Input service processes the JCL created for these jobs in the same manner as any other standard job.

Jobs initially placed on direct-access storage devices (DASD) and subsequently analyzed by JES3 input service are placed on the JES3 spool.

- Converter/interpreter processing (JCL conversion)

JES3 chooses the address space where it converts the job's JCL. The selected address space then reads the job's JCL from spool and converts and interprets the JCL. During this processing, JES3 flushes any job with JCL errors and determines the job-referenced data sets that require volume mounting. JES3 passes the information about the required resources to the JES3 main device scheduler (MDS) if a job requires devices, volumes, or data sets.

### 3.43 JES3 job flow (2)

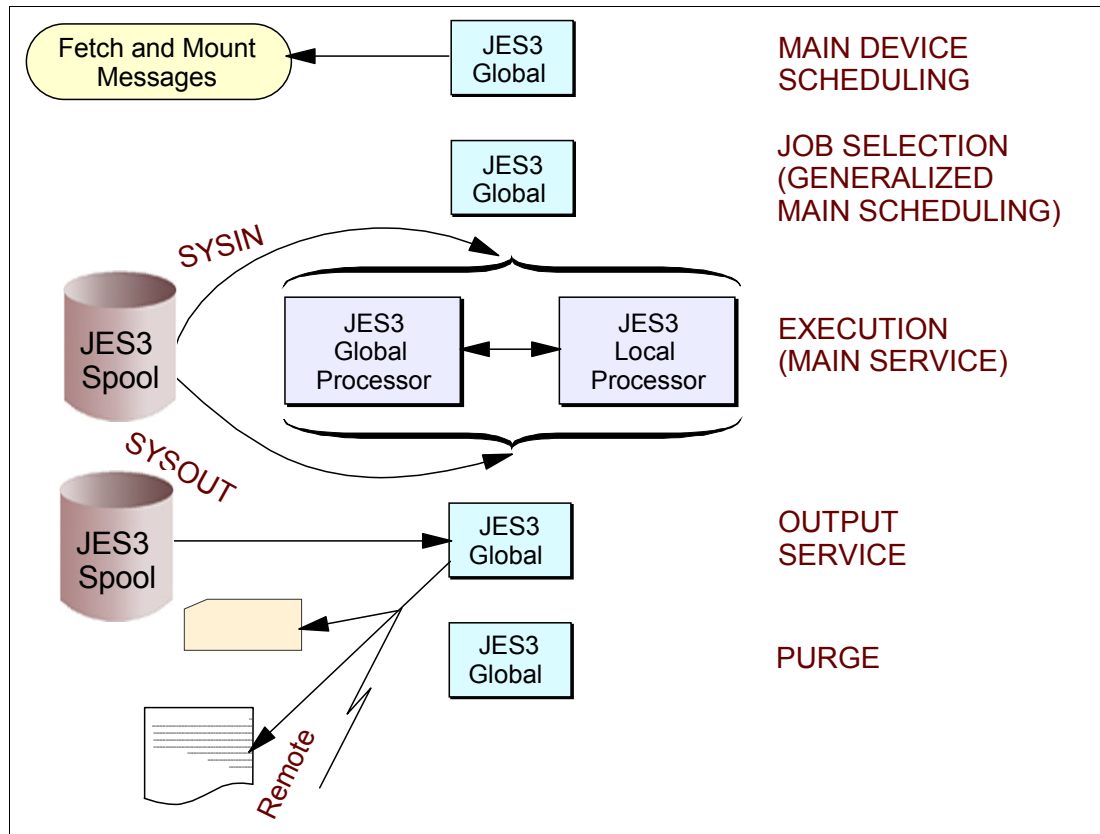


Figure 3-43 JES3 job flow (2)

#### JES3 job flow

After the first two phases, during which jobs are entered into the system and then passed through the converter and interpreter, the next phases are:

- ▶ **Main device scheduling (job resource management)**  
MDS ensures that resources (devices, volumes, and data sets) needed by the job are allocated before the job processes. MDS sends fetch messages to the tape and disk libraries, and mount messages to the setup operators to mount the volumes needed.
- ▶ **Generalized main scheduling (job selection)**  
JES3 schedules the job for processing based on specifications determined by the installation.
- ▶ **Processing (main service - job execution)**  
Jobs can run on the global or a local. z/OS controls the job during this phase. During processing, output generated by the job is usually written to the JES3 spool.
- ▶ **Output processing**  
Once processing is complete, JES3 processes the job's output data from the JES3 spool. Output data sets are printed and punched by JES3 output service when an available device matches the data set's requirements, such as forms, carriage forms control buffer (FCB), and train. JES3 output service informs the operator of any setup requirements of the data sets. The devices involved can be either local or remote, as defined by the job.

- Purge processing

After output processing is complete, the purge function releases all spool space associated with the completed job.

### **JES3 from a system programmer's point of view**

One way of visualizing JES3 is as a control program that performs job entry and job exit services. Another way of visualizing JES3 is as a language. Saying that JES3 is a control program is analogous to saying FORTRAN is a compiler. In reality, there is a FORTRAN language and a FORTRAN compiler. Similarly, there is a JES3 language and there is a JES3 program. Just as a FORTRAN programmer uses the FORTRAN language to define work the compiler is to do, JES3 system programmers use the JES3 language to define the work that JES3 is to do.

The JES3 language is made up of initialization statements, control statements, and commands. Operators and application programmers use parts of the language, but it is system programmers who define the total JES3 environment. Often, what is specified on JES3 initialization statements can be overridden with JES3 commands or control statements. While system programmers may not themselves use commands or control statements, they may have to instruct operators or application programmers on when and how they should be used.

A further level of defining how JES3 should do its work comes in the form of installation exits supplied with JES3. For example, in the area of controlling the print output from jobs, a system programmer can:

- Define initial values in the initialization stream
- Allow the operator to make certain modifications to the initial values
- Write an installation exit routine to further modify or control what should happen to the output

## 3.44 JES3 job flow review

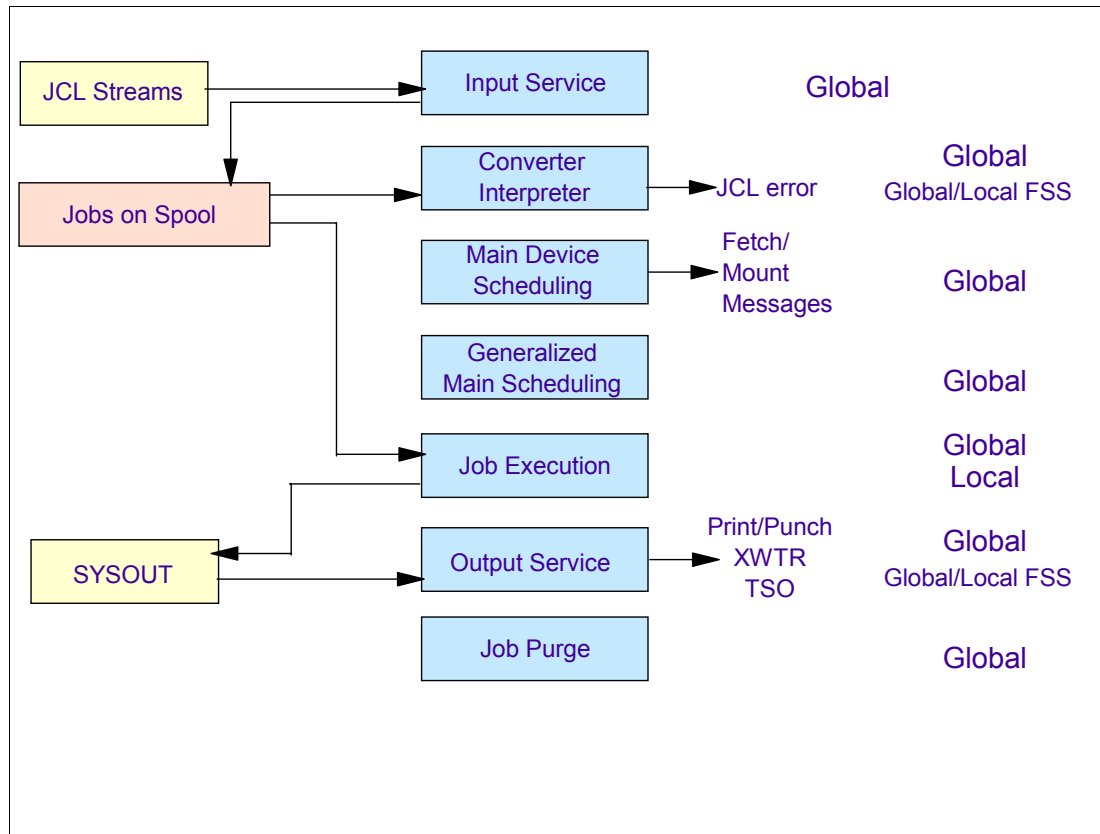


Figure 3-44 JES3 job flow review

### Review of JES3 job flow

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

- ▶ Creation and scheduling of a card reader job.
- ▶ Reading of the input stream by a DSP.
- ▶ Building of JCT entries for each job in the input stream.
- ▶ Execution of 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.

### JES3 input service

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 knows what resources the job will require 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. 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.

### 3.45 JES3 job flow: Scheduler elements

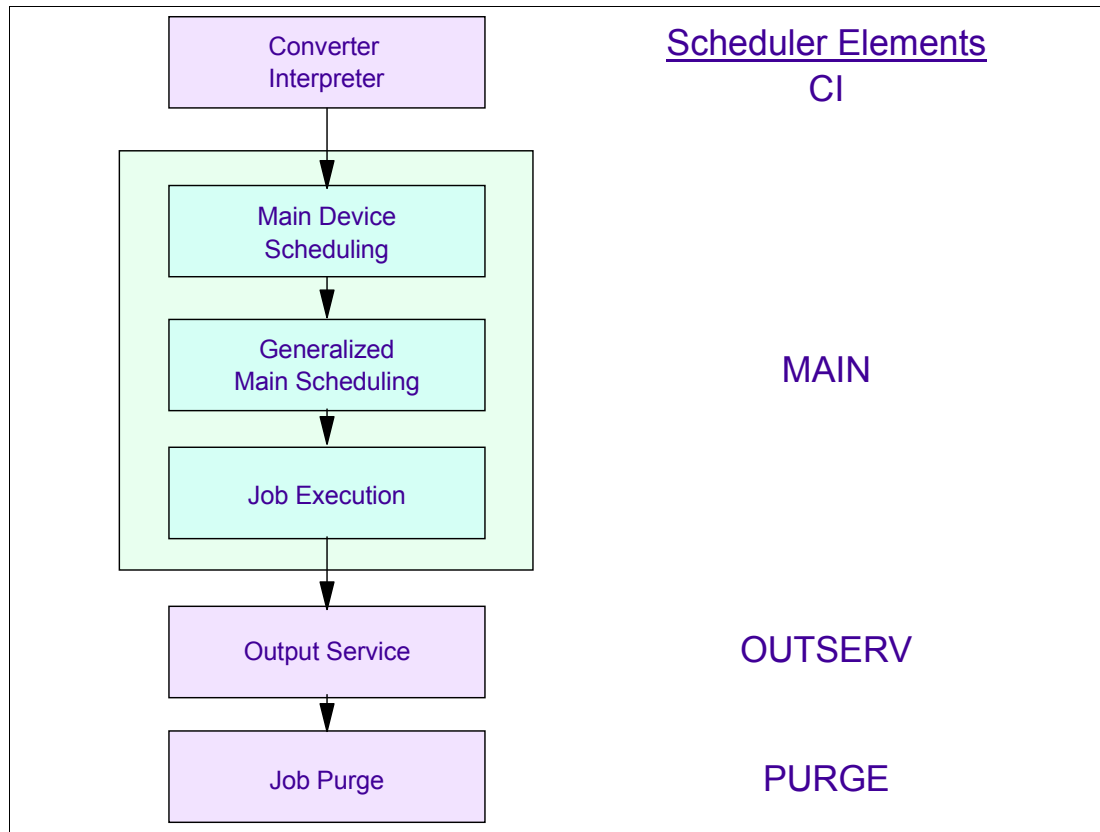


Figure 3-45 JES3 job flow: Scheduler elements

#### JES3 scheduler elements

JES3 processing performed on behalf of jobs consists of distinct phases. A typical job originating in the input stream requires the following phases:

- ▶ Converting the job's JCL to a form usable by MVS.
- ▶ Insuring I/O resources needed by the job are available, and passing the job to MVS.
- ▶ Handling the job's SYSOUT data.
- ▶ Removing the job from the system.

Each of the four phases of a job is represented by a scheduler element (SE) in the JCT entry for the job shown in the Figure 3-45. Every scheduler element denotes a unit of work JES3 must perform to process the job. Each is represented on the FCT chain by one or more DSPs that performs the work required for that type of SE. Scheduler elements needed for job A are listed here (the actual DSP name is shown in capital letters).

There are two types of jobs in the JES3 complex:

- ▶ The standard job
- ▶ The non-standard job

## Standard jobs

Every job containing JCL is considered a standard job. JES3 places into the JCT entries of all standard jobs the same four scheduler elements in the same order:

- ▶ Converter/Interpreter (CI)
- ▶ Main service (MAIN)
- ▶ Output service (OUTSERV)
- ▶ Purge (PURGE)

Each JCT entry contains information about the job and an entry for each SE. Each SE entry contains flags indicating that SE's status: inactive, active, or complete. JSS uses these flags to determine which SE is next for scheduling/ending functions. The first SE that is not marked complete is selected, and is marked either complete or active.

## 3.46 JES3 standard job

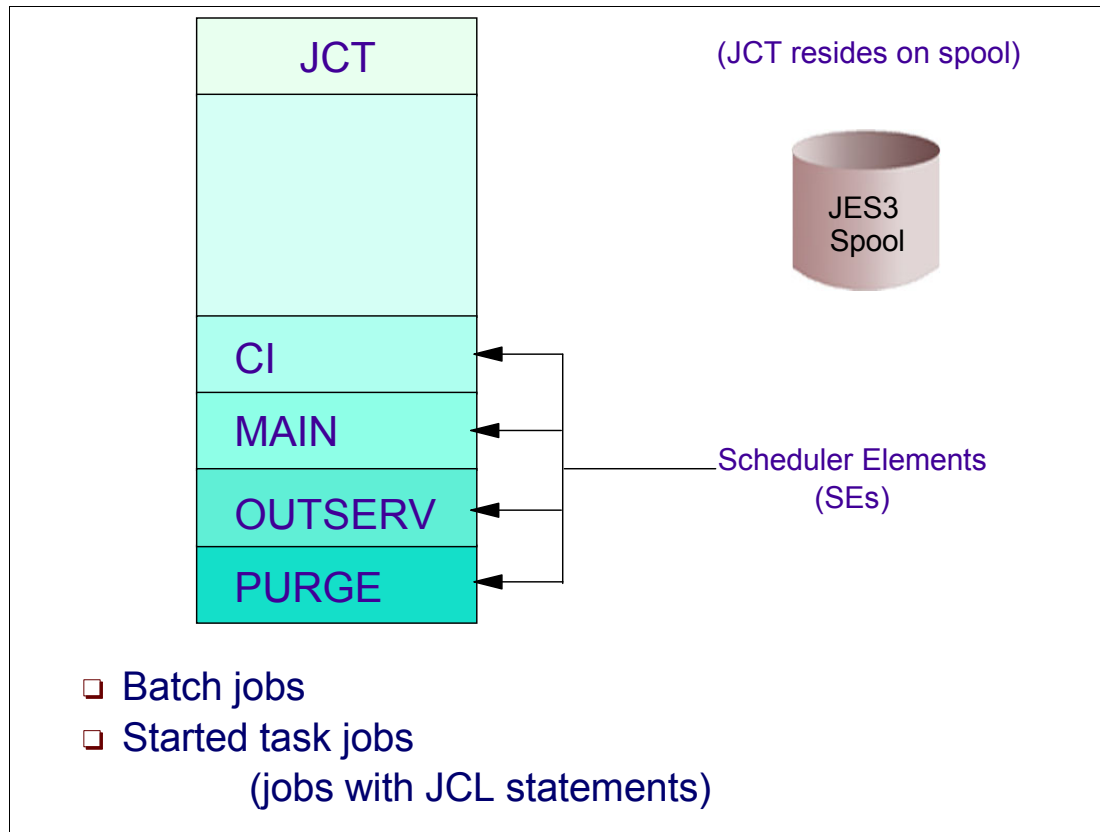


Figure 3-46 JES3 standard job

### JES3 standard jobs

The names in parentheses are formal names of the DSPs that are executed to perform the work. The number or type of scheduler elements to be used for standard jobs can be changed by means of a user exit routine; such a change would apply to all standard jobs.

JES3 job management consists of the following phases:

- ▶ Input service (when the jobs is read in to the system).
- ▶ Converter/interpreter service
- ▶ Resource allocation
- ▶ Job selection and scheduling
- ▶ Output service
- ▶ Purge

**Note:** Resource allocation and JOB selection and scheduling is performed by the MAIN scheduler element.



### 3.47 JES3 non-standard job

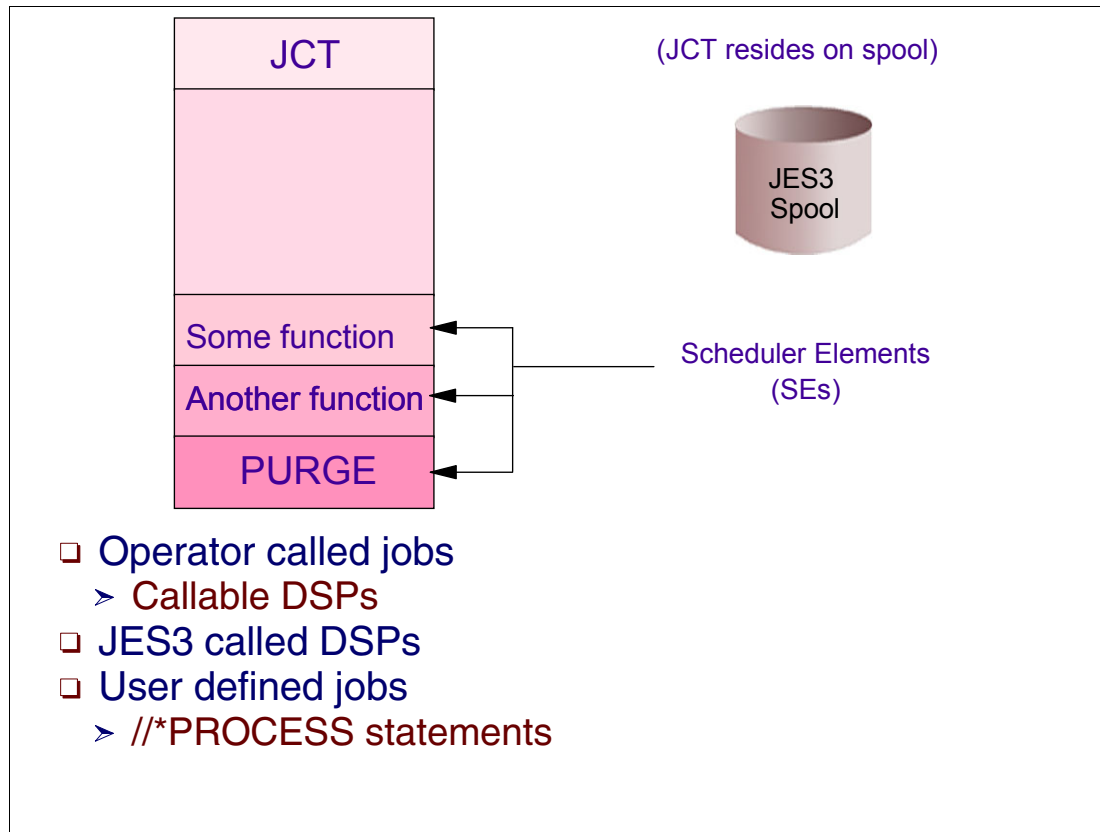


Figure 3-47 JES3 non-standard job

#### JES3 non-standard jobs

A non-standard job is a job for which the user determines the number of segments and their order of execution. To specify a different set of scheduler elements on a job-by-job basis, code special JES3 control statements and include them with JCL statements for the jobs.

Called jobs, which are non-standard, are created by operator request. The operator uses JES3 `*CALL` commands to make the requests. Called jobs are unique because they are not defined by JCL; there is, in fact, no JCT involved at all. These jobs are internally generated by JES3 in response to the `*CALL` command, and their JCT entries always contain two scheduler elements:

- ▶ One to represent the DSP needed for the request
- ▶ One to remove the called job from the system (PURGE)

## 3.48 Creating a batch job

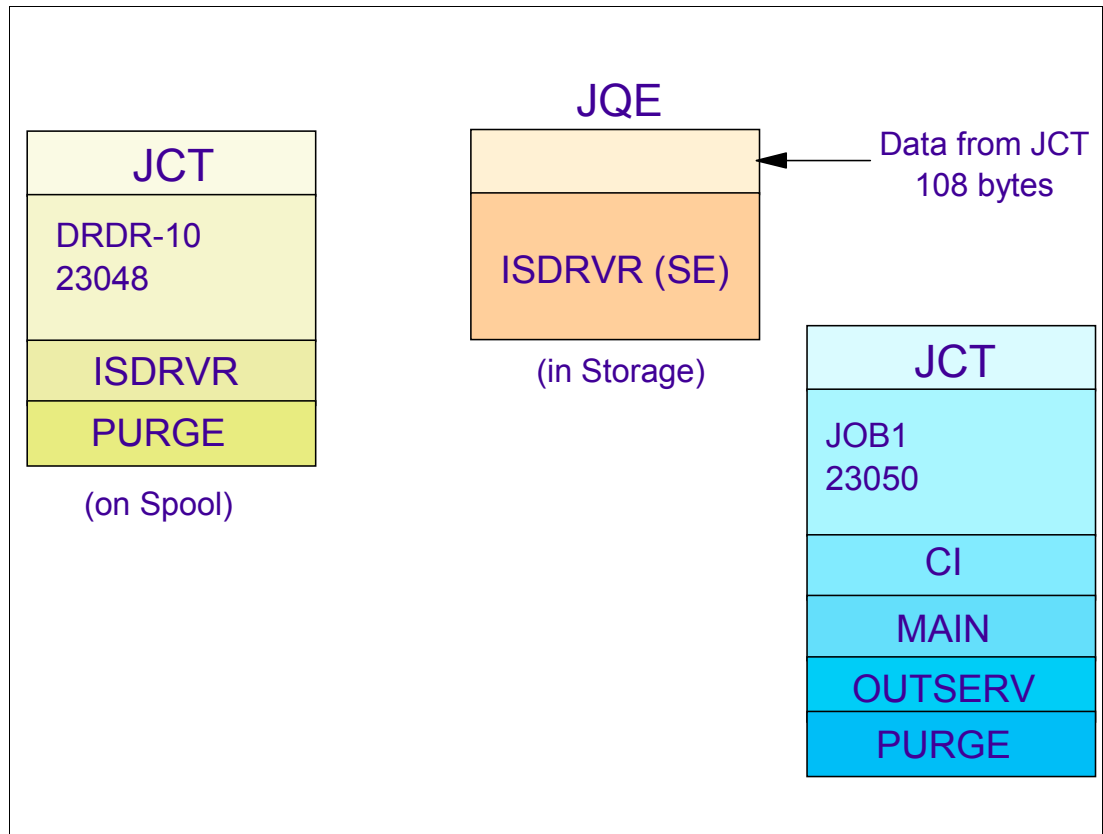


Figure 3-48 Creating a batch job

### Creating batch jobs

For each job in the input stream, ISDRVR creates a JCT entry for each MVS job in the batch.

Information placed into the JCT entry for a job includes:

- ▶ A JES3 job number
- ▶ The jobname from the JOB statement
- ▶ The job priority and class
- ▶ A symbolic origin for the job (for returning job output to the origin)
- ▶ Information from JES3 control statements
- ▶ Scheduler elements representing the DSPs needed to process the job

Since the sample job is a standard job, the scheduler elements will be CI, MAIN, OUTSERV, and PURGE

### 3.49 Converter/interpreter processing

- ☐ First scheduler element for standard job
- ☐ Interface to MVS converter/interpreter
- ☐ Construct SWA - write to spool
- ☐ Flush jobs with JCL errors
- ☐ Determine devices required
- ☐ Determine volumes required
- ☐ Establish data set awareness
- ☐ 13 user exits
- ☐ Uses subtasks - number is user defined
- Controllable by operator

Figure 3-49 Converter/interpreter processing

#### Converter/interpreter processing

The CI DSP serves as an interface between JES3 and the JES3 CI subtasks that invoke the MVS converter/interpreter subtasks. The interface responsibilities of the CI DSP are:

- ▶ To fail the job if JCL errors are detected by the MVS converter/interpreter
- ▶ To fail any job that contains more JCL statements than the limit allows
- ▶ To delay processing of any job that would temporarily cause the C/I subtasks to process more JCL statements than the system limit allows
- ▶ To copy the SWA control blocks that are produced by the MVS converter/interpreter onto spool

The CI DSP stores I/O requirements in job-related control blocks called the job summary table (JST) and job volume table (JVT). It writes these control blocks to spool along with others for the job. These control blocks and some that are kept in main storage allow JES3 to maintain a complex-wide awareness of the status of volumes and data sets.

During JES3 initialization, JES3 attaches an installation-defined number of converter/interpreter (C/I) tasks. These C/I tasks are subtasks to the JES3 primary task. When a C/I subtask processes a job, the scheduler work area (SWA) in the address space in which the service is invoked (global functional subsystem) provides temporary storage for the job's JCL statements. These statements remain in the SWA until the C/I subtask finishes processing the job. When several C/I subtasks run concurrently, the SWA contains the JCL statements for all the jobs these subtasks are processing.

## 3.50 Main scheduler element

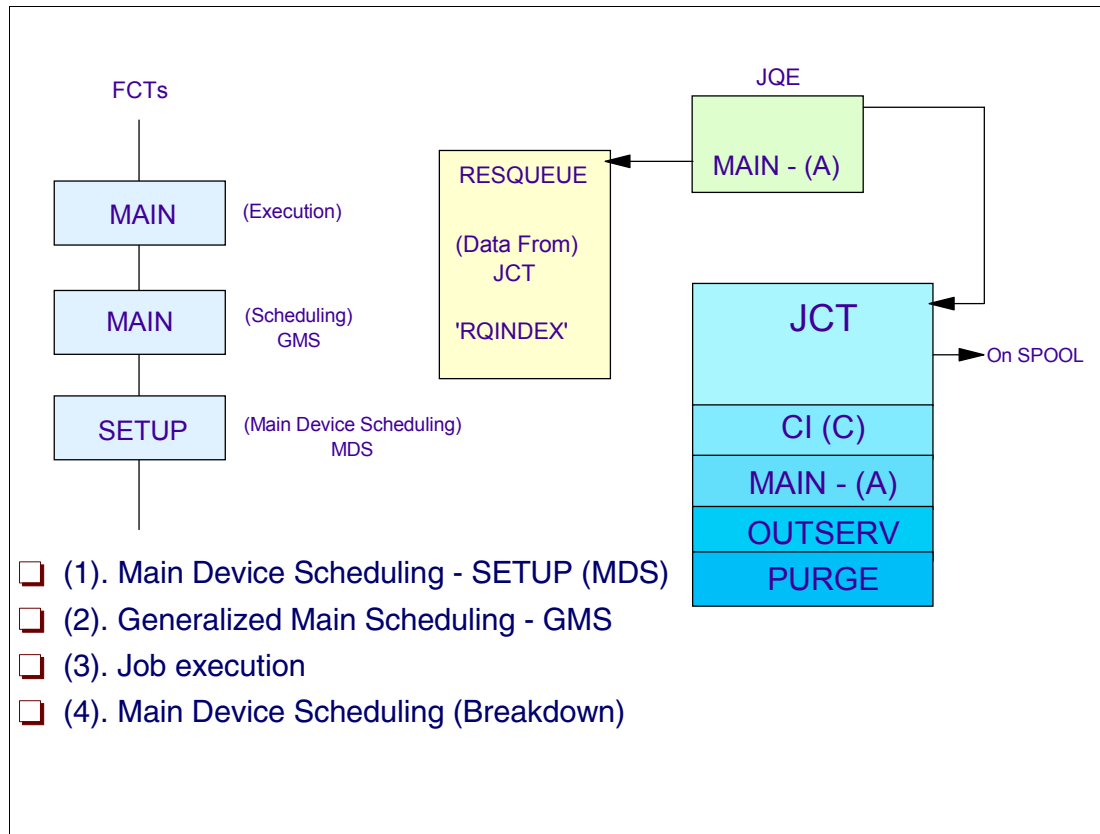


Figure 3-50 Main scheduler element

### Main scheduler element processing

When the MAIN scheduler element is scheduled, the following functions occur under the FCTs, as illustrated in Figure 3-50. Main device scheduling is the allocation of device, volume, and data set resources to jobs, the selection of jobs to be passed to MVS initiators for execution, and the freeing of allocated resources after the jobs are executed.

For standard jobs, main service processing begins when the job segment scheduler reaches the MAIN scheduler element in the job's JCT entry. The MAIN scheduler element represents two DSPs: setup (SETUP) and generalized main scheduling (MAIN). Both of these DSPs are resident and each has a permanent entry in the FCT chain, so the job segment scheduler need not construct the FCT entries.

The work performed by the SETUP and MAIN DSPs is crucial to JES3 processing. The goals of the SETUP and MAIN DSPs are effective resource utilization and maximum job throughput. The processing sequence is:

- ▶ Initial setup processing to prepare I/O resources (MDS)
- ▶ Generalized main scheduling to select and pass a job to an initiator
- ▶ Job execution
- ▶ Breakdown processing to relinquish I/O resources

## 3.51 Main device scheduling (MDS)

- ❑ MDS processing options
  - Volume fetch
  - Job setup
  - High watermark setup (HWS)
  - Explicit setup - `//*MAIN SETUP=`
  - NONE - (makes MDS optional)
- ❑ Schedules resources to jobs
  - Devices - volumes - data sets
- ❑ Objectives
  - Satisfy resource requirements before execution

Figure 3-51 Main device scheduling - MDS

### Main device scheduling

Main device scheduling (MDS) represents the second phase of setup processing. Converter/interpreter is the first phase. The converter/interpreter routines construct a job summary table (JST) that lists required data sets and devices, and a job volume table (JVT) that describes the volumes the main's device scheduling routines will fetch and allocate. Volumes that are mounted will be verified.

Main device scheduling functions are optional and may be bypassed for a job as well as a complex. The purpose of MDS is to allocate I/O resources among competing jobs and release resources after they have been used. JES3 setup processing is defined by JES3 initialization statement parameters, JCL control statements, and JES3 operator commands. JES3 setup is available through the following options:

- ▶ Volume fetch
- ▶ Job setup
- ▶ High-watermark setup
- ▶ Explicit setup

## 3.52 Main scheduler element processing

- ❑ Generalized Main Scheduling - GMS
  - User defined algorithms - jobs to initiators
- ❑ Job execution
  - Monitor jobs during execution
  - Operator communication
- ❑ Main device scheduling - Breakdown
  - Return resources - devices, volumes, data sets

*Figure 3-52 Main scheduler element processing*

### Main scheduler element processing

JES3 job scheduling 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 scheduling involves the routines invoked by the MAIN DSPs, which are represented by the MAIN scheduler elements on the job control table entry.

GMS selects jobs to be processed by MVS initiators. GMS selects a job for execution using the job selection algorithms established at JES3 initialization. MAINPROC, SELECT, CLASS, and GROUP initialization statements control the key variables in the job scheduling and job execution process.

During execution, the operator can monitor and cancel jobs.

When a step completes or the job ends, MDS breakdown occurs, which allows the freeing of resources, devices, volumes, and data sets.

### 3.53 OUTSERV scheduler element processing

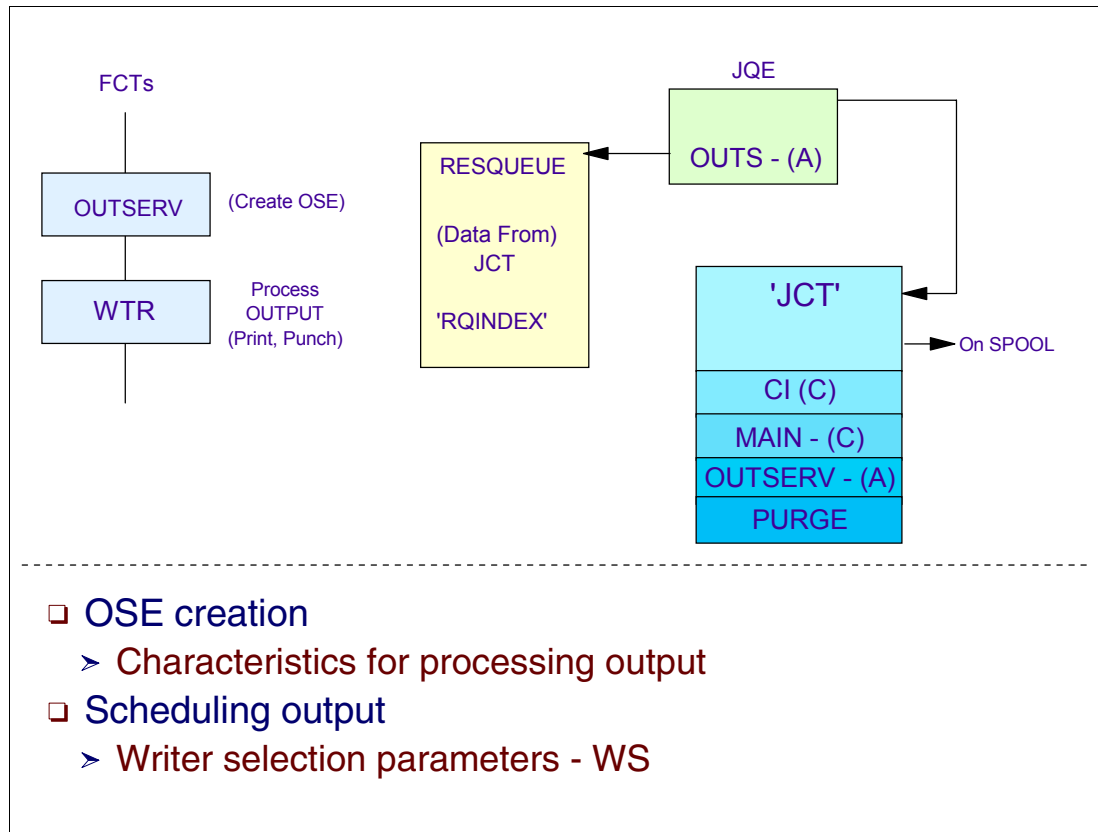


Figure 3-53 OUTSERV scheduler element processing

#### OUTSERV scheduler element processing phases

Following are the processing phases for output processing:

- ▶ Phase 1 - Queuing of output
- ▶ Phase 2 - Scheduling of output
- ▶ Phase 3 - Printing and punching of output

Phases 1 and 2 occur in the JES3 global address space on the global processor. Phase 3 can run in the global address space or in a functional subsystem (WTR FSS).

- ▶ The OUTSERV DSP summarizes output data sets at two points of processing:
  - For normal jobs, when a job completes main service processing
  - For spin-off data sets, when a job or a DSP moves a data set directly to output service for processing before the job ends
- ▶ JES3 creates output service elements (OSEs) to summarize the output data set characteristics for scheduling. An OSE represents all of a job's output data sets with similar scheduling characteristics. One job can have several OSEs. OSEs are later used to schedule output to specific devices.
- ▶ The JES3 WTR DSP handles printing and punching of output. The WTR DSP reads from spool and writes to the specific output device, thereby performing device dependent functions. Output can be sent to a variety of devices. For locally attached devices that are driven by JES3 in the global address space, the WTR DSP writes directly to the device. For non-locally attached devices that are not driven by the JES3 global address space, the WTR DSP passes output to the appropriate interface. This can be RJP, SNARJP, NJE services, the internal writer, or a functional subsystem (FSS).

## 3.54 Output service processing

- ❑ Output writers
  - Hot writers - \*X WTR,.....
  - Dynamic writers
- ❑ Output types processed
  - Print and punch
  - External writer - PSO
  - TSO users via OUTPUT command - PSO
- ❑ 10 user exits
- ❑ SMF type 6 records

*Figure 3-54 Output service processing*

### Output service processing

Hot writers give operations personnel total control of output handling. Operators enter commands to call and control hot writers. Hot writers remain available, even when there is nothing to print.

Dynamic writers are often used for volume printing on stock paper. These writers allow JES3 to control changing the setup characteristics for devices, thereby reducing the amount of control operators have over when and how writing is performed.

The output types processed are by device type. This characteristic indicates which type of device is to receive the output. Data sets that were defined as “print type” will be sent to printers, “punch type” will be routed to punches, and “sys type” will be routed to TSO or to external writers.



## 3.55 Purge processing

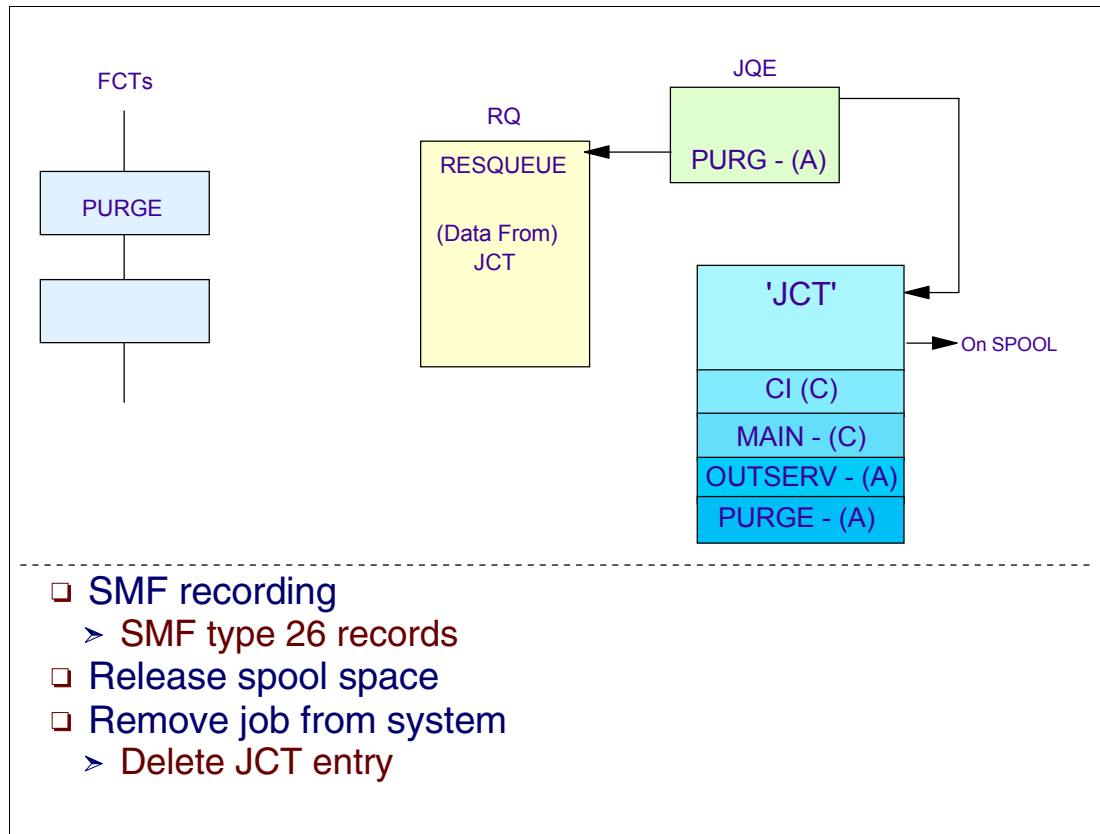


Figure 3-55 Purge processing

### 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. Purge processing consists of the following steps:

- ▶ JES3 releases all spool space assigned to the job and updates resident control blocks.
- ▶ JES3 writes SMF record type 25, which contains device allocation information.
- ▶ JES3 writes SMF record type 26, which contains final job accounting information.
- ▶ JES3 informs the operator that the job has been purged.
- ▶ The JCT entry is deleted.

## 3.56 JES3 functions not in JES2

- ☐ Main Device Scheduling (MDS)
- ☐ Generalized Main Scheduling (GMS)
- ☐ Dependent Job Control (DJC)
- ☐ Deadline scheduling
- ☐ Interpreter processing
- ☐ Priority aging

*Figure 3-56 JES3 functions not in JES2*

### **Main device scheduling**

JES3 provides a device management facility called the main device scheduler (MDS) that can wholly or partially support the MVS allocation process. The purpose of MDS is to satisfy job resource requirements (the devices, volumes, and data sets needed) before and during job execution, thus allowing execution to proceed without allocation delays. MDS also allows controlled multisystem access to commonly accessible data sets in the loosely coupled environment.

### **Generalized main scheduling**

Each time an MVS initiator requests work, generalized main scheduling (GMS) selects and schedules a job for execution. The job that GMS selects depends primarily upon initialization parameters that you have specified. Deadline scheduling and dependent job control (DJC), additional GMS functions, enable you to control when jobs execute. With deadline scheduling, you specify a deadline by which you want the job to run. JES3 periodically increases the job's selection priority in an attempt to run the job by the specified deadline. DJC allows you to create a network of related jobs.

### **Dependent job control**

Dependent job control (DJC) is a means of coordinating the processing of jobs. With DJC, one job will not run unless some other job has run first. (DJC has an MVS counterpart for conditional execution of job steps.) DJC requires no advance preparation with JES3

initialization statements. Application programmers can specify all relationships between jobs on JES3 control statements they submit with the affected jobs.

### **Deadline scheduling**

Deadline scheduling is a way of scheduling jobs by time of day or by week, month, or year. Job priorities remain in force, but as deadlines approach, JES3 increases the priorities, thereby increasing the likelihood the jobs will be processed before a certain deadline. System programmers must prepare for deadline scheduling with JES3 initialization statements that tell when a priority is to be increased, and by how much. Then application programmers can put deadlines on JES3 control statements they include with the jobs.

### **Interpreter processing**

Converter/Interpreter (C/I) service controls the conversion of JCL statements to Converter/Interpreter text and then into control blocks. This service comprises primarily the JES3 CI and POSTSCAN dynamic support programs, the C/I subtasks under which the MVS C/I routines run, and the initiator support routines. C/I service controls the conversion and interpretation of a job's JCL. The three principal phases of C/I service are:

- ▶ Converter/Interpreter phase: Uses the MVS C/I routines to convert and interpret JCL into scheduler control blocks. At this time, the scheduler control blocks are created in the scheduler work area (SWA).
- ▶ Prescan phase: Extracts job requirements from the scheduler control blocks for use in the postscan phase. At the end of the prescan phase, the scheduler control blocks are moved from the SWA to JES3 spool.
- ▶ Postscan phase: Locates data sets and gathers information for use in JES3 device setup.

### **Priority aging**

When all initiators are busy, throughput of certain jobs might fall below normal expectations. To help in these situations, JES3 uses the additional scheduling function of priority aging. Priority aging can help ensure that jobs that have been waiting to run have a chance of being selected to run before those jobs that just entered the system. By using priority aging, an installation can increase the priority of a waiting job. The longer the job waits, the higher its priority becomes, up to a limit, and the greater its chances of being selected to run.

### 3.57 Dynamic support program (DSP)

#### ❑ JES3 functions called DSPs

- Readers, writers, converters, .... etc.
- Many functions started automatically
- Other functions started by the operator
  - Operator callable DSPs    \*X dspname
  - Called DSPs become JES3 jobs

\*X WTR,OUT=PRT1

\$S PRT1 (JES2)

|            |
|------------|
| JCT        |
| WTR<br>123 |
| WTR        |
| PURGE      |

Figure 3-57 Dynamic support program (DSP)

#### Dynamic support programs (DSPs)

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, writes, converts, 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.

## 3.58 JES3 and consoles

- ❑ JES3 may be operated from any console attached to any system
- ❑ From any console, an operator can
  - Direct a command to any system
  - Receive the response to that command
- ❑ Any console can be set up to receive messages
  - From all systems
  - Specific systems

*Figure 3-58 JES3 and consoles*

### **JES3 and consoles**

JES3 may 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.

## 3.59 Issuing commands

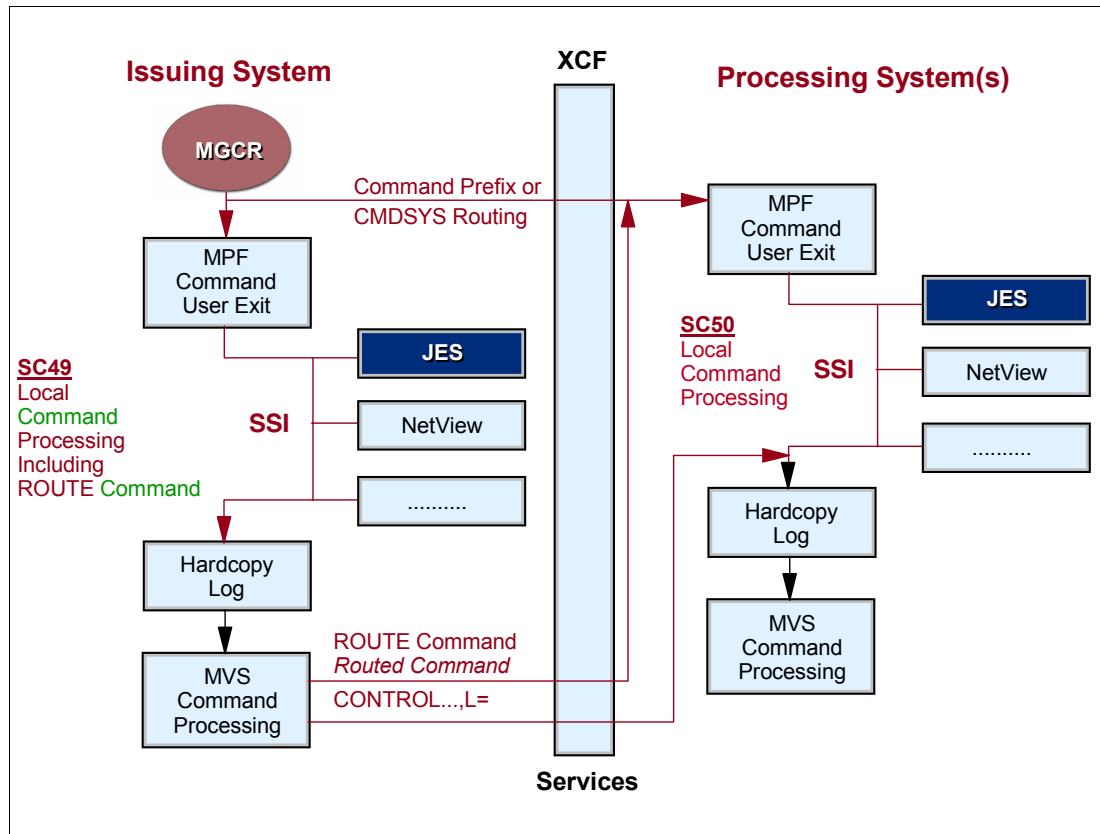


Figure 3-59 Issuing commands

### Command processing

You must establish a command prefix for each JES3 system since commands are routed also by the prefix. The routing of commands occurs as follows:

- Command prefix and CMDSYS routing

If the command contains a prefix or the console has a CMDSYS specification that directs the command to another system, the command is immediately transmitted using XCF services to the processing system.

**Note:** For command prefix and CMDSYS routing, the command is first transported to the receiving system before any system symbol substitution takes place.

A REPLAY command is sent to the system where the WTOR was issued. If the REPLAY command text contains system symbols, substitution occurs on the receiving system.

- MPF command user exit

If the command is not transmitted to another processing system, it is processed on the issuing system by the installation MPF command exit routines. The exits are specified using the .CMD statement in the MPFLSTxx parmlib member. These exits can perform authority checking, modify the command text, or the command processing.

**Note:** For commands containing system symbols, substitution has occurred before the exit is entered.

► SSI processing

The command is broadcast on the subsystem interface (SSI) to all active subsystems. Each subsystem inspects the command and decides whether to process it. The subsystems base the decision on the command prefix characters of the command string. For example, by default, NetView looks for a percent sign (%) and process the commands starting with the % sign.

When a subsystem decides to process the command, the command is passed to subsystem processing, and a return code is set to indicate that the command was processed by a subsystem.

**Note:** At this point in processing, all system symbol substitution has occurred. The original command text is also available.

► Hardcopy log

Once the command has been examined by all active subsystems, it is logged to the &hc.log (usually SYSLOG or OPERLOG).

**Note:** The hardcopy log contains the command before any system symbols contained in the command have been substituted, and it also contains the command after substitution has occurred.

► MVS command processing

If none of the subsystems has marked the command as having been processed, it is assumed to be an MVS command and is passed to the appropriate MVS command processor. If a command processor does not exist, an error message is issued stating that the command is invalid.

## 3.60 Consoles on each system

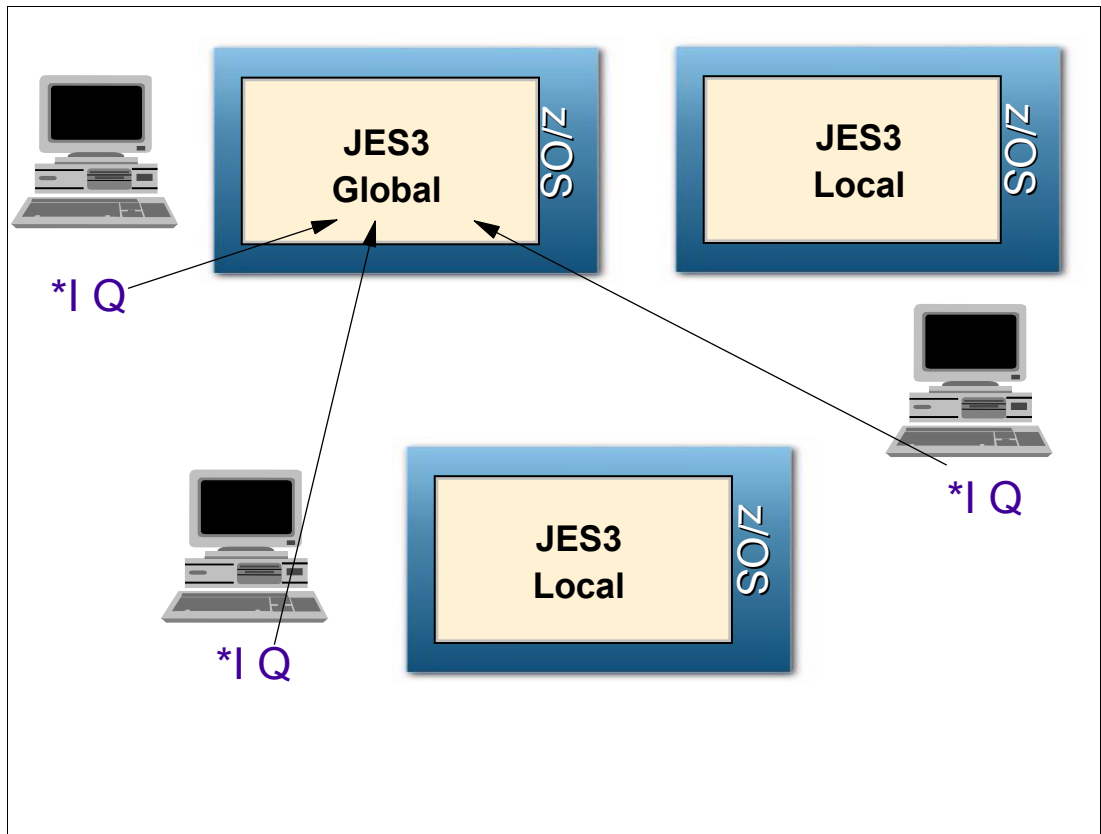


Figure 3-60 Consoles on each system

### Defining consoles

Consoles are devices that you use to enter commands and receive messages from JES3, MVS, and application programs. Consoles fall into one of the following classes:

- ▶ Multiple console support (MCS) consoles
- ▶ Remote job processing (RJP) consoles

MCS consoles are devices that you can physically attach to global or local processors. These consoles allow you to control the sysplex. Refer to *z/OS MVS Planning: Operations*, SA22-7601 for information about MCS consoles in a sysplex environment.

RJP consoles are devices that you attach to the JES3 global as part of a remote workstation using telecommunications lines. RJP permits you to submit jobs and receive output at workstations that can reside at some distance from your installation.

See *z/OS JES3 Initialization and Tuning Reference*, SA22-7550 for more information about defining RJP consoles.



## 3.61 JES3 commands

- ❑ \*INQUIRY commands are used to request information - (\*I)
- ❑ \*MODIFY commands used to change specifications in jobs - (\*F)
- ❑ \*CALL (\*X), \*START (\*S), \*RESTART (\*R), \*CANCEL (\*C), and \*FAIL commands used to control dynamic support programs (DSPs)
- ❑ \*VARY (\*V) commands and the \*MODIFY,V (\*F V) command are used to control devices and other JES3 resources
- ❑ \*SWITCH and \*FREE commands used to control RJP consoles
- ❑ \*MESSAGE commands are used to communicate with other consoles and with remote nodes
- ❑ \*SEND (\*T) commands route commands to a remote node
- ❑ \*DUMP and \*RETURN are used to terminate JES3

Figure 3-61 JES3 commands

### JES3 operator control

Commands are requests you make to the system. You can use commands to control devices, to change specifications previously made to the system, or to display information about the operator's console. The first element of a command is a prefix that informs JES3 that a command is being entered. The default JES3 command prefix is the asterisk character (\*).

These are the basic JES3 commands and their purposes:

- ▶ \*INQUIRY commands are used to request information. No action will be taken except that the requested information will be displayed.
- ▶ \*MODIFY commands are used to change specifications given in jobs or previous commands, or during initialization.
- ▶ \*CALL, \*START, \*RESTART, \*CANCEL, and \*FAIL commands are used to control dynamic support programs (DSPs), such as utilities.
- ▶ \*VARY commands and the \*MODIFY,V command are used to control devices and other JES3 resources.
- ▶ \*SWITCH and \*FREE commands are used exclusively to control RJP consoles.
- ▶ \*MESSAGE commands are used to communicate with other consoles and with remote nodes.
- ▶ \*SEND commands are used to route commands to a remote node.
- ▶ \*DUMP and \*RETURN commands are used to end JES3.
- ▶ \*TRACE command is used to trace certain JES3 events.

## 3.62 Controlling JES3 jobs

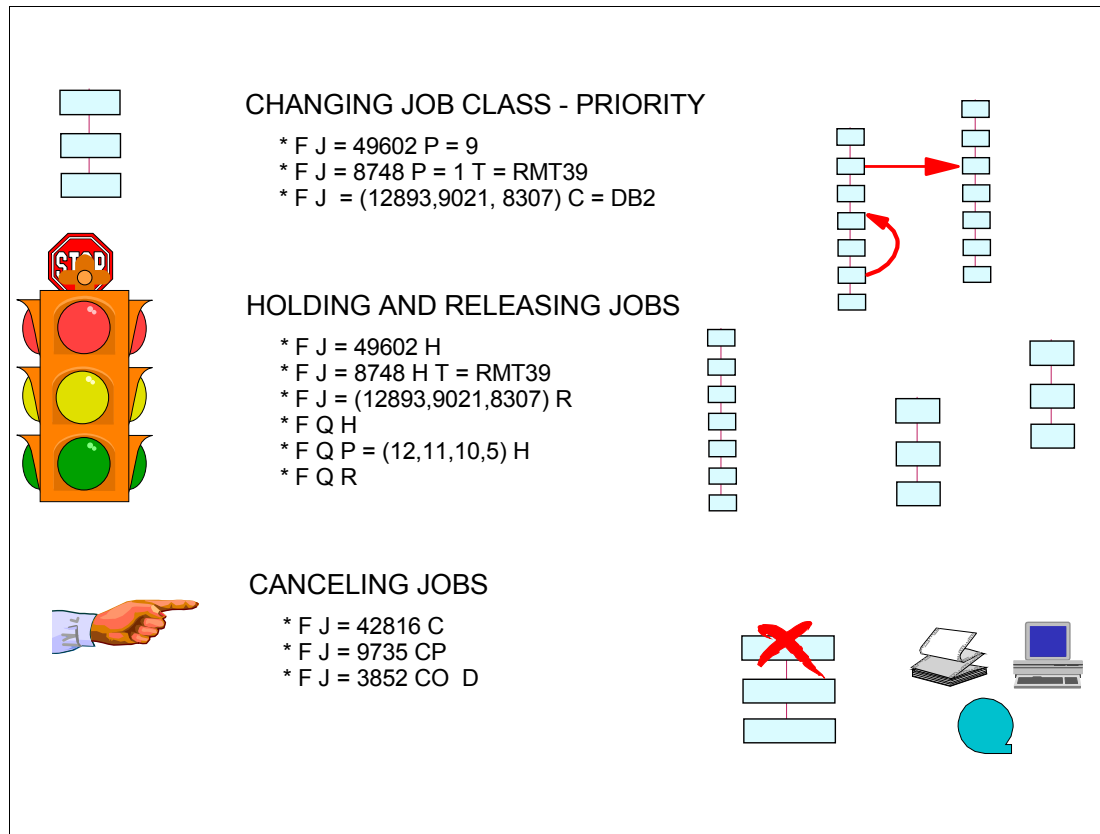


Figure 3-62 Controlling JES3 jobs

### Controlling JES3 jobs

Use the **\*MODIFY,J=** or the **\*F,J=** command to:

- ▶ Hold a job, **\*F,J=nnnn H**
- ▶ Release a job, **\*F,J=nnnn R**
- ▶ Cancel a job, **\*F,J=nnnn C** or **CP** or **CO**
- ▶ Change a job's priority, **\*F,J=nnnn P=nn**
- ▶ Change a job's job class, **\*F,J=nnnn C=class**
- ▶ Change a job's JESMSG LG logging status, **\*F,J=nnnn LOG** or **NOLG**

Use the **\*MODIFY,Q** or **\*F,Q** command to:

- ▶ Hold or release all the jobs in the JES3 job queue, **\*F Q H** or **R**
- ▶ Hold or release jobs of a designated priority, **\*F Q P=nnn H** or **R**

## 3.63 Controlling job input

### ❑ DSP dictionary entry for ISDRVR

```
ISDRVR  IATYDSD      PRTY = 4,PABLE = YES,      *
                                CSECT = IATISDT,REENT = YES,  *
                                DRVR = IATISDV,MAXCT = 2
```

### ❑ Inquiry DSP entry ISDRVR

```
*I X,D=ISDRVR
IAT8475 (IQDX) - ISDRVR  MXCT=000002 USE=000000 MOD=YES
IAT8472 (IQDX) - ISDRVR  IS NOT IN HOLD
```

### ❑ Modify DSP entry ISDRVR

```
*F X,D=ISDRVR,MC=5
IAT8484 (MODX) - ISDRVR  OLDMXCT=000002 NEWMXCT=5
```

Figure 3-63 Controlling job input

## Controlling job input

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 represented 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.

### IATYDSD macro

The IATYDSD macro generates an entry for a dynamic support program (DSP) in the DSP dictionary (module IATGRPT or, in a C/I FSS address space, module IATGRPTF). An entry in the table is required for each DSP in order for it to be recognized as part of JES3. The following are considerations:

|        |                                                                                                                                                                                                                                                                                                                     |
|--------|---------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| ISDRVR | The 1 to 8 character name of the DSP whose entry is being created by this macro. If the DSP is to be callable, this name will appear as the argument of an <b>*X,dspname</b> command. If the DSP is to be processable, this name will appear in the <b>/*PROCESS dspname JCL</b> statements. The label is required. |
| PRTY   | The priority to be assigned to the DSP, in the range from 1 through 255. This priority becomes the FCT priority when the DSP is activated.                                                                                                                                                                          |
| REENT  | Specifies that the DSP is reenterable.                                                                                                                                                                                                                                                                              |
| DRVR   | The name of the DSP driver module to be loaded, if necessary, for each use of the DSP.                                                                                                                                                                                                                              |

|       |                                                                                                                                                                                                                                                                                                                                      |
|-------|--------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| CSECT | The name of the data CSECT to be loaded by the job segment scheduler driver (module IATGRJR) for each use of the DSP.                                                                                                                                                                                                                |
| MAXCT | The maximum number of copies of this DSP that may be concurrently active. The number must be within the range 1 through 65535. If this parameter is not specified for a DSP specifying REENT=YES, no MAXCT limit is imposed. This parameter is dynamically alterable by using the <b>*MODIFY</b> command unless you specify MUCC=NO. |

JSS schedules all DSPs into execution by building an FCT and adding the FCT to the FCT chain in priority sequence.

## 3.64 Commands for job queue status

```
❏ *I Q - *I Q,N=ALL - *I Q,N=25

*I Q
IAT8674 JOB NJECONS (JOB02539) P=15 NJECONS (ACTIVE)
IAT8674 JOB NJERDR (JOB02540) P=15 NJERDR (ACTIVE)
IAT8674 JOB NJERDR (JOB13925) P=15 NJERDR (ACTIVE)
IAT8674 JOB DEADLINE (JOB19683) P=15 DEADLINE (ACTIVE)
IAT8674 JOB SNARJP (JOB13067) P=15 SNARJP (ACTIVE)
IAT8674 JOB MONITOR (JOB13361) P=15 MONITOR (ACTIVE)
IAT8674 JOB SDSF (JOB17432) P=15 CL=A OUTSERV (PENDING WTR)
IAT8674 JOB DFSMSHSM (JOB17433) P=15 CL=A OUTSERV (PENDING WTR)
IAT8674 JOB OAM (JOB17434) P=15 CL=A OUTSERV (PENDING WTR)
IAT8674 JOB OPTSO (JOB17436) P=15 CL=A OUTSERV (PENDING WTR)
IAT8595 INQUIRY ON JOB QUEUE STATUS COMPLETE, 10 JOBS DISPLAYED
```

Figure 3-64 Commands for job queue status

### Displaying job queue status

Use the **\*INQUIRY,Q** command to display:

- ▶ A list of jobs waiting for a DSP.
- ▶ The names of the spool data sets assigned to a spool partition and whether the partition is defined as the default partition, the initialization partition, or an overflow partition.
- ▶ The size of the partition and the amount of space currently available.
- ▶ The status of a partition, and users of the largest spool space.
- ▶ The status of a spool data set and the name of the spool partition the data set belongs to.
- ▶ All the defective tracks currently known to JES3.
- ▶ The amount of space available on all the JES3 spool data sets in the complex.
- ▶ A list of jobs of a particular category in the JES3 job queue.
- ▶ The amount of space remaining in the JES3 job queue.
- ▶ The status of the spool integrity function.
- ▶ A list of jobs that use the specified scheduling environment that are either waiting to be scheduled or have been scheduled for main service. Information about why the job is waiting can also be displayed.
- ▶ A list of jobs with the specified service class that are either waiting to be scheduled for main service or have been scheduled for main service. Information about why the job is waiting can also be displayed.
- ▶ Whether or not multiple batch jobs with the same name may be scheduled for the MAIN SE at the same time.

You might want to use this command to help determine if a performance problem is the result of JES3 using a high percentage of the available spool space in one or more spool partitions.

## 3.65 Jobs in operator hold

```
*I Q,H,OP
IAT8674 JOB A          (JOB09773) P=01 CL=A          HOLD=(OP) CI
IAT8674 JOB B          (JOB09774) P=01 CL=A          HOLD=(OP,N) CI
IAT8674 JOB C          (JOB09775) P=01 CL=A          HOLD=(OP,N) CI
IAT8674 JOB D          (JOB09777) P=01 CL=A          HOLD=(OP,N) CI
IAT8674 JOB E          (JOB09779) P=01 CL=A          HOLD=(OP,N) CI
IAT8698 HELD JOB SUMMARY:
IAT8696      5 JOBS IN OPERATOR HOLD
IAT8595 INQUIRY ON JOB QUEUE STATUS COMPLETE,      5 JOBS DISPLAYED
```

Dependent Job Control (DJC)  
(DJC Network)

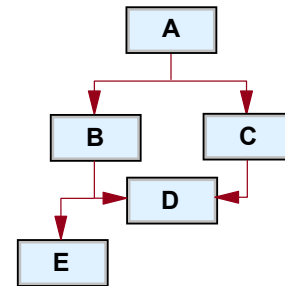


Figure 3-65 Jobs in operator hold

### Controlling jobs in hold status

You can use the command **\*I Q,H. OP** to display the status of all jobs in operator hold, with the total summary count.

You can use the option **H** to display the status of jobs held with:

- ▶ Operator hold
- ▶ DJC operator hold
- ▶ Error hold
- ▶ Spool hold
- ▶ Priority hold
- ▶ DJC net hold
- ▶ Automatic restart management hold

## 3.66 Jobs in hold

```
*I Q H
IAT8674 JOB A          (JOB09773) P=01 CL=A          HOLD=(OP) CI
IAT8674 JOB B          (JOB09774) P=01 CL=A          HOLD=(OP,N) CI
IAT8674 JOB C          (JOB09775) P=01 CL=A          HOLD=(OP,N) CI
IAT8674 JOB D          (JOB09777) P=01 CL=A          HOLD=(OP,N) CI
IAT8674 JOB E          (JOB09779) P=01 CL=A          HOLD=(OP,N) CI
IAT8696      4 JOBS IN DJC NET HOLD
IAT8595 INQUIRY ON JOB QUEUE STATUS COMPLETE,      5 JOBS DISPLAYED

*I Q,H,SUMM
IAT8698 HELD JOB SUMMARY:
IAT8696      5 JOBS IN OPERATOR HOLD
IAT8696      0 JOBS IN DJC OP HOLD
IAT8696      4 JOBS IN DJC NET HOLD
IAT8696      0 JOBS IN ERROR HOLD
IAT8696      0 JOBS IN SPOOL HOLD
IAT8696      0 JOBS IN ARM HOLD
IAT8696      0 JOBS IN PRIORITY HOLD
IAT8595 INQUIRY ON JOB QUEUE STATUS COMPLETE,      0 JOBS DISPLAYED
```

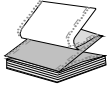
Figure 3-66 Jobs in hold


### Command for jobs in hold

The output of the **\*I Q,H** command is shown in Figure 3-66.

If you use the **SUMM** option you will display the summary of jobs for each hold type.

## 3.67 Managing output service jobs

**WRITER**  
**\*I U J = ACCTLST**  


**HOLD**  
**\*I U Q = HOLD J = 24105**  


**WHAT IS THE STATUS OF OUTPUT FOR A JOB ?**

**\*I U J = ACCTLOD REG = ?**  
**\*I U J = DB2DB1 DD = ?**  
**\*I U J = COBTST1 H = Y**  
**\*I U J = COBTST1 H = ( SYS OPER USER )**

SYSOUT CLASS (CL) DESTINATION (D) DEVICE GROUP (DG)  
FORMS (F) LINES (L) PAGES (PG) TIME ON Q (AGE)

**WHAT JOBS ARE WAITING FOR EXTERNAL WRITERS?**

**\*I U Q = HOLD W = FICHE J = ?**

Figure 3-67 Managing output service jobs

### Commands for job output

Use the **\*INQUIRY,U** or the **\*I,U** command to display job output in a JES3 system. The job output can be at various places within the system and your selection of the proper **Q=** keyword value on the **\*INQUIRY,U** command dictates what output you want.

The choices are generally the following:

- ▶ **\*INQUIRY,U,Q=BDT**  
To display SNA/NJE job output.
- ▶ **\*INQUIRY,U,Q=HOLD**  
To display job output on the HOLD queue.
- ▶ **\*INQUIRY,U,Q=WTR**  
To display job output on the WTR service queue.
- ▶ **\*INQUIRY,U**  
To display job output on the WTR service queue. (Q=WTR is the default.)

A command can have a length of up to 126 characters if the command is issued from an input device that permits that command length.



## 3.68 Start JES3 with a hot start

- ☐ 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
- ☐ Warm start requires IPL of all systems

Figure 3-68 Start JES3 with a hot start

### Starting JES3

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

JES3 does not read the initialization stream during a hot start. Instead, JES3 initializes itself by using the parameter values established during the previous cold or warm start. In addition, certain changes made by the operator after the previous restart remain in effect. For a list of these changes, see *z/OS JES3 Commands*, SA22-7540-03.

If you perform an IPL of MVS before the hot start, JES3 restarts jobs that were running on the global and are eligible for restart. If a job is not eligible for restart, JES3 processes it based on the job's failure options. For an explanation of the failure options, see the **FAILURE** parameter on the **CLASS** initialization statement. JES3 also restarts all FSSs that were running on the global and reschedules all jobs that were active in the FSS at the time of the IPL. If you do not perform an IPL of MVS before the hot start, all jobs and FSSs that were running before the hot start continue to run after the hot start. During a hot start, JES3:

- ▶ Examines the job queue to verify that it can restart each job
- ▶ Prompts the operator if it locates a job that contains invalid control information

- ▶ Removes jobs that contain invalid control information
- ▶ Records the job control information associated with the removed jobs (they are recorded in the data set defined by the //JES3SNAP DD statement)
- ▶ Continues to process valid jobs that were in the job queue before the hot start. Internal job numbering resumes with the next available job number.

Jobs and FSSs running on locals before the hot start continue to execute after the hot start. Any changes to the definition of an FSS brought about by using the **\*MODIFY** operator command before the hot start remain in effect across a hot start with or without an IPL of MVS. Overflow partition relationships, including any changes brought about by using the **\*MODIFY** operator command before the hot start, also remain in effect across a hot start.

## Warm start

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.
- ▶ Following the construction of the MVS Base Control Program.

During a warm start, JES3 reads and processes the initialization stream, saves jobs that are in the job queue, and terminates active jobs according to the job's failure options.

After a warm start, you must IPL and then restart all local processors. All FSS address spaces terminate at IPL time and are restarted by JES3 after IPL. JES3 then reschedules all jobs that were active in the FSS address space at the time of warm start. Any changes to the definition of an FSS address space brought about by using the **\*MODIFY** operator command before the warm start are lost. JES3 continues to process jobs that were in the job queue at the time of the warm start.

## 3.69 JES3 DLOG function

- ❑ DLOG started as address space on global
  - Establishes EMCS console
  - Receives messages from all systems in sysplex
  - Formats messages in JES3 format
    - Writes to SYSLOG
- ❑ Operator command to start or stop DLOG
  - \*F O,DLOG=ON/OFF

Figure 3-69 JES3 DLOG

### JES3 DLOG function

JES3 provides an option to activate a DLOG, where records are written in the JES3 format rather than the MVS format.

The DLOG message traffic is managed by MCS. On the global, JES3 activates an extended MCS console with the HARDCOPY=YES attribute to receive the hardcopy message set. The messages received by the DLOG EMCS console are in the MDB format and are converted to the JES3 DLOG format and then written using the WTL macro service to the global JES3 system's SYSLOG (alias DLOG).

JES3 initialization assigns SYSLOG unconditionally to be the MVS hardcopy medium (VARY SYSLOG, HARDCPY) on all systems. If a hardcopy log to a device is required (MLOG is requested in the initialization stream), it is written by JES3 to a JES3-managed device. In addition, if DLOG is specified in the initialization stream, and the JES3 global is restarting without IPL, JES ensures that the log is active on the global processor (WRITELOG START). During local connect processing, JES3 causes the accumulated log data on each local to be written to spool (WRITELOG).

When JES3 is installed, you have the option to use the OPERLOG as the only hardcopy media. Because JES3 can no longer unconditionally assign SYSLOG to the MVS hardcopy media (VARY SYSLOG, HARDCPY) on all systems, this processing is removed. However, if DLOG is requested in the initialization stream, JES3 global initialization continues to ensure that SYSLOG is assigned the hardcopy media and that a SYSLOG task in the console address space is active.

The initial state for DLOG is defined on the CONSTD initialization statement with the parameter DLOG=ON or DLOG=OFF, as shown in Figure 3-69 on page 227. The old CONSTD keyword HARDCOPY= is deleted.

The operator can turn the DLOG on or off with a command.

## 3.70 JES3 start procedure

```
//JES3      PROC
//IEFPROC   EXEC  PGM=IATINTK,DPRTY=(15,15)
//STEPLIB   DD  DISP=SHR,DSN=SYS1.VnRnMn.SIATLIB
//CHKPNT    DD  DISP=SHR,DSN=SYS1.JES3CKPT
//CHKPNT2   DD  DISP=SHR,DSN=SYS1.JS3CKPT2
//JES3JCT    DD  DISP=SHR,DSN=dsn
//SPOOL1     DD  DISP=SHR,DSN=dsn
.
.
//SPOOLnn    DD  DISP=SHR,DSN=dsn
//JES3OUT    DD  DISP=SHR,DSN=dsn
//JES3SNAP   DD  UNIT=AFF=JES3OUT
//JESABEND   DD  UNIT=AFF=JES3OUT
//SYSABEND   DD  UNIT=AFF=JES3OUT
//JES3DRDS   DD  DISP=SHR,DSN=dsn
//IATPLBST   DD  DISP=SHR,DSN=SYS1.PROCLIB
.
.
//IATPLBnn   .
//JES3IN     DD  DISP=SHR,DSN=SYS1.PARMLIB(JES3IN00)
```

Figure 3-70 JES3 start procedure

### How to start JES3

JES3 runs as a started task in the z/OS environment. Therefore, z/OS must be active before you can start JES3. Moreover, JES3 must be active on the global before you can start JES3 on the local mains.

The required JES3 procedure DD statements are explained as follows:

- IEFPROC** Specifies the name of the JES3 job step task, load module IATINTK.
- CHKPNT** and **CHKPNT2** Define the JES3 checkpoint data set(s). At least one of the two checkpoint data sets must be allocated and cataloged prior to JES3 operation. Each checkpoint data set must be allocated as a single extent which begins and ends on a cylinder boundary. The size of each data set should be at least 2 cylinders on a 3380, 3390, or 9345 spool volume.
- JES3JCT** Defines the JES3 job control table (JCT) data set. This data set must be allocated and cataloged 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** and **Spoolnn** Define 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 cataloged prior to JES3 operation.

|                 |                                                                                                                                                                                                                                                                                                                                       |
|-----------------|---------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| <b>JES3OUT</b>  | 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. |
| <b>IATPLBST</b> | Defines the installation's standard procedure library.                                                                                                                                                                                                                                                                                |
| <b>JES3IN</b>   | Defines the data set containing the JES3 initialization stream. This data set must be a blocked or unblocked partitioned data set. In our example, the initialization stream is read from SYS1.PARMLIB(member JES3IN00).                                                                                                              |

## 3.71 JES3 initialization stream

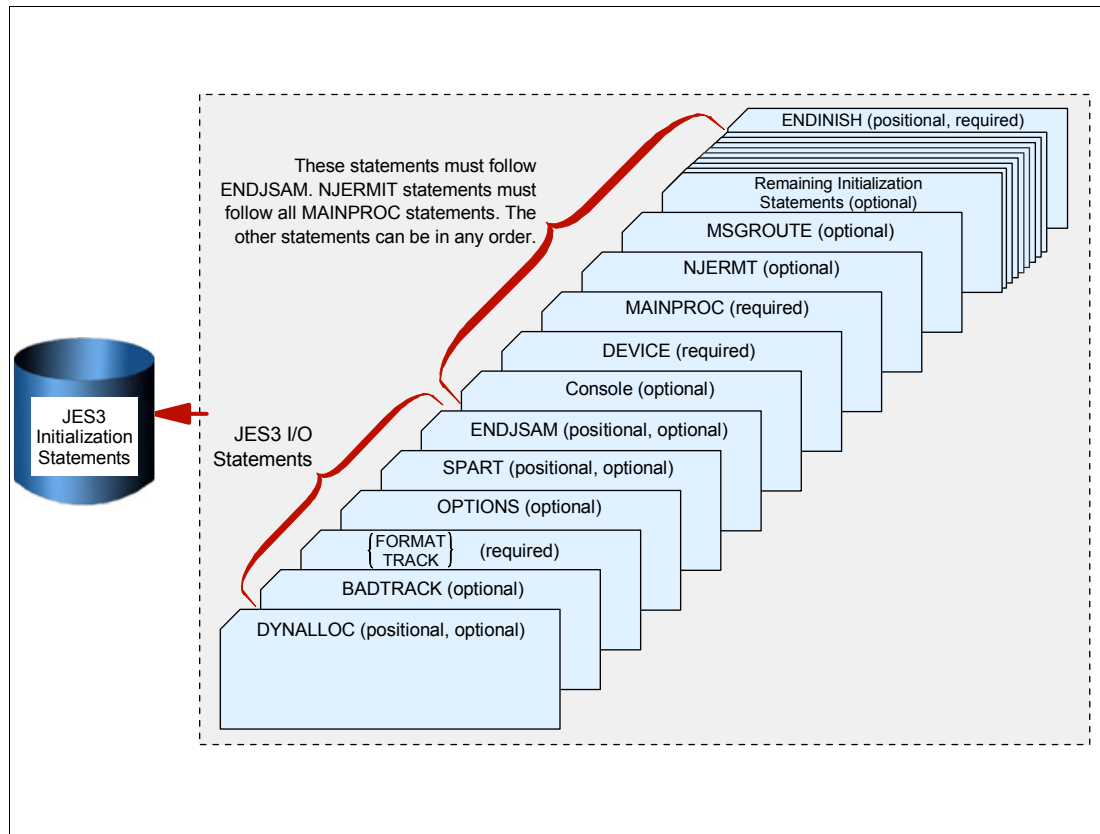


Figure 3-71 JES3 initialization stream

### JES3 initialization deck

Here are the required JES3 initialization statements:

- ▶ **DEVICE,DTYPE=SYSMAIN,JNAME=main1,JUNIT=(,main2,,~OFFIONÀ,...)**

Use this 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. You must code one of these DEVICE statements for each main on every processor in the complex, as follows:

```
DEVICE,DTYPE=SYSMAIN,JNAME=SY1,JUNIT=(,SY1,,ON,)
```

- ▶ **ENDINISH**

Use the ENDINISH statement to identify the end of the initialization statements in the initialization stream.

- ▶ **ENDJSAM**

Use the ENDJSAM initialization statement to indicate the end of the JES3 spool initialization statements.

- ▶ **FORMAT,DDNAME=ddname,SPART=partitionname {,STT=(cylnum,cylnum)} {,STTL=(cylnum,numtrkgps)}**

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 as follows:

```
FORMAT,DDNAME=SP00L1,STT=(30,31),SPART=PART1
```

The corresponding DD statement in the JES3 start procedure is:

```
//SP00L1 DD DSN=dsn,DISP=OLD,UNIT=SYSDA,VOL=SER=xxxxxx
```

- MAINPROC,NAME=main,SYSTEM=JES3,FIXPAGE=,ID=msgprefix,MDEST=,PRTPAGE=,RID=,SELECT=,SID=,SPART=partitionname,TRKGRPS=,USRPAGE=

Use the MAINPROC initialization statement to define a uniprocessor or a multiprocessor as a JES3 main. The initialization stream must include one MAINPROC statement for each main that you wish to define to JES3. Each MAINPROC statement must have a DEVICE statement that specifies DTYPE=SYSMAIN associated with it.

The NAME=main specifies the name of a JES3 main. The name should match the purpose of the main, or at least the hardware it is using. You also use this name in operator commands and in the JES3 CLASS, DEVICE, GROUP, MSGROUTE, and SETACC initialization statements to refer to the main. It must be the first or second parameter on the statement.

The SPART=partitionname specifies the spool partition that JES3 is to use for jobs that execute on the main defined by this statement. To specify the default spool partition, omit the SPART parameter.

- TRACK,DDNAME=ddname,SPART=partitionname {,STT=(cylnum,cylnum)}  
{,STTL=(cylnum,numtrkgps)}

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.

```
TRACK,DDNAME=SP00L1  
TRACK,DDNAME=SP00L2
```

Associated DD statements in the JES3 procedure:

```
//SP00L1 DD DSN=dsn,DISP=OLD,UNIT=3330,VOL=SER=xxxxxx  
//SP00L2 DD DSN=dsn,DISP=OLD,UNIT=3330,VOL=SER=MVSRW2
```

Depending on how your z/OS initial program load (IPL) procedures are set up, JES3 can be started automatically or you can start JES3 manually by entering a z/OS **START** command from the master console. The manual method allows you to determine if the IPL and system configuration conditions are acceptable before starting JES3. If the JES3 subsystem definition in the IEFSSNxx parmlib member specifies the NOSTART subparameter, JES3 must be started manually. Otherwise JES3 will start automatically.

## 3.72 JES3 startup messages

```
S JES3

IAT3040 STATUS OF JES3 PROCESSORS IN COMPLEX
IAT3040 SC50 ( ), SC49 ( ), SC43 <UP>

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

r 175,h

IAT4030 0001 SPOOL DATA SET IN USE
IAT4075 MAXIMUM NUMBER OF JOBS IS LIMITED TO 0012236 BY JCT DATA SET
SIZE
IXZ0001I CONNECTION TO JESXCF COMPONENT ESTABLISHED, 638
GROUP WTSCPLX9 MEMBER SC43
```

Figure 3-72 JES3 startup messages

### JES3 messages during a start

Figure 3-72 shows the messages that are displayed when you start JES3. The first message shows the status of JES3 processors in the complex; after that, JES3 sends a request to the operator asking him the type of start.

Here are the ways to start JES3:

- ▶ Hot start
- ▶ Hot start with analysis
- ▶ Hot start with refresh
- ▶ Hot start with refresh and analysis
- ▶ Warm start
- ▶ Warm start with analysis
- ▶ Warm start to replace a spool data set
- ▶ Warm start with analysis to replace a spool data set
- ▶ Cold start
- ▶ Local start (used to start local mains only)

The type of start you select depends on why you need to start or restart JES3. In our example the operator chose hot start (h). During hot start JES3 displays information about the spool configuration as well as other information, as shown in Figure 3-72.



## 3.73 JES3 startup

```
IXZ0001I CONNECTION TO JESXCF COMPONENT ESTABLISHED, 697  
GROUP WTSCPLX9 MEMBER JES3DLOG  
IAT7114 DLOG INITIALIZATION SUCCESSFUL
```

```
*IAT3100 JES3 OS2.10.0 SYSTEM HOTSTART ON 2002.013 AS SC43
```

### Start scheduling jobs

❑ Job segment scheduler (JSS) - issue following command

```
*S JSS
```

Figure 3-73 JES3 startup

### JES3 start up processing

After a successful system hot start, the messages in Figure 3-73 are displayed. To start scheduling jobs the operator has to issue the following command.

```
*S JSS
```

Before starting job scheduling, you can use JES3 commands to cancel jobs, change the status of jobs, and change the status of devices. During a hot start with analysis, you can release jobs in spool hold status after reinstating a spool data set that contains data for the jobs, and you can vary devices online or offline. You can make adjustments for any system data that might have been lost during the restart.

You can also make any changes to the system that were made before a hot start or a warm start, but did not remain in effect after the restart. When you are satisfied that the system is ready to begin processing, enter a **\*START,JSS** command to start job scheduling.

### 3.74 JES3 messages following \*S JSS

```
IAT6394 MONITOR FUNCTION ACTIVE
IAT7131 NJECONS NOW ACTIVE
IAT9225 NJERDR IS ACTIVE
IAT9225 NJERDR IS ACTIVE
IAT5919 100 INITIAL VERIFY RESPONSES RECEIVED FROM SC43
. . . . .

IAT2006 PREMATURE JOB TERM - JOB ASCHINT (JOB19360) - CANCEL
- SC43
IAT7450 JOB INITJES3 (INT19367) PURGED

IAT2645 ***** SC43 CONNECT COMPLETE *****
S INIT.A IATMSMS
S INIT.A IATMSMS
S INIT.A IATMSMS
S INIT.A IATMSMS
S JES9CI.CI9,,(JES3,8,IATINFC)
```

Figure 3-74 JES3 messages following \*S JSS

#### JES3 connect processing

The messages shown in Figure 3-74 are displayed by JES3 to indicate which functions are active and other actions that JES3 took during job scheduling initialization.

It also shows that the system is connected to the complex and which initiators are available (started) for job processing.

## 3.75 JES3 start types

|     |   |                                                         |
|-----|---|---------------------------------------------------------|
| H   | - | Hot Start                                               |
| HA  | - | Hot Start with Analysis                                 |
| HR  | - | Hot Start with Refresh                                  |
| HAR | - | Hot Start with Refresh and Analysis                     |
| W   | - | Warm Start                                              |
| WA  | - | Warm Start with Analysis                                |
| WR  | - | Warm Start to Replace a Spool Data Set                  |
| WAR | - | Warm Start with Analysis to Replace a<br>Spool Data Set |
| C   | - | Cold Start                                              |
| L   | - | Local Start                                             |

Figure 3-75 JES3 start types

### JES3 start types

The types of starts and restarts for the JES3 global processor are shown in Figure 3-75. You must use a cold start when starting JES3 for the first time. For subsequent starts (restarts), you can use any one of the start types, depending on the circumstances at the time. Thus, the other types of starts are actually restarts.

JES3 initialization statements are read as part of cold start, warm start, and hot start with refresh processing. If JES3 detects any error in the initialization statements, it prints an appropriate diagnostic message on the console or in the JES3OUT data set. JES3 ends processing if it cannot recover from the error. In this case, you must notify the system programmer to ensure that the error is corrected before restarting JES3.

JES3 configuration and processing options are specified with the initialization statements. Because many options affect the overall performance of the system, initialization statements must be provided by your system programmer. After starting JES3 on a local main, you can use the **ROUTE** command to start JES3 on all the local processors once JES3 global initialization is complete. For example:

```
ROUTE *OTHER,S JES3
```

### JES3 local start

Use a local start to restart JES3 on a local main after a normal shutdown on the local, after JES3 ends due to a failure in either the local JES3 address space or z/OS, or after partitioning a multiprocessor. You must also start each local main after you perform a cold

start or any type of warm start on the global, or after you use a hot start to remove or reinstate a spool data set on the global processor. You can perform a local start any time the global is active. Local start processing uses the initialization stream placed on the spool data sets during global main initialization.

**Note:** Do not start a main if another main having the same name is active in the complex.

Before you begin, be sure that all spool data sets available to the global are also available to this local.

## Start JES3

If you are performing an IPL of the local main and the MSTRJCL contains the automatic **START** command, the command is displayed on your console immediately after the z/OS master scheduler is initialized. You can enter the z/OS command:

**START JES3**

The first message from JES3 is:

```
IAT3040      STATUS OF JES3 PROCESSORS IN COMPLEX
              main(status) (,main(status)...)
```

A series of entries on the following lines of the message contain the name of each main followed by a code for that main's status as recorded in the complex status record (CSR). During a start for the system, this entry (the status of the global) is:

```
main< >
```

On subsequent starts, this entry could be:

```
main<IN>
```

This indicates that the previous start ended abnormally during initialization; this main is still recorded in the CSR as being in initialization as follows:

```
main<UP>
```

This main is recorded in the CSR as currently being active (up). Consult with the system programmer before continuing with a cold start.

Next, JES3 issues the following message:

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

To continue the start, enter the appropriate letters shown in the message, as follows:

```
R nn,C
```

To cancel the start, enter:

```
R nn,CANCEL
```

**Note:** Because cold start processing involves reinitializing the spool and losing all jobs, JES3 issues the following message requiring you to confirm your request for a cold start before JES3 takes irreversible action:

```
nn IAT3033      CONFIRM JES3 COLD START REQUEST (U) OR CANCEL
```

To continue the cold start, enter:

```
R nn,U
```

To cancel the cold start, enter:

```
R nn,CANCEL
```

Any other reply restarts the sequence, beginning with message IAT3011.

## Start JSS

After JES3 initialization has finished, you should start the job segment scheduler (JSS). You start job scheduling after JES3 issues message IAT3100 notifying you that initialization processing is complete on the global:

```
IAT3100  JES3 xxxxx SYSTEM type START ON yyyy.ddd AS main
```

where xxxxx is the release level of JES3; type is COLD, WARM, or HOT; yyyy.ddd is the Julian date; and main is the JES3 name of the main.

Before starting job scheduling, you can use JES3 commands to cancel jobs, change the status of jobs, and change the status of devices. During a hot start with analysis, you can release jobs in spool hold status after reinstating a spool data set that contains data for the jobs, and you can vary devices online or offline. You can make adjustments for any system data that might have been lost during the restart. You can also make any changes to the system that were made before a hot start or a warm start but did not remain in effect after the restart.

When you are satisfied that the system is ready to begin processing, enter the following command to start job scheduling:

```
*START,JSS
```

After you enter the \*START,JSS command, ensure that the global is varied online to JES3. If it is not, enter the **\*MODIFY,V,main,ON** command to vary the processor online, ensuring that the subsystem interface, the z/OS system commands, and the system log are initialized. JES3 then issues the following message:

```
IAT2645      ***** main CONNECT COMPLETE *****
```

If you do not want the global to run jobs, you can now vary the main offline to JES3 scheduling with the **\*MODIFY,V,main,OFF** command. At this point, you can resubmit any jobs that were canceled or whose processing was altered as a result of the restart.

## How to stop JES3

Stopping JES3 can be done on either of the global and local systems, as follows:

### *Stopping local processors*

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

1. Enter a **\*F, V,main,OFF** command for the local main to prevent JES3 from scheduling any further jobs on the processor.

2. Enter the **\*RETURN** command with the proper password to end JES3 after all jobs on the local main have completed processing.
3. Enter the **HALT EOD** command on the local main to ensure that important statistics and data records in storage are not permanently lost.

Your installation may not want to wait for all jobs to complete normally, for example:

- ▶ Jobs will not end due to 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.

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

### ***Stopping the global processor***

Before stopping the global, you should stop the local mains as described previously. 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.

Once all system activity has completed, enter the **\*RETURN** command to end JES3.

**\*RETURN,password,FSS=fssname** or

**\*RETURN,password,FSS=ALL** or

**\*RETURN,password,FSS=NONE**

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.

## 3.76 TME 10 OPC

- ❑ Understanding TME 10 OPC
- ❑ What is TME 10 OPC
- ❑ What is an OPC Tracker
- ❑ What is an OPC Controller
- ❑ Workload Management Tool
- ❑ Users of TME 10 OPC

Figure 3-76 TME® 10™ OPC

### Operations planning and control

The TME 10 Operations Planning and Control (TME 10 OPC) Licensed Program is the IBM foundation for enterprise workload management. TME 10 OPC provides a comprehensive set of services for managing and automating the workload. Whether you manage a single-image z/OS system or multi-vendor networks and systems from a single point of control, TME 10 OPC helps you manage and automate the production workload.

TME 10 OPC builds operating plans from your descriptions of the production workload.

TME 10 OPC consists of a base product, the tracker, and a number of features. All the systems in your complex require the base product. The *tracker* is the link between the system that it runs on and the TME 10 OPC controller.

One z/OS system in your complex is designated the controlling system and runs the controller feature. From this system, you can automatically plan, control, and monitor your entire production workload. Only one controller feature is required, even when you want to start standby controllers on other z/OS systems in a sysplex.

- ▶ **Tivoli® OPC** makes it possible for the scheduling manager to maintain current and future production processing across your enterprise. Tivoli OPC benefits the scheduling manager by:
  - Automatically schedule all production workload activities.

- Automatically resolve the complexity of production workload dependencies and drive the work in the most efficient way.
- Supporting the simulation of future workloads on the system. The scheduler can evaluate, in advance, the effect of changes in production workload volumes or processing resources.
- Giving a real-time view of the status of work as it flows through the system so that the scheduler can quickly:
  - Respond to customer queries about the status of their work.
  - Identify problems in the workload processing.
- Providing facilities for manual intervention.
- Managing many workload problems automatically. The TME 10 OPC production-workload-restart facilities, hot standby, automatic recovery of jobs and started tasks, and data set cleanup provide the scheduler with comprehensive error- and disaster-management facilities.
- Providing a log of changes to the TME 10 OPC production workload data through the audit-trail facility. This assists the scheduler in resolving problems caused by user errors.
- Managing unplannable work.
- The **operations manager** uses the reporting, planning, and control functions to:
  - Improve the efficiency of the operation.
  - Improve control of service levels and quality.
  - Set service level agreements for end-user applications and for services provided.
  - Improve relationships with end-user departments.
  - Increase the return on your IT investment.
  - Develop staff potential.
- The **Shift Supervisor** uses TME 10 OPC, especially in multisystem complexes, where local and remote systems are controlled from a central site. Also it can help the shift supervisor to:
  - Monitor and control the flow of production work through multisystem complexes.
  - Control the use of mountable devices.
  - Separate information about work status from system and other information.
  - Provide end users with status information directly.
  - Manage the workload if a system failure occurs.
  - Make changes to the current plan in response to unplanned events, such as equipment failures, personnel absences, and rush jobs.
- The **application programmer**. The TME 10 OPC user-authority checking enables application development groups to use all the planning and control functions of TME 10 OPC in parallel with—but in isolation from—production systems and services. For the application programmer, it can be a valuable tool for application development staff, enabling them to:
  - Package new applications for regular running
  - Test new JCL in final packaged form
  - Test new applications and changes to existing ones
  - Restart or rerun unsuccessful jobs.



- ▶ The **console operators** can be freed from these time-consuming tasks:
  - Starting and stopping started tasks.
  - Preparing JCL before job submission.
  - Submitting jobs.
  - Verifying the sequence of work.
  - Reporting job status.
  - Performing data set cleanup in recovery and rerun situations.
  - Responding to workload failure.
  - Preparing the JCL for step-level restarts.
- ▶ The **workstation operators** gets help in:
  - Providing complete and timely status information.
  - Providing up-to-date ready lists that prioritize the work flow.
  - Providing online assistance in operator instructions.
- ▶ The **end users** and **help desks** can be informed about the status of workload processing. They can use the ISPF dialogs or the Workload Monitor/2 to check the status of the processing of their applications themselves, from a personal workstation. End users can make queries using the Workload Monitor/2 without having to be familiar with TME 10 OPC, ISPF, or TSO, and without having to be logged on to a local system. The Workload Monitor/2 also provides alerts that can automatically advise end users when their work has completed or ended in error. The help desk can use the Workload Monitor/2 in the same way to answer queries from end users about the progress of their workload processing.

### 3.77 TME 10 OPC platforms

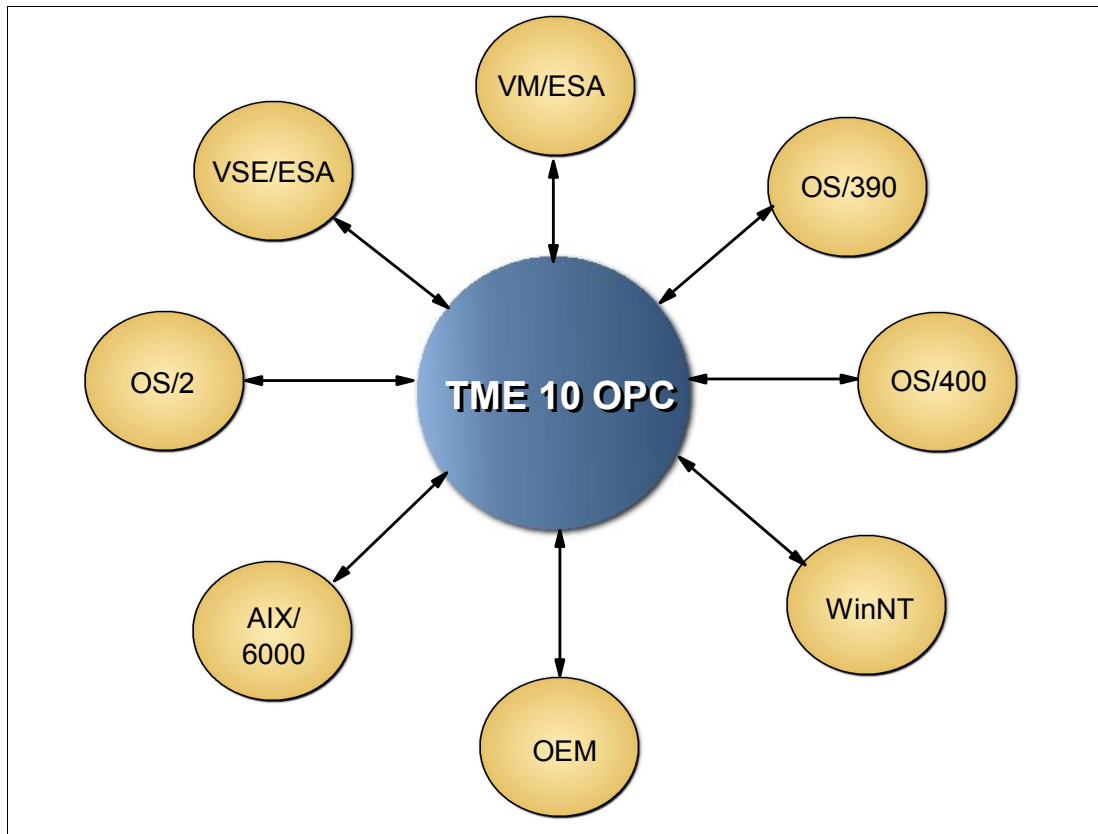


Figure 3-77 TME 10 OPC platforms

#### TME 10 platforms

TME 10 OPC is a state-of-the-art production workload manager, designed to help you meet your present and future data processing challenges. Its scope encompasses your entire enterprise information system, including heterogeneous environments.

Pressures in today's DP environment are making it increasingly difficult to maintain the same level of services to customers. Many installations find that their batch window is shrinking. More critical jobs must be finished before the morning online work begins. Conversely, requirements for the integrated availability of online services during the traditional batch window put pressure on the resources available for processing the production workload. More and more, 7/24 (7 days a week, 24 hours a day) is not only a DP objective but a requirement.

Users and owners of DP services are also making more use of batch services than ever before. The batch workload tends to increase each year at a rate slightly below the increase in the online workload. Combine this with the increase in data usage by batch jobs, and the end result is a significant increase in the volume of work.

Furthermore, there is a shortage of people with the required skills to operate and manage increasingly complex DP environments. The complex interrelationships between production activities—between manual and machine tasks—have become unmanageable without a workload management tool.

TME 10 OPC provides leading-edge solutions to problems in production workload management. It can automate, plan, and control the processing of your enterprise's entire

production workload, not just the batch subset. TME 10 OPC functions as an “automatic driver” for your production workload, to maximize the throughput of work and optimize your resources, but it also allows you to intervene manually as required.

When TME 10 OPC interfaces with other system management products, it forms part of an integrated automation and systems management platform for your DP operation.

TME 10 OPC forms operating plans based on user descriptions of the operations department and its production workload. These plans provide the basis for your service level agreements and give you a picture of the production workload at any point in time. You can simulate the effects of changes in your production workload and in resource availability by generating trial plans.

Good planning is the cornerstone of any successful management technique. Effective planning also helps you maximize return on your investments in information technology.

TME 10 OPC automates, monitors, and controls the flow of work through your enterprise's entire DP operation, on both local and remote systems. From a single point of control, TME 10 OPC analyzes the status of the production work and drives the processing of the workload according to installation business policies. It supports a multiple end-user environment, enabling distributed processing and control across sites and departments within your enterprise.

## 3.78 z/OS OPC configuration

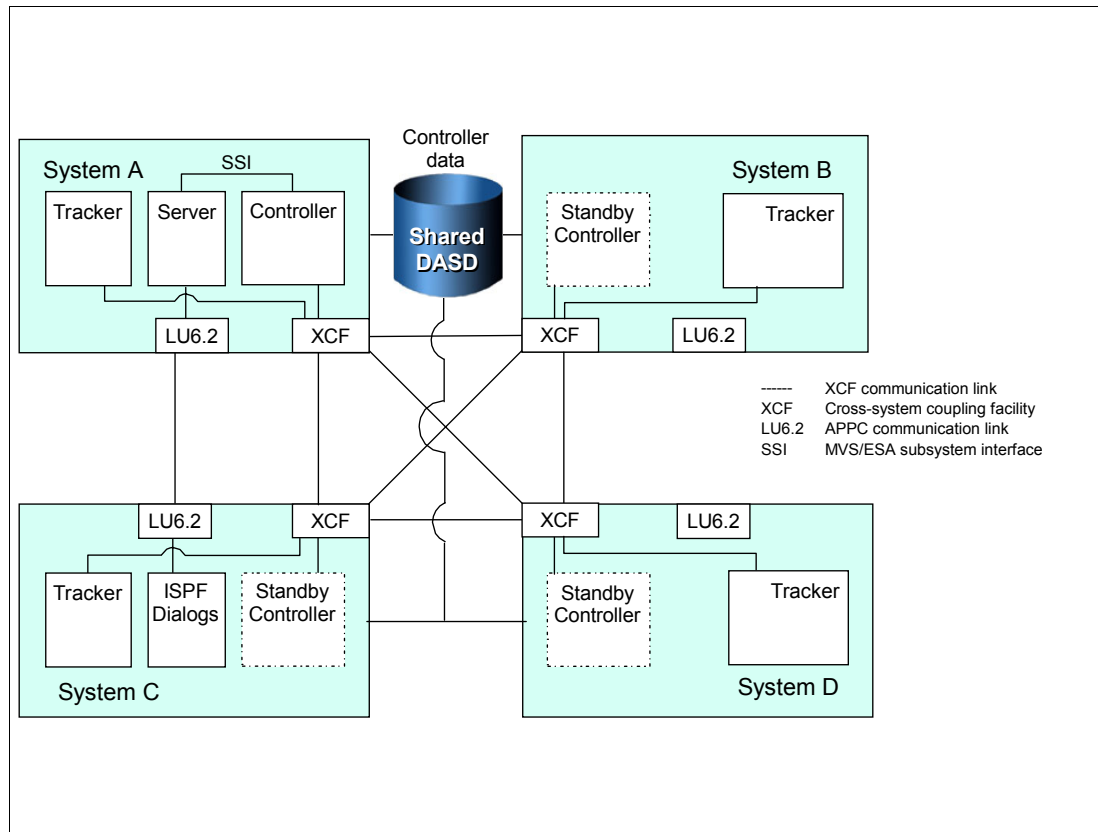


Figure 3-78 z/OS OPC configuration

### What is a TME 10 OPC Tracker

A tracker is required for every controlled z/OS system in a TME 10 OPC configuration. This includes, for example, local controlled systems within shared DASD or sysplex configurations.

The tracker runs as a z/OS subsystem and interfaces with the operating system (through JES2 or JES3, and SMF), using the subsystem interface and the operating system exits. The tracker monitors and logs the status of work, and passes the status information to the controller via shared DASD, XCF, or ACF/VTAM.

You can exploit the z/OS cross-system Coupling Facility (XCF) to connect your local z/OS systems. Rather than being passed to the controlling system via shared DASD, work status information is passed directly via XCF connections. XCF lets you exploit all of TME 10 OPC's production-workload-restart facilities and its hot standby function.

The tracker on a remote z/OS system passes status information about the production work in progress to the controller on the TME 10 OPC controlling system. All communication between TME 10 OPC subsystems on controlling and remote systems is done via ACF/VTAM.

TME 10 OPC lets you link remote systems using ACF/VTAM networks. Remote systems are frequently used locally "on premises" to reduce the complexity of the DP installation.

Every controlled AS/400® system needs the Tracker Agent for OS/400® feature installed. When this feature is used, status information and job logs for the production workload are passed back to the controlling z/OS system using APPC services. The TME 10 OPC

controller submits jobs and commands to one system in an AS/400 cluster; from there the work is directed to the target AS/400 system.

Controlled UNIX systems require the OPC Tracker Agent feature for the appropriate implementation of UNIX, including the z/OS Open Edition environment. When one of these features is used, status information and job logs for the production workload are passed back to the controlling z/OS system using TCP/IP services. The TME 10 OPC controller submits jobs and commands directly to the target UNIX system.

If you use Load Leveler to manage workload distribution in your heterogeneous workstation environment, the OPC Tracker Agents for AIX® and the other UNIX implementations can integrate with Load Leveler 1.2.

To control the workgroup workload on OS/2® workstations and servers, you need the Tracker Agent for OS/2 feature. When this feature is used, status information and job logs for the production workload are passed back to the controlling z/OS system using TCP/IP services. The TME 10 OPC controller submits jobs and commands directly to the target workstation system. To control the workgroup workload on Windows® NT workstations and servers, you need the Tracker Agent for Windows NT® feature. When this feature is used, status information and job logs for the production workload are passed back to the controlling z/OS system using TCP/IP services. The TME 10 OPC controller submits jobs and commands directly to the target workstation system.

Remote systems controlled by other operating environments can also communicate with the TME 10 OPC controller using supplied programs. Although not a full-function tracker agent today, these routines provide you with the opportunity to centralize control and automate the workload on these operating environments. The supplied sample programs demonstrate communication using a variety of methods. The programs can be tailored for a specific environment.

TME 10 OPC Tracker Agents are functionally equivalent to the base z/OS tracker, with the exception that automatic data set cleanup and the job-completion checker (JCC) functions are not provided for non-z/OS platforms. This means you can use TME 10 OPC functions like automatic recovery and automatic job tailoring for your non-z/OS workload. Many installations run business applications on a variety of UNIX platforms. TME 10 OPC provides tracker agents that can manage your workload in several operating system environments. At the time of this writing, tracker agents for OS/400, AIX/6000, HP-UX, SunOS™, Sun™ Solaris™, Pyramid MIPS ABI, Digital OpenVMS, Digital UNIX, AIX, and System/390® Open Edition operating systems are available, as well as the tracker agents for OS/2 and Windows NT platforms, and the base tracker for z/OS.

## **What is a TME 10 OPC controller**

The controller is the focal point of your TME 10 OPC configuration. It uses the information in the database to determine which jobs to run, when they should run, and where they should run. It contains the controlling functions, the dialogs, and TME 10 OPC's own batch programs. Only one TME 10 OPC controller is required to control the entire TME 10 OPC installation, including local and remote systems.


The system that the controller is started on is called the TME 10 OPC controlling system. TME 10 OPC systems that communicate with the controlling system are called controlled systems. You need to install in your z/OS system at least one controller for your production systems. It can control the entire TME 10 OPC configuration, both local and remote. You can use the TME 10 OPC controller to provide a single, consistent control point for submitting and tracking the workload on any operating environment.

TME 10 OPC provides open interfaces to let you integrate the planning, scheduling, and control of work units such as online transactions, file transfers, or batch processing in any operating environment that can communicate with z/OS.

### **TME 10 OPC as a workload management tool**

TME 10 OPC allows you to:

- ▶ Manage your workload on a variety of platforms from a single point of control.
- ▶ Run jobs on the right day at the right time.
- ▶ Start jobs in the correct order.
- ▶ Resolve complex dependencies between jobs.
- ▶ Take into account business days and holidays across divisions, states, and countries.
- ▶ Provide plans for the future workload to help you manage peak processing (year-end work, for example).
- ▶ Optimize hardware resources by allocating work to specific machines.
- ▶ Initiate automatic recovery actions in the event of hardware or software failure.
- ▶ Maintain logs of work that has run and that is available for viewing or post-processing.



## LPA, LNKLST, and authorized libraries

In order to maximize the retrieving modules performance, the MVS operating system, and now z/OS, has been designed to maintain in memory those modules that are needed for fast response to the operating system as well as for the critical applications. Link pack area (LPA), LNKLST, and authorized libraries described here are the cornerstone of the fetching process. The role of VLF and LLA components are described at the appropriate level.

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

You should consider these factors when deciding where to place individual modules or libraries containing multiple modules in the system-wide search order for modules:

- ▶ The search order the system uses for modules
- ▶ How placement affects virtual storage boundaries
- ▶ How placement affects system performance
- ▶ How placement affects application performance

## 4.1 Link pack area (LPA)

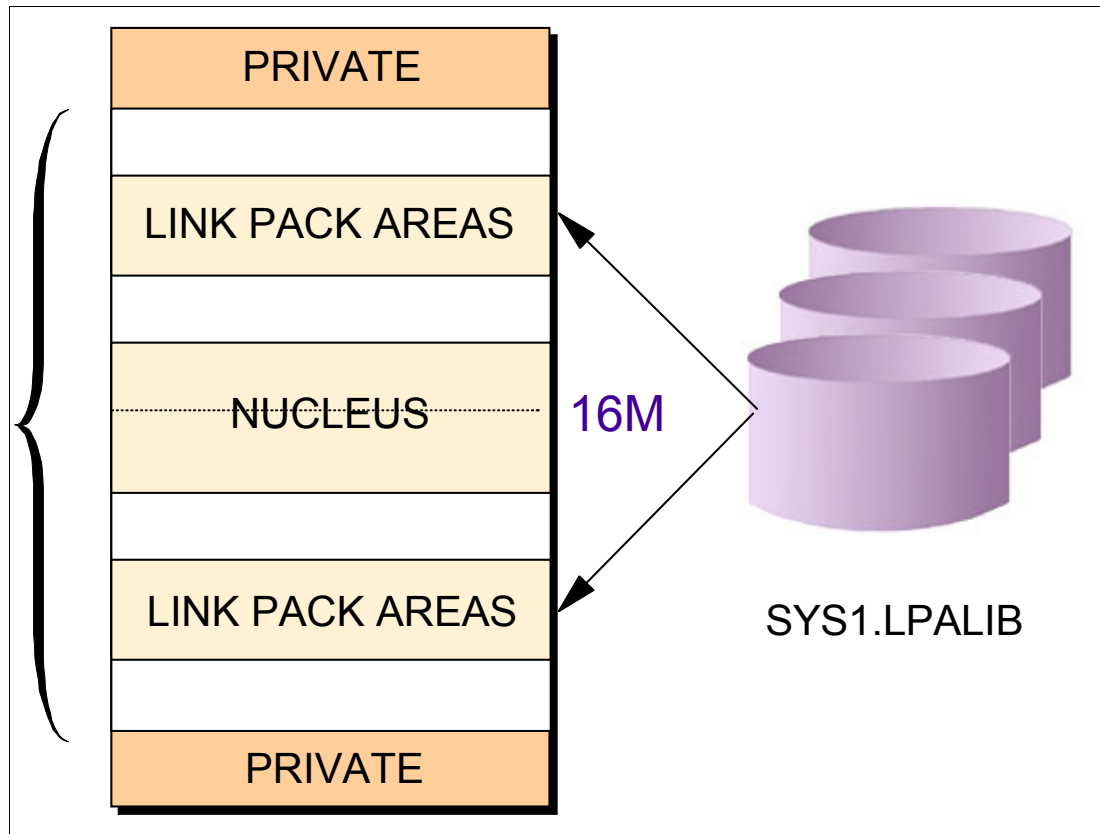


Figure 4-1 Link pack area (LPA)

### The link pack area (LPA)

Link pack area (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.

### FLPA and central storage

Modules placed anywhere in LPA are always in virtual storage, and modules placed in FLPA are also always in central storage. Whether modules in LPA, but outside FLPA, 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 will be in central storage at any given time.

LPA pages are only stolen, and never paged out, because there are copies of all LPA pages in the LPA page data set. But the results of paging out and page stealing are usually the same; unless stolen pages are reclaimed before being used for something else, they will not be in central storage when the module they contain is called.

### Modules referenced in LPA

LPA modules must be referenced very often to prevent their pages from being stolen. When a page in LPA (other than in FLPA) is not continually referenced by multiple address spaces, it



tends to be stolen. One reason these pages might be stolen is that address spaces often get swapped out (without the PLPA pages to which they refer), and a swapped-out address space cannot refer to a page in LPA.

When all the pages containing an LPA module (or its first page) are not in central storage when the module is called, the module will begin to run only after its first page has been brought into central storage.

### **Modules in CSA**

Modules can also be loaded into CSA by authorized programs. When modules are loaded into CSA and shared by multiple address spaces, the performance considerations are similar to those for modules placed in LPA. (However, unlike LPA pages, CSA pages must be paged out when the system reclaims them.)

## 4.2 LPA subareas

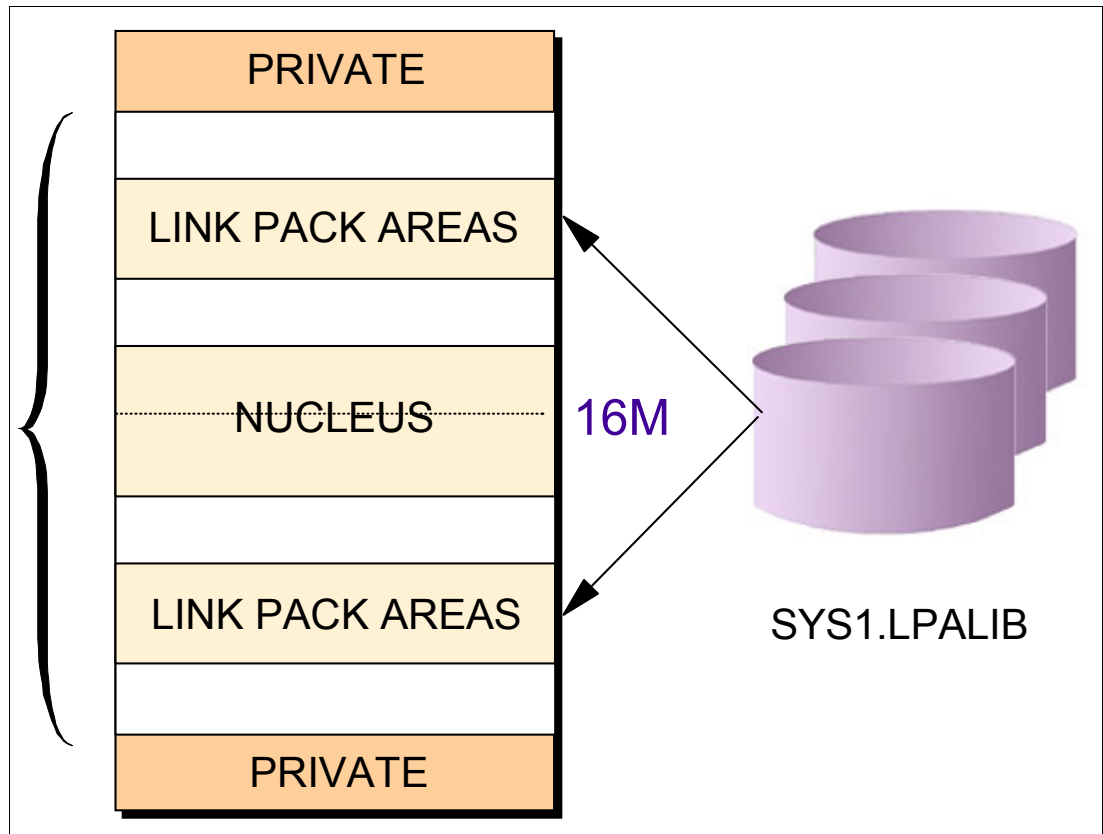


Figure 4-2 LPA subareas

### Subareas of LPA

The link pack area (LPA) 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 the LPA has a counterpart in the extended common area (that is above the 16 megabyte line) as follows:

**FLPA** 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).

An installation can elect to have some modules that are normally loaded in the pageable link pack area (PLPA) loaded into the fixed link pack area (FLPA). This area should be used only for modules that significantly increase performance when they are fixed rather than pageable. Modules placed in the FLPA must be reentrant and refreshable.

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 the PLPA, the system's paging algorithms will tend to keep that module in central storage. The best

candidates for the FLPA are modules that are infrequently used but are needed for fast response to some terminal-oriented action.

**PLPA** During initialization, both primary and secondary (or duplexed) PLPAs are established. The PLPA is established initially from the LPALST concatenation.

On the PLPA page data set, ASM maintains records that have pointers to the PLPA and extended PLPA pages. During quick start and 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 and segment tables, and uses them for the current IPL.

If CLPA is specified at the operator's console, 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 on the PLPA page data set.

The secondary PLPA is saved on the optional duplex paging data set for recovery purposes. If any uncorrectable I/O error occurs on a page-in request from the PLPA, the duplexed PLPA is used.

**MLPA** This area may be used to contain reenterable routines from APF-authorized libraries that are to be part of the pageable extension to the link pack area 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 for each IPL (including quick start and warm start IPLs). The MLPA is allocated just below the FLPA (or the PLPA, if there is no FLPA); the extended MLPA is allocated above the extended FLPA (or the extended PLPA if there is no extended FLPA). When the system searches for a routine, the MLPA is searched before the PLPA.

**Note:** Loading a large number of modules in the MLPA can increase fetch time for modules that are not loaded in the LPA. This could affect system performance.

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.

## 4.3 Pageable link pack area

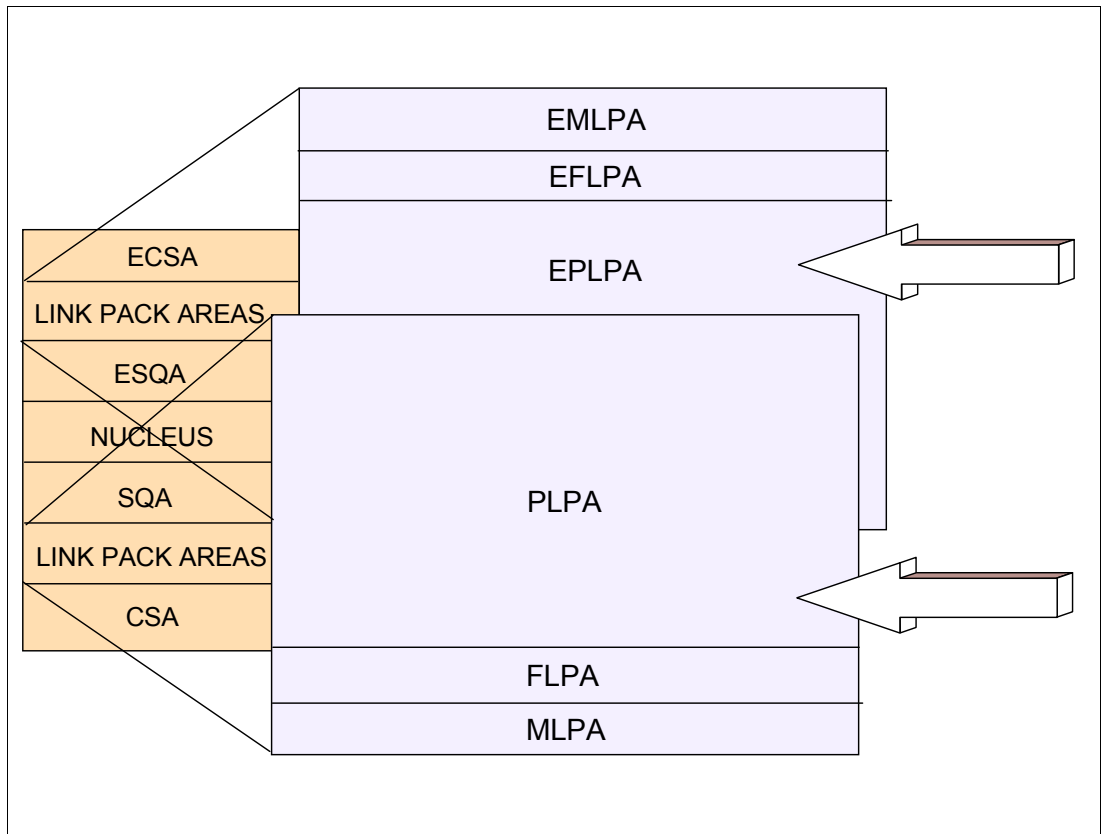


Figure 4-3 Pageable link pack area

### Pageable link pack area (PLPA/extended PLPA)

The PLPA is an area of common storage which is loaded at IPL time (when you do a cold start, specify CLPA in your IPL).

This area contains SVC routines, access methods, and other read-only system programs, along with any read-only reenterable 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.

### 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 reentrant and executable.

## 4.4 LPA parmlib definitions

### ❑ LPALSTxx parmlib member specification

```
LPA= {aa  
      { (aa,bb,...[,L]) }  
(YEASYSxx parameter)
```

### ❑ Operator can override parmlib specification

```
IEA101A SPECIFY SYSTEM PARAMETERS FOR OS/390 02.07.00 HBB6607  
R 00,LPA=03  
IEE600I REPLY TO 00 IS;LPA=03
```

Figure 4-4 LPA parmlib definitions

### Specifying LPA parameters in parmlib

The two characters (A through Z, 0 through 9, @, #, or \$) represented by aa (or bb and so forth) in Figure 4-4, 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.

The LPA parameter is only effective 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.

An example of overriding the value of the LPA parameter in the IEASYSxx during system initialization is as follows:

```
IEA101A SPECIFY SYSTEM PARAMETERS FOR z/OS 02.05.00 HBB6605  
r 00,LPA=03  
IEE600I REPLY TO 00 IS;LPA=03
```

LPALST03 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 reenterable 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.

**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 cataloged 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 to 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.

## 4.5 Coding a LPALSTxx parmlib member

```
File Edit Edit_Settings Menu Utilities Compilers Test Help
-----
EDIT          SYS1.PARMLIB(LPALST6C) - 01.03          Columns 00001
Command ==>   Scroll ==>
***** Top of Data *****
000010 EJES.SEJELPA,
000100 SYS1.SERBLPA,
000200 SYS1.LPALIB,
000300 ISF.SISFLPA,
000500 ING.SINGMOD3,
000600 NETVIEW.SCNMLPA1,
000700 SDF2.V1R4M0.SDGILPA,
000800 REXX.SEAGLPA,
001000 SYS1.SIATLPA,
001100 EOY.SEOYLPA,
001200 SYS1.SBDTLPA,
001300 CEE.SCEELPA,
001400 ISP.SISPLPA,
001500 SYS1.ISAMLPA,
001600 SYS1.SORTLPA,
001700 SYS1.SICELPA,
001800 EUV.SEUVLPA,
001900 TCPIP.SEZALPA,
```

Figure 4-5 LPALSTxx parmlib member example

### LPALSTxx parmlib member

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.

## 4.6 Fixed link pack area

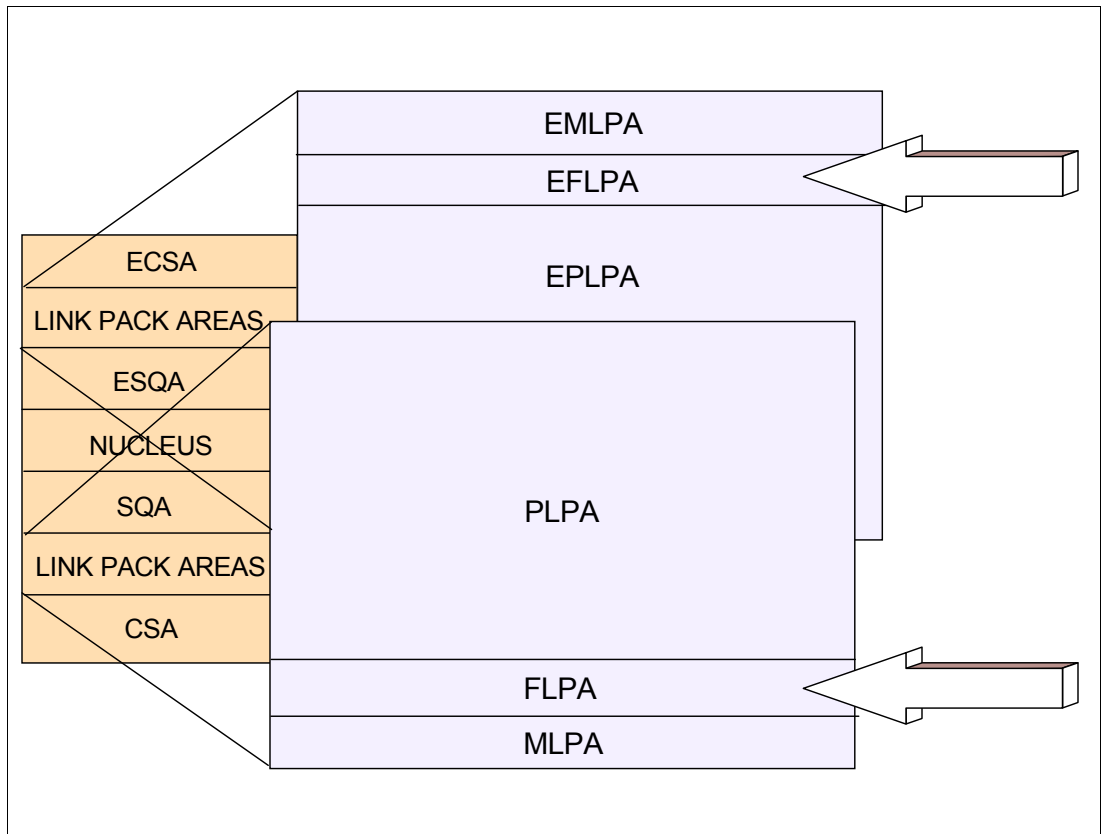


Figure 4-6 Fixed link pack area

### Fixed link pack area (FLPA or extended FLPA)

The FLPA is loaded at IPL time, with those modules listed in the active IEAFIXxx member of SYS1.PARMLIB, and also contains modules which must be kept in fixed storage frames (that is, they cannot be paged out).

This area should be used only for modules that significantly increase performance when they are fixed rather than pageable. Modules placed in the FLPA must be reentrant and refreshable. The best candidate for the FLPA are modules that are infrequently used but are needed for fast response.

### Contents of the FLPA or extended FLPA

Modules from the LPALST concatenation, the LNKST 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.

Modules specified in IEAFIXxx are placed in the FLPA or extended FLPA, depending on the RMODE of the modules. Modules with RMODE of 24 are placed in the FLPA, while those with an RMODE of ANY are placed in the extended FLPA.



## 4.7 Coding the IEAFIXxx member

### Specify - (INCLUDE - LIBRARY - MODULES)

#### ➤ Consider module order

- Saves search time

```
Menu Utilities Compilers Help
-----
BROWSE SYS1.PARMLIB(IEAFIX01) - 01.01          Line 00000000 Col 001 080
Command ==>                                     Scroll ==> PAGE
***** Top of Data *****
INCLUDE LIBRARY(SYS1.LPALIB)
      MODULES(IEAVAR00,      /* 7K RCT INIT/TERM      */
              IEAVAR06,      /* RCT INIT/TERM ALIAS  */
              IGC0001G,      /* 456 RESTORE(SVC17)   */
              ICHRFC00,      /* RACF IMS/CICS        */
              ICHRF00,       /* RACF IMS/CICS        */
              ICHSFR00)      /* RACF IMS/CICS        */
INCLUDE LIBRARY(SYS1.SVCLIB) MODULES(IGC000RC IGC09302 IGC09303)
***** Bottom of Data *****
```

Figure 4-7 The IEAFIXxx parmlib member

### Specifying modules in the FLPA or extended FLPA

Use the IEAFIXxx member in the SYS1.PARMLIB to specify the names of the modules that are to be fixed in central storage for the duration of an IPL.

**Note:** These libraries should be cataloged in the system master catalog.

Modules specified in IEAFIXxx are loaded and fixed in the order which they are specified in the member. To keep search time within reasonable limits, do not allow the FLPA to become excessively large.

### Coding IEAFIXxx in SYS1.PARMLIB

The statements or parameters used in IEAFIXxx are:

- |                |                                                                                                            |
|----------------|------------------------------------------------------------------------------------------------------------|
| <b>INCLUDE</b> | Specifies modules to be loaded as temporary extensions to the existing pageable link pack area (PLPA).     |
| <b>LIBRARY</b> | Specifies a qualified data set name. The specified library must be cataloged in the system master catalog. |
| <b>MODULES</b> | Specifies a list of 1 to 8 character module names.                                                         |

### Rules for specification of IEAFIXxx

- Each statement must begin with the INCLUDE keyword.

- ▶ The library name follows the LIBRARY keyword and is enclosed in parentheses.
- ▶ The list of modules follows the MODULES keyword and is enclosed in parentheses. Any number of modules' names can be specified.
- ▶ The statement is assumed to be all information from one INCLUDE keyword to the next INCLUDE keyword or until end-of-file.
- ▶ Use all columns except 72 through 80.

The visual shows an example of IEAFIXxx

### **Advantages of FLPA or extended FLPA**

Because fixed modules are not paged, you save I/O time and paging overhead by placing moderately used modules into the FLPA. This can shorten the LPALST concatenation. When a module is requested, the program manager searches the list of fixed routines before it examines the pageable LPA directory.

**Note:** The price for this performance improvement 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.

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

## 4.8 Specifying the IEAFIXxx member

- ❑ Operator response to IEA101A - system parameters
  - R 00,FIX=aa - (overrides IEASYSxx FIX=)
- ❑ Display contents of IEAFIXxx on console
  - Use L option
- ❑ Override the page protection default
  - Use NOPROT option

```
FIX= {aa                                     }  
      { (aa, [,L] [,NOPROT])                 }  
      { (aa,bb, . . . [,L] [,NOPROT]) }  
      }
```

Figure 4-8 Specifying the IEAFIXxx parmlib member

### How to specify FIX parameter during system initialization

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, the system displays the contents of the IEAFIXxx parmlib members at the operator's console as the system processes the members.
- NOPROT** By default, the LPA modules in the IEAFIXxx parmlib members are page-protected in storage. However, the NOPROT option allows an installation to override the page protection default.

**Syntax format for FIX parameter:** FIX= {aa} {(aa,[,L][,NOPROT]) }  
{(aa,bb,...[,L][,NOPROT])}

An example of overriding the value of the FIX parameter that was specified in the IEASYSxx during system initialization follows; member 01 is selected during system initialization:

```
IEA101A SPECIFY SYSTEM PARAMETERS FOR z/OS 02.05.00 HBB6605  
R 00,FIX=01  
IEE600I REPLY TO 00 IS;FIX=01
```

## 4.9 Modified link pack area

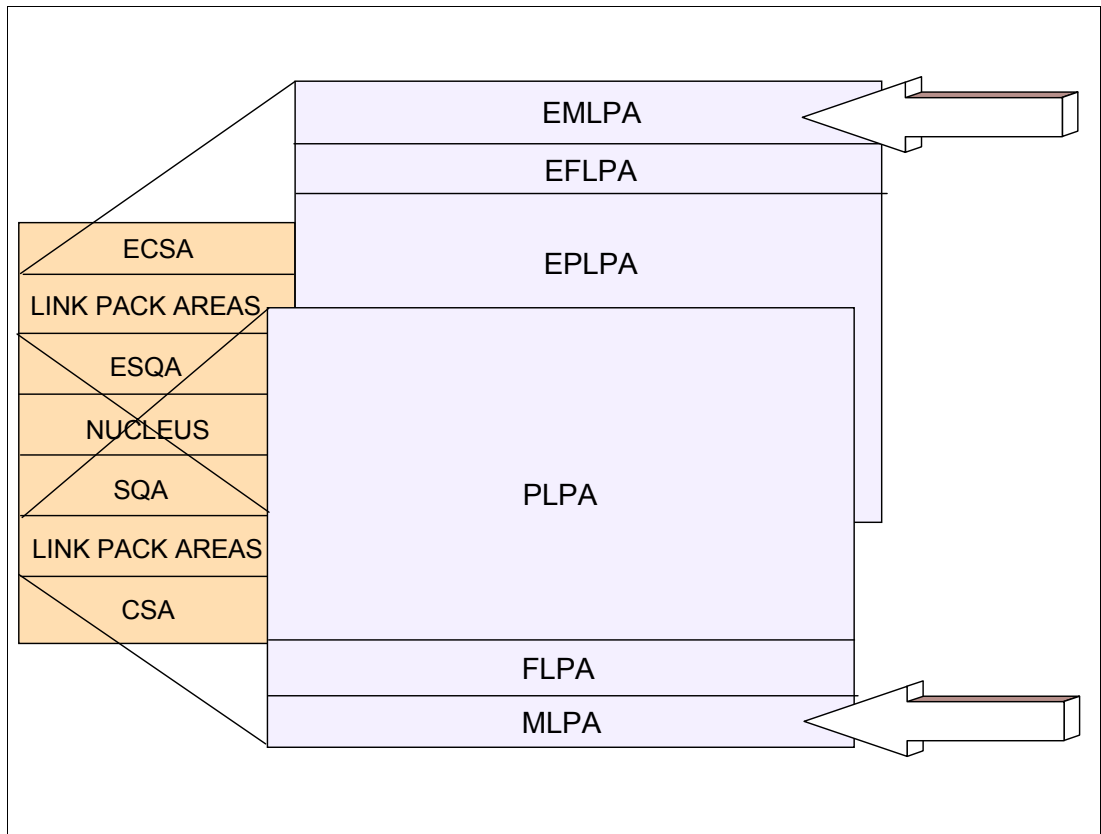


Figure 4-9 Modified Link Pack Area

### Modified link pack area (MLPA/extended MLPA)

This area may be used to contain reenterable routines from APF-authorized libraries that are to be part of the pageable extension to the link pack area during the current IPL.

The MLPA exists only for the duration of the IPL. Therefore, if an MLPA is desired, the 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 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 modules specified in the IEALPAXx are placed in the MLPA or the extended MLPA, depending 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.

## 4.10 Coding the IEALPAxx member

### Specify - (INCLUDE - LIBRARY - MODULES)

```
Menu  Utilities  Compilers  Help
-----
BROWSE  SYS1.PARMLIB(LEALPA02) - 01.01          Line 00000000 Col 001 080
Command ==>                                     Scroll ==> PAGE
***** Top of Data *****
INCLUDE LIBRARY (ISP.SISPLOAD)
          MODULES (ISPASUBS, ISPEX, ISPEXEC,
                  ISPLINK, ISPLNK,
                  ISPQRY)
INCLUDE LIBRARY (SYS1.LINKLIB)
          MODULES (IFBSEXIT)
***** Bottom of Data *****
```

Figure 4-10 Coding the IEALPAxx parmlib member

### How to code IEALPAxx of the SYS1.PARMLIB

Figure 4-10 is an example of the IEALPAxx member.

The important syntax rules for IEALPAxx are:

- ▶ Each statement must begin with the INCLUDE keyword.
- ▶ The library name follows the LIBRARY keyword and is enclosed in parentheses.
- ▶ The list of modules follows the MODULES keyword and is enclosed in parentheses. Any number of module names can be specified.
- ▶ The statement is assumed to be all information from one INCLUDE keyword to the next INCLUDE keyword or until end-of-file.
- ▶ Use all columns except 72 through 80.

## 4.11 Specifying the IEALPAxx member

- ❑ Operator response to IEA101A - system parameters
  - R 00,MLPA=02 - (overrides IEASYSxx MLPA=)
- ❑ Display contents of IEALPAxx on console
  - Use L option
- ❑ Override the page protection default
  - Use NOPROT option

```
MLPA= {aa                                     }
      { (aa , [ ,L] [ ,NOPROT] )             }
      { (aa ,bb , . . . [ ,L] [ ,NOPROT] ) }
```

Figure 4-11 Specifying the IEALPAxx member

### Specifying the MLPA member at system initialization

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:

- 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 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:** MLPA= {aa} {(aa[,L][,NOPROT]) }  
{(aa,bb,...[,L][,NOPROT])}

An example of overriding the value of the MLPA parameter in the IEASYSxx during system initialization follows; IEALPA02 was selected during system initialization:

```
IEA101A SPECIFY SYSTEM PARAMETERS FOR z/OS 02.05.00 HBB6605
R 00,MLPA=02
IEE600I REPLY TO 00 IS;MLPA=02
```

## 4.12 Dynamic LPA functions

- ❑ LPA statement in PROGxx parmlib member
  - LPA ADD
  - LPA DELETE
  - LPA CSAMIN
- ❑ SETPROG LPA operator command
- ❑ D PROG,LPA operator command
- ❑ CSVDYLPA macro

Figure 4-12 Dynamic LPA functions

### Managing dynamic LPA content

Dynamic link pack area (DLPA) was introduced in OS/390 Release 4, which has the ability to dynamically add or delete modules from the link pack area (LPA) after the IPL. This facility allows optional and new products to be loaded into the system without an IPL. It also enables you to display modules residing in LPA.

A module added dynamically will be found before one of the same name added during the IPL. This allows you to test new modules before you put them into production. Modules added dynamically to the LPA will be loaded into the common area (CSA) or extended common area (ECSA).

### How to perform dynamic LPA functions

The dynamic LPA functions may be invoked in one of the following ways:

- ▶ The PROGxx parmlib member includes the LPA statements, which are used to define what modules can be added to or deleted from LPA after the IPL. You use the **SET** command to validate the PROGxx parmlib member; for example, **SET PROG=xx**.
- ▶ The **SETPROG LPA** command may be used to initiate a change (add or delete) to the LPA.
- ▶ The **DISPLAY PROG,LPA** command may be used to display information about modules that have been added to LPA.
- ▶ The CSVDYLPA macro allows an authorized program to initiate dynamic LPA services.

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

## 4.13 The LNKLST

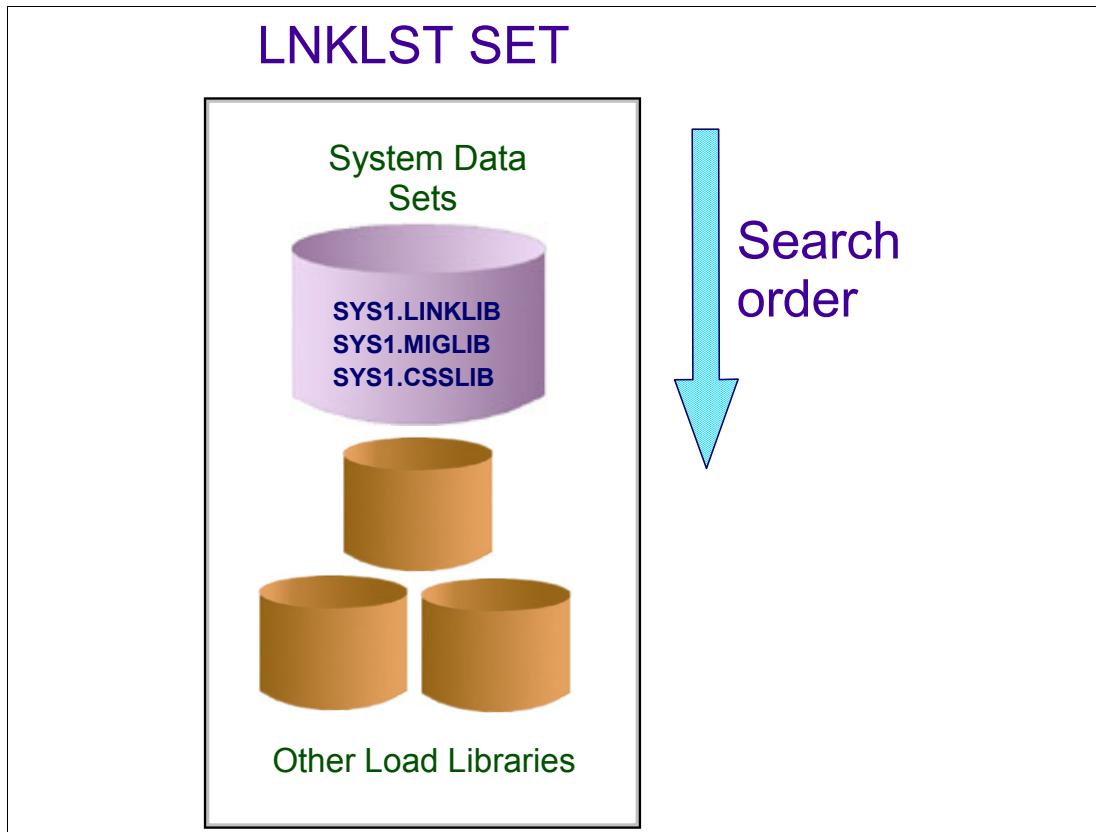


Figure 4-13 Overview of the LNKLST

### Overview of the LNKLST

This section explains the LNKLST and the related mechanisms utilized to improve the performance of locating and fetching modules. It is important to understand the relationship between the LNKLST and LLA/VLF.

When determining which data sets LLA is to manage, try to limit these choices to production load libraries that are rarely changed. Because LLA manages LNKLST libraries by default, you need only 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.

Because you obtain the most benefit from LLA when you have both LLA and VLF functioning, you should plan to use both. When used with VLF, LLA reduces the I/O required to fetch modules from DASD by causing selected modules to be staged in VLF data spaces. 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. This includes LNKLST libraries. A library must remain cataloged for the entire time it is managed by LLA. The benefits of LLA apply only to modules that are retrieved through the following macros: LINK, LINKX, LOAD, ATTACH, ATTACHX, XCTL, and XCTLX.

LLA does not stage load modules in overlay format. LLA manages the directory entries of overlay format modules, but the modules themselves are provided through program fetch. If



you want to make overlay format modules eligible for staging, you must re-linkedit the modules as non-overlay format. These reformatted modules might occupy more storage when they execute and, if LLA does not stage them, might take longer to be fetched.

The LNKLIST is a group of system and user-defined load libraries which form part of the search order the system uses for programs.

Modules (programs), whether stored as load modules or programs objects, must be loaded into both virtual storage and central storage before they can be run.

By default, the LNKLIST begins with these system data sets:

- ▶ SYS1.LINKLIB
- ▶ SYS1.MIGLIB
- ▶ SYS1.CSSLIB

You can change this order and add other load libraries to the LNKLIST concatenation.

## The LNKLIST concatenation

The LNKLIST concatenation is established at IPL time. It consists of SYS1.LINKLIB, followed by the libraries specified in the LNKLISTxx member(s) of SYS1.PARMLIB. The LNKLISTxx member is selected through the LNK parameter in the IEASYSxx member of the SYS1.PARMLIB. In addition, the system also automatically concatenates data sets SYS1.MIGLIB and SYS1.CSSLIB to SYS1.LINKLIB.

The building of the LNKLIST 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 may be included in the linklist. However, to include a user cataloged data set in the LNKLIST concatenation, 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 LNKLIST 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 has been exceeded, the system writes error message IEA328E to the operator's console.

These data sets are concatenated in the order in which they appear in the LNKLISTxx member(s), and the system creates a data extent block (DEB) that describes the data sets concatenated to SYS1.LINKLIB and their extents. This contains details of each physical extent allocated to the linklist. These extents remains in effect for the duration of the IPL. After this processing completes, the library lookaside (LLA) is started and manages the LNKLIST data sets and can be used to control updates to them.

## Specifying the LNK parameter

An example of overriding the value of the LNK parameter in the IEASYSxx during system initialization is as follows:

```
IEA101A SPECIFY SYSTEM PARAMETERS FOR z/OS 02.05.00 HBB6605
R 00, LNK=03
IEE600I REPLY TO 00 IS; LNK=03
```

For more information, see *z/OS MVS Initialization and Tuning Reference*, SA22-7592, and *z/OS MVS System Commands*, SA22-7627.

## 4.14 Dynamic LNKLST functions

- ❑ LNKLST statement in PROGxx parmlib member
  - SET PROG=xx command
- ❑ SETPROG LNKLST operator command
- ❑ D PROG, LNKLST operator command
- ❑ CSVDYNL macro

Figure 4-14 Dynamic LNKLST commands

### Managing dynamic LNKLST content

You can change the current LNKLST set dynamically through the **SET PROG=xx** and **SETPROG LNKLST** commands.

A LNKLST set remains allocated until there are no longer any jobs or address spaces associated with it. If the current LNKLST set is dynamically changed, any job or address space associated with the previous LNKLST set continues to use the data sets until the job or address space finishes processing. 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 that are started after the new current LNKLST set is activated use the new current LNKLST set.

An active LNKLST set cannot be modified. Once the last job or address space associated with a LNKLST set terminates, the LNKLST set is no longer active. The only other way to deactivate a LNKLST set is with LNKLST UPDATE.

Through SET PROG=xx and SETPROG LNKLST, you can also remove the definition of a LNKLST set from the system, associate a job or address space with the current LNKLST set, or locate a specific module associated with a data set in the LNKLST set.

You can update the LNKLST content dynamically using the following 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.

- ▶ You can simply use the **SETPROG LNKST** operator command to update the LNKST directly.
- ▶ You can also use the **D PROG, LNKST** command to display information about the LNKST set.
- ▶ Finally, you can use CSVDYNL macro programming service in an authorized program to change the LNKST concatenation for associated jobs and address spaces.

For more information, see *z/OS MVS Initialization and Tuning Reference*, SA22-7592, and *z/OS MVS System Commands*, SA22-7627.

## 4.15 Library lookaside (LLA)

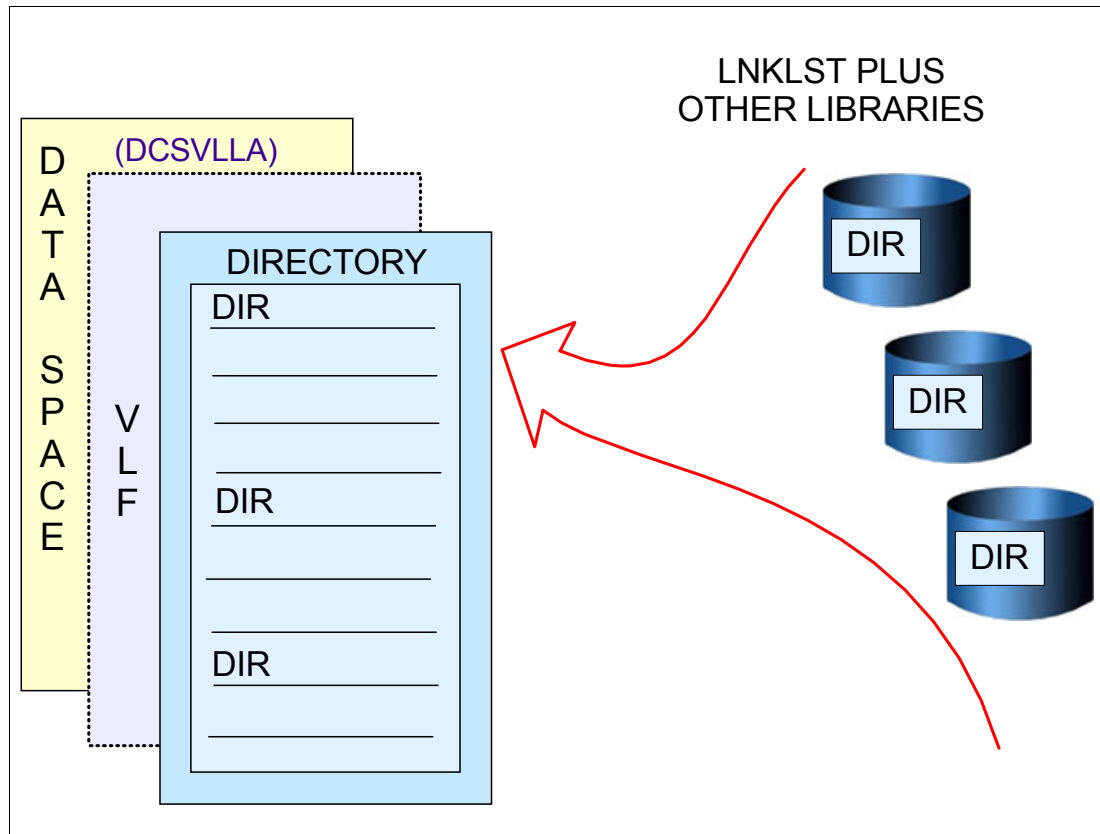


Figure 4-15 Library lookaside (LLA)

### The library lookaside (LLA) overview

Library lookaside (LLA) is an address space that maintains a copy of the directory entries of the libraries that it manages. Since the entries are cached, the system does not need to read the data set directory entries to find out where a module is stored before fetching it from DASD. This greatly reduces I/O operations.

The main purpose of using LLA is to improve the performance of module fetching on your system.

### How LLA improves performance

LLA improves module fetch performance in the following ways:

1. By maintaining (in the LLA address space) copies of the library directories the system uses to locate load modules. The system can quickly search the LLA copy of a directory in virtual storage instead of using costly I/O to search the directories on DASD.
2. By placing (or staging) copies of selected modules in a virtual lookaside facility (VLF) data space DCSVLLA when you define the LLA class to VLF, and start VLF. The system can quickly fetch modules from virtual storage, rather than using slower I/O to fetch the modules from the DASD.
3. By determining which modules, if staged, would provide the most benefit to module fetch performance. LLA evaluates modules as candidates for staging based on statistics it collects about the members of the libraries it manages (such as module size, frequency of fetches per module (fetch count), and the time required to fetch a particular module). If

necessary, you can directly influence LLA staging decisions through installation exit routines (CSVLLIX1 and CSVLLIX2).

## Planning LLA use

Before you can use LLA, you have to do the following:

1. Determine which libraries should be LLA-managed libraries.
2. Code the CSVLLAxx parmlib member to include, modify, and remove the LLA-managed libraries as well as to optimize the performance of the search processing and selectively refresh LLA directory entries.
3. Learn how to control the LLA through the operator commands **Start**, **Stop**, and **Modify LLA**. LLA directory entries may be selectively refreshed via the operator command **F LLA, REFRESH**.

## LLA-managed libraries

By default, LLA address space caches directory entries for all the modules in the data sets included in the linklist concatenation (defined in LNKLSTxx or PROGxx parmlib member).

You can also identify other user-defined load libraries that contain frequently used modules to be managed by LLA.

## 4.16 CSVLLAxx SYS1.PARMLIB member

```
LIBRARIES(libname1,libname2,...[-LNKLST-],...)  
    MEMBERS(mmbr1,mmbr2,..)  
LNKMEMBERS(mmbr1,mmbr2,...)  
REMOVE(libname1,libname2,...[-LNKLST-],...)  
GET_LIB_ENQ(YES|NO)  
PARMLIB(dsn) SUFFIX(xx)  
FREEZE(libname1,libname2,...[-LNKLST-],...)  
NOFREEZE(libname1,libname2,...[-LNKLST-],...)  
EXIT1(ON|OFF)  
EXIT2(ON|OFF)  
PARMSUFFIX(xx)
```

```
START LLA,LLA=xx  
MODIFY LLA,UPDATE=xx  
MODIFY LLA,REFRESH
```

Figure 4-16 CSVLLAxx parmlib member

### CSVLLAxx parmlib member

You use the CSVLLAxx parmlib member to specify which libraries (in addition to the LNKLST concatenation) library lookaside is to manage.

**Note:** If you do not supply a CSVLLAxx member, LLA will by default manage only those libraries 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.
- ▶ Libraries for which LLA is to use the performance-enhancing FREEZE/NOFREEZE option.
- ▶ 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 build the LLA directory. This command is issued during system initialization by the IBM-supplied IEACMD00 parmlib member; the command can also be entered by the operator.

The parameters for the **S LLA** command are:

|                 |                                                                                                                                                                                                                                      |
|-----------------|--------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| <b>LLA</b>      | Invokes the LLA procedure and creates the LLA address space.                                                                                                                                                                         |
| <b>LLA=xx</b>   | Indicates which CSVLLAnn parmlib member LLA is to use. If you omit <b>LLA=xx</b> , LLA will build its directory using only the LNKST libraries.                                                                                      |
| <b>SUB=MSTR</b> | Indicates that the name of the subsystem that will process 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. |

**Note:** We recommend that you specify **SUB=MSTR** on the **START LLA** command to prevent LLA from failing if JES fails.

**Syntax format 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 modify LLA

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.

► **MODIFY LLA,UPDATE=xx**

This command rebuilds LLA's directory only for specified libraries or 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.

**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 MODIFY LLA command to refresh LLA's cache with the latest version of the directory information from the DASD. Otherwise, the system will continue to use an older version of a load module. If you accidentally delete a data set from the current LNKST set, the system will continue working, and continue finding modules in the deleted library! This is because the physical addresses of the members are still held by LLA even though it is no longer possible to open the directory. This only works, of course, until the physical space on the DASD is reused by something else. You will also find yourself in the situation where you need to compress a data set from the LNKST set. You should refresh LLA's cache after the compress; otherwise, the directory entries in the LLA for every module moved during the compress process would be invalid, leading to abends whenever a user attempted to load one of these, until a refresh is done.

Use the following to do a complete update of the LLA:

```
F LLA,REFRESH  
CSV210I  LIBRARY LOOKASIDE REFRESHED
```



## 4.17 Compressing LLA-managed libraries

- ❑ Issue a **MODIFY LLA,UPDATE=xx** command
  - CSVLLAxx parmlib member has a **REMOVE** statement identifying the library that needs to be compressed
- ❑ Compress the library
- ❑ Issue a **MODIFY LLA,UPDATE=xx** command
  - CSVLLAxx parmlib member includes a **LIBRARIES** statement to return the compressed library to LLA management

Figure 4-17 Compressing LLA-managed libraries

### How to compress LLA-managed libraries

The procedure for compressing an LLA-managed library is:

1. Create a new CSVLLAxx member that includes a **REMOVE** statement identifying the library that needs to be compressed.

```
Menu Utilities Compilers Help
-----
BROWSE SYS1.PARMLIB(CSVLLA02) - 01.01          Line 00000000 Col 001 080
***** Top of Data *****
REMOVE(SYS1.LINKLIB)
***** Bottom of Data *****
```

Then issue the **F LLA,UPDATE=xx** command, as follows:

```
F LLA,UPDATE=02
IEE252I MEMBER CSVLLA02 FOUND IN SYS1.PARMLIB
CSV210I LIBRARY LOOKASIDE UPDATED
```

2. Compress the library.

There are two ways of compressing a data set:

- You can issue the line command **Z** against the data set you want to compress from the ISPF panel.

```

Menu Options View Utilities Compilers Help
-----
DSLIS - Data Sets Matching SYS1.LINKLIB                      Row 1 of 1
Command ==>  Scroll ==> PAGE

Command - Enter "/" to select action          Message          Volume
-----
Z      SYS1.LINKLIB                                MPRES1
***** End of Data Set list *****

```

- Or you can submit a job to compress the data set using the utility IEBCOPY as shown in the following sample JCL.

```

//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
/*

```

3. Create a new CSVLLAxx member that includes a LIBRARIES statement to return the compressed library to LLA management.

```

Menu Utilities Compilers Help
-----
BROWSE SYS1.PARMLIB(CSVLLA03) - 01.01          Line 00000000 Col 001 080
***** Top of Data *****
LIBRARIES(SYS1.LINKLIB)
***** Bottom of Data *****

```

Then issue the **F LLA,UPDATE=xx** command, as follows:

```

F LLA,UPDATE=03
IEE252I MEMBER CSVLLA03 FOUND IN SYS1.PARMLIB
CSV210I LIBRARY LOOKASIDE UPDATED

```

## 4.18 Virtual lookaside facility (VLF)

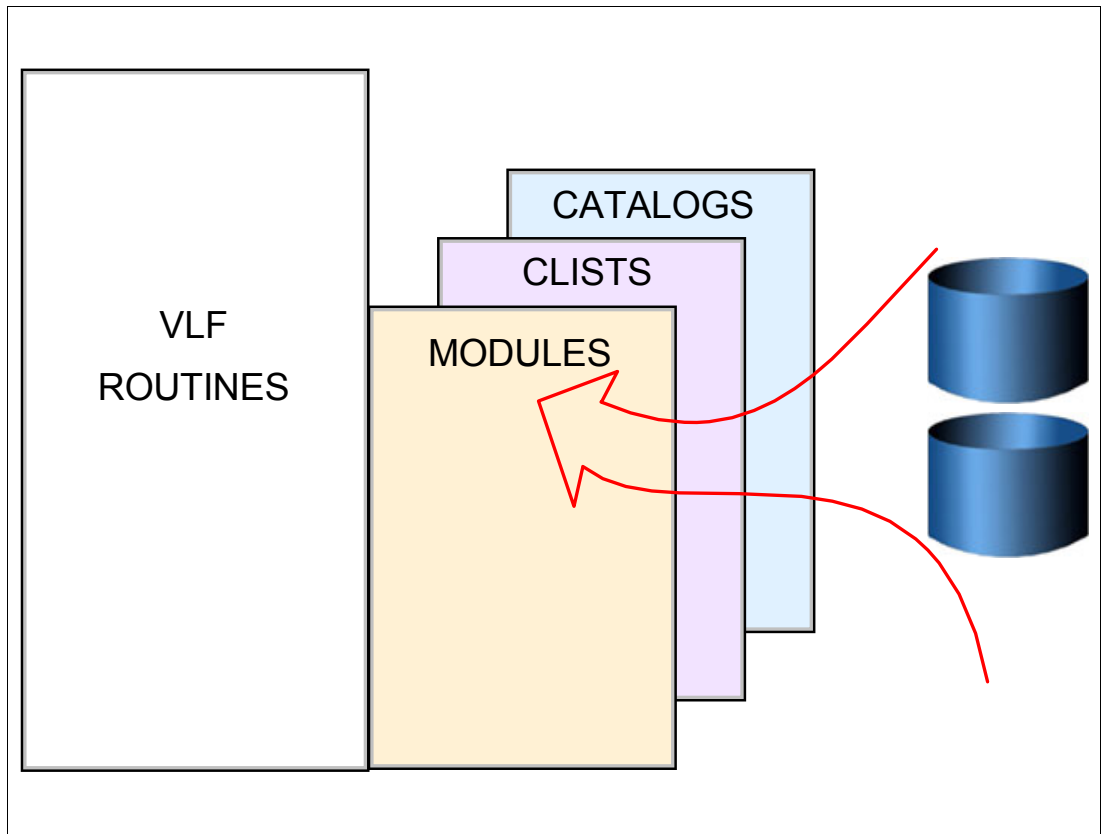


Figure 4-18 Virtual lookaside facility (VLF) overview

### Virtual lookaside facility (VLF) overview

The virtual lookaside facility (VLF) is a set of services that can improve the response time of applications that must retrieve a set of data for many users. VLF creates and manages data spaces to store an application's most frequently used data. When the application makes a request for data, VLF checks its data space to see if the data is there. If the data is present, VLF can rapidly retrieve it without requesting I/O to DASD.

To take advantage of VLF, an application must identify the data it needs to perform its task. The data is known as a data object. Data objects should be small-to-moderate in size, named according to the VLF naming convention discussed later in this section, and associated with an installation-defined class of data objects.

Certain IBM products or components such as LLA, TSO/E, CAS, and RACF use VLF as an alternate way to access data. Since VLF uses virtual storage for its data spaces, 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 will obtain the most benefit from LLA when you have both LLA and VLF functioning; you should plan to use both. When used with VLF, LLA reduces the I/O required to fetch modules from the DASD by causing selected modules to be staged in VLF data spaces.

LLA does not use VLF to manage library directories. When using 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 your system performance, there are several factors you have to consider. Some of these factors are:

- ▶ VLF works best with two kinds of data:
  - Data objects that are members of partitioned data sets, located through a partitioned data set (PDS) concatenation.
  - Data objects that, while not PDS members, could 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.

- ▶ 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 processor storage and not be paged out to auxiliary storage.
- ▶ VLF works best with relatively small objects because less virtual storage is expended to reduce the number of I/O operations.
- ▶ 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, expanded, or auxiliary storage constraint, this strategy is probably not a good choice.

## VLF planning

Before you can use VLF, you have to:

1. Start VLF using the IBM-supplied default COFVLFxx parmlib member.
2. Update COFVLFxx to include the VLF classes associated with the applications or products.

## 4.19 COFVLFxx parmlib member

```
CLASS    NAME(classname)
        {EDSN(dsn1) [VOL(vol)] EDSN(dsn2)...}
        {EMAJ(majname1) EMAJ(majname2)...}

        [MAXVIRT(nnn)]

        S VLF,SUB=MSTR[,NN=xx]

        P VLF
```

Figure 4-19 Defining the COFVLFxx parmlib member

### COFVLFxx parmlib member

You use the COFVLFxx 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 COFVLFxx 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 end users concurrently

Some of the more important statement and parameters for COFVLFxx are:

|                        |                                                                                                                                                                                                                              |
|------------------------|------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| <b>CLASS</b>           | Each group of objects that VLF processes must have a CLASS statement defining it. The CLASS statement indicates that the following parameters define that particular group of objects to VLF.                                |
| <b>NAME(classname)</b> | NAME(classname) specifies the name of the VLF class. The classname may be 1 to 7 alphanumeric characters including @, #, and \$. IBM-supplied VLF class names begin with the letters A through I, for example, NAME(CSVLLA). |

**EDSN(dsn)]VOL(vol)]** For a PDS class, EDSN(dsn) identifies a partitioned data set name whose members are eligible to be the source for VLF objects in the class. 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). Without the volume serial number, an uncataloged data set is not included in the eligible data set name list. The system issues an informational message to the operator to identify any data set where the system cannot find the catalog entry to extract the volume. 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.

**EMAJ(majname)** EMAJ identifies an eligible major name (majname) for a non-PDS class, a class that does not have a major name and minor name related to partitioned data sets and their members. The majname can be 1 to 64 alphanumeric characters except comma(,), blank, or right parenthesis ()), for example EMAJ(LLA).

**Note:** Do not use the EMAJ and the EDSN parameter on the same CLASS statement.

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

:The following shows an example of a COFVLFxx parmlib 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 *****

```

### Starting and stopping VLF

To start VLF, you use the **START VLF, SUB=MSTR, N=xx** command. This command identifies the COFVLFxx parmlib member to build VLF. This command is issued by the IBM-supplied COMMND00 parmlib member during system initialization; the command can also be entered by the operator.

The parameters for the **S VLF** command are: **S VLF, SUB=MSTR[, NN=xx]**. This invokes the VLF procedure that starts the virtual lookaside facility (VLF), where:

**NN=xx** Indicates that the system is to start VLF using the COFVLFxx member of the SYS1.PARMLIB (default COFVLF00).

**Syntax for S VLF command:** **S VLF, SUB=MSTR[, NN=xx]**

To stop VLF use the **P VLF** command.

The parameter for the **P VLF** command is:

**VLF** The jobname assigned to the virtual lookaside facility. Using this parameter stops VLF with the message COF033I.

**Note:** Stopping VLF can degrade your system performance.

## 4.20 Authorized libraries

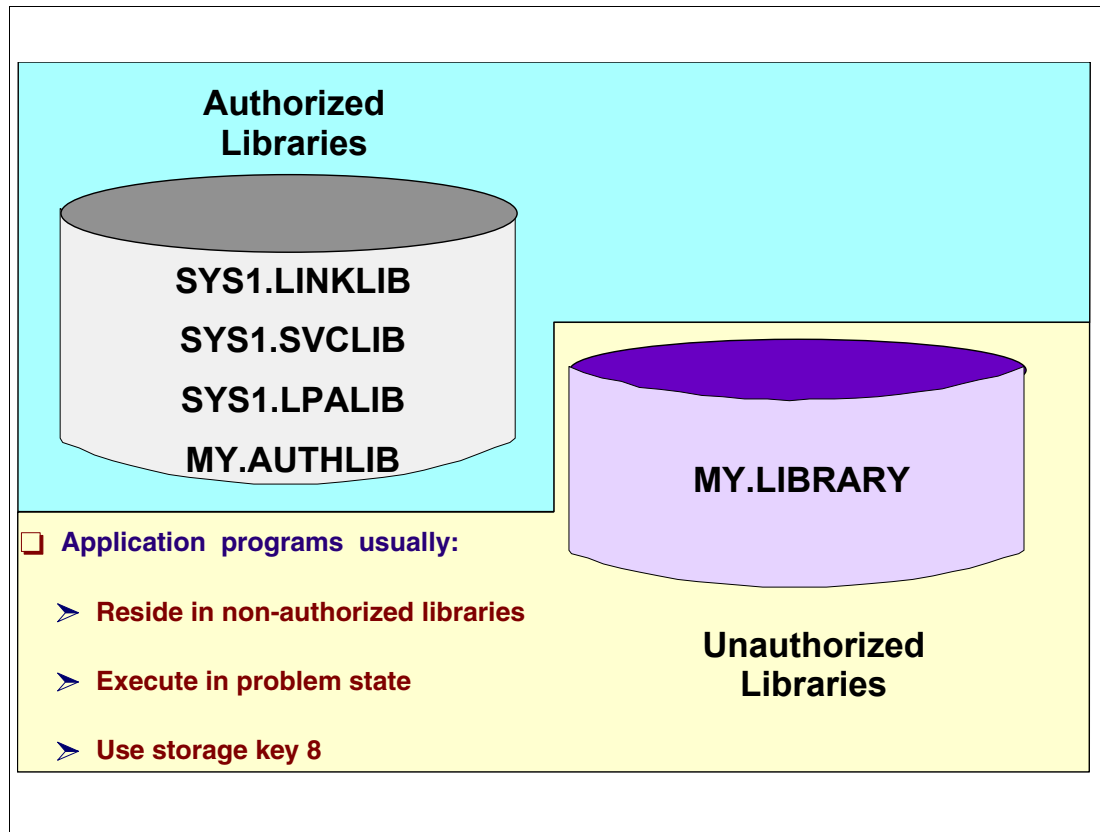


Figure 4-20 Authorized libraries overview

### Authorized libraries

z/OS offers a mechanism called the authorized program facility (APF) to restrict the access to sensitive system functions or user programs. APF was designed to avoid integrity exposures. The installation identifies what libraries contain those special functions or programs. Those libraries are then called APF (authorized program facility) libraries.

The modified link pack area (MLPA) may be used to contain reenterable routines from APF-authorized libraries that are to be part of the pageable extension to the link pack area 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 for each IPL (including quick start and warm start IPLs).

The MLPA is allocated just below the FLPA (or the PLPA, if there is no FLPA); the extended MLPA is allocated above the extended FLPA (or the extended PLPA if there is no extended FLPA). When the system searches for a routine, the MLPA is searched before the PLPA.

**Note:** Loading a large number of modules in the MLPA can increase fetch time for modules that are not loaded in the LPA. This could affect system performance.

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.



## Authorized libraries overview

Many system functions, such as entire supervisor calls (SVC) or special paths through SVCs, are sensitive. Access to these functions must be restricted to only authorized programs to avoid compromising the security and integrity of the system.

The system considers a program authorized if the program has one or more of the following characteristics:

- ▶ Runs in supervisor state (bit 15 of the PSW is zero).
- ▶ Runs with PSW key 0 to 7 (bits 8 through 11 of the PSW are in the range 0 to 7).
- ▶ Runs under an APF-authorized job step task.

The authorized program facility allows your installation to identify system or user programs that can use sensitive system functions.

Libraries that contain authorized programs are known as authorized libraries. APF-authorized programs must reside in one of the following authorized libraries:

- ▶ SYS1.LINKLIB
- ▶ SYS1.SVCLIB
- ▶ SYS1.LPALIB
- ▶ An authorized library specified by your installation

**Note:** By default, the libraries in the LNKST concatenation are considered authorized unless you specified LNKAUTH=APFTAB in the IEASYSxx parameter list. However, if the system accesses the libraries in the LNKST concatenation through JOBLIB 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 IEAAPFxx 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 modules in PLPA are marked as coming from an authorized library. When accessed through a STEPLIB, JOBLIB, or TASKLIB DD statement, SYS1.LPALIB is considered an authorized library only if you have included it in the APF list.

## Authorized program facility

You can use the authorized program facility (APF) to identify system or user programs that can use sensitive system functions. For example, APF:

- ▶ Restricts the use of sensitive system SVC routines (and sensitive user SVC routines, if you need them) to APF-authorized programs
- ▶ Allows the system to fetch all modules in an authorized job step task only from authorized libraries, to prevent programs from counterfeiting a module in the module flow of an authorized job step task

## Authorizing a program

To authorize a program, you must first assign the authorized code to the first load module of the program. APF prevents authorized programs from accessing any load module that is not in an authorized library. When the system attaches the first load module of a program, the system considers the program APF-authorized if the module meets both the following criteria:

1. The module is contained in an authorized library.

2. The module is link-edited with authorized code, AC=1 (to indicate that you want to authorize the job step task). This code is contained in a bit setting in the partitioned data set (PDS) directory entry for the module.

### Link-editing 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. 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',...
```

This method causes the system to consider every load module created by the step to be authorized.

The authorization code of a load module has meaning only when it resides on an APF-authorized data set and when the load module executes as the first program of a job-step attach. If no authorization code is assigned in the linkage editor step, the system assigns the default of authorized. Thus, if a load module tries to execute functions or SVCs that require authorization, the system abnormally ends the program and issues abend code X'306' reason code X'04'.

### Guidelines for using APF

When you use APF authorization, you must control which programs are stored in the authorized libraries. If the first module in a program sequence is authorized, the system assumes that the flow of control to all subsequent modules is known and secure as long as these subsequent modules come from authorized libraries. To ensure that this assumption is valid you should:

- ▶ 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 modules with the same name could lead to accidental or deliberate mix-up in module flow, possibly introducing an integrity exposure.
- ▶ Link edit 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, thus ensuring that a user cannot call modules out of sequence, or bypass validity checking or critical-logic flow.

It is recommended that you protect the libraries in the APF list with generic security product profiles so only selected users can store programs in those libraries.

## 4.21 Authorizing libraries

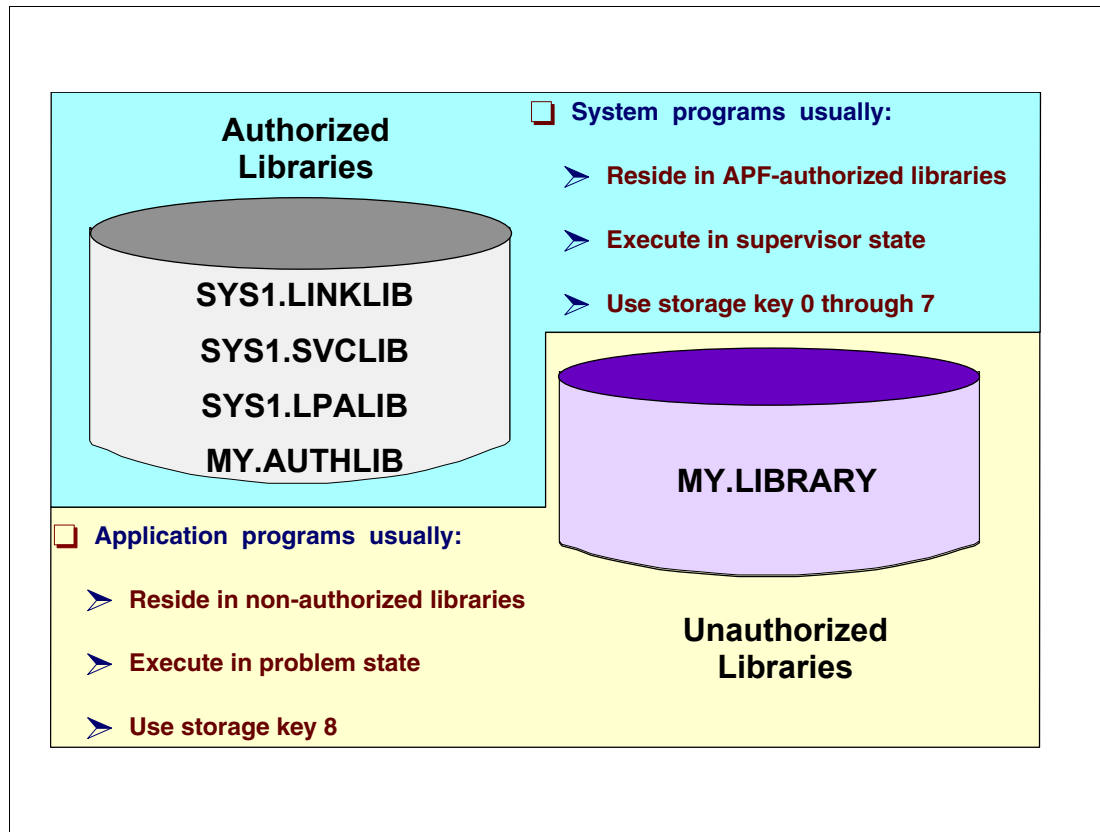


Figure 4-21 Authorizing libraries

### Specifying program libraries to be APF authorized

The APF list is built during IPL using those libraries specified in the IEAAPFxx or PROGxx parmlib members, 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

Use the IEAAPFxx member in the SYS1.PARMLIB to specify program libraries that are to receive APF authorization. List the data set names of the libraries, along with the volume where the library resides.

To specify the volume where the library resides, use one of the following:

- The volume serial number of the volume on which the library resides.
- Six asterisks (\*\*\*\*\*) to indicate that the library resides on the current SYSRES volume.
- \*MCAT\* to indicate the library resides on the volume on which the system master catalog resides.
- Nothing after the library name, to indicate that the library is managed by the storage management system (SMS).

:Following is an example of the IEAAPFxx:

```
Menu  Utilities  Compilers  Help
-----
BROWSE  SYS1.PARMLIB(PROGTT)  - 01.01          Line 00000000 Col 001 080
Command ===>                      Scroll ===> PAGE
***** Top of Data *****
SYS1.VTAMLIB      *****
SYS1.SICELINK     *****
SYS1.LOCAL.VTAMLIB TOTCAT,
ISP.SISPLoad      *MCAT*
```

## PROGxx parmlib member coding

You can use the APF statement 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:

1. The system automatically adds SYS1.LINKLIB and SYS1.SVCLIB to the APF list at IPL.
2. 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.
3. 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 statements and parameters for the APF statement are:

|                               |                                                                                                                                                                                                                                                                                                                                                                                                                                                                                             |
|-------------------------------|---------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| <b>FORMAT(DYNAMIC STATIC)</b> | Specifies that the format of the APF list is dynamic or static.                                                                                                                                                                                                                                                                                                                                                                                                                             |
| <b>ADD DELETE</b>             | Indicates whether you want to add or delete a library from the APF list.                                                                                                                                                                                                                                                                                                                                                                                                                    |
| <b>DSNAME(dsname)</b>         | The 44-character name of a library that you want to add or delete from the APF list.                                                                                                                                                                                                                                                                                                                                                                                                        |
| <b>SMSIVOLUME(volume)</b>     | The identifier for the volume containing the library specified on the DSNAME parameter, which is one of the following: <ul style="list-style-type: none"><li>– SMS, which indicates that the library is SMS-managed.</li><li>– A six character identifier for the volume.</li><li>– ***** , which indicates that the library is located on the current SYSRES volume.</li><li>– *MCAT* , which indicates that the library is located on the volume containing the master catalog.</li></ul> |

**Syntax format for APF statement:** APF FORMAT(DYNAMIC|STATIC) APF ADD | DELETE  
DSNAME(dsname) SMS | VOLUME(volname)

The following shows an example of 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 *****
```

**Note:** It is recommended that you use the PROGxx instead of IEAAPFxx parmlib 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 IEAAPFxx, followed by those libraries in the PROGxx. If you are currently using the IEAAPFxx parmlib member to define the APF list, you can convert the format of IEAAPFxx 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.

## 4.22 Dynamic APF functions

- ❑ APF statement in PROGxx parmlib member
- ❑ SETPROG APF operator command
- ❑ D PROG,APF operator command

Figure 4-22 Dynamic APF functions

### Managing dynamic APF

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 follows:

```
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
```

To add a library to the APF list use the following command:

```
Example of the APF ADD: 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.

```
Syntax for SETPROG APF command: SETPROG APF{,FORMAT={DYNAMIC|STATIC}}
{,{ADD|DELETE},DSNAME|LIBRARY=libname,{SMS|VOLUME=volume}}
```

To change the format of the APF list to dynamic:

```
SETPROG APF,FORMAT=DYNAMIC
CSV410I APF FORMAT IS NOW DYNAMIC
```

To add a library to the APF list:

```
SETPROG APF,ADD,DSNAME=SCRIPT.R40.DCFLOAD,VOLUME=MPRES2
CSV410I DATA SET SCRIPT.R40.DCFLOAD ON VOLUME MPRES2 ADDED TO APF LIST
```

To delete a library from the APF list:

```
SETPROG APF,DELETE,DSNAME=SCRIPT.R40.DCFLOAD,VOLUME=MPRES2
CSV410I DATA SET SCRIPT.R40.DCFLOAD ON VOLUME MPRES2 DELETED FROM APF
```

**Note:** If you try to add 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: D PROG,APF[,ALL ] |,DSNAME=libname
|,ENTRY=xxx |,ENTRY=(xxx-yyy) ],L={a|cc|cca|name|name-a} ]
```

To display the whole APF list:

```
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
```

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:

```
D PROG,APF,ENTRY=1
CSV450I 05.24.55 PROG,APF DISPLAY 979
FORMAT=DYNAMIC
ENTRY VOLUME DSNAME
  1 MPRES1 SYS1.LINKLIB
```

To display all the libraries in the range from 1 to 5:

```
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
```





## System Modification Program/Enhanced (SMP/E)

SMP/E 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 ensure that all products are installed at the proper level so all elements of the system can work together. At first, that might not sound too difficult, but as the complexity of the software configuration increases, 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 can help you maintain it.

This chapter covers the following topics:

- ▶ SMP/E overview
- ▶ Concept of SMP/E element
- ▶ Data sets used by SMP/E
- ▶ Consolidate software inventory (CSI)
- ▶ SMP/E commands you need to know
- ▶ SMP/E dialogs
- ▶ Java™ archive (JAR) file support
- ▶ GIMZIP and GIMUNZIP introduction
- ▶ GIMXSID service routine
- ▶ SMP/E release levels
- ▶ SMP/E hints

## 5.1 SMP/E overview

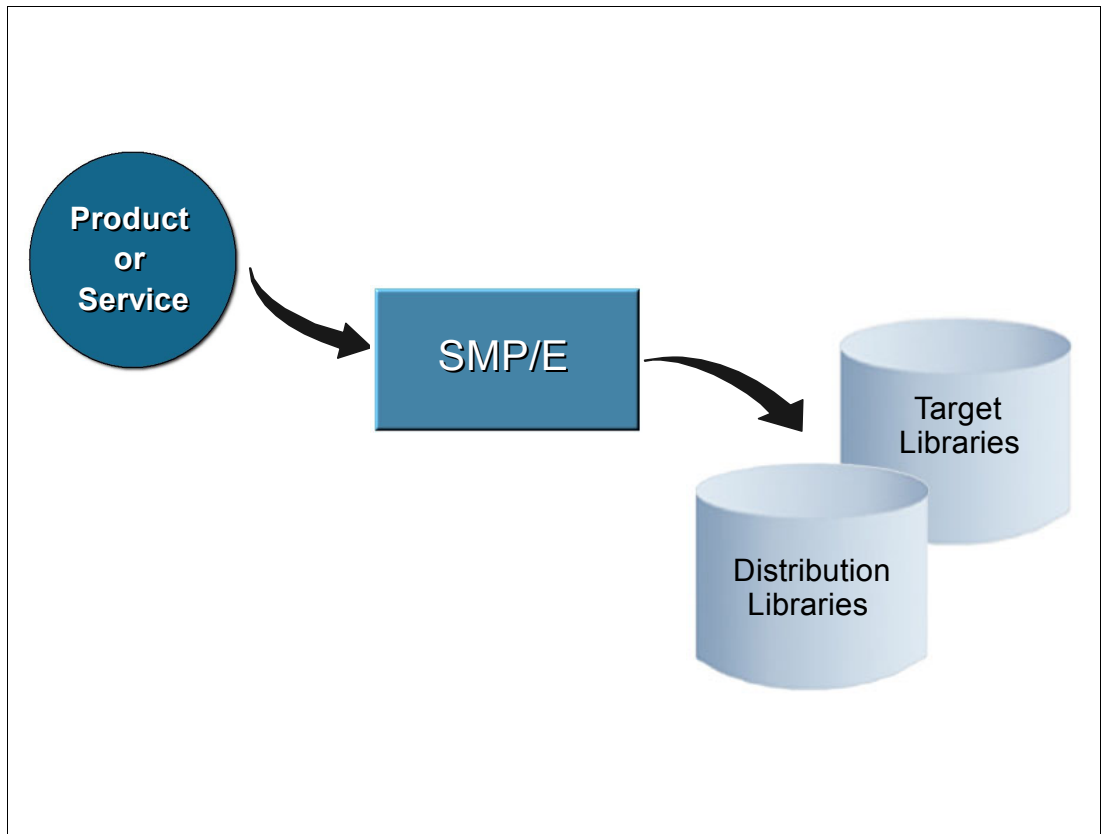


Figure 5-1 SMP/E overview

### SMP/E overview

SMP/E is the basic tool for installing and maintaining software in z/OS systems and subsystems. 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 for CBPDOs, ProductPacs, ServicePacs and selective follow-on service for CustomPacs. In addition, SMP/E 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). SMP/E dialogs help you interactively query the SMP/E database as well as create and submit jobs to process SMP/E commands.

These are some of the types of software that can be installed by SMP/E:

- ▶ Products and service provided in CBPDOs and CustomPac offerings
- ▶ Products and service from IBM Software Distribution Centers not provided in CBPDOs or CustomPac offerings

- ▶ Service provided in Expanded Service Options (ESOs)
- ▶ Other products and service

SMP/E can install software from any of these source, provided it is packaged as a **system modification**, or **SYSMOD**. When it processes SYSMODs, it installs the elements in the appropriate libraries and updates its own records of the processing it has done. It installs program elements into two types of libraries:

- ▶ **Target libraries**

Target libraries contain the executable code needed to run your system (for example, the libraries from which you run your production system or your test system).

- ▶ **Distribution libraries**

Distribution libraries (DLIBs) contain the master copy of each element for a system. They are used as input to the SMP/E GENERATE command or 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.

## 5.2 SYSMODs

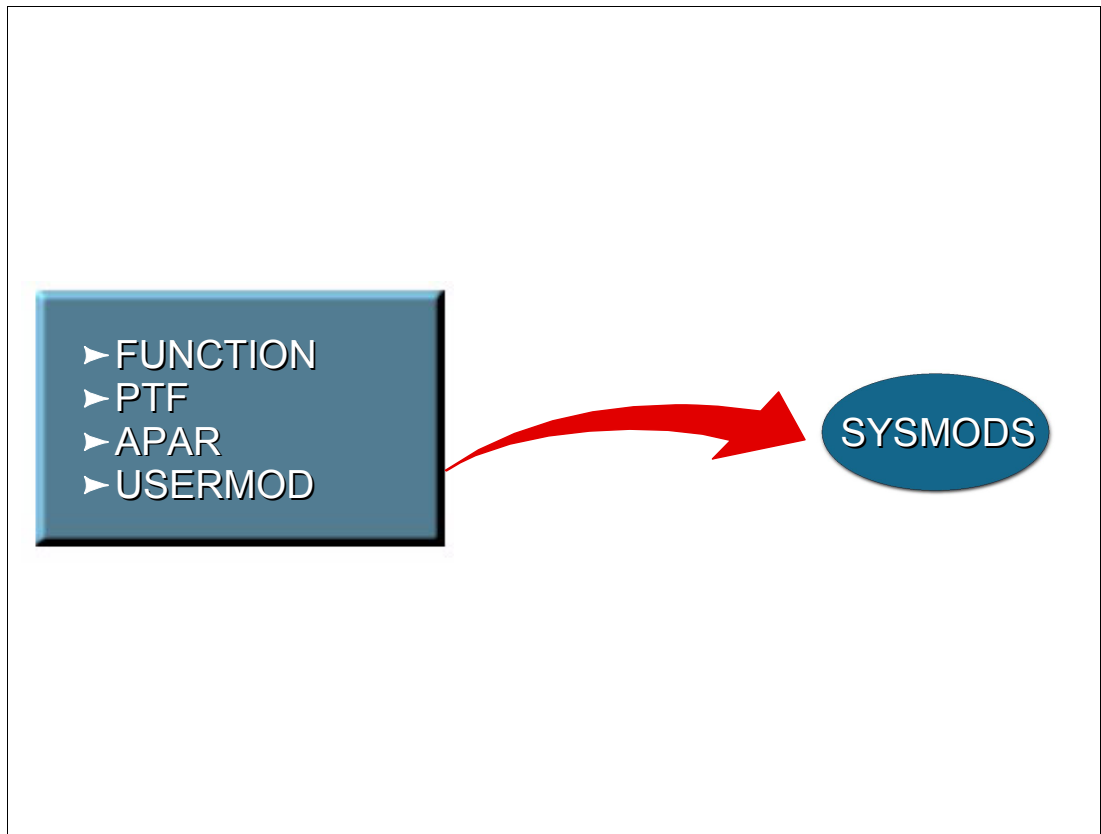


Figure 5-2 SYSMODs

### SYSMODs

Software, whether it is a product or service, consists of *elements* such as macros, modules, source, and other types of data (such as CLISTs or sample procedures). For software to be installed by SMP/E, 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 may also be installed on the same z/OS system. The combination of elements and control information is called a system modification, or SYSMOD.

You 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. In all cases, you are making system modifications.

The SYSMOD is the actual package containing information SMP/E needs to install and track system modifications. SYSMODs are composed of two parts:

- ▶ Modification control statements (MCS), 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.

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 modification identifier). The FMID is a 7 character identifier that needs to be unique to distinguish one product from another. Function SYSMODs for base functions are applicable to any z/OS environment, although they may 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.

► **PTFs**

These are IBM-supplied, tested fixes for reported problems. They are meant to be installed in all environments. PTFs may be used as preventive service to avert certain known problems that may have not yet appeared on your system, or they may 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. Each PTF has a unique, 7-character name called a SYSMOD ID.

► **APAR fixes**

Authorized Program Analysis Reports (APARs) are temporary fixes designed to fix or bypass a problem for the first reporter of the problem. These fixes may 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. Each APAR has a unique, 7-character name called a SYSMOD ID.

► **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. Each USERMOD has a unique, 7-character name called a SYSMOD ID.

**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. See *z/OS Packaging Rules*, SC23-3695, for detailed information on packaging software.

## 5.3 Concept of SMP/E elements

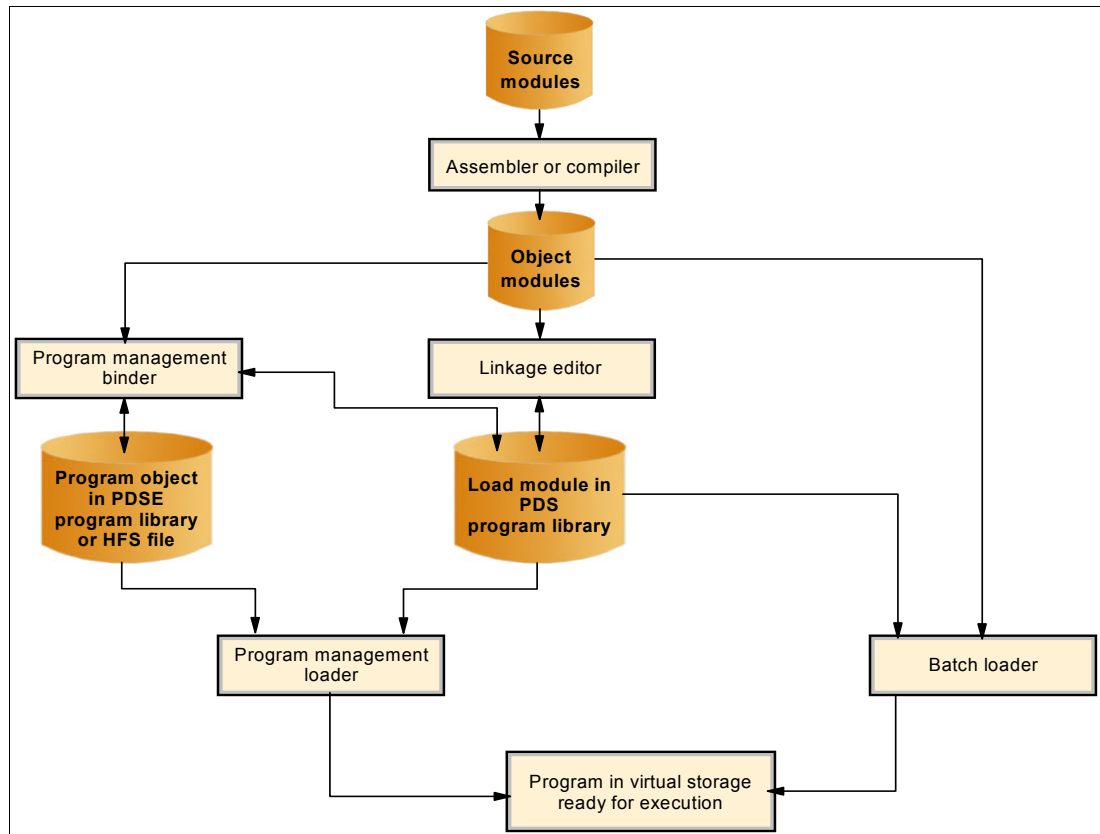


Figure 5-3 Concept of SMP/E elements

### Concept of SMP/E elements

An *element*, in SMP/E, is a part of a product, such as a macro, module, dialog panel, or sample code.

We describe here the major element types dealt with in SMP/E. But, before we go further, we need to review some z/OS programming concepts, such as:

- ▶ Machine instruction - A *machine instruction* asks the CPU to execute a specified action. This action falls in one of seven classes: general, decimal, floating-point-support (FPS), binary-floating-point (BFP), control, I/O, and hexadecimal-floating-point (HFP) instructions. Any machine instruction has an *operation code*, which specifies the action to be performed, and *operands*, the data to be modified by the action.
- ▶ Machine language - *Machine language* is language that is used directly by the CPU.
- ▶ Program - *Program* is a general term for any combination of statements that can be interpreted by a computer or language translator, and that serve to perform a specific function. Programs fall in three general classes:
  - Source code - *Source code* is the logical combination of the source statements written in an specific language by a programmer that constitute the input to a language translator for a particular translation (Cobol, PL/I, Assembler). In Assembler, for example, it is possible for the programmer to use a structured language named *Macros*. Writing Macros makes Assembler programming easier. With a Macro, the

programmer enters a few options and the Assembler transforms these options into object code.

- Object code - *Object code* is the output from a language translator (such as a compiler or an Assembler), where the source statements are converted to machine instructions. Before an object code can be executed, it must be processed by the linkage editor.
- Executable code - *Executable code* is the output of the *Binder*. Binder is a z/OS utility program that “link edits” object codes, thereby creating a fully executable code. The input for the Binder is the Object code (or even Executable code). Executable code is kept as special data set types, such as: PDS, PDSE, and HFS. *Program Management* is the name of the z/OS component that loads executable code in memory and manages its libraries. In z/OS there are two types of executable code:
  - Load modules stored in PDS data sets
  - Program Objects stored in PDSE or HFS data sets

**Note:** Program Objects remove many of the restrictions of the Load Modules and have support for new functionality.

Coming back to SMP/E, it works with SYSMODs as input. SYSMODs are modifications (usually corrections) to existing codes. Depending on the correction, the element type can be a Macro or a Source code. In this case, it must be assembled and link edited by the Binder in order to generate the Load Module (or Program Object) replacing the former one. It also can be a specific Load Module (or Program Object) ready to replace the one in error.

Some of the SMP/E element types that are addressed by SYSMODs are the following:

|                |                                                                               |
|----------------|-------------------------------------------------------------------------------|
| <b>SRC</b>     | Module source that has been installed in the target or distribution libraries |
| <b>MAC</b>     | Macros that have been installed in the target or distribution libraries       |
| <b>MOD</b>     | Modules used to create load modules in the target libraries                   |
| <b>PROGRAM</b> | Program objects                                                               |
| <b>HFS</b>     | Elements installed in a hierarchical file system (HFS)                        |
| <b>JAR</b>     | Java Archive files                                                            |

## 5.4 Changing the elements of the system

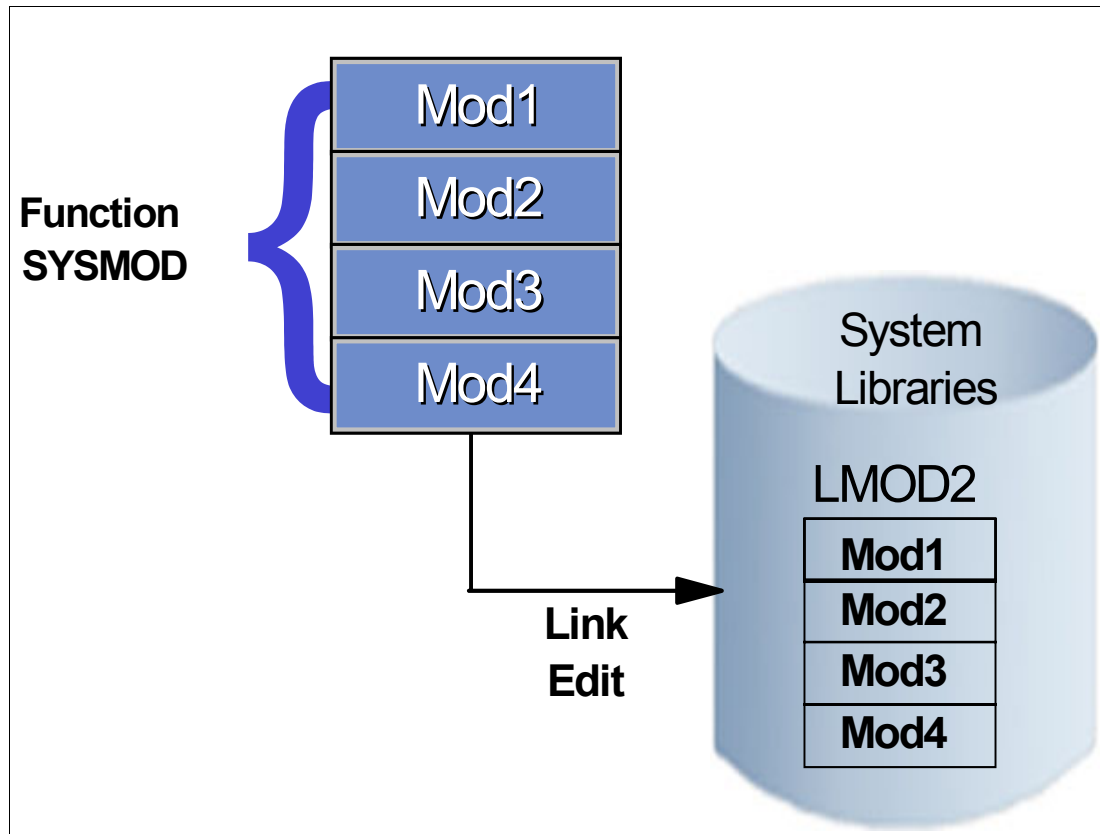


Figure 5-4 Changing the elements of the system

### Introducing an element (Function SYSMOD)

One way you can modify your system is to introduce new elements into that system. To accomplish this with SMP/E, you can install a function SYSMOD. The function SYSMOD introduces a new product, a new version or release of a product, or updated functions for an existing product into the system. All other types of SYSMODs are dependent upon the function SYSMOD, because they are all modifications of the elements originally introduced by the function SYSMOD.

When we refer to installing a function SYSMOD, we are referring to the placing of all the product's elements in the system data sets, or libraries. Examples of these libraries are SYS1.LINKLIB, SYS1.LPALIB, and SYS1.SVCLIB.

Figure 5-4 depicts the process of creating executable code in the production system libraries where the installation of a function SYSMOD link-edits object modules Mod1, Mod2, Mod3, and Mod4 to create load module LMOD2. The executable code created in load module LMOD2 is installed in the system libraries through the installation of the function SYSMOD.

Figure 5-5 is an example of a simple function SYSMOD that introduces four elements.



```

++FUNCTION(FUN0001)      /* SYSMOD type and identifier. */.
++VER(Z038)              /* For MVS SREL */.
++MOD(MOD1) RELFILE(1)   /* Introduce this module */
    DISTLIB(AOSFB)       /* in this distribution library. */.
++MOD(MOD2) RELFILE(1)   /* Introduce this module */
    DISTLIB(AOSFB)       /* in this distribution library. */.
++MOD(MOD3) RELFILE(1)   /* Introduce this module */
    DISTLIB(AOSFB)       /* in this distribution library. */.
++MOD(MOD4) RELFILE(1)   /* Introduce this module */
    DISTLIB(AOSFB)       /* in this distribution library. */.

```

*Figure 5-5 Example of the function SYSMOD MCS with four element*

## 5.5 Fixing problems with an element (PTF SYSMOD)

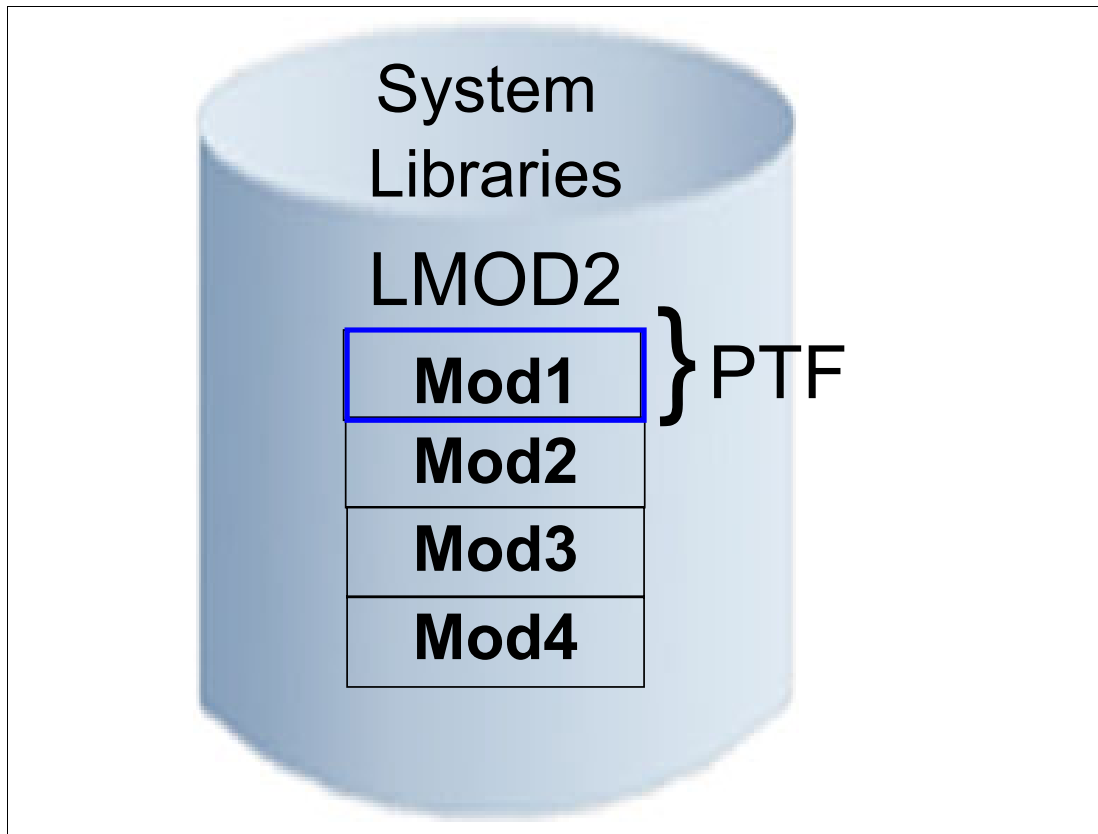


Figure 5-6 Preventing or fixing problems with an element (PTF SYSMOD)

### PTF SYSMOD

When a problem with a software element is discovered, IBM supplies its customers with a tested fix for that problem. This fix comes in the form of a program temporary fix (PTF). Although you may not have experienced the problem the PTF is intended to prevent, it is wise to install the PTF on your system. The PTF SYSMOD is used to install the PTF, thereby preventing the occurrence of that problem on your system.

Usually, PTFs are designed to replace or update one or more complete elements of a system function.

In Figure 5-6, we see a previously installed load module LMOD2. If we want to replace the element Mod1, we should install a PTF SYSMOD that contains the module Mod1. This PTF SYSMOD replaces the element in error with the corrected element. As part of the installation of the PTF SYSMOD, SMP/E relinks LMOD2 to include the new and corrected version of Mod1.

PTF SYSMODs are always dependent upon the installation of a function SYSMOD. In some cases, some PTF SYSMODs may also be dependent upon the installation of other PTF SYSMODs. These dependencies are called *prerequisites* (a typical PTF prerequisite is shown in Figure 5-13 on page 305).

Figure 5-7 on page 299 is an example of a simple PTF SYSMOD MCS.

```
++PTF(PTF0001)      /* SYSMOD type and identifier. */.  
++VER(Z038) FMID(FUN0001) /* Apply to this product. */.  
++MOD(MOD1)         /* Replace this module */  
    DISTLIB(AOSFB)   /* in this distribution library. */.  
...  
... object code for module  
...
```

*Figure 5-7 Sample PTF SYSMOD MCS*

## 5.6 Fixing a problem with an element (APAR SYSMOD)

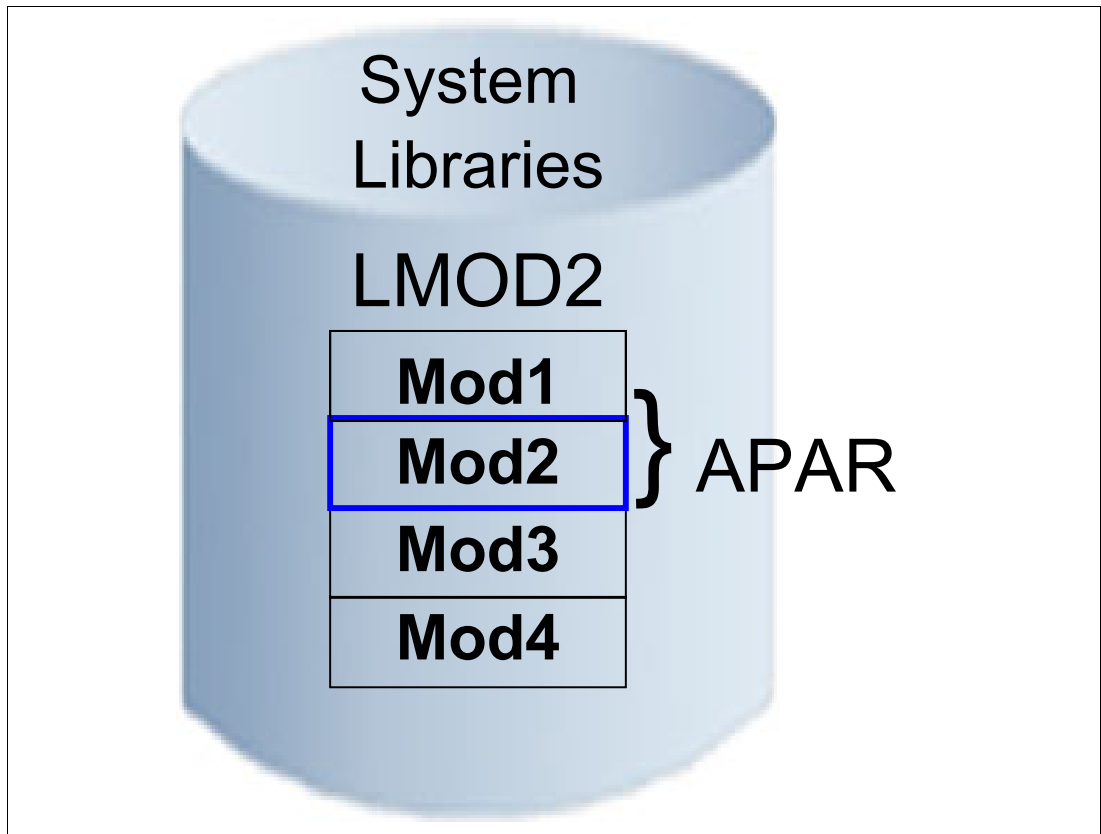


Figure 5-8 Fixing a problem with an element (APAR SYSMOD)

### APAR SYSMOD

You may sometimes find it is necessary to correct a serious problem that occurs on your system before a PTF is ready for distribution. In this situation, IBM supplies you with an Authorized Program Analysis Report (APAR). An APAR is a fix designed to quickly correct a specific area of an element or replace an element in error. You install an APAR SYSMOD to implement a fix, thereby updating the incorrect element.

In Figure 5-8, the highlighted section is an area of Mod2 containing an error. The processing of the APAR SYSMOD provides a modification for object module Mod2. During the installation of the APAR SYSMOD, Mod2 is updated (and corrected) in load module LMOD2.

The APAR SYSMOD always has the installation of a function SYSMOD as a prerequisite, and can also be dependent upon the installation of other PTF or APAR SYSMODs.

Some APARs do not change modules; instead, they registers problems while the final solution is being developed. Another APAR function is to give tips about customization and to alert for probable customer errors.

Figure 5-9 on page 301 is an example of a simple APAR SYSMOD.

:

```
++APAR(APAR001)          /* SYSMOD type and identifier. */.  
++VER(Z038) FMID(FUN0001) /* Apply to this product      */.  
                        PRE(UZ00004) /* at this service level. */.  
++ZAP(MOD2)              /* Update this module          */.  
                        DISTLIB(A0SFB) /* in this distribution library. */.  
...  
... zap control statements  
...
```

*Figure 5-9 Example of simple APAR SYSMOD MCS*

## 5.7 Customizing an element (USERMOD SYSMOD)

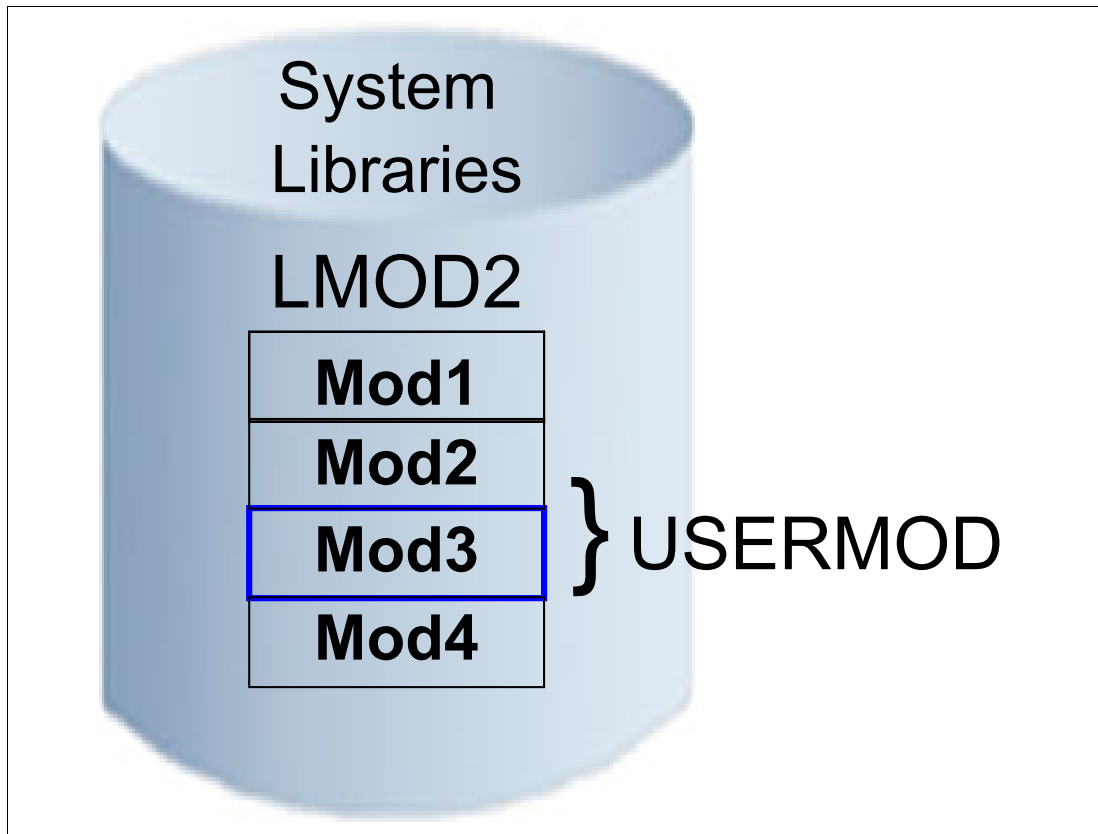


Figure 5-10 Customizing an Element (USERMOD SYSMOD)

### USERMOD SYSMOD

If you have a requirement for a product to perform differently from the way it was designed, you might want to customize that element of your system. IBM provides you with certain modules that allow you to tailor IBM code to meet your specific needs. After making the desired changes, you add these modules to your system by installing a USERMOD SYSMOD. This SYSMOD can be used to replace or update an element, or to introduce a totally new user-written element into the system. In either case, the USERMOD SYSMOD is built by you either to change IBM code or to add your own code to the system.

In Figure 5-10, Mod3 has been updated through the installation of a USERMOD SYSMOD.

Prerequisites for USERMOD SYSMODs are the installation of a function SYSMOD, and possibly the installation of other PTF, APAR, or USERMOD SYSMODs.

Figure 5-11 on page 303 is an example of a simple USERMOD SYSMOD.

```

++USERMOD(USRM0D1)          /* SYSMOD type and identifier. */.
++VER(Z038) FMID(FUN0001)    /* Apply to this product      */.
      PRE(UZ00004)           /* at this service level.    */.
++SRCUPD(JESMOD3)           /* Update this source module */.
      DISTLIB(AOSFB)         /* in this distribution library. */.
...
... update control statements ...
...

```

*Figure 5-11 Example simple USERMOD SYSMOD MCS*

## **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 may be dependent upon other PTF SYSMODs.
- ▶ APAR SYSMODs may be dependent upon PTF SYSMODs and other APAR SYSMODs.
- ▶ USERMOD SYSMODs may 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.

## 5.8 PTF replacement

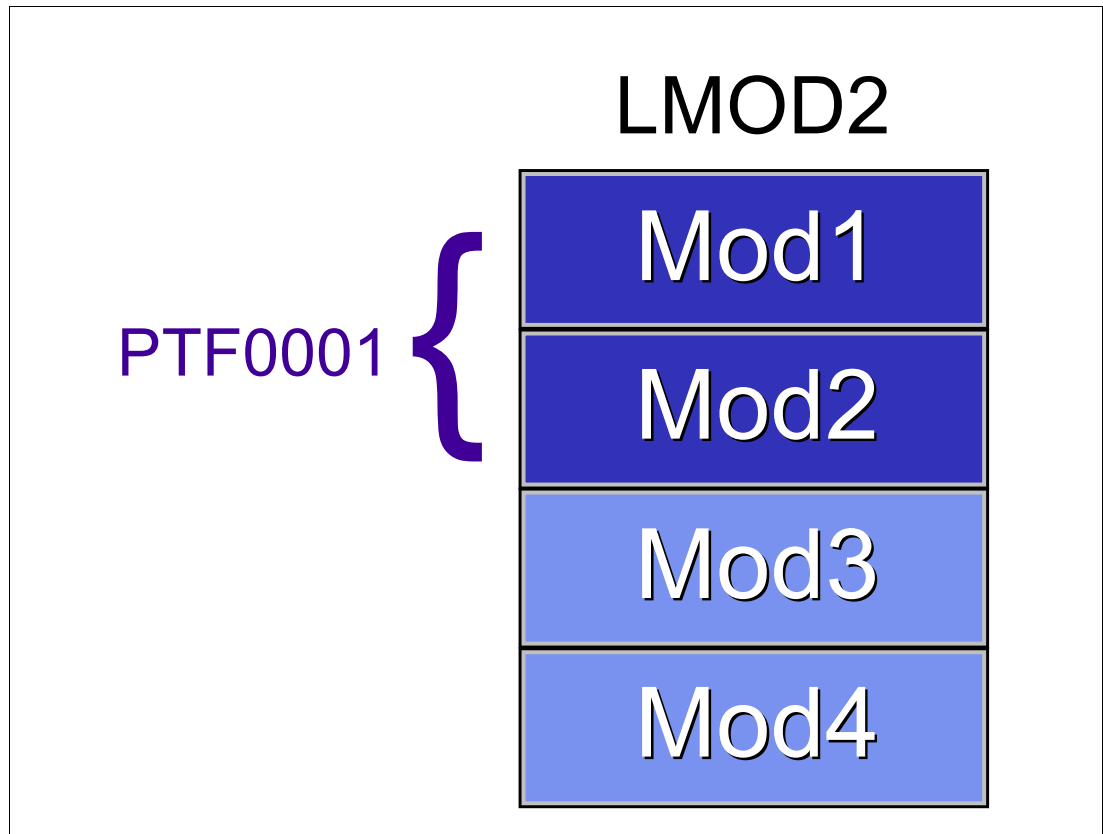


Figure 5-12 PTF replacement

The importance of keeping track of system elements and their modifications becomes readily apparent when we examine the z/OS maintenance process. Often, a PTF contains multiple element replacements. In the example in Figure 5-12, PTF0001 contains replacements for two modules, MOD1 and MOD2. Although load module LMOD2 contains four modules, only two of those modules are being replaced.

But what happens if a second PTF replaces some of the code in a module that was replaced by PTF0001? Consider the example in Figure 5-13 on page 305.



## 5.9 PTF prerequisite

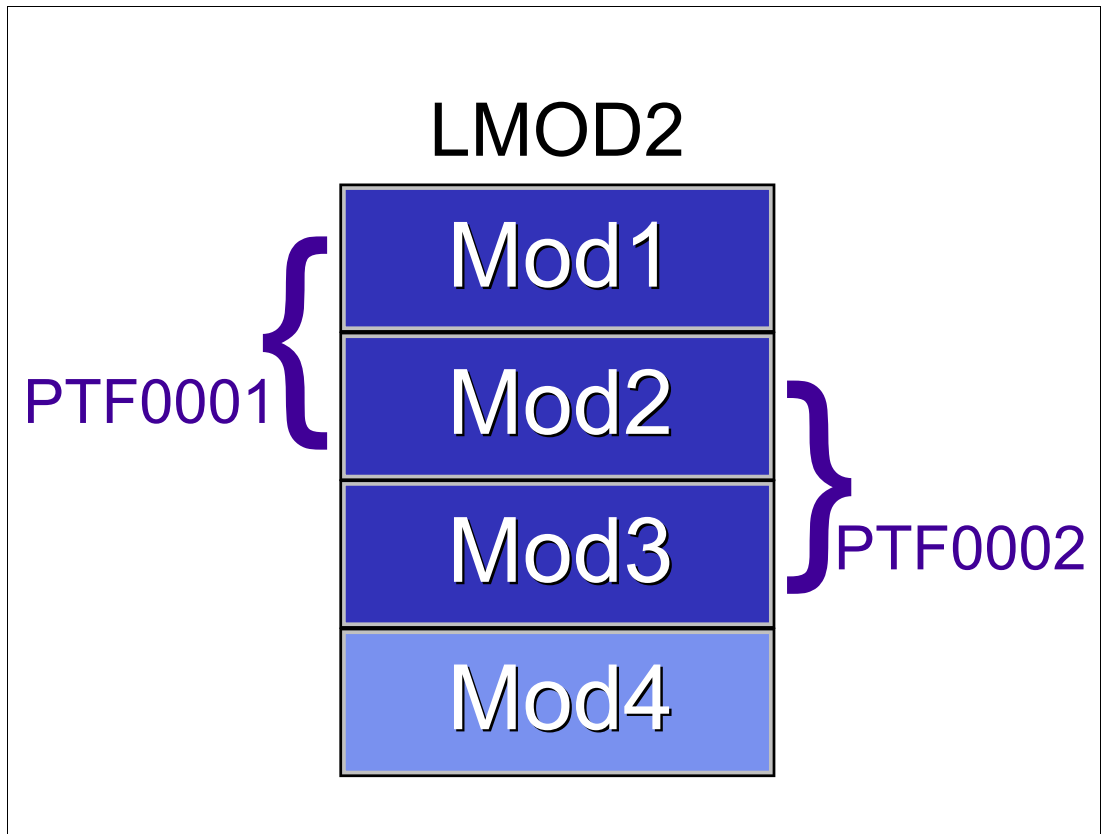


Figure 5-13 PTF prerequisite

In this example, PTF0002 contains replacements for MOD2 and MOD3. For MOD1, MOD2, and MOD3 to interface successfully, PTF0001 must be installed before PTF0002. That's because MOD3 supplied in PTF0002 may depend on the PTF0001 version of MOD1 to be present. It is this dependency that constitutes a prerequisite. SYSMOD prerequisites are identified in the modification control statements (MCS) part of the SYSMOD package.

## 5.10 Load module construction

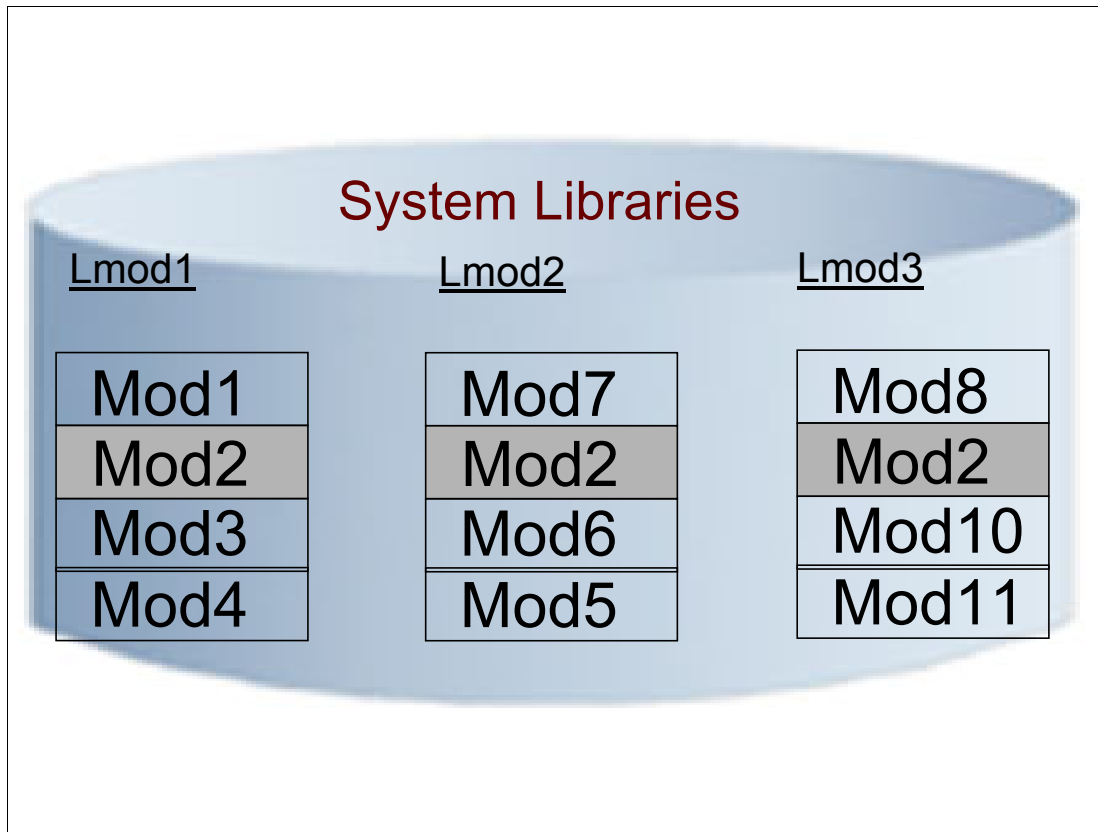


Figure 5-14 Load module construction

In Figure 5-14, the same Mod2 module is present in LMOD1, LMOD2, and LMOD3. When a PTF is introduced that replaces the element Mod2, that module must be replaced in all the load modules in which it exists. Therefore, it is imperative that we keep track of all load modules and the modules they contain.

You can now appreciate how complicated the tracking of system elements and their modification levels can become. Let's take a brief look at how we implement the tracking capabilities of SMP/E.

### Tracking and controlling prerequisites

To track and control elements successfully, all elements and their modifications and updates must be clearly identified to SMP/E. SMP/E relies on modification identifiers to accomplish this. There are three modification identifiers associated with each element:

- ▶ *Function modification identifiers (FMIDs)* that identify the function SYSMOD that introduced the element into the system.
- ▶ *Replacement modification identifiers (RMIDs)* that identify the last SYSMOD (usually a PTF SYSMOD) to replace the element.
- ▶ *Update modification identifiers (UMIDs)* that identify the SYSMODs that have updated an element since it was last replaced.

SMP/E uses these modification identifiers to track all SYSMODs installed on your system. This ensures that they are installed in the proper sequence. Now that you realize the need for

element tracking and know the types of things SMP/E tracks, let's look at how SMP/E performs its tracking function.

## 5.11 Data sets used by SMP/E

### □ Some important SMP/E data sets:

- |                    |            |
|--------------------|------------|
| ➤ SMPCSI           | ➤ SMPSCDS  |
| ➤ SMPCNTL          | ➤ SMPSTS   |
| ➤ SMPHOLD          | ➤ SMPPTFIN |
| ➤ SMPLOG / SMPLOGA | ➤ SMPRPT   |
| ➤ SMPLTS           | ➤ SMPOUT   |
| ➤ SMPMTS           | ➤ SMP SNAP |
| ➤ SMPPTS           | ➤ SYSLIB   |
| ➤ SMPPTSNN         |            |

Figure 5-15 Data sets used by SMP/E

### SMP/E data sets

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:

- ▶ *Target libraries* contain the executable code needed to run your system (for example, the libraries from which you run your production system or your test system).
- ▶ *Distribution libraries* (DLIBs) contain the master copy of each element for a system. They are used as input to the SMP/E GENERATE command or 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.

To install elements in these libraries, SMP/E uses a database made up of several types of data sets:

#### **SMPCSI (CSI)**

These data sets are VSAM databases used by SMP/E to record status and other information about the various target and distribution libraries being supported. The data set specified by the ddname SMPCSI is the CSI containing the global zone. (This data set is also known as the master CSI.) These data sets can be reorganized by IDCAMS.

#### **SMPPTS**

The SMPPTS data set (PTS) is used as a repository for SYSMODs. It contains one member for each SYSMOD that was received. Each member is called an MCS entry and is an exact copy of the SYSMOD as it was received from the SMPPTFIN data set. The name of an MCS

entry matches the SYSMOD ID of the SYSMOD it contains. The PTS is related to the global zone that contain information about the received SYSMODs. Only one PTS can be used for a given global zone. Therefore, you can look at the global zone and PTS as a pair of data sets to be processed concurrently.

|                 |                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                               |
|-----------------|-------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| <b>SMPPTSnn</b> | SMPPTS spill data sets can be used to store SYSMODs when the SMPPTS data set becomes full. This type of processing is called spill processing. SMPPTS spill data sets are used by SMP/E in the same way as the primary SMPPTS data set is used. The first SMPPTS spill data set must be specified with a ddname of SMPPTS1, the second SMPPTS2, and so on, up to a maximum of SMPPTS99. Do not skip any ddnames in this sequence; if a spill data set is omitted, SMP/E ignores any data sets that may follow the omitted data set (for example, if you specify only SMPPTS1 and SMPPTS3, then SMP/E uses only SMPPTS1 and ignores SMPPTS3). You must specify a valid data set name for an SMPPTS data set. NULLFILE and DD DUMMY are invalid for SMPPTS spill data sets. <i>This data set is available since SMP/E V3R1.</i> |
| <b>SMPSCDS</b>  | The SMPSCDS data set (SCDS) contains backup copies of target zone entries that are created during APPLY processing. The backup copies are used during RESTORE processing to return the entries to the way they were before APPLY processing. Each target zone must have its own SMPSCDS data set; that data set must be unique to that target zone. This SMPSCDS data set must also be used with the related distribution zone.                                                                                                                                                                                                                                                                                                                                                                                               |
| <b>SMPMTS</b>   | This data set is a target library for macros that exist only in a distribution library (such as macros in SYS1.AMODGEN). This data set allows the current version of these macros to be used for assemblies during APPLY processing. Therefore, the MTS is related to a specific target zone, and each target zone must have its own MTS data set. The SMPMTS data set must be in the SYSLIB concatenation for APPLY and RESTORE processing. It can be in the SYSLIB concatenation for ACCEPT processing.                                                                                                                                                                                                                                                                                                                     |
| <b>SMPSTS</b>   | The SMPSTS data set (STS) is a target library for source that exists only in a distribution library. This data set allows the current version of these modules to be used for assemblies during APPLY processing. Each target zone must have its own SMPSTS data set, which may not be shared by any other target zone. This SMPSTS data set can also be used with the related distribution zone.                                                                                                                                                                                                                                                                                                                                                                                                                             |
| <b>SMPLTS</b>   | The SMPLTS data set is used by SMP/E to save the base version of a product's load modules that use callable services. To reduce SMPLTS space requirements, SMP/E now saves a base version of a load module in the SMPLTS data set only if it contains both CALLLIBS and XZMOD subentries. If a load module contains CALLLIBS subentries, but no XZMOD subentries, this load module is not saved in the SMPLTS.                                                                                                                                                                                                                                                                                                                                                                                                                |

**Note:** Since SMP/E Version 3 Release 2 or z/OS 1.5, the size of the SMPLTS is very much reduced when compared to prior SMP/E releases. In prior releases of SMP/E or z/OS it was a very large data set.

Other data sets used by SMP/E are known as the *utility and work data sets*. Some of the important utility and work data sets are:

|                 |                                                                                                                                                                                                                                                                                                                                                                                                    |
|-----------------|----------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| <b>SMPTLIB</b>  | This data set is used as temporary storage for relative files that are loaded from SMPPTFIN during RECEIVE processing. They are deleted when the associated SYSMOD is deleted by REJECT, RESTORE, or ACCEPT processing.                                                                                                                                                                            |
| <b>SMPCTL</b>   | Contains the SMP/E commands to be processed. <i>This data set may reside in a UNIX file system since SMP/E V3R1.</i>                                                                                                                                                                                                                                                                               |
| <b>SMPNTS</b>   | The SMPNTS (SMP/E Network Temporary Store) is a directory of UNIX file system files that is used for temporary storage of network transported packages that were received during SMP/E RECEIVE processing. <i>This data set is available since SMP/E V3R1.</i>                                                                                                                                     |
| <b>SMPHOLD</b>  | Contains ++HOLD and ++RELEASE statements to be processed by the RECEIVE command. This may refer to an actual data set, or it may refer to a file on a tape (such as file 4 on an ESO or Internet data set). <i>This data set may reside in a UNIX file system since SMP/E V3R1.</i>                                                                                                                |
| <b>SMPLOG</b>   | Contains time-stamped records of SMP/E processing. The records in this data set can be written automatically by SMP/E or added by the user through the LOG command. The data set also contains messages issued by SMP/E, as well as detailed information about the data set allocation. Each zone should have its own SMPLOG data set.                                                             |
| <b>SMPLOGA</b>  | A secondary log data set (SMPLOGA) should be defined to hold log data when the SMPLOG data set is full. Otherwise, the extra log data is written to the SMPOUT data set, with the date and time stamp encrypted. Each zone should have its own SMPLOGA data set.                                                                                                                                   |
| <b>SMPPTFIN</b> | Contains SYSMODs and ++ASSIGN statements to be processed by the RECEIVE command. It can refer to an actual data set or to a file on a tape (such as file 1 on a ESO tape). <i>This data set may reside in a UNIX file system since SMP/E V3R1.</i>                                                                                                                                                 |
| <b>SMRPT</b>    | Contains the reports produced during SMP/E processing. If SMRPT is not defined, all report output goes to the SMPOUT data set. <i>This data set may reside in a UNIX file system since SMP/E V3R1.</i>                                                                                                                                                                                             |
| <b>SMPOUT</b>   | Contains messages issued during SMP/E processing, as well as dumps of the VSAM RPL, if any dumps were taken. It may also contain LIST output and reports if the SMPLIST and SMRPT data sets are not defined. <i>This data set may reside in a UNIX file system since SMP/E V3R1.</i>                                                                                                               |
| <b>SMPSNAP</b>  | The SMPSNAP data set is used for snap dump output. When a severe error occurs, such as an abend or severe VSAM return code, SMP/E requests a snap dump of its storage before doing any error recovery.                                                                                                                                                                                             |
| <b>SYSLIB</b>   | <p>The SYSLIB is a concatenation of macro libraries that are to be used by the assembler utility. For the APPLY and RESTORE processing, the data sets should be concatenated in this order:</p> <ul style="list-style-type: none"><li>▶ SMPMTS</li><li>▶ MACLIB</li><li>▶ MODGEN</li><li>▶ Target system macro libraries (such as libraries specified for SYSLIB on the ++MAC statement)</li></ul> |

- Distribution macro libraries (such as libraries specified for DISTLIB on the ++MAC statement)

**Note:** Since SMP/E Version 3 Release 2 the data sets in the SYSLIB concatenation will be allocated only if an assembly operation will be performed during the APPLY, ACCEPT, and RESTORE commands. This reduces SMP/E processing to allocate the data sets, as well as eliminating possible contention for the data sets.


## 5.12 Dynamic allocation of SMP/E data sets

### DD statements



```
//SMPMTS DD DSN=SMPE.MVST100.SMPMTS,DISP=SHR
//SMPCSI DD DSN=SMPE.GLOBAL.CSI,DISP=SHR
//SMPRPT DD SYSOUT=*
```

### DDDEF entries



```
Entry Type: DDDEF      Zone Name: MVST100
Entry Name: SMPMTS     Zone Type: TARGET

DSNAME: SMPE.MVST100.SMPMTS

VOLUME: OS3RS1  UNIT: 3390  DISP: SHR
SPACE :      PRIM:      SECOND:      DIR:
SYSOUT CLASS:      WAITFORDSN:
PROTECT:
DATACLAS:          MGMTCLAS :
STORCLAS:          DSNTYPE  :
```

Figure 5-16 Dynamic allocation of SMP/E data sets

### Dynamic allocation of SMP/E data sets

The processing of SMP/E commands requires a variety of data sets. You can either provide the 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.

### Sources of information for dynamic allocation

SMP/E can use several sources of information to allocate data sets dynamically:

- ▶ DDDEF entries
- ▶ SMPPARM member GIMDDALC
- ▶ Standard defaults

During any attempt to allocate a data set, SMP/E first looks for a DD statement specified in the job. If no DD statement was specified, SMP/E then looks for a DDDEF entry in the zone. If no DDDEF entry was found, SMP/E then checks to see if there is an SMPPARM data set. If



so, and there is a GIMDDALC member within it, SMP/E looks for a GIMDDALC control statement for the corresponding ddname.

### ***DDDEF Entries***

You can use DDDEF entries to provide SMP/E with information it needs to allocate any of the following:

- ▶ Permanent data sets, such as target libraries, distribution libraries, and SMP/E data sets
- ▶ Temporary data sets
- ▶ SYSOUT data sets
- ▶ Work data sets
- ▶ Pathnames for elements and load modules residing in a UNIX file system

The name of the DDDEF entry must match the ddname of the data set it describes and the entry must exist in the zone that uses the data set. DDDEF entries provide more flexibility than DD statements; they enable different zones to use different data sets for the same ddname and they use resources more efficiently because they allow SMP/E to allocate only the data sets it needs.

DDDEF entries can include the following information:

- ▶ Data set name
- ▶ Unit type
- ▶ Volume serial number
- ▶ Initial data set status: NEW, OLD, MOD, or SHR
- ▶ Final data set status: KEEP, DELETE, or CATALOG
- ▶ How the data set is to be allocated: blocks, cylinders, or tracks
- ▶ Primary and secondary values for space allocation
- ▶ Whether the data set should be RACF-protected
- ▶ Whether the data set is SMS-managed
- ▶ Directory information used to allocate the pathname for an element or load module residing in a UNIX file system.

### ***SMPPARM member GIMDDALC***

Another way to provide SMP/E with information about data sets is through GIMDDALC control statements in SMPPARM member GIMDDALC. It is used to specify data sets that are to be dynamically allocated by SMP/E. The following types of data sets may be specified in GIMDDALC:

- ▶ Data sets to be allocated to a SYSOUT class (or to the terminal for foreground execution)
- ▶ Data sets to be allocated as temporary data sets
- ▶ SMPTLIB data sets

**Note:** If you previously used module GIMMPDFT to define allocation information for temporary data set, you must now use member GIMDDALC in the SMPPARM data set; module GIMMPDFT no longer exists since SMP/E V3R1.

### ***Standard Defaults***

SMPOUT and SYSPRINT are critical for SMP/E to operate properly. Therefore, in case they are not defined, SMP/E has built-in defaults for them.

- ▶ SMPOUT is allocated either as SYSOUT (for background processing) or to the terminal (for foreground processing).
- ▶ SYSPRINT is allocated as SYSOUT.

## 5.13 How dynamic allocation works

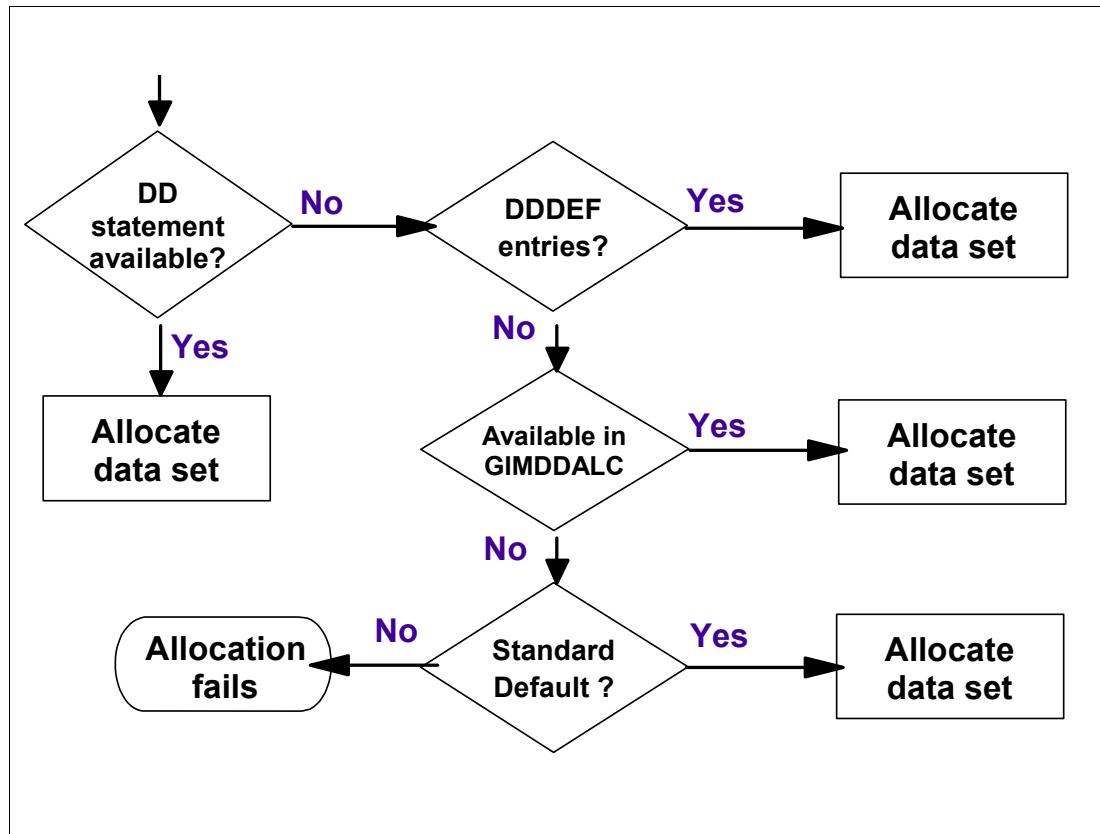


Figure 5-17 How dynamic allocation works

### How dynamic allocation works

Once SMP/E has determined which data sets are needed for the command it is processing, SMP/E checks whether DD statements have been provided for any of those data sets. SMP/E uses information from those DD statements in allocating the data sets to which they apply. If any data sets lack DD statements, SMP/E must allocate them dynamically. To get the information it needs to do this, SMP/E checks the following sources in the order shown:

- ▶ DDDEF entries. If the zone specified on the SET command contains a DDDEF entry for the required data set, SMP/E uses that entry to allocate the data set. Otherwise, it checks the next source.
- ▶ SMPPARM member GIMDDALC. If a GIMDDALC control statement has been defined in SMPPARM member GIMDDALC, SMP/E uses that information to allocate the data set. Otherwise, it checks the next source.
- ▶ Standard defaults. If the data set is for SMPOUT or SYSPRINT, SMP/E uses the standard default to allocate the data set. Otherwise, data set allocation fails.

**Note:** When a data set is part of a concatenation (such as the SYSLIB concatenation), SMP/E does not do the previously described checking. All data sets in a concatenation must be defined the same way. For example, if a DDDEF entry specifies a concatenation, all the specified entries must also be defined by DDDEF entries.

- ▶ For the cross-zone phase of APPLY and RESTORE processing, a DD statement cannot be used to allocate the SYSLIB for a cross-zone load module. This library can be allocated only through a DDDEF entry in the cross zone containing the LMOD entry for the cross-zone load module.
- ▶ The DD name SMPDUMMY is always allocated as a DUMMY data set. SMP/E ignores any allocation information specified for SMPDUMMY by a DDDEF entry or GIMDDALC member. If SMPDUMMY was previously allocated outside of SMP/E, SMP/E frees the SMPDUMMY DD and then reallocates it as a DUMMY data set.

## 5.14 Consolidated software inventory (CSI)

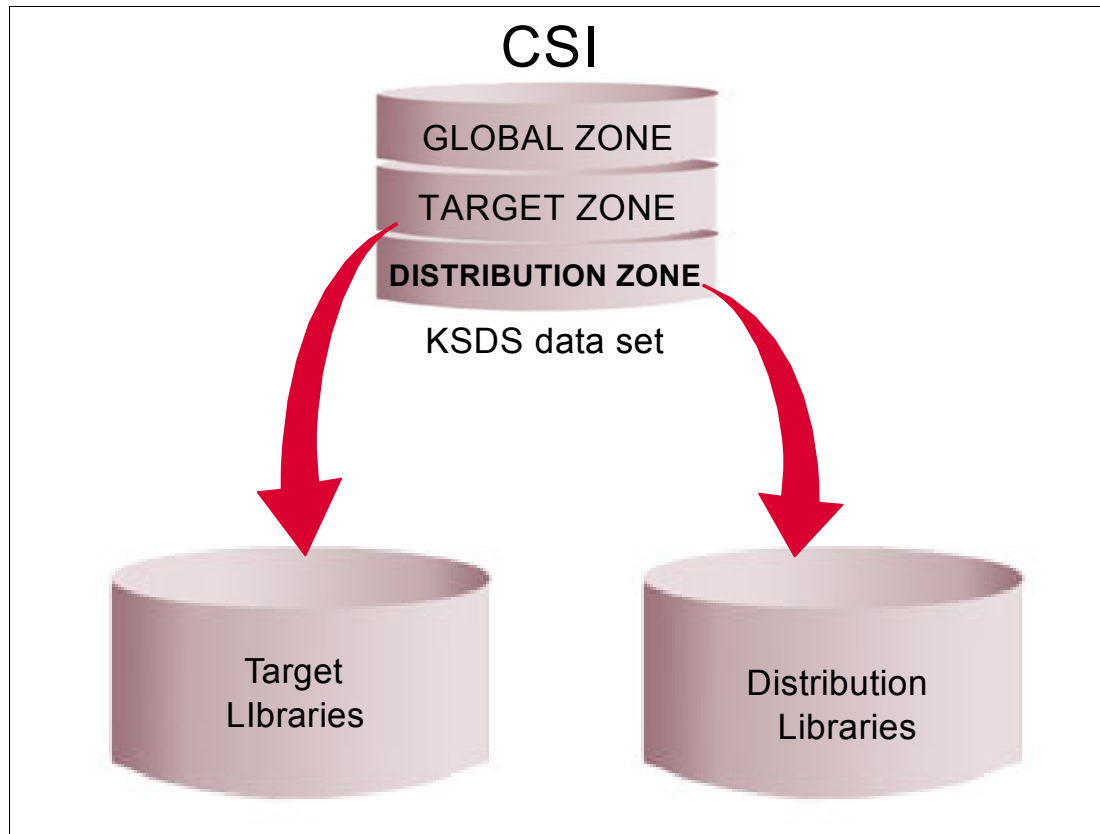


Figure 5-18 Consolidated software inventory

### Consolidated software inventory

The CSI data sets contain all the information SMP/E needs to track the target and distribution libraries. The CSI contains an entry for each element in its system, which describes the element name, type, history, how the element was introduced into the system, and a pointer to the element in the target and distribution libraries. The CSI does not contain the element itself, but rather a description of the element it represents.

### The organization of the CSI data set

In the CSI, entries for the elements in the distribution and target libraries are grouped according to their installation status. These groupings are known as SMP/E zones.

There are three types of zones in a CSI:

|                    |                                                                                                                                                                                                                                                                                                                                                                |
|--------------------|----------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| <b>Global zone</b> | Contains entries needed to identify and describe each target and distribution zone to SMP/E and stores information about SMP/E processing options. Contains status information for all SYSMODs SMP/E has begun to process and holds exception data for SYSMODs requiring special handling or that are in error. This data set is also known as the master CSI. |
| <b>Target zone</b> | Contains information that describes the content, structure, and status of the target libraries. It also contains a pointer to the related distribution zone, which can be used in APPLY, RESTORE, and LINK                                                                                                                                                     |

when SMP/E is processing a SYSMOD and needs to check the level of the elements in the distribution libraries.

**Distribution zone**

Contains information that describes the content, structure, and status of the distribution libraries. Each distribution zone also points to the related target zone, which is used when SMP/E is accepting a SYSMOD and needs to check if the SYSMOD has already been applied in the target zone.

## 5.15 How to organize CSI data sets

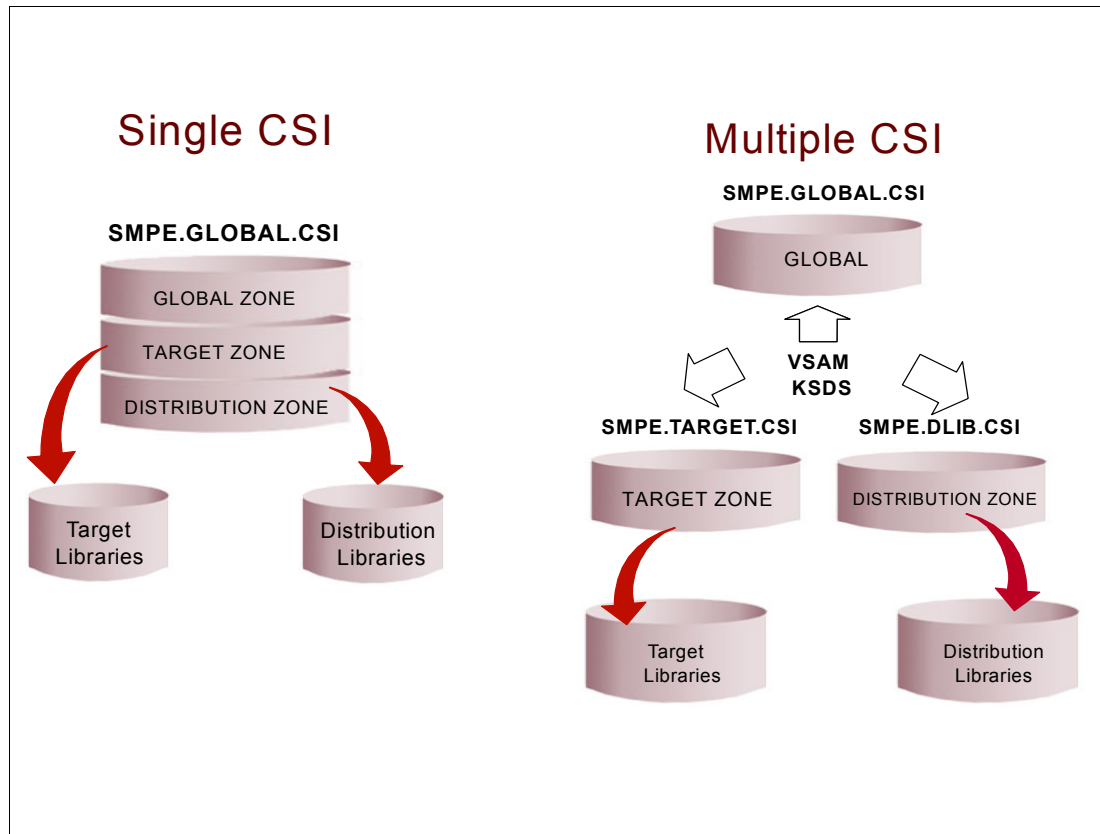


Figure 5-19 How to organize CSI data sets

### How to organize CSI data sets

Before you allocate any CSI data sets, you must decide how to organize those data sets. There are two basic structures for CSI data sets:

#### ► Single CSI

In single CSI structure, you can define one CSI to keep track of all the system activity. The single CSI data has one global zone and one or more target and distribution zones.

Among the advantages of using this structure are the following:

- The single CSI data set optimizes the use of direct access storage.
- The single CSI data set puts your whole establishment in one VSAM data set. This provides you with a single control point and one source of information for your whole system.

**Note:** Single-CSI systems do have a drawback. When SMP/E needs to process a zone, it cannot request access to that specific zone; it must request access to the CSI data set containing that zone. As a result, if you have a single-CSI system, you can run only one background SMP/E job at a time.

► Multiple CSI

Each zone resides in a separate VSAM data set. The target and distribution zones are connected by ZONEINDEX entries to a single global zone. The global zone must be in one of the CSI data sets.

## How to allocate a CSI data set

After you have decided which CSI structure to adopt for your installation, you can allocate a CSI data set using the VSAM access method services.

CSI data sets are keyed VSAM clusters and are allocated by use of access method services. The GIMSAMPU member in SYS1.SAMPLIB is a sample job to allocate and prime CSI and SMP/E operational data sets. The following sample job step, which is taken from the sample job in GIMSAMPU, allocates a CSI data set with enough space to have multiple target or distribution zones and then initializes the CSI with the zpool record.

Figure 5-20 shows sample JCL for allocating a CSI data set with enough space to have multiple target and distribution zones, as follows:

- The high-level qualifier should not be SYS1 if the CSI data set is to be cataloged in a user catalog. The low-level qualifier *must* be CSI.
- The CSI is a key-VSAM (KSDS) data set.
- SMP/E does not support cross-system sharing of the CSI; you cannot specify 4 as the cross-system value for SHAREOPTIONS.

```
//DEFZONES EXEC PGM=IDCAMS
//SYSPRINT DD SYSOUT=*
//GIMZPOOL DD DSN=SYS1.MACLIB(GIMZPOOL),DISP=SHR
//SYSIN DD *
    DEFINE CLUSTER( +
        NAME(SMPE.GLOBAL.CSI) +
        VOLUMES(vol1id) +
        CYLINDERS(100 10) +
        FREESPACE(10 5) +
        KEYS(24 0) +
        RECORDSIZE(24 143) +
        SHAREOPTIONS(2 3) +
        ) +
    DATA ( +
        NAME(SMPE.GLOBAL.CSI.DATA) +
        CONTROLINTERVALSIZE(8192) +
        ) +
    INDEX (NAME(SMPE.GLOBAL.CSI.INDEX) +
        CONTROLINTERVALSIZE(4096) +
        )
    REPRO INFILE(GIMZPOOL) +
        OUTDATASET(SMPE.GLOBAL.CSI)
/*
```

Figure 5-20 Sample JCL for allocating a CSI data set

## 5.16 Defining zones for your system

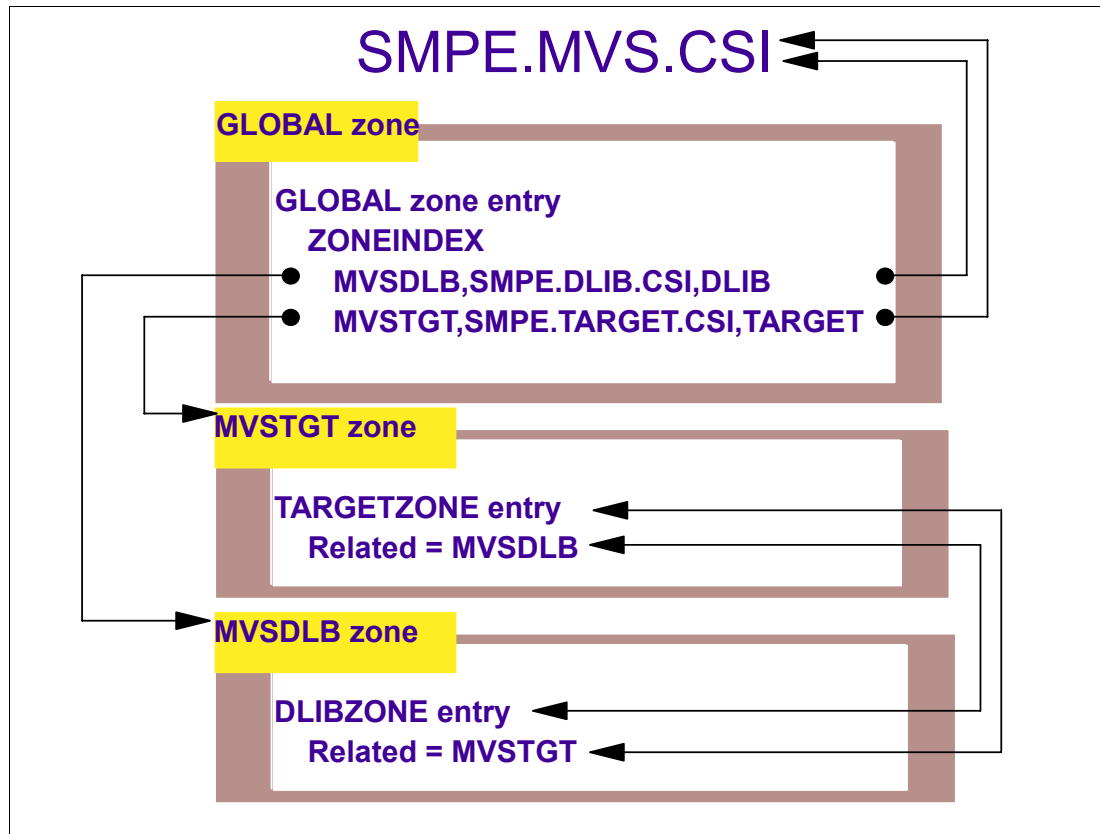


Figure 5-21 Defining zones for your system

### Defining zones for your system

Once you have allocated and initialized the CSI data sets, you need to create within them the entries SMP/E uses to maintain your system. The first entries you need to define are the zone definition entries:

**GLOBALZONE entry** A global zone is created by defining a GLOBALZONE entry. The GLOBALZONE entry contains processing related information for SMP/E. It is also used by SMP/E as an index to target and distribution zones, either in the same CSI or in different CSI data sets. The GLOBALZONE entry must be defined before you can do other processing for that global zone.

**TARGETZONE entry** A target zone is created by defining a TARGETZONE entry. The TARGETZONE entry contains information SMP/E uses to process a specific target zone and the associated target libraries. It must be defined before you can do any other processing for that target zone.

**DLIBZONE entry** A distribution zone is created by defining a DLIBZONE entry. The DLIBZONE entry contains the information SMP/E uses to process a specific distribution zone and the associated distribution libraries. It must be defined before you can do any other processing for that distribution zone.



After you have defined the zones for your system, you can create other entries. SMP/E zones contain two basic types of entries:

- ▶ Entries controlling SMP/E processing

You generally define processing control entries through the SMP/E administration dialogs or with the UCLIN command.

- ▶ Entries describing the structure and status of the target and distribution libraries

Status and structure entries are generally created by SMP/E when you install SYSMODs, run the JCLIN command, or copy entries from one zone to another.

**Note:** SMP/E provides a member in SYS1.SAMPLIB (GIMSAMPU) containing sample UCLIN statements to define entries for a basic z/OS system. You can access this member by use of standard system utilities. The sample definitions are syntactically correct and can be used as the basis for your CSI entries. This sample is not complete for all systems, but it is an example of the types of information various entries need. For examples of UCLIN to define entries, see the UCLIN command in *SMP/E Commands*, SA22-7771, which shows the UCLIN syntax for each entry type, and SMP/E Data Set Entries in *SMP/E Reference*, which contains a description of the syntax plus examples and notes on its use.

## 5.17 SMP/E commands you need to know

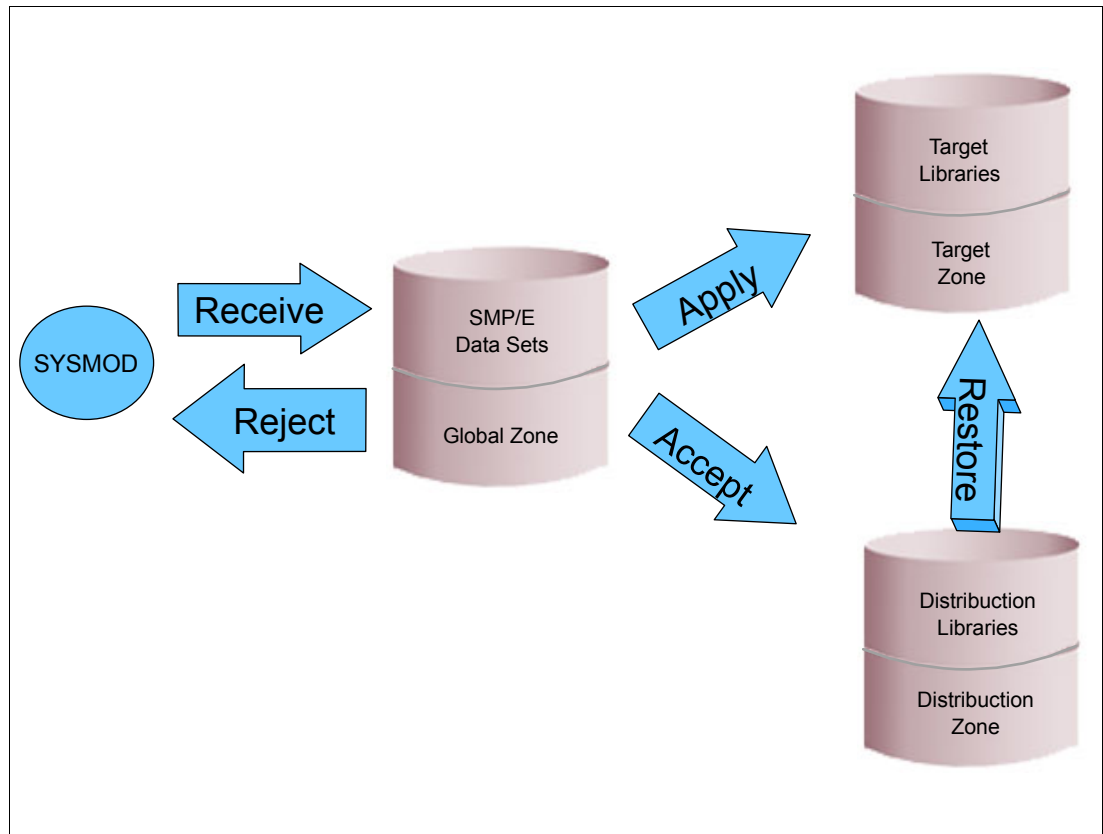


Figure 5-22 SMP/E commands you need to know

### SMP/E commands you need to know

Now that you are familiar with SMP/E and what it can do, you are probably wondering what you need to know to get started using SMP/E. Let's take a look at the basic processing commands you need to know to use SMP/E. For more detailed information about these commands, see *SMP/E Commands*, SA22-7771.

### Setting the zone you want to work on

Before processing SMP/E commands, you must first set the zone on which you want SMP/E to work (global, target, or distribution). You do this by issuing the SET BOUNDARY command. This command identifies the zone and, therefore, the libraries, upon which subsequent SMP/E commands are to act.

The SET BOUNDARY command can also be used to request a particular set of predefined processing options.

### Receiving the SYSMOD

For SMP/E to install a SYSMOD (Function, PTF, APAR or USERMOD), the SYSMOD must be *received* into data sets that can be used by SMP/E. The RECEIVE command performs the task of copying the SYSMOD from the distribution medium to data sets used by SMP/E (SMPPTS, SMPTLIB, and global zone within the CSI). For the RECEIVE command, the SET BOUNDARY command must specify the global zone.

**Note:** The RECEIVE command, in SMP/E Version 3 Release 3 (z/OS 1.6), has been enhanced to assign the source ID specified on the SOURCEID operand of the command to SYSMODs found in the SMPPTFIN input stream, even if they are already received. Formerly, the source ID was not assigned to SYSMODs that are already received.

## Rejecting SYSMODs

The REJECT command allows you to clean up the global zone, SMPPTS, and associated entries and data sets. REJECT 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. For the REJECT command, the SET BOUNDARY command must specify the global zone.

**Note:** Since SMP/E Version 3 Release 3 (z/OS 1.6), the CHECK operand is now allowed on the REJECT command. The CHECK operand indicates whether REJECT should perform a trial run of the command without actually updating any zones or data sets. This provides a way to test for errors that might occur during actual processing and to receive reports on the changes that would be made.

## Applying the SYSMOD

The APPLY command is used to cause SMP/E to install the elements supplied by a SYSMOD into the operating (or target) system libraries. For the APPLY command, the SET BOUNDARY command must specify the target zone associated with the target libraries in which the SYSMODs are installed.

## Restoring the target libraries to the previous level

At times, you may want to remove a SYSMOD that has been applied to the target libraries. For example, perhaps you installed a fix for a problem and 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. For the RESTORE command, the SET BOUNDARY command must specify the target zone from which the SYSMODs should be removed. SMP/E uses the RELATED field in the TARGETZONE entry to determine which distribution zone describes the elements to replace the restored elements.

## Accepting the SYSMOD

After you have performed a SYSMOD RECEIVE and APPLY, you need to *accept* the elements into the distribution libraries for backup. However, this should be done only after you are satisfied with the performance and stability of the elements of the SYSMOD. Once 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. Once you have accepted a SYSMOD into the distribution libraries, you cannot use RESTORE to remove it from the target libraries. For the ACCEPT command, the SET BOUNDARY command must specify the distribution zone associated with the distribution libraries where the SYSMODs will be installed.

## Displaying SMP/E data

The SMP/E CSI and other primary data sets contain a great deal of information you may find useful when installing new elements or functions, preparing user modifications, or debugging problems. There are several ways SMP/E allows you to display that information, as well as information about modules, macros, and other elements:

- LIST commands

- ▶ REPORT commands
- ▶ QUERY dialogs

**Note:** Since SMP/E Version 3 Release 3 (z/OS 1.6), the CSI QUERY dialog allows a wildcard (pattern) for the entry name specification. A selection list of all entry names that match the specified pattern will be displayed when using a wildcard.

## 5.18 The RECEIVE process

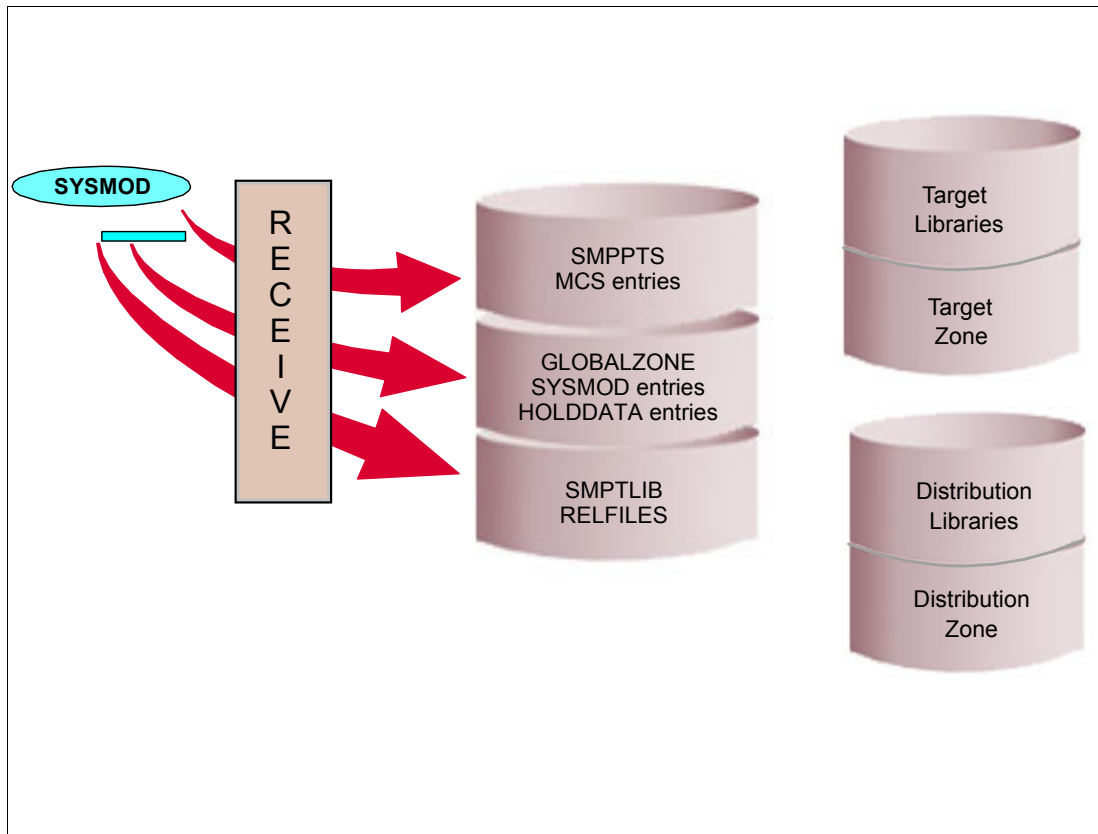


Figure 5-23 The RECEIVE process

### The RECEIVE process

During receive processing, SMP/E reads data from tape files, DASD data sets, or files in a UNIX file system into the global zone, the SMPPTS, and temporary data (SMPTLIBs) for later processing.

The RECEIVE command processes data from the following sources:

- ▶ The SMPPTFIN data set is a sequential data set with LRECL=80, BLKSIZE=multiple of 80, and RECFM=FB, which contains the modification control statements (MCSs) defining the SYSMODs, as well as any related ++ASSIGN, ++FEATURE, and ++PRODUCT statements.

If you want to receive from multiple product tapes, you cannot concatenate them on a single SMPPTFIN DD statement. Instead, you must process each tape in a separate step, using separate SMPPTFIN DD statements. Refer to the documentation supplied with the tape for exact information on how to code the SMPPTFIN DD statement. For example, with a product tape, the program directory contains this information.

This data set may reside in the UNIX file system. Specify PATHOPTS(ORDONLY) and either FILEDATA=BINARY or FILEDATA=TEXT on the DD statement for this data set, if it is to reside in the HFS. If you specify FILEDATA=BINARY, each line of input must be padded with blanks to the 80-byte record length because there is no end of record marker for FILEDATA=BINARY. If you specify FILEDATA=TEXT, use the newline character (X'15') to mark the end of each line.

- ▶ The SMPHOLD data set is a sequential data set with LRECL=80, BLKSIZE=multiple of 80, and RECFM=FB, which contains exception SYSMOD data (++HOLD and ++RELEASE statements).

This data set may reside in the UNIX file system. Specify PATHOPTS(ORDONLY) and either FILEDATA=BINARY or FILEDATA=TEXT on the DD statement for this data set if it is to reside in the HFS. If you specify FILEDATA=BINARY, each line of input must be padded with blanks to the 80-byte record length because there is no end of record marker for FILEDATA=BINARY. If you specify FILEDATA=TEXT, use the newline character (X'15') to mark the end of each line.

- ▶ A package on a TCP/IP connected FTP server.
- ▶ The SMPNTS directory.

When receiving a SYSMOD, SMP/E creates two entries:

- ▶ An MCS entry is created on the SMPPTS. This entry is an exact copy of the SYSMOD as it appeared in the SMPPTFIN data set.
- ▶ A SYSMOD entry is created in the global zone. This entry contains information that describes the installation requirements and element content of the SYSMOD.

When receiving the HOLDDATA, SMP/E also creates (or modifies) two entries:

- ▶ A HOLDDATA entry is created (or modified) in the global zone. This entry is an exact copy of the ++HOLD statements as they appeared in the SMPHOLD data set. The name of the entry is the ID of the SYSMOD affected by this ++HOLD statement. The HOLDDATA entry for a single SYSMOD can contain multiple ++HOLD statements.

**Note:** When a ++RELEASE statement is processed, SMP/E removes the corresponding ++HOLD statement from the HOLDDATA entry. When all ++HOLD statements are removed, the HOLDDATA entry is automatically deleted.

- ▶ A SYSMOD entry is created (or modified) in the global zone. This entry contains information that describes the exception SYSMOD conditions.
- ▶ For each ++HOLD statement processed, SMP/E updates the global zone SYSMOD entry to add a HOLD reason ID subentry. There are three types of HOLD reason ID subentries, HOLDERROR, HOLDSYSTEM, and HOLDUSER, corresponding to the three categories of exception SYSMODs.

**Note:** When a ++RELEASE statement is processed, SMP/E removes the corresponding reason ID from the global zone SYSMOD entry.

## Managing exception SYSMOD through 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 may be required before SYSMOD installation (SYSTEM HOLD).
- ▶ The user may want to perform some actions before installing the SYSMOD (USER HOLD).

## 5.19 Sources of HOLDDATA

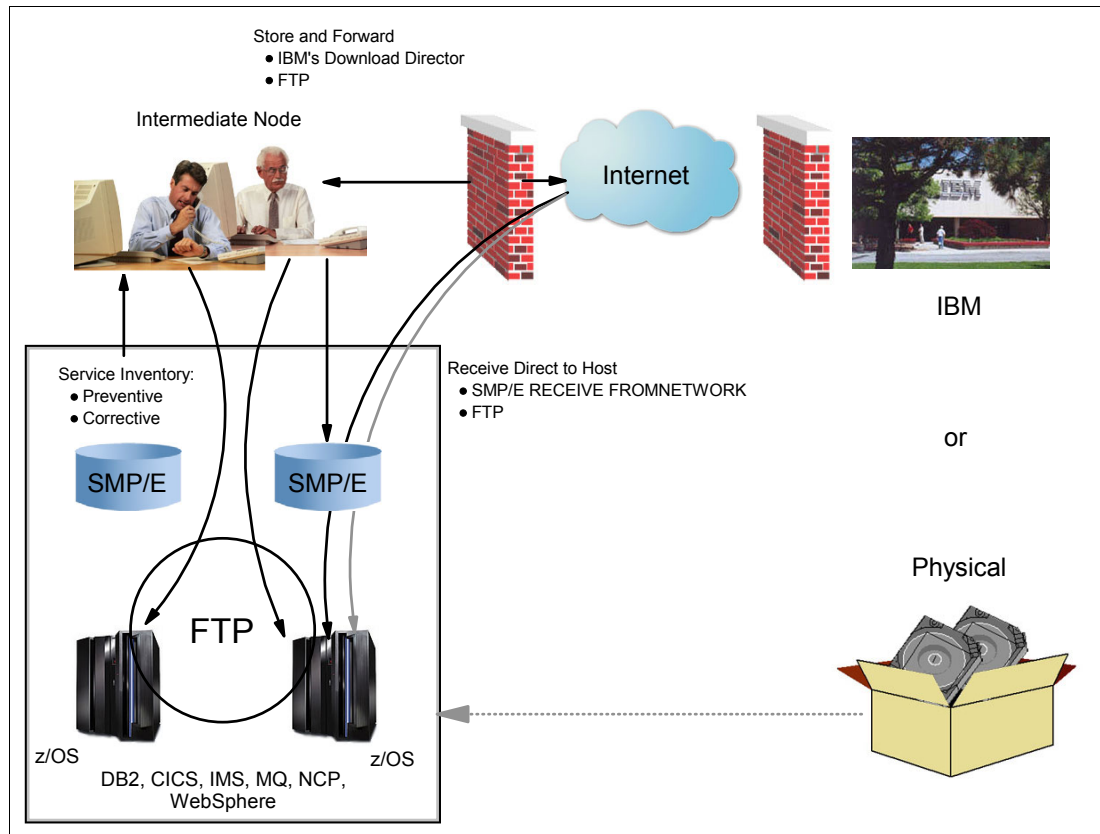


Figure 5-24 Sources of HOLDDATA

### Sources of HOLDDATA

The main sources of HOLDDATA provided by IBM are:

► Custom Build Product Delivery Option (CBPDO) tapes

This contains HOLDDATA that has been customized to your product set. That is, it contains only data applicable to PTFs for those products within a given feature that you have ordered.

► Expanded Service Option (ESO) tapes

IBM regularly creates the service levels shipped on these tapes, then custom-builds ESOs for users and makes the tapes available through either subscription orders or special request orders.

► Preventive service planning (PSP)

Once a service level has been created, there is no further opportunity to change the HOLDDATA on that tape, even though new errors are reported. PSP files have been set up to hold this additional HOLDDATA.

► IBM Web pages service

**Note:** Figure 5-24 shows the complete process of a file transfer RECEIVE, not only HOLDDATA. The IBM Web pages service is shown in the left part of the figure, which also shows the back and forth dialogues between IBM and the customer commonly associated with these services.

## 5.20 Reports for RECEIVE processing

- ❑ **Summary Report**
  - listing with all SYSMODs processed
- ❑ **Exception SYSMOD Data Report**
  - quick summary of HOLDDATA information
- ❑ **File Allocation Report**
  - listing of all data sets used
- ❑ **Receive Product Summary Report**
  - summarize for ++FEATURE and ++PRODUCT MCS

Figure 5-25 RECEIVE processing reports

When RECEIVE processing is complete, the following reports will help you to analyze the results:

- ▶ **Summary Report** provides you with a listing with all the SYSMODs that were processed during the RECEIVE command run. It shows you which SYSMODs were received, which were not received, and why they were not received.

An example of a “RECEIVE Summary Report” is shown in Figure 5-26.

|                                                                      |              |                  |                     |                              |          |
|----------------------------------------------------------------------|--------------|------------------|---------------------|------------------------------|----------|
| PAGE 0001 - NOW SET TO GLOBAL ZONE DATE 04/16/99 TIME 13:37:0 GIMSMP |              |                  |                     |                              |          |
| RECEIVE SYSMODS.                                                     |              |                  |                     |                              |          |
| PAGE 0002 - NOW SET TO GLOBAL ZONE DATE 04/16/99 TIME 14:08:0 GIMSMP |              |                  |                     |                              |          |
| RECEIVE SUMMARY REPORT                                               |              |                  |                     |                              |          |
| SYSMOD                                                               | STATUS       | TYPE             | SOURCEID            | STATUS FIELD                 | COMMENTS |
| UQ01266                                                              | NOT RECEIVED | PTF <sup>1</sup> | NO APPLICABLE ++VER |                              |          |
|                                                                      | NOT ASSIGNED |                  | PUT9710             | SYSMOD NOT IN RECEIVE STATUS |          |
| UQ09543                                                              | NOT RECEIVED | PTF <sup>2</sup> | ALREADY RECEIVED    |                              |          |
|                                                                      | ASSIGNED     |                  | PUT9710             |                              |          |
| UQ09340                                                              | RECEIVED     | PTF <sup>3</sup> |                     |                              |          |
|                                                                      | ASSIGNED     |                  | PUT9710             |                              |          |

Figure 5-26 Sample RECEIVE Summary Report



Figure 5-26 notes:

- 1 - PTF UQ01266 was not received because there is no applicable subsystem or system release (SREL) defined in the GLOBALZONE that matches the ++VER statement.
  - 2 - PTF UQ09543 was not received because it has already been received before.
  - 3 - UQ09340 was successfully received.
- **Exception SYSMOD Data Report** provides you with a quick summary of the HOLDDATA information processed during the RECEIVE command run. It lists the SYSMODs requiring special handling or that are in error, and those SYSMODs no longer requiring special handling or that have had an error fixed.

An example of an Exception SYSMOD Data Report is shown in Figure 5-27.

```

PAGE 0106 - NOW SET TO GLOBAL ZONE  DATE 04/16/99  TIME 14:08:02  GIMSMP

                RECEIVE ++HOLD/++RELEASE SUMMARY

NOTE:  SMD NF   - SYSMOD NOT RELEASED - NOT FOUND IN THE GLOBAL ZONE
       RSN NF   - SYSMOD NOT RELEASED - NOT HELD FOR THIS REASONID
       INT HLD  - SYSMOD NOT RELEASED - CANNOT RELEASE INTERNAL SYS HOLD

SYSMOD  TYPE STATUS REASON FMID      ++HOLD MCS STATEMENTS
UQ09660 SYS  HELD   ACTION HDB4410  ++ HOLD(UQ09660) SYS FMID(HDB4410)
1++ HOLD(UQ09660) SYS FMID(HDB4410)
      COMMENT
      (**Action for PN04918)

```

Figure 5-27 Sample RECEIVE Exception Data Report

Figure 5-27 note:

- 1 - UQ09660 is an exception SYSMOD. Normally, for exception SYSMOD with the reason ID as ACTION, you have to perform certain actions before you can install the SYSMOD. For example, you may have to IPL the system for the fix to take effect.
- **File Allocation Report** provides you with a list of data sets used for RECEIVE processing and supplies information about these data sets.

An example of a File Allocation Report is shown in Figure 5-28.

```

PAGE 0106 - NOW SET TO GLOBAL ZONE  DATE 04/16/99  TIME 14:08:02  GIMSMP
                SMP RECEIVE  FILE ALLOCATION REPORT

ZONE DDNAME DDDEFNAM SMPDDNAM TYPE  -----DATA SET OR PATH-----

      SMPCTL          PERM  IBMSSAA.SMPE.MOF2.OS390.JCL
      SMPCSI          PERM  DBAP.DB2V410.SMPG.CSI
      SMPLOG          SYSIO  IBMSSAA.IBMSSAA1.JOB29770.D0000129
      SMPLOGA         SYSIO  IBMSSAA.IBMSSAA1.JOB29770.D0000130
      SMPOUT          SYSIO  IBMSSAA.IBMSSAA1.JOB29770.D0000127
      SMPPTFIN        PERM  SMPPTFIN
      SMPPTS SMPPTS    1PERM  DBAP.DB2V410.SMPPTS
      SMPRPT          SYSIO  IBMSSAA.IBMSSAA1.JOB29770.D0000128
      SYSUT1 SYSUT1    PERM  SYS99106.T133706.RA000.IBMSSAA1
      SYSUT2 SYSUT2    PERM  SYS99106.T133706.RA000.IBMSSAA1
      SYSUT3 SYSUT3    PERM  SYS99106.T133706.RA000.IBMSSAA1

```

Figure 5-28 Sample RECEIVE File Allocation Report

Figure 5-28 note:

1 - This shows that SMP/E dynamically allocates the SMPPTS data set through DDDEF.

- **Receive Product Summary Report** is produced at the completion of RECEIVE processing to summarize the processing that occurred for ++FEATURE and ++PRODUCT MCS.

**Note:** If no ++FEATURE and ++PRODUCT MCS are processed, this report is not generated.

An example of a RECEIVE Product Summary Report is shown in Figure 5-29.

```
++PRODUCT(5647-A01,2.5.0) DESCRIPTION(OS/390)
  SREL(Z038).
++FEATURE(OS3250BA) DESCRIPTION(OS/390 Base)
  PRODUCT(5647-A01,2.5.0)
  FMID(HBB6605,HMP1B00).
++FEATURE(OS3250DD) DESCRIPTION(OpenEdition DCE User Privacy DES)
  PRODUCT(5647-A01,2.5.0)
  FMID(JMB3125).
++FEATURE(OS3250LD) DESCRIPTION(Language Environment Decryption)
  PRODUCT(5647-A01,2.5.0)
  FMID(JMWL755).
```

Figure 5-29 Sample RECEIVE Product Summary Report

## Summary

When the RECEIVE command is used to load a SYSMOD into SMP/Es storage area, it does the following:

- Copies the MCS for each SYSMOD to the SMPPTS data set
- Loads elements into SMPTLIB data sets for SYSMODs using the relative file packaging method
- Records what is received in the global zone
  - SYSMOD entries
  - HOLDDATA entries
- Reports the results of processing

## 5.21 RECEIVE examples

- ☐ Receiving only HOLDDATA
- ☐ Receiving only SYSMOD
- ☐ Receiving both SYSMOD and HOLDDATA
- ☐ Receiving selected SYSMOD and HOLDDATA

Figure 5-30 RECEIVE examples

### RECEIVE examples

The following examples are provided to help you use the RECEIVE command.

#### How to receive only HOLDDATA

There may be times when you do not want to receive the SYSMODs from a service tape, but you do want to receive the HOLDDATA. Because the HOLDDATA provides information about SYSMODs requiring special handling or that are in error, it is important for you to receive the HOLDDATA into the SMP/E storage repository as soon as possible. Figure 5-31 shows sample JCL to process only the HOLDDATA.

```
//RECHOLD JOB (),'MVSSP',NOTIFY=&SYSUID,CLASS=A,MSGLEVEL=(1,1)
//SMPE EXEC PGM=GIMSMP1
//SMPCSI DD DSN=SMPE.GLOBAL.CSI,DISP=SHR
//SMPHOLD DD DISP=SHR,DSN=SYS1.OS251140.HOLDDATA,2
// VOL=SER=S1140C,LABEL=(1,SL),UNIT=3480
//SMPRPT DD SYSOUT=*
//SMPDOUT DD SYSOUT=*
//SYSPRINT DD SYSOUT=*
//SMPCNTL DD *
SET BDY (GLOBAL) .3
RECEIVE HOLDDATA .4
```

Figure 5-31 Sample JCL to process only HOLDDATA

Figure 5-31 notes:

- 1 - This is the main SMP/E program.
- 2 - In the SMPHOLD DD statement, you specify the data set that contains the HOLDDATA. In this case, the HOLDDATA comes with the CBPDO tape.
- 3 - For the RECEIVE command, the SET BOUNDARY command must specify the global zone.
- 4 - The HOLDDATA parameter indicates that the applicable data form SMPHOLD DD should be received. In this case, HOLDDATA is received for all FMIDs defined in the GLOBAL zone.

**Note:** To indicate the ends of the commands, for each SMPE zone, a dot must be used. For more information, see the SMP/E Commands manual.

You must provide SMP/E with the most current HOLDDATA possible to get the most benefit from this support. It is usually available on IBM service Web pages, together with Recommended Service Upgrade (RSU).

Recommended Service Upgrade (RSU) is a preventive service philosophy that applies to z/OS. RSU is intended to reduce the volume of PTFs customers must apply for preventive maintenance and to reduce the chance of encountering a PTF in error (PE), resulting in a more stable system.

We recommend that customers APPLY all RSU PTFs as preventive maintenance on their z/OS systems. However, customers must make the final decision as to what maintenance they will install.

For information about the latest recommended level of service, see the CST and RSU Web site:

<http://www.ibm.com/servers/eserver/zseries/zos/servicetst/>

## How to receive only SYSMODs

This example assumes that you have previously received the HOLDDATA from a service tape and are now ready to install the SYSMODs. Before you can install these SYSMODs (using the SMP/E APPLY and ACCEPT commands), you must first receive them. Figure 5-32 shows sample JCL to process only the SYSMODs.

```
//RECSYSM JOB (), 'MVSSP', NOTIFY=&SYSUID, CLASS=A, MSGLEVEL=(1,1),
//          MSGCLASS=X
//SMPE      EXEC PGM=GIMSMP
//SMPPTFIN DD DISP=SHR, DSN=SYS1.OS251140.PTF, 1
//          VOL=SER=S1140C,
//          LABEL=(2,SL), UNIT=3480
//SMPCTL DD *
//          SET BDY (GLOBAL) .
//          RECEIVE SYSMODS SOURCEID(MVSPUT1) .2
```

Figure 5-32 Sample JCL to process only the SYSMODs

Figure 5-32 notes:

- 1 - SMPPTFIN DD statement indicates the data sets which contains MCSs defining the SYSMODs to be received.

2 - SYSMODS parameter indicates that only the data from SMPPTFIN should be received. SOURCEID specifies a one-to-seven character source identifier to be assigned to the SYSMODs being received. SMP/E assigns this source ID to all the SYSMODs processed by this RECEIVE command.

## How to receive both SYSMOD and HOLDDATA

In the course of maintaining your system, you need to install both the SYSMODs and process the related HOLDDATA. You can accomplish this by using the sample JCL shown in Figure 5-33.

```
//RECPTF1 JOB (), 'MVSSP', NOTIFY=&SYSUID, CLASS=A, MSGLEVEL=(1,1),  
//          MSGCLASS=X  
//SMPE     EXEC PGM=GIMSMP  
//SMPPTFIN DD DISP=SHR, DSN=SYS1.OS251140.PTF,  
//          VOL=SER=S1140C,  
//          LABEL=(2,SL), UNIT=3480  
//SMPHOLD  DD DISP=SHR, DSN=SYS1.OS251140.HOLDDATA,  
//          VOL=SER=MPWRK1  
//SMPCNTL  DD *  
//          SET BDY (GLOBAL) .  
//          RECEIVE .1
```

Figure 5-33 Sample JCL to receive both the SYSMODs and HOLDDATA

Figure 5-33 note:

1 - This will receive all the SYSMODs and the related HOLDDATA for all the FMIDs that were defined in the global zone.

## How to receive selected SYSMODs and HOLDDATA

The SMP/E RECEIVE command also allows you to select individual SYSMODs or exception HOLDDATA applicable to the selected SYSMODs. In the example shown in Figure 5-34, the commands caused SMP/E to receive two SYSMODs specified plus any HOLDDATA entries that are applicable to the SYSMODs.

```
//RECPTF2 JOB (), 'MVSSP', NOTIFY=&SYSUID, CLASS=A, MSGLEVEL=(1,1),  
//          MSGCLASS=X  
//SMPE     EXEC PGM=GIMSMP  
//SMPPTFIN DD DISP=SHR, DSN=SYS1.OS251140.PTF,  
//          VOL=SER=S1140C,  
//          LABEL=(2,SL), UNIT=3480  
//SMPHOLD  DD DISP=SHR, DSN=SYS1.OS251140.HOLDDATA,  
//          VOL=SER=MPWRK1  
//SMPCNTL  DD *  
//          SET BDY (GLOBAL) .  
//          RECEIVE S(UW41970,UW41979)1  
//          SYSMODS  
//          HOLDDATA  
//          SOURCEID(MVSPUT1) . /*
```

Figure 5-34 Sample JCL to receive selected SYSMODs and HOLDDATA

Figure 5-34 note:

1 - Only two SYSMODs and applicable HOLDDATA were received using this command.

## 5.22 The REJECT process

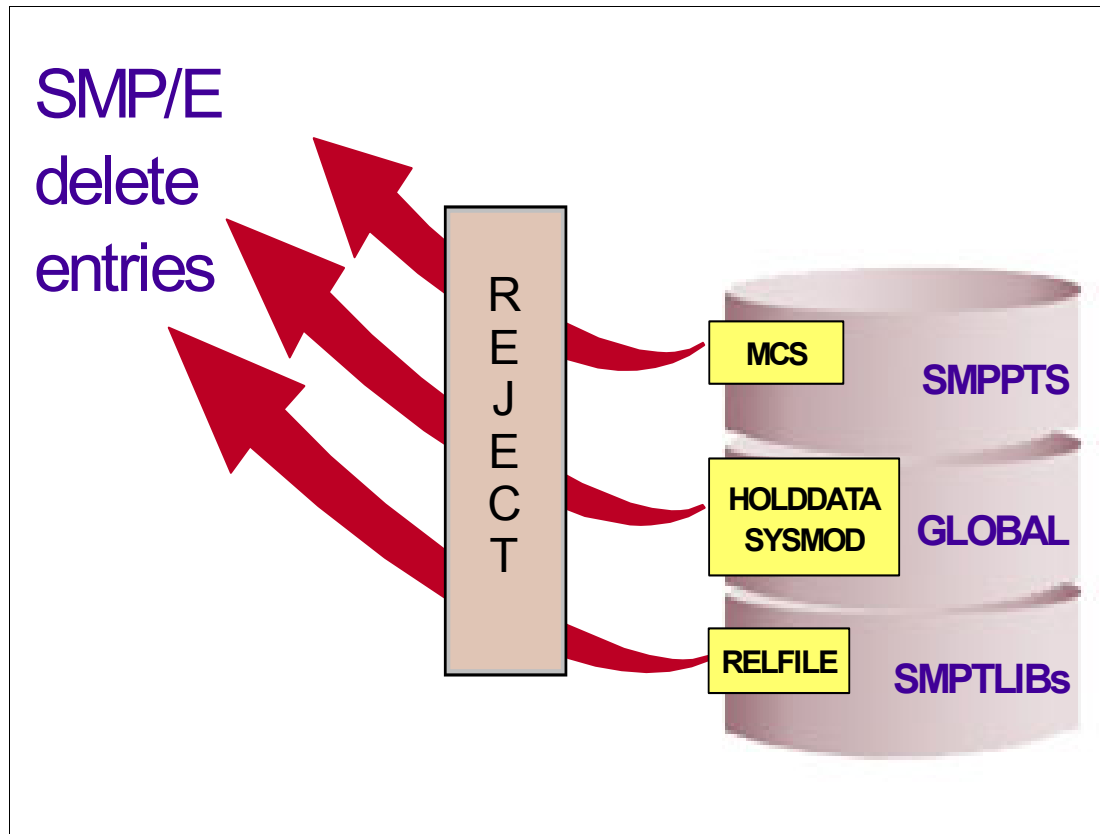


Figure 5-35 Rejecting SYSMODs

### Rejecting SYSMODs

The SMP/E REJECT command allows you to clean up the global zone, SMPPTS, and associated entries and data sets. Since SMP/E Version 3 Release 3 or z/OS v1R6 you can use the CHECK operand on the RECEIVE command to test for errors that might occur during actual processing of it. The REJECT processing deletes a SYSMOD's global zone entry, HOLDDATA entry (if any), PTF MCS entry, and also scratches any TLIB data sets allocated for the SYSMOD. REJECT is helpful if the SMPPTS is being used as a permanent database for all SYSMODs, including those that have been installed. You can also use it to purge old data from the global zone and SMPPTS. There are times when you need to reverse the process of receiving SYSMODs into your system. After receiving a SYSMOD, you can use the SMP/E REJECT command to remove that SYSMOD from the PTS and the global zone.

### Processing modes of the REJECT command

To use the REJECT command, you must first determine which processing mode of the command you want to use. The mode you choose depends on the data you want to delete.

**Note:** The processing modes are mutually exclusive. No two modes can be used together.

There are five modes of REJECT processing:

- CHECK mode

Since SMP/E Version 3 Release 3 or z/OS v1R6 you can use the CHECK operand on the RECEIVE command to test for errors that might occur during actual processing of it.

- ▶ **MASS mode**  
SMP/E rejects all SYSMODs that have been received but not installed.
- ▶ **SELECT mode**  
SMP/E rejects specific SYSMODs that have been received but not applied.
- ▶ **PURGE mode**  
SMP/E rejects all SYSMODs that have been accepted into the specified distribution zones. This is used when SYSMODs are not automatically deleted once they have been accepted into the distribution libraries. This is true when NOPURGE is coded in the OPTIONS entry used to process the distribution zone.
- ▶ **NOFMID mode**  
SMP/E rejects all SYSMODs applicable to functions that are not part of the system. This can be used to delete service for all functions that have been deleted from the global zone. It can also be used to delete FEATURE and PRODUCT entries for all functions that have been deleted from the global zone.

### **Entries deleted by the REJECT command**

Regardless of whichever REJECT processing mode you select, SMP/E will delete the following:

- ▶ The SMPPTS MCS entry
- ▶ The global zone SYSMOD entry
- ▶ The associated FMID subentry in the GLOBALZONE entry, as appropriate
- ▶ The eligible HOLDDATA entries
- ▶ The associated SMPTLIB data sets, if the SYSMOD was packaged in RELFILE format

### **Reports for REJECT processing**

At the end of REJECT processing, output from the REJECT command includes reports, as well as statistics written to SMPOUT and SMPLOG.

Two reports are produced during REJECT processing:

- ▶ **File Allocation report**  
This report provides a list of data sets allocated during the REJECT process and also information regarding these data sets.
- ▶ **REJECT Summary report**  
This report is produced at the completion of REJECT processing to summarize the processing that occurred for SYSMODs and other data.

SMP/E writes statistics to SMPOUT and SMPLOG to summarize what happened during REJECT processing. These statistics follow the message issued at the completion of REJECT processing.

For more information regarding these reports, see *SMP/E Commands*, SA22-7771.

## 5.23 REJECT examples

- ☐ Rejecting in MASS mode
- ☐ Rejecting in SELECT mode
- ☐ Rejecting in PURGE mode
- ☐ Rejecting in NOFMID mode

Figure 5-36 REJECT examples

### REJECT examples

The following examples are provided to help you to use the REJECT command.

#### How to reject in MASS mode

In MASS mode, you can reject all SYSMODs that have been received but not installed yet. You can use the sample JCL shown in Figure 5-37 to perform this task.

```
//REJSYSM JOB (), 'MVSSP', NOTIFY=&SYSUID, CLASS=A, MSGLEVEL=(1,1),
//      MSGCLASS=X
//SMPE     EXEC PGM=GIMSMP
//SMPCSI   DD DSN=SMPE.GLOBAL.CSI, DISP=SHR
//SMPRPT   DD SYSOUT=*
//SMPOUT   DD SYSOUT=*
//SYSPRINT DD SYSOUT=*
//SMPCNTL  DD *
        SET      BDY(GLOBAL)          /* Set to global zone.      */
        REJECT   APARS                  /* Reject all received-only */
                FUNCTIONS                /* SYSMODs.                 */
                PTFS
                USERMODS.
/*
```

Figure 5-37 Sample JCL for rejecting SYSMODs in MASS mode



**Note:** This method may reject SYSMODs you wanted to keep, and if that happens you would have to receive them again.

Using the MASS mode, you can also reject only PTFs that have been received but not applied. The commands in Figure 5-38 show how it can be done.

```
SET      BDY(GLOBAL) .      /* Set to global zone.      */
REJECT .                      /* Reject all received-only
                             PTFs.                          */
```

Figure 5-38 Rejecting PTFs that have been received but not applied

**Note:** If no SYSMOD types are specified on a REJECT command for MASS mode, SMP/E rejects only PTFs.

### How to reject in SELECT mode

Sometimes, you want to reject certain SYSMODs from your system. You can use the REJECT command in the SELECT mode if you have applied and accepted a USERMOD into your system. You want to reject the current version, update the SYSMOD, and then reapply and reaccept it. You can use the command shown in Figure 5-39 to perform this task.

```
SET      BDY(GLOBAL) .      /* Set to global zone.      */
REJECT   S(MYMOD01)1        /* Reject this SYSMOD      */
          BYPASS(            /* even though it was      */
          ACCEPTCHECK        /* accepted and            */
          APPLYCHECK) .      /* applied.                */
```

Figure 5-39 Rejecting a SYSMOD in SELECT mode

Figure 5-39 note:

1 - MYMOD01 is a USERMOD.

**Note:** By default, the REJECT command will only reject SYSMODs that are received but not installed. However, you can specify the BYPASS operand to prevent SMP/E from checking where the SYSMODs have been installed.

You can also use the REJECT command to reject HOLDDATA from your system. Assuming that you want to delete a HOLDDATA entry with no associated SYSMOD entry, you can use the command shown in Figure 5-40 to accomplish this task.

```
SET      BDY(GLOBAL) .      /* Set to global zone.      */
REJECT   S(UW04356)         /* For this SYSMOD,        */
          HOLDDATA .         /* reject the HOLDDATA.    */
```

Figure 5-40 Rejecting HOLDDATA from the system

**Note:** The REJECT command will delete both the SYSMOD and its associated HOLDDATA entry if it exists in the system.

## How to reject in PURGE mode

When you have specified NOPURGE in the OPTIONS entry, SMP/E will not delete the GLOBAL SYSMOD and the SMPPTS MCS entry after you have accepted the SYSMODs. You can use the commands shown in Figure 5-41 to reject SYSMODs that have been accepted into your system.

```
SET      BDY(GLOBAL) .      /* Set to global zone.      */
REJECT   PURGE(DLIB1) .      /* Reject SYSMODs installed
                             in this DLIB zone      */
```

Figure 5-41 Rejecting in PURGE mode

## How to reject in NOFMID mode

Assume you had received, applied, and accepted function HMX1101. The function was automatically deleted from the global zone and SMPPTS when it was accepted. You have also received service for the function. Assume you have now decided to install an updated version of the function. To prepare for this, you want to delete the FMID of the current function from the GLOBALZONE entry, as well as delete the service and associated HOLDDATA that were received for that function.

You can use the command shown in Figure 5-42 to perform the delete.

```
SET      BDY(GLOBAL) .      /* Process global zone.      */
REJECT   DF MID(HMX1101)     /* Delete FMID and reject    */
          NOFMID .           /* for FMIDs not in GZONE.   */
```

Figure 5-42 Rejecting in NOFMID mode

**Note:** This deletes SYSMODs and associated HOLDDATA for all functions that are not defined in the GLOBALZONE entry. It is not limited to entries for HMX1101. To limit the REJECT command to specific functions, use the FORFMID operand in another REJECT mode.

Rather than manually determine which FMIDs to delete, you can use the sample programs GIMCRSAM and GIMPRSAM (provided in SYS1.SAMPLIB) to help you create a REJECT NOFMID command for the FMIDs that are superseded or deleted in the DLIB zones you specify.

## 5.24 The APPLY process

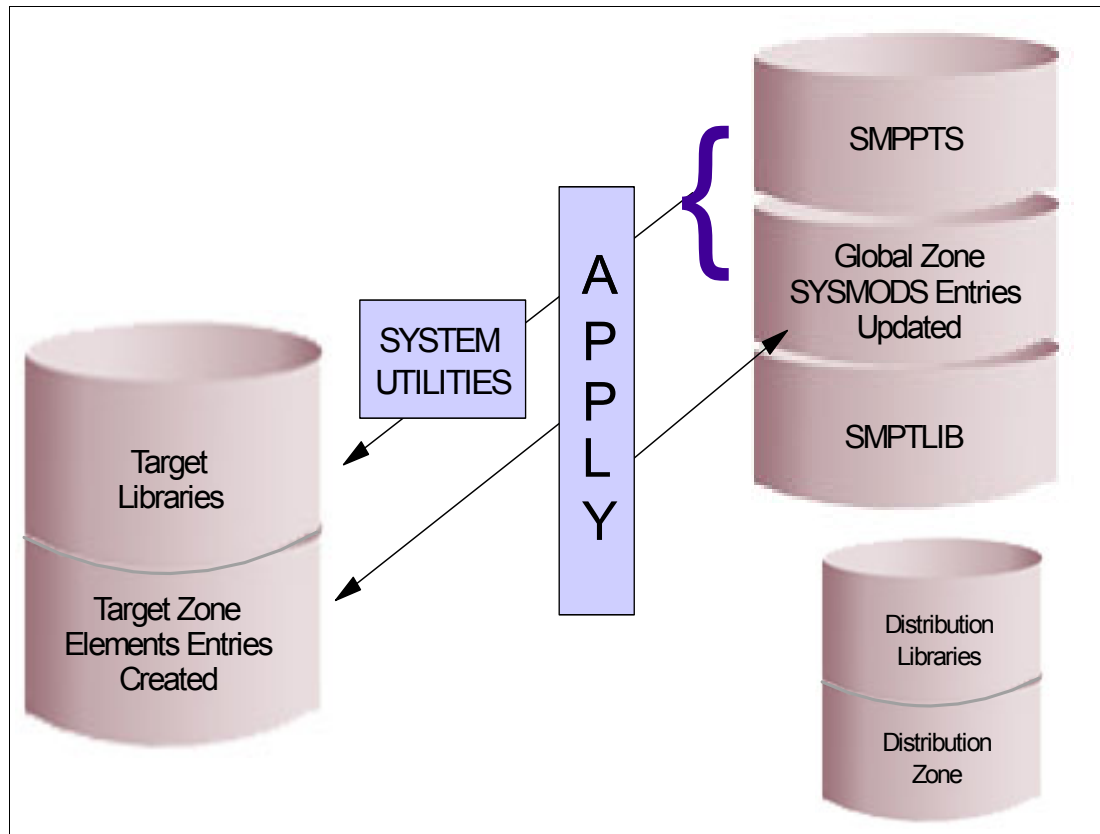


Figure 5-43 The APPLY process

### The APPLY process

After the SYSMODs have been received, you can use the APPLY command to install them into the appropriate target system libraries. The APPLY command calls system utilities, which are responsible for the actual updating of those libraries.

### Selecting SYSMODS

You can specify operands on the APPLY command that tell SMP/E which of the received SYSMODs are to be selected for installation in the target libraries. SMP/E checks to make sure all other required SYSMODs (prerequisites) have been installed or are being installed concurrently and in the proper sequence.

### Selecting elements

During APPLY processing, SMP/E uses the information provided in the selected SYSMODs to determine which elements should be installed in the target libraries. The selection of elements is monitored by SMP/E to make sure that the correct functional level of each element is selected.

### Checking the APPLY process

SMP/E provides you with an option to stop APPLY processing just before any updating takes place so you can ensure all prerequisites are satisfied before the installation of the SYSMODs. This helps you see what will happen (and helps you detect problem SYSMODs) without actually updating the target libraries.

## Updating the target library

After the proper SYSMODs have been selected and the proper functional and service level of each element has been determined, and after ensuring that the installation of another SYSMOD does not cause any current service regress, the APPLY command directs SMP/E to call the system utilities. It is the system utilities that actually place the elements into the target libraries described in the target zone. The source of the elements is the SMPTLIB data sets, the SMPPTS data set, or the indirect libraries, depending on how the SYSMOD was packaged.

### Note:

- ▶ Because the APPLY command updates the system libraries, you should never use it on a live production system. When you process the APPLY command, you should always use a copy of the target libraries and target zone. By using a copy, you minimize the risk of new code causing an outage of your system. This process of copying is called *cloning* and is explained in detail in the *OS/390 Software Management Cookbook*, SG24-4775.
- ▶ We recommend that you always use the CHECK subparameter before you update the libraries with the new version of the modules, even when applying in a test system. It can avoid unnecessary workload.

## How SMP/E keeps track of APPLY processing

SMP/E updates the information about the SYSMODs that have been applied. Remember, the target zone reflects the contents of the target libraries. Therefore, after the utility work is complete, and the target libraries have been updated, the APPLY command records the functional and service levels of the new elements in the target zone.

- ▶ A SYSMOD entry is created in the target zone for each SYSMOD that has been applied. Element entries (such as MOD and LMOD) are also created in the target zone for those elements that have been installed in the target libraries.
- ▶ SYSMOD entries in the global zone are updated to reflect that the SYSMOD has been applied to the target zone.
- ▶ BACKUP entries are created in the SMPSCDS data set so the SYSMOD can later be restored, if necessary.

The APPLY process is controlled by:

- ▶ The information in the target zone reflecting the status and structure of the target system libraries
- ▶ Information on the SYSMODs indicating their applicability
- ▶ Information in the OPTIONS and UTILITY entries
- ▶ Operands on the APPLY command

## Reports for Apply processing

When APPLY processing is complete, these reports will help you analyze the results:

- ▶ **SYSMOD Status Report** provides you with a summary of the processing that took place for each eligible SYSMOD, based on the operands you specified on the APPLY command. It shows you which SYSMODs were applied, which were not applied, and why.
- ▶ The **Element Summary Report** provides you with a detailed look at each element affected by APPLY processing. It tells you in which libraries the elements were installed.
- ▶ The **Causer SYSMOD Summary Report** provides you with a list of SYSMODs that caused other SYSMODs to fail, and describes the errors that must be fixed to successfully

process the SYSMODs. This report can reduce the amount of work involved in figuring out which errors caused SYSMODs to fail.

- ▶ The **File Allocation Report** provides you with a list of the data sets used for APPLY processing and supplies information about these data sets.

Additional reports may be produced depending on the work being done and the content of the SYSMODs. For more information about all the reports produced by the APPLY command (and samples of actual reports), see the APPLY command in *SMP/E Commands SA22-7771*.

## Summary

As explained in this section, when the APPLY command is used to install a SYSMOD in the target libraries, the command does the following:

- ▶ Selects SYSMODs to install
- ▶ Checks that all other required SYSMODs have been (or are being) installed
- ▶ Based on SYSMODs, selects elements to install
- ▶ Directs SMP/E to call the system utilities to update the target libraries
- ▶ Records what is applied:
  - Target zone: Creates SYSMOD entries and element entries
  - Global zone: Updates SYSMOD entries
  - SMPSCDS data set: Creates BACKUP entries
- ▶ Reports the results of processing

Remember, you should never perform APPLY processing on a live production system!

## 5.25 APPLY examples

- ❑ Applying all SYSMODs
  - using SOURCEID for grouping SYSMODs
- ❑ Applying with GROUP
  - SMP/E automatically installs all requisites
- ❑ Applying with CHECK
  - SMP/E does not install SYSMOD, for testing only

Figure 5-44 APPLY examples

### APPLY examples

The following examples will help you to use the APPLY command.

#### Applying all SYSMODs from a given source

If the SOURCEID operand was used during RECEIVE processing to group all those SYSMODs processed, you can choose to install only that set of SYSMODS. This can be done with the SOURCEID operand of the APPLY command. Suppose you received an ESO containing service levels PUT9901 and PUT9902. The ESO contained ++ASSIGN statements that assigned each PTF a SOURCEID value corresponding to the service level that it is part of. Now you want to install all the applicable PTFs from those tapes into the target libraries described by zone MVSTST1. You can use the sample JCL shown in Figure 5-45 to perform this task.

```
//APPLY    JOB 'accounting info',MSGLEVEL=(1,1)
//APPLY    EXEC SMPPROC
//SMPTLIB  DD  UNIT=3380,VOL=SER=TLIB01
//SMPCNTL  DD  *
      SET   BDY(MVSTST1)           /* Process MVSTST1 tgt zone. */.
      APPLY SOURCEID(PUT9901,      /* Process these service */
                PUT9902)          /* levels */.
                GROUP             /* and any requisites. */.
/*
```

Figure 5-45 Sample JCL to apply all SYSMODs from a given source

## Applying with the GROUP operand

At times, you may know that a particular SYSMOD is required on your system, but you may not know all its requisite SYSMODs. By using the GROUP operand of the APPLY command, you can have SMP/E determine all the requisites and automatically install them. This method is often used during the installation of new functions. Suppose you want to install a new function HYY2102, plus all its service, plus any requisite SYSMODs. You can do this with the commands shown in Figure 5-46.

|       |                  |                              |     |
|-------|------------------|------------------------------|-----|
| SET   | BDY(MVSTST1)     | /* Process MVSTST1 tgt zone. | */. |
| APPLY | FORFMID(HYY2102) | /* For one function.         | */  |
|       | FUNCTIONS PTFS   | /* Function and PTFS         | */  |
|       | GROUP            | /* plus requisites.          | */. |

Figure 5-46 Applying with the GROUP operand

## Applying with the CHECK operand

In this example, SMP/E was directed to automatically include SYSMODs needed for the selected function and service. At times, you may want to review which SYSMODs are included before you actually install them. This can be done by using the CHECK operand of the APPLY command, as shown in Figure 5-47.

|       |                  |                              |     |
|-------|------------------|------------------------------|-----|
| SET   | BDY(MVSTST1)     | /* Process MVSTST1 tgt zone. | */. |
| APPLY | FORFMID(HYY2102) | /* For one FMID.             | */  |
|       | FUNCTIONS PTFS   | /* Functions and PTFS        | */  |
|       | GROUP            | /* plus requisites           | */  |
|       | CHECK            | /* in check mode.            | */. |

Figure 5-47 Applying with the CHECK operand

After running this command, check the SYSMOD Status Report to see which SYSMODs would have been installed if you had not specified CHECK. If the results are acceptable, run the commands again, without the CHECK operand, to actually install the SYSMODs.

## 5.26 The APPLY CHECK process

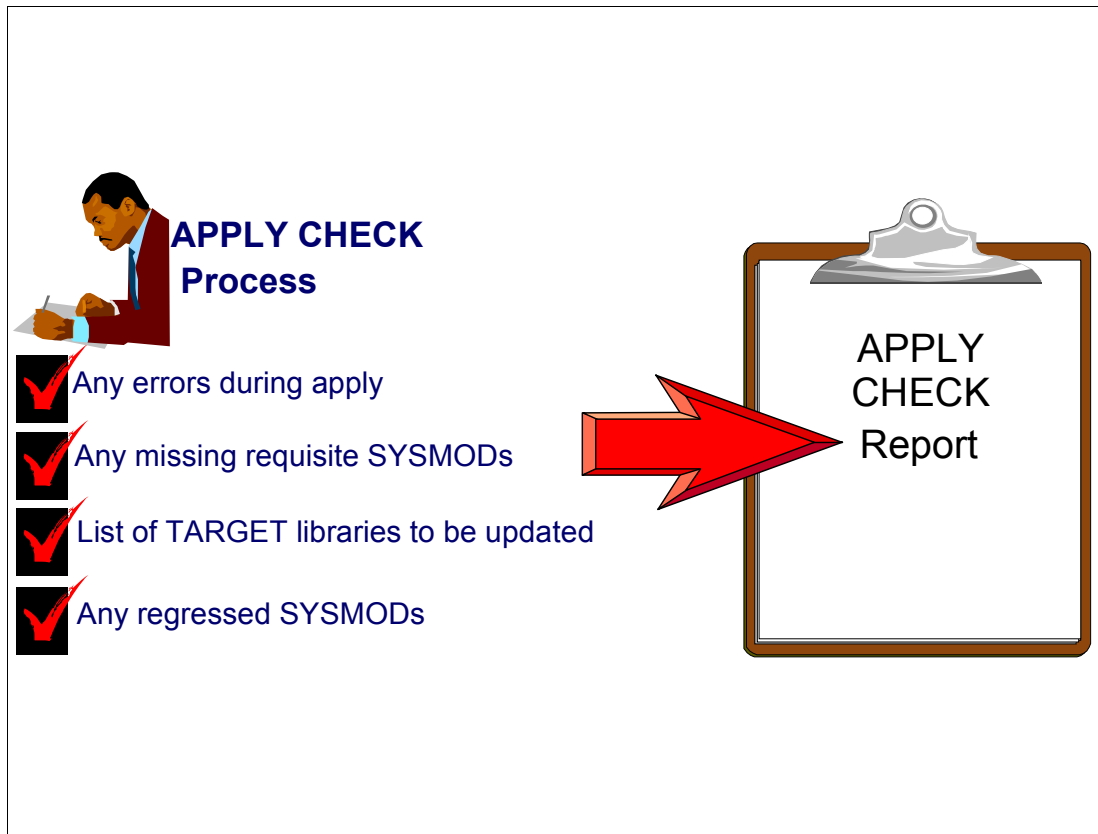


Figure 5-48 The APPLY CHECK process

### The APPLY CHECK process

The purpose of this command is to determine:

- ▶ Whether any errors will occur while the new function is being applied (except for errors that occur as a direct result of an update, such as a target library running out of space). This includes missing DDDEF entries.
- ▶ Whether any requisite SYSMODs are missing.
- ▶ A list of target libraries that will be updated during the actual apply. This will allow you to back up those libraries before you apply the new elements.
- ▶ The SYSMODs, if any, that will be regressed.
- ▶ SYSMOD in hold exception status.
- ▶ Processing of online JCLIN.

### Researching the APPLY CHECK reports

As a result of running the APPLY CHECK job, SMP/E produces various messages and reports that you should now use to do further research. Here are some of the errors that may have been detected:

- ▶ Some DD statements may be missing. Check the program directory or the *SMP/E Reference*, SA22-7772, to determine why they are required and how they should be specified.



- ▶ Some APAR fixes or USERMODs may be regressed. If so, you must determine why. For APAR fixes, you have to get the version of the APAR fix applicable to the new product. For USERMODs, you have to rework the modification to make it applicable to the new function, or eliminate the modification if the product being installed provides the same function. When doing the actual APPLY operation, you may need to specify the BYPASS operand to inform SMP/E that you have resolved these problems.
- ▶ Some prerequisite or requisite PTFs may be missing. If so, you should determine whether they can be obtained. Some may already be on an ESO tape you have in-house but have not received; others may not have been shipped, in which case you have to get an early copy of them by contacting the IBM Support Center. Although you can also avoid these conditions by using the BYPASS operand, you are advised not to do this because the regressions have not been resolved.
- ▶ Some elements may not have been selected for installation. For each such element, if the current functional owner (that is, FMID) is an IBM product, there may not be a problem; this condition is common and occurs because there are multiple functions with common elements. Check the program directory or installation guide for the product you are installing to determine whether this condition is normal or if it indicates a problem.  
  
If the FMID is not one for an IBM product, further research is necessary. Contact the current owner of the element to determine how that product is related to the one you are installing.
- ▶ Some of the PTFs may not have been selected for installation because of exception SYSMOD conditions identified by the ++HOLD MCSs. When installing a new function, you may want to research these PTFs further. You can use the reason ID and the comments specified in the ++HOLD MCS to determine which of the following actions is most appropriate:
  - Bypass the condition using the BYPASS(HOLDERR) operand.
  - Do not install the PTF.
  - Obtain a fix for the APAR.

### Getting additional SYSMODs

After doing the research step, you may decide that additional SYSMODs are needed. If so:

- ▶ Obtain the additional SYSMODs by using CBPDO, ESO, CSSF Information/Access, SoftwareXcel Extended, or the IBM Support Center.
- ▶ Receive the additional SYSMODs, using the same source ID value as used when processing the CBPDO tape.
- ▶ Rerun the APPLY CHECK job.
- ▶ Repeat this process until no errors are reported.

## 5.27 The RESTORE process

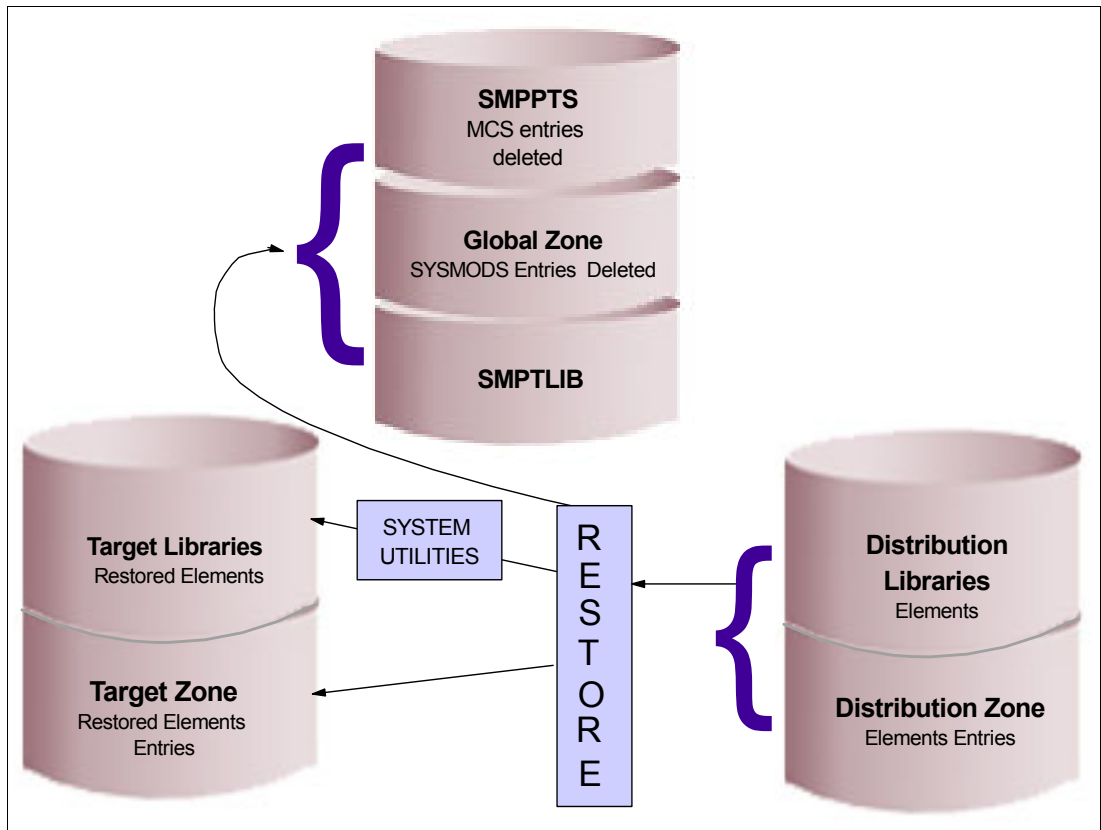


Figure 5-49 The RESTORE process

### The RESTORE process

If you discover that a particular SYSMOD is causing a problem in your target libraries, you can remove it and replace the elements affected by it with the previous level of those elements, which is obtained from the backup (or distribution) libraries.

The RESTORE command replaces the affected elements in the target libraries with the unchanged versions from the distribution libraries.

You can use the RESTORE command to remove SYSMODs from the target libraries and restore them to a previous level. The RESTORE command reverses APPLY processing, but has no effect on ACCEPT processing.

### Removing SYSMODs

SMP/E ensures the eligibility of the selected SYSMODs and checks whether other SYSMODs are affected before continuing with RESTORE processing. Because of the various relationships and dependencies among the many SYSMODs, this checking is very important to the integrity of your system. In fact, to ensure that the requisites for a SYSMOD being restored are processed appropriately, SMP/E may require the whole chain of prerequisites to be restored.

### Selecting elements

During RESTORE processing, SMP/E uses the information provided in the selected SYSMODs to determine which elements in the target zone should be replaced by elements in

the related distribution libraries. The selection of elements is monitored by SMP/E to make sure that the correct functional level of each element is selected.

### Replacing the elements in the target libraries

When SMP/E is satisfied that the proper SYSMODs have been selected, it uses information from the target zone to determine which distribution zone describes the elements necessary to replace the SYSMOD's elements in the target libraries. The RESTORE command directs SMP/E to call system utilities that replace the elements in the target libraries with the previous level of the elements from the related distribution libraries.

### How SMP/E keeps track of RESTORE processing

SMP/E updates the information about the SYSMODs that have been restored. Remember, the target zone reflects the contents of the target libraries. Therefore, after the utility work is complete, and the target libraries have been updated, the target zone is updated to accurately reflect the status of those libraries.

- ▶ All information in the target zone pertaining to the restored SYSMOD is removed. The element entries in the target zone are restored to reflect the distribution zone level of the elements.
- ▶ The global zone SYSMOD entries and MCS statements, which are stored in the SMPPTS data set, are deleted for those SYSMODs that have been restored. Any SMPTLIB data sets created during RECEIVE processing are also deleted for the restored SYSMOD. SMP/E automatically performs this global zone clean-up, unless you specify otherwise.

### Reports for RESTORE processing

When RESTORE processing is complete, these reports will help you analyze the results:

- ▶ The **SYSMOD Status Report** provides you with a summary of the processing that took place for each eligible SYSMOD, based on the operands you specified on the RESTORE command. It shows you which SYSMODs were restored, which were not restored, and why.
- ▶ The **Element Summary Report** provides you with a detailed look at each element replaced or modified by RESTORE processing. It tells you in which libraries the elements were restored.
- ▶ The **Causer SYSMOD Summary Report** provides you with a list of SYSMODs that caused other SYSMODs to fail, and describes the errors that must be fixed to successfully process the SYSMODs. This report can reduce the amount of work involved in figuring out which errors caused SYSMODs to fail.
- ▶ The **File Allocation Report** provides you with a list of the data sets used for RESTORE processing and supplies information about these data sets.

Depending on the work being done and the content of the SYSMODs, additional reports may be produced. For more information about all the reports produced by the RESTORE command (and samples of actual reports), see *SMP/E Commands*, SA22-7771.

### Summary

When the RESTORE command is used to remove a SYSMOD from the target libraries it does the following:

- ▶ Removes the SYSMOD from the indicated target zone
- ▶ Calls system utilities to replace the SYSMODs elements in the target libraries with elements from the related distribution libraries

- ▶ Records what is restored:
  - Target zone: Restores element entries to reflect their distribution zone level and deletes all information about restored SYSMOD.
  - Global zone: Deletes SYSMOD entries and MCS statements in SMPPTS for restored SYSMOD. Any SMPTLIB data sets created during RECEIVE processing are also deleted for the restored SYSMOD. (This global zone processing is optional.)
  - SMPSCDS data set: Deletes BACKUP entries for restored SYSMOD.
- ▶ Reports the results of processing

**Note:** Not all SYSMODs can be restored. For example, SMP/E cannot restore a SYSMOD that deletes another SYSMOD or that deletes a load module during APPLY processing.

## 5.28 Restore examples

- ❑ Restoring a single SYSMOD
  - Removing 1 PTF
- ❑ Restoring multiple PTFs to remove 1 PTF
  - Using 2 methods
- ❑ Restoring PTFs with GROUP operand
  - Indicates that SMP/E to determine additional SYSMODs to be restored

Figure 5-50 Restore examples

### Restore examples

The following examples can help you to use the RESTORE command.

#### Restoring a single SYSMOD

Assume you have applied only PTF UZ00001, that an error was detected during testing, and that you want to remove the PTF from your system. You can use the RESTORE command shown in Figure 5-51 to do this.

|         |            |                        |     |
|---------|------------|------------------------|-----|
| SET     | BDY(TGT1)  | /* Set to target zone. | */. |
| RESTORE | S(UZ00001) | /* Restore 1 PTF.      | */. |

Figure 5-51 Removing a single PTF

If you want to clean up all of the SMP/E records for this PTF (the global zone and the SMPPTS data set), you can use the REJECT command after RESTORE processing completes, as shown in Figure 5-52.

|        |             |                        |     |
|--------|-------------|------------------------|-----|
| SET    | BDY(GLOBAL) | /* Set to global zone. | */. |
| REJECT | S(UZ00001)  | /* Reject 1 PTF.       | */. |

Figure 5-52 Cleaning up the SMP records after the reject

## Restoring multiple PTFs to remove one PTF

Assume you have applied the PTFs named UZ00001, UZ00002, and UZ00003 to your system, and that during testing an error is found in module XYMOD01. Because the current service level of that module is UZ00003 (we can say that the RMID of the module is UZ00003), you want to restore that PTF from the system.

You have two choices:

1. Restore PTF UZ00001, UZ00002, UZ00003, and then reapply UZ00001 and UZ00002, as shown in Figure 5-53.

|         |            |                          |     |
|---------|------------|--------------------------|-----|
| SET     | BDY(TGT1)  | /* Set to target zone.   | */. |
| RESTORE | S(UZ00001, | /* Restore all 3 PTFs.   | */  |
|         | UZ00002,   | /*                       | */  |
|         | UZ00003)   | /*                       | */. |
| APPLY   | S(UZ00001  | /* Then re-apply the two | */  |
|         | UZ00002)   | /* that may be ok.       | */. |

Figure 5-53 Restoring and reapplying PTFs

2. Accept PTFs UZ00001 and UZ00002, if you are sure that they have no errors, then restore UZ00003 as shown in Figure 5-54.

|         |            |                           |     |
|---------|------------|---------------------------|-----|
| SET     | BDY(DLIB1) | /* Set to DLIB zone.      | */. |
| ACCEPT  | S(UZ00001, | /* Accept two good PTFs.  | */  |
|         | UZ00002)   | /*                        | */. |
| SET     | BDY(TGT1)  | /* Set to target zone.    | */. |
| RESTORE | S(UZ00003) | /* Restore the 1 bad PTF. | */. |

Figure 5-54 Accepting some PTFs and then restoring another

The end result in both cases is that module XYMOD01 from PTF UZ00002 is in the target libraries.

## Restoring PTFs using the GROUP operand

In the previous example, when you wanted to restore the three PTFs, you specified all three in the select list. In a simple case like this, that was very easy; in practice, however, many PTFs are related to one another, and it may not be easy to determine which PTFs must be restored in order to remove the bad one. The GROUP operand can be used to assist in determining this. The commands shown in Figure 5-55 can be run to determine which PTFs must be restored to restore UZ00003.

|         |            |                             |     |
|---------|------------|-----------------------------|-----|
| SET     | BDY(TGT1)  | /* Set to target zone.      | */. |
| RESTORE | S(UZ00003) | /* Restore this one PTF     | */  |
|         | GROUP      | /* plus any related PTFs,   | */  |
|         | CHECK      | /* in check mode this time. | */. |

Figure 5-55 Restoring PTFs using the Group operand

After running these commands, the various SMP/E reports can be used to determine that PTFs UZ00001, UZ00002, and UZ00003 should be restored. You can then determine the correct action: restore all, or accept some and then restore.

## 5.29 The ACCEPT process

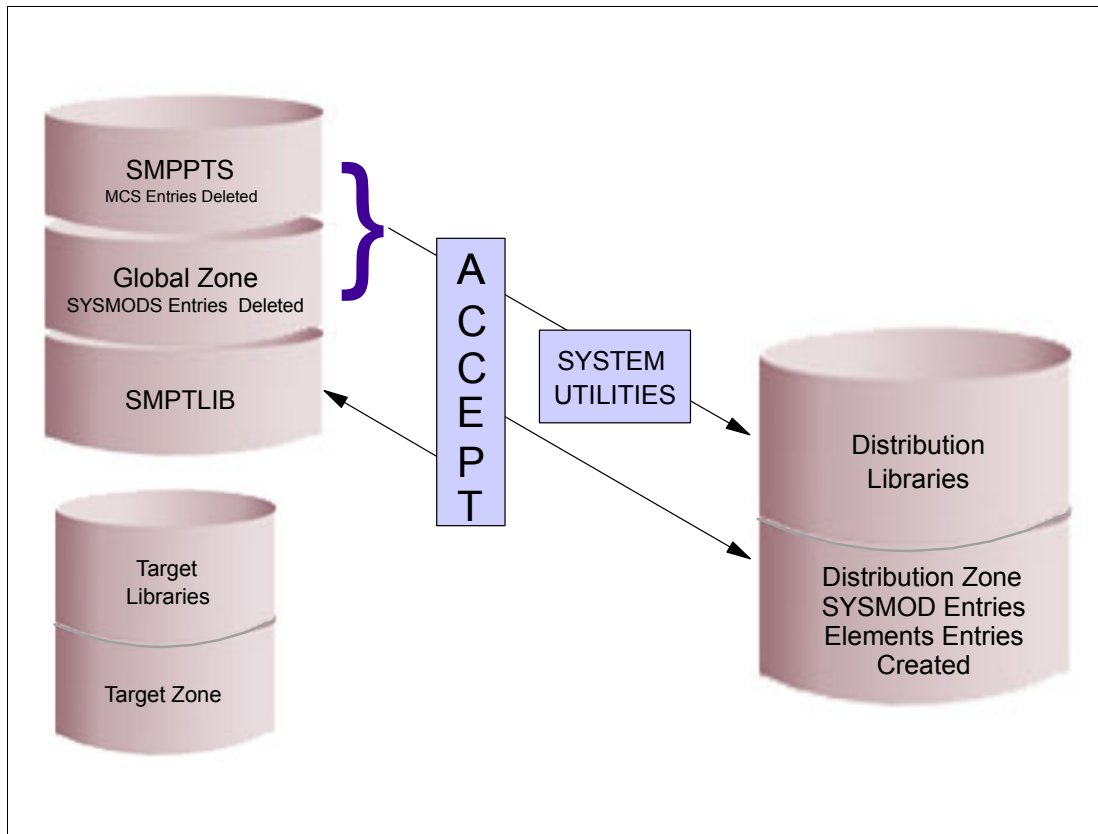


Figure 5-56 The ACCEPT process

### The ACCEPT process

You can use the ACCEPT process to install a SYSMOD in the distribution libraries. The ACCEPT process is very similar to the APPLY process with one important exception: ACCEPT processing is *irreversible*.

After you are satisfied that an applied SYSMOD has performed reliably in your target system, you can install it in your distribution libraries.

When you accept a PTF, be aware that if in the future you need to restore a PTF that changes the same module of the accepted PTF, this module will be copied from the DLIB to your target library and it will retrograde to the level of the last PTF accepted. If you have never accepted a PTF, the module will retrograde to the base level, or at least, to the level that was installed.

### Selecting SYSMODs

You can specify operands on the ACCEPT command that tell SMP/E which of the received SYSMODs are to be selected for installation in the distribution libraries. SMP/E ensures that all other required SYSMODs have been installed or are being installed concurrently and in the proper sequence.

### Selecting elements

During ACCEPT processing, SMP/E uses the information provided in the selected SYSMODs to determine which elements should be installed in the distribution libraries. The selection of

elements is monitored by SMP/E to make sure that the correct functional level of each element is selected.

## Updating the distribution libraries

After the proper SYSMODs have been selected and the proper functional and service level of each element has been checked, SMP/E calls the system utilities (in the same manner as APPLY and RESTORE) to place the elements into the distribution libraries described in the distribution zone. The source of the elements is the SMPTLIB data sets, the SMPPTS data set, or the indirect libraries, depending on how the SYSMOD was packaged.

**Note:** When ACCEPT processing has been completed, there is no way it can be undone.

## How SMP/E keeps track of ACCEPT processing

SMP/E updates the information about the SYSMODs that have been accepted. Remember, the distribution zone reflects the contents of the distribution libraries. Therefore, after the utility work is complete, and the distribution libraries have been updated, the distribution zone is updated to accurately reflect the status of those libraries.

- ▶ A SYSMOD entry is created in the *distribution zone* for each SYSMOD that has been accepted. Element entries (such as MOD and LMOD) are also created in the distribution zone for the elements that have been installed in the distribution libraries.
- ▶ *Global zone* SYSMOD entries and MCS statements in the SMPPTS data set are deleted for those SYSMODs that have been accepted into the distribution zone. Any SMPTLIB data sets created during RECEIVE processing are also deleted. If you do not want SMP/E to do this global zone clean-up, you have the option to indicate this to SMP/E, and the information is saved.

The ACCEPT process is controlled by:

- ▶ The information in the distribution zone reflecting the status and structure of the distribution libraries
- ▶ Information on the SYSMODs indicating their applicability
- ▶ Information in the OPTIONS and UTILITY entries
- ▶ Operands on the ACCEPT command

## ACCEPT CHECK process

The purpose of the CHECK option is to perform a test run informing you of possible error conditions and providing reports of SYSMOD status, libraries that will be updated, regression conditions, and SYSMODs that will be deleted. During CHECK processing, the list of distribution zone entries is maintained in storage; data is written to the distribution zone as a temporary storage medium. CHECK processing deletes any data written to the distribution zone. Consequently, no permanent updates are made to the distribution zone.

We recommend that you always use the ACCEPT CHECK prior to accepting the SYSMODs.

## Reports for ACCEPT processing

When ACCEPT processing is complete, these reports will help you analyze the results:

- ▶ The **SYSMOD Status Report** provides you with a summary of the processing that took place for each eligible SYSMOD, based on the operands you specified on the ACCEPT command. It shows you which SYSMODs were accepted, which were not accepted, and why.



- ▶ The **Element Summary Report** provides you with a detailed look at each element affected by ACCEPT processing. It tells you in which libraries the elements were installed.
- ▶ The **Causer SYSMOD Summary Report** provides you with a list of SYSMODs that caused other SYSMODs to fail, and describes the errors that must be fixed to successfully process the SYSMODs. This report can reduce the amount of work involved in figuring out which errors caused SYSMODs to fail.
- ▶ The **File Allocation Report** provides you with a list of the data sets used for ACCEPT processing and supplies information about these data sets.

Additional reports may be produced depending on the work being done and the content of the SYSMODs. For more information about all the reports produced by the ACCEPT command (and samples of actual reports), see *SMP/E Commands*, SA22-7771.

## Summary

When the ACCEPT command is used to install a SYSMOD in the distribution (or backup) libraries, it does the following:

- ▶ Selects SYSMODs to install
- ▶ Checks that all other required SYSMODs have been (or are being) installed
- ▶ Based on SYSMODs, selects elements to install
- ▶ Directs SMP/E to call the system utilities to update the distribution libraries
- ▶ Records what is accepted:
  - Distribution zone: Creates SYSMOD entries and element entries.
  - Global zone: Deletes SYSMOD entries and MCS statements in SMPPTS. Any SMPDLIB data sets created during RECEIVE processing are also deleted. (This global zone processing is optional.)
- ▶ Reports the results of processing

Remember, once you have accepted a SYSMOD, it cannot be restored!

## 5.30 ACCEPT examples

- ❑ Accepting all SYSMODs
  - Using SOURCEID operand
- ❑ Accepting using GROUP operand
  - SMP/E includes all requisite SYSMODs not accepted
- ❑ Accepting using GROUPEXTEND operand
  - SMP/E includes superseding SYSMODs for requisite SYSMODs in error

Figure 5-57 ACCEPT examples

### ACCEPT examples

The following examples will help you use the ACCEPT command.

#### Accepting all SYSMODs from a given source

If you used the SOURCEID operand during RECEIVE processing to group all the SYSMODs processed, you may choose to install only that set of SYSMODs. You can do this with the SOURCEID operand of the ACCEPT command. Suppose you received an ESO containing service levels PUT9901 and PUT9902. The ESO contained ++ASSIGN statements that assigned each PTF a SOURCEID value corresponding to the service level it is part of. Now you want to install all the applicable PTFs from those tapes into the distribution libraries described by zone MVSDLB1. You can do this with the commands shown in Figure 5-58.

```
SET      BDY(MVSDLB1)      /* Process MVSDLB1 DLIB zone. */.  
ACCEPT   SOURCEID(PUT9901, /* Process these service      */  
          PUT9902)         /* levels              */  
          GROUP            /* and any requisites. */.
```

Figure 5-58 Accepting all SYSMODs from a given source

#### Accepting with the GROUP operand

At times, you may know that a particular SYSMOD is required on your system, but you may not know all its requisite SYSMODs. You can use the GROUP operand of ACCEPT to have

SMP/E determine all the requisites and install them automatically. This method is often used when a new function is being installed. Suppose you want to install a new function, HPT1234, with all its service and any requisite SYSMODs. You can do this with the commands shown in Figure 5-59.

|        |                  |                                   |
|--------|------------------|-----------------------------------|
| SET    | BDY(MVSDLB1)     | /* Process MVSDLB1 DLIB zone. */. |
| ACCEPT | FORFMID(HYY1234) | /* For one function. */           |
|        | FUNCTIONS PTFS   | /* Functions and PTFS */          |
|        | GROUP            | /* plus requisites. */.           |

Figure 5-59 Accepting SYSMODs with ACCEPT

The FORFMID operand indicates that only SYSMODs applicable to this function should be installed. The FUNCTIONS operand indicates that HYY1234 can be installed. The PTFS operand indicates that only PTFS for HYY1234 should be installed (no APARs or USERMODs are included). The GROUP operand indicates that all requisite SYSMODs should also be accepted. These requisites can be applicable to other functions, but may not be APARs or USERMODs.

### Accepting with the GROUPEXTEND operand

Assume you want SMP/E to automatically include the requisites for some SYSMODs you plan to install. However, you are not sure whether all of the requisites are available. (They may not have been received, or they might be held because they are in error.) In these cases, you would like SMP/E to check whether a superseding SYSMOD is available for the unsatisfied requisites. To have SMP/E do this additional checking, you can use the GROUPEXTEND operand as shown in Figure 5-60.

|        |                  |                                   |
|--------|------------------|-----------------------------------|
| SET    | BDY(MVSDLB1)     | /* Process MVSDLB1 DLIB zone. */. |
| ACCEPT | FORFMID(HYY1234) | /* For one function. */           |
|        | FUNCTIONS PTFS   | /* Functions and PTFS plus */     |
|        | GROUPEXTEND      | /* requisites or supersedes. */.  |

Figure 5-60 Using the GROUPEXTEND operand

SMP/E accepts HYY1234 and any functions or PTFS applicable to HYY1234. Because of the GROUPEXTEND operand, SMP/E also accepts all requisites for those SYSMODs, even if the requisites are not applicable to HYY1234. If SMP/E cannot find a requisite, it looks for a SYSMOD that supersedes the requisite and uses it to satisfy the requirement. Likewise, if a requisite is being held for an error reason ID, SMP/E looks for a SYSMOD that supersedes the requisite, or that either satisfies or supersedes its error reason ID, and uses it to satisfy the requirement.

**Note:** You must be careful in using the ACCEPT command. When you accept a SYSMOD, you cannot restore it by using SMP/E.

## 5.31 Other useful SMP/E commands

- ☐ LIST
- ☐ REPORT ERRSYSMOD
- ☐ UCLIN
- ☐ UNLOAD
- ☐ ZONECOPY
- ☐ ZONEDELETE
- ☐ ZONERENAME

Figure 5-61 Other useful SMP/E commands

### Other useful SMP/E commands

In the course of managing your system, 90 percent of the time you will only use the RECEIVE, APPLY, and ACCEPT commands. However, there are times when you need to use other SMP/E commands to perform a specific task, such as retrieving information from SMP/E data sets, or cloning a new target zone from the existing target zone.

Some of the more useful SMP/E commands are:

► LIST command

You can use the LIST command to retrieve information stored in SMP/E data sets.

► REPORT ERRSYSMOD command

This command helps you to identify exception SYSMODs and also any resolving SYSMODs for the held SYSMODs.

► UCLIN command

With the UCLIN command, you can add, delete, or replace entries in the following SMP/E data sets:

- SMPCSI
- SMPMTS
- SMPSCDS
- SMPSTS

The UCLIN command only updates entries in SMP/E data sets, hence it does nothing to any elements or load modules in any product libraries.

**Note:** Be sure you understand the relationships between the various entries before you make any UCLIN changes, as this helps to ensure that any UCLIN changes made are complete and consistent with one another. Remember, SMP/E does not check how the UCLIN changes might affect the other entries.

- ▶ UNLOAD command

Sometimes you want to make changes to a set of DDDEF entries in the target zone. Instead of using the SMP/E dialog and making changes to each DDDEF entry one by one, you can use the UNLOAD command, which causes SMP/E to unload all the DDDEF entries from the target zone. The output produced is in the form of UCLIN statements, to which you can make the necessary updates for the DDDEF entries.

- ▶ ZONECOPY command

The ZONECOPY command can be used to copy an entire target or distribution zone from an existing CSI data set to another CSI data set. SMP/E copies the data from the input zone to the other CSI and renames the receiving zone. This is useful when you need to set up a new target or distribution zone based on the existing zones.

- ▶ ZONEDELETE command

There are times when you want to delete the information about an old level of the product after installing a new level of product in its own target and distribution zone.

- ▶ ZONERENAME command

Sometimes the current name of a zone does not conform to the naming standard established in your environment and thus you want to assign a new name to the zones. You can do that by using the ZONERENAME command, which allows you to change the name of an existing target or distribution zone.

In this section, we briefly introduce the usage of the LIST and REPORT ERRSYSMODS commands. For more information on the rest of the commands, see *SMP/E Commands*, SA22-7771.

## 5.32 Using the LIST command

### LIST command

➤ Used to display information for selected entries in:

- Global zone, target zone, distribution zone
- SMPPTS
- SMPLOG
- SMPSCDS

*Figure 5-62 Using the LIST command*

### Using the 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 may 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.

To list entries in a CSI data set, you must specify the name of the zone containing the entries to be listed on the SET BOUNDARY command.

To list entries in a data set other than the CSI (such as the SMPLOG or SMPSCDS), you must specify the zone associated with that data set on the SET BOUNDARY command:

► SMPLOG

Specify the zone containing the DDDEF entry for the particular SMPLOG data set to be listed.

► SMPSCDS

Specify the target zone containing the DDDEF entry for the particular SMPSCDS data sets to be listed.

**Note:** Make sure that the data you request to be listed is valid for the specified zone type.

## Listing entries in a particular zone

There are times when you need to check all the DDDEF entries defined in the target zone. You can use the sample JCL shown in Figure 5-63 to perform this task.

```
//LIST1 JOB ('MVSSP',NOTIFY=&SYSUID,CLASS=A,MSGLEVEL=(1,1),
//      MSGCLASS=X
//SMPE EXEC PGM=GIMSMP
//SMPCSI DD DSN=SMPE.GLOBAL.CSI,DISP=SHR
//SMPRPT DD SYSOUT=*
//SMPOUT DD SYSOUT=*
//SYSPRINT DD SYSOUT=*
//SMPCNTL DD *
        SET BDY (MVST100). /* Set to target zone */
        LIST DDDEF . /* List all DDDEF entries */
```

Figure 5-63 Listing entries in a particular zone

If you want to check all the DDDEF entries defined in the global zone and all the zones defined to that global zone, you can use the LIST command shown in Figure 5-64 (the global and any zone defined to it can be used in the SET BOUNDARY command).

```
SET BDY(MVST100) .
LIST DDDEF ALLZONES .
```

Figure 5-64 Listing all DDDEF entries in the global zone and all defined zones

Suppose you need to determine whether target zone MVST100 contains entries for any elements owned by function HXY1100. To do this, use the LIST command with the FORFMID operand, as shown in Figure 5-65.

```
SET      BDY(MVST100)      /* Set to target zone.      */.
LIST     SYSMOD PTFS       /* List SYSMOD that are PTFS */.
        FORFMID(HXY1100)  /* for this FMID.          */.
```

Figure 5-65 Listing SYSMODS that are PTFs for a specific SYSMOD

**Note:** If you just use the SYSMOD operand, SMP/E lists all SYSMOD entries in that zone specified in the SET BOUNDARY command. You can limit which SYSMOD entries are listed by coding one or more SYSMOD qualifiers, such as APARS, PTFS, FUNCTION, and so forth.

## Reports

The following reports are produced during LIST processing:

- ▶ The File Allocation report
- ▶ The LIST Summary report

For more information about these reports, see *SMP/E Commands*, SA22-7771.

## 5.33 Using the REPORT ERRSYSMOD command

### REPORT ERRSYSMOD

- To determine exception SYSMODs
- To determine resolving SYSMODs for held SYSMODs
- Writes SMP/E commands to SMPPUNCH for receiving, applying, and accepting resolving SYSMODs

*Figure 5-66 Using the REPORT ERRSYSMOD command*

### Using the REPORT ERRSYSMODS command

This command helps you to determine whether any SYSMODs you have already installed are now exception SYSMODs. It also helps you to determine whether any resolving SYSMODs are available for held SYSMODs.

For target and distribution zones, REPORT ERRSYSMODS lists installed SYSMODs for which ++HOLD statements were subsequently received and whose error reason IDs have not yet been resolved.

For the global zone, REPORT ERRSYSMODS lists received SYSMODs for which ++HOLD statements with error reason IDs have been received.

**Note:** When using the REPORT ERRSYSMODS command, the SET BOUNDARY command must specify GLOBAL.

### Examples of the REPORT ERRSYSMODS command

The following are some examples showing how you can use the REPORT ERRSYSMODS command to get the information that you want.

Assume you have received the latest HOLDDATA, and you want to know whether it affects any of the SYSMODs that have already been applied to both the target and distribution zone. To do this, use the REPORT ERRSYSMODS, as shown in Figure 5-67 on page 361.



```

SET    BDY(GLOBAL)1 .
REPORT ERRSYSMODS
      ZONES(TZONE1,DZONE1)2
      BEGINDATE(07 01 02)3
      ENDDATE(08 01 02) .

```

Figure 5-67 Checking if ERRSYSMOD affects any already applied SYSMODs

Figure 5-67 notes:

- 1** - Remember you have to specify GLOBAL in the SET BOUNDARY (SET BDY) command when you use the REPORT ERRSYSMODS command.
- 2** - Specify the name of target and distribution zone to report on.
- 3** - The BEGINDATE and ENDDATE indicates the begin and end date of the ++HOLD statement received by SMP/E for the REPORT ERRSYSMODS processing. The format of the date is *mm dd yy*.

There are situations where you are only interested in the effect of the HOLDDATA on a certain function SYSMOD, for example HBB6607. You can use the FORFMID operand to achieve the objective as is shown in Figure 5-68.

```

SET    BDY(GLOBAL)
REPORT ERRSYSMODS
      ZONES(TZONE1)
      FORFMID(HBB6607)1
      BEGINDATE(07 01 02)
      ENDDATE(08 01 02) .

```

Figure 5-68 Checking the effect of ERRSYSMOD on a specific SYSMOD

Figure 5-68 note:

- 1** - Specifying the FORFMID operand will limit the list of SYSMODs to those that have the ++HOLD statement with the FMID specified.

## Reports for REPORT ERRSYSMODs

At the end of the REPORT ERRSYSMODS processing, information about the held SYSMODs and any resolving SYSMODs is written to the Exception SYSMOD report. At the same time, commands needed to install the resolving SYSMODs are provided in the SMPPUNCH data set.

### ► Exception SYSMOD report

This report is produced at the completion of REPORT ERRSYSMODS processing during which exception SYSMOD checking was done and HOLDERROR reason IDs were not resolved for SYSMODs installed in the specified zone.

The report shows the exception SYSMODs that were previously installed, the HOLDERROR reason IDs (APAR numbers) that have made them exception SYSMODs, resolving SYSMODs that have not yet been installed, and the hold class and hold symptoms for each APAR.

The exception SYSMOD reports produced by a given REPORT ERRSYSMODS command are arranged alphanumerically by zone name. Each report begins on a new page. The information gathered for each zone is sorted in ascending order, first by FMID, then by SYSMOD name, then APAR number, and finally, by resolving SYSMOD.

Figure 5-69 shows an example of exception SYSMOD report.

```
***** TOP OF DATA *****
PAGE 0001 - NOW SET TO GLOBAL ZONE          DATE 04/21/99  TIME 10:16:50

EXCEPTION SYSMOD REPORT FOR ZONE MVST100     DATE: 01/01/98 - 03/01/99

      HOLD      SYSMOD      APAR      ---RESOLVING SYSMOD--- HOLD      HOLD
      FMID      NAME        NUMBER    NAME        STATUS RECEIVED CLASS  SYMPTOMS
      -----
1  HMJ4102      UW31189      AN80203  UW32213  GOOD    YES        PE        IPL
2  HQA5140      UW42146      AN90025  UW43610  HELD    NO         PE

PAGE 0002 - NOW SET TO GLOBAL ZONE          DATE 04/21/99  TIME 10:16:5

EXCEPTION SYSMOD REPORT SUMMARY              DATE: 01/01/98 - 03/01/99

      ZONE      FMID          TOTAL APARS3 TOTAL RESOLVING4
              AGAINST FMID      SYMSMODS AGAINST FMID
      -----
MVST100  HMJ4102              1              1
              HQA5140          1              1
```

Figure 5-69 Sample Exception SYSMOD Report

Figure 5-69 notes:

- 1** - UW31189 is an exception SYSMOD for function HMJ4102. The reason for being held is that it is a PE (PTF in error). However, there is a resolving SYSMOD UW32213, which has already been received and has no known problems.
- 2** - UW42146 is a PE for function HQA5140, and the supposed resolving SYSMOD, UW43610, is being held for the error described in APAR AN90025.
- 3** - This indicates the total number of APARs that have error holds against the FMID.
- 4** - This indicates the total number of resolving SYSMODs found against the FMID. The number includes all received APARs and all PTFs (both received and unreceived, held and unheld) against the FMID.

► The SMPPUNCH data set

In order to help you to install resolving SYSMODs for exception SYSMODs, SMP/E writes the necessary commands to the SMPPUNCH data set:

- SET BOUNDARY
- RECEIVE (for unreceived resolving SYSMODs)
- RESETRC (sets the return codes for the preceding commands to zero)
- Either APPLY (for target zones) or ACCEPT (for distribution)

However, there are cases where nothing is written to SMPPUNCH for a specified zone. This occurs when:

- There are no exception SYSMODs for the specified zone.
- There are no resolving SYSMODs for any of the exception SYSMODs, or all resolving SYSMODs identified are held.
- The specified zone is the global zone.
- NOPUNCH operand was specified on the REPORT ERRSYSMODS command.

Figure 5-70 shows an example of SMPPUNCH output.

```
SET BDY (GLOBAL ). /* REMOVE COMMENT IF DOING RECEIVE
RECEIVE SELECT (
    UW50483
    UW50095
    UW49599
    .
    .
    .
    UW55773
    UW56914 )
    SYSMODS.
                                REMOVE COMMENT IF DOING RECEIVE */
RESETRC.
SET BDY (MPRDTZ ).
APPLY SELECT (
    /* UW50483                RESOLVES AW29065 FOR HBB6605 FMID(HBB6605) */
    UW52639 /* PTF            RESOLVES AW29065 FOR HBB6605 FMID(HBB6605) */
    .
    .
    .
    UW53498 /* PTF            RESOLVES AW35984 FOR UW51679 FMID(HPRF220) */
    )
    GROUP.
```

Figure 5-70 Sample SMPPUNCH output

For more information on the LIST, REPORT ERRSYSMODS, and others SMP/E commands, see *SMP/E Commands*, SA22-7771.

## 5.34 SMP/E dialogs

```
----- SMP/E PRIMARY OPTION MENU ----- SMP/E 33.07
===> 0 _

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

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   ===> 'ZOSR06.GLOBAL.CSI'
(Leave blank for a list of SMPCSI data set names.)

Specify YES to have DD statements for SYSOUT and temporary
data sets generated. Specify NO, to use DDDEFS.
Generate DD statements ===> NO
```

Figure 5-71 SMP/E dialogs

### SMP/E dialogs

You can get to the SMP/E dialogs from the SMP/E primary option menu. To provide access to that menu, you must connect the dialogs to ISPF. Sample ISPF panels are provided with z/OS to enable panels for most z/OS elements, including SMP/E. These panels are in the SISPPENU data set after APPLY processing. The panels supplied are:

- |                 |                                                                                                                                                                                           |
|-----------------|-------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| <b>SR@390</b>   | This is an ISPF Primary Options Menu. It is identical to the ISR@PRIM Primary Options Menu, except that it includes the additional options 12 and 13, which point to the next two panels. |
| <b>ISR@390S</b> | This is a secondary panel, with options used by system programmers and administrators, including SMP/E.                                                                                   |
| <b>ISR@390U</b> | This is a secondary menu panel, including the options used by most ISPF users.                                                                                                            |

Use one of the methods documented in the Program Directory for this level of SMP/E to use ISR@390 instead of ISR@PRIM. The SMP/E dialogs will then be accessible through panel ISR@390S.

## 5.35 Customize the SMP/E dialogs

```
====>                                SMP/E DIALOG SETTINGS                                More:      +

Reset to use SMP/E's default settings? ==> NO      (Yes or No)

Enter or verify the information below used to allocate temporary data sets.
These temporary data sets contain generated JCL jobs, output for jobs which
have completed execution, and MCS entries for viewing:

UNIT   ==> SYSALLDA
VOLUME ==>

Enter or verify the information below used to create DD statements in
generated JCL jobs:

Space for SMP/E Utility Work data sets (SYSUTn):
      Block Size Primary Secondary
SYSUT1 3120      380      760
SYSUT2 3120      380      760
SYSUT3 3120      380      760
SYSUT4 3120       38      100

Space for SMP/E Work data sets (SMPWRKn):
      Block Size Primary Secondary Directory Blocks
SMPWRK1 3120      364      380      500
SMPWRK2 3120      364      380      500
SMPWRK3 3120      364      380      500
SMPWRK4 3120      364      380      500
SMPWRK6 3120      364      380      500

Unit for SYSUTn and SMPWRKn data sets:
UNIT ==> SYSDA

Data set name to use for SMPPARM. If a name is specified, an SMPPARM DD
statement will be generated.
DSN ==>

Enter or verify the information below used to allocate permanent SMPWRK3
data sets created and used by the SYSMOD Management dialogs:

UNIT   ==>
VOLUME ==>
PREFIX ==> (TS0 Prefix is used if no prefix is specified)

To save the changes, press ENTER .
To ignore the changes, enter END .
```

Figure 5-72 Customize the SMP/E dialogs

### Customize the SMP/E dialogs

The access by ISPF to the SMP/E dialogs is faster and easier. Through these screens you can obtain functions such as generation of SMP/E jobs and submit of SMP/E processes. Also it is possible using dialogs to find any information about products, for example:

- Modules
- Macros
- Clist
- Which libraries these elements are stored in

Several examples are mentioned here; for complete details refer to the *SMP/E User's Guide*, SA22-7773.

After you install the SMP/E dialogs, you can change the default values they use. When you select option 0 (SETTINGS) on the SMP/E Primary Option Menu, SMP/E displays panel GIM@PARM, which allows you to:

- Modify the values for allocating temporary SMP/E utility (SYSUTn) and SMP/E work (SMPWRKn) data sets. The options you specify are saved permanently in the ISPF profile pool for use later by other SMP/E dialog processes.
- Specify a data set name to be used by SMP/E for the SMPPARM data set during background execution. The SMPPARM data set is used to define exit routines and specify allocation information used by SMP/E processing. If a data set name is specified on panel

GIM@PARM, SMP/E generates an SMPPARM DD statement in all JCL jobs created by the SMP/E dialogs that invoke SMP/E.

### **JOB statement customization**

If you have never entered a JOB statement on panels GIMCGSUB, GIMRCSUB, or GIMSB01, then SMP/E primes those panels with the default JOB statement shown in Figure 5-73.

```
//useridA JOB (ACCOUNT),'NAME'  
/*
```

*Figure 5-73 JOB statement customization*

This initial default JOB statement is not customizable, but when you enter a JOB statement on panels GIMCGSUB, GIMRCSUB, or GIMSB01, the statement you enter is saved in your ISPF profile pool and will be used as the new default when you use those panels again.

## 5.36 Query selection menu

```
====> 1_

1  CSI QUERY          - Display SMPCSI entries
2  CROSS-ZONE QUERY   - Display status of an entry in
                        all zones
3  SOURCEID QUERY     - Display SOURCEIDs for specified zone
D  DESCRIBE           - Overview of using QUERY
T  TUTORIAL           - Information on using QUERY

To return to the SMP/E primary option menu, enter END .
```

Figure 5-74 Query selection menu

### Query selection menu

Let's assume you want to find out which SYSMODs have been applied to a particular target zone on your system. You can accomplish this task using the QUERY SELECTION MENU and selecting the CSI QUERY, option 1, as shown in Figure 5-74.

## 5.37 CSI query panel

CSI QUERY

===>

Specify the zone, entry type, and name to be queried:

ZONE NAME

===> MVST600

Name of the zone to be queried.  
To display a list of all zones,  
leave blank

ENTRY TYPE

===> SYSMOD

Entry type to be queried.  
To display a list of all valid  
entry types, leave ENTRY TYPE  
and ENTRY NAME blank

ENTRY NAME

===> \_

Entry name to be queried.  
Leave blank or use a wildcard  
(entry name pattern) to display  
a selection list.

To return to the Query selection menu, enter END .

Figure 5-75 CSI query panel

### CSI query panel

When the CSI QUERY panel is displayed, you can indicate that you want SMP/E to check target zone MVST600 for all SYSMOD entries.



## 5.38 CSI query - Select entry

```
====>          CSI QUERY - SELECT ENTRY          Row 1 to 26 of 22,097
   SCROLL ==> PAGE

Select one entry to query from TARGET zone MVST600 :

S      NAME      ACTION
AA01511
AA01516
AA01517
AA01535
AA01558
AA01587
AA01614
AA01628
AA01645
AA01655
AA01667
AA01691
AA01718
AA01720
AA01728
AA01739
AA01752
AA01760
AA01767
AA01777
AA01783
AA01801
AA01805
AA01822
AA01827
AA01860
```

Figure 5-76 CSI query - Select entry

### CSI query - Select entry

Because the ENTRY NAME was left blank on the CSI QUERY panel, SMP/E displays another panel, shown in Figure 5-76, that lists all the SYSMOD entries in target zone MVST600.

## 5.39 CSI query - SYSMOD entry

```

                                     CSI QUERY - SYSMOD ENTRY
Row 1 to 1 of 1
===> -                               SCROLL ==> PAGE

To return to the previous panel, enter END .

Primary Command: FIND

Entry Type:  SYSMOD                      Zone Name: MVST600
Entry Name:  ZQ94033                     Zone Type: TARGET
Description:

Type:                               Status:
FMID:                               SUPBY   UK02460
Date/Time:

-----
SUPBY   UQ96160  UQ96309

```

Figure 5-77 CSI query - SYSMOD entry

### CSI query - SYSMOD entry

This panel displays all relevant information about the SYSMOD ZQ94033. As you can see, this SYSMOD is superseded by UK02460; this means that it is contained in or replaced by the SYSMODs UK02460. A superseded SYSMOD (ZQ94033) is functionally lower than the SYSMOD that superseded it. The SYSMOD ZQ94033 was superseded by SYSMODs UQ96160 and UQ96309 before it was superseded by UK02460.

As you can see, the QUERY dialog panels provide a quick and easy way for you to obtain information about your system.

**Note:** The CSI QUERY dialog allows a wildcard (pattern) for the entry name specification since SMP/E Version 3 Release 3. A selection list of all entry names that match the specified pattern will be displayed when using a wildcard. Patterns of the form ABC\* or \*ABC may be specified, where ABC is a string from 0 to 7 characters long.

## 5.40 Building SMP/E jobs using the dialog

```
----- SMP/E PRIMARY OPTION MENU ----- SMP/E 33.07
==> 4 _

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

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 ==> 'ZOSR06.GLOBAL.CSI'
(Leave blank for a list of SMPCSI data set names.)

Specify YES to have DD statements for SYSOUT and temporary
data sets generated. Specify NO, to use DDDEFs.
Generate DD statements ==> NO
```

Figure 5-78 Building SMP/E jobs using the dialog

### Building SMP/E jobs using the dialog

You can use the SMP/E dialog to generate jobs, for RECEIVE, APPLY, LIST, and so forth.

Suppose that you want to list all PTFs that are RECEIVED for an FMID. In the SMP/E Primary Option Menu, select Option 4, Command Generation. You must also enter the SMPCSI DATA SET field, or enter it on the next displayed panel. If you specify the name of the CSI, you receive the panel shown in Figure 5-78.

## 5.41 Command generation - Selection menu

```

                                     COMMAND GENERATION SELECTION MENU

==> 32 _

Select one of the following:
  10 RECEIVE      20 RESETRC      30 LIST BACKUP    40 ZONECOPY
  11 APPLY        21 JCLIN        31 LIST LOG         41 ZONEEDIT
  12 ACCEPT       22 UCLIN        32 LIST              42 ZONEDELETE
  13 REJECT       23 CLEANUP      33 UNLOAD           43 ZONEEXPORT
  14 RESTORE      24 GENERATE      34 REPORT          44 ZONEIMPORT
  15 LINK         25 LOG           35 BUILD MCS        45 ZONEMERGE
                                     46 ZONERENAME
                                     47 GZONEMERGE

Enter or verify the following:
  ZONE NAME      ==> (required)
  OPTIONS NAME   ==> OPTIONS name or
                    blank
  SMP/E PROCESS PARAMETER ==> WAIT WAIT or END

To return to the SMP/E primary option menu enter the END command
```

Figure 5-79 Command generation - Selection menu

### Command generation - Selection menu

Using this panel, SMP/E lists the commands that you can select. To illustrate the use of this panel, we selected option **32**, the LIST command, as shown in Figure 5-79.

The SMP/E data sets contain a great deal of information (the global zone, target zones, distribution zones, SMPPTS, SMPLOG, and SMPSCDS) that you may find useful when installing a new function, preparing a user modification, or debugging a problem. You can use the SMP/E LIST command to display that information.

SMP/E can display all the entries of a specified type (such as MOD, MAC, SYSMOD, and so on), or it can display information for selected entries. In addition, for SYSMOD entries, SMP/E provides some additional operands you can specify to list groups of SYSMODs that meet certain criteria.

In the following six figures, we take you through a set of screens that show the generation of a LIST command.

## 5.42 Command generation - Select zone

```
====>                                COMMAND GENERATION - SELECT ZONE                                Row 1 to 25 of 29
  SCROLL ====> PAGE

Select the zone that is to be processed:
(Enter HELP to view the zone types allowed for each command.)

S   NAME      TYPE      CSI DATA SET
s - GLOBAL    GLOBAL    ZOSR06.GLOBAL.CSI
    MVSD600    DLIB      ZOSR06.MVS.DLIB.CSI
    MVSD610    DLIB      ZOSR06.MVS.DLIB.CSI
    MVSD611    DLIB      ZOSR06.MVS.DLIB.CSI
    MVSD612    DLIB      ZOSR06.MVS.DLIB.CSI
    MVSTA00    TARGET    ZOSR06.MVSTA00.TARGET.CSI
    MVSTA10    TARGET    ZOSR06.MVSTA00.TARGET.CSI
    MVSTA11    TARGET    ZOSR06.MVSTA00.TARGET.CSI
    MVSTA12    TARGET    ZOSR06.MVSTA00.TARGET.CSI
    MVSTB00    TARGET    ZOSR06.MVSTB00.TARGET.CSI
    MVSTB10    TARGET    ZOSR06.MVSTB00.TARGET.CSI
    MVSTB11    TARGET    ZOSR06.MVSTB00.TARGET.CSI
    MVSTB12    TARGET    ZOSR06.MVSTB00.TARGET.CSI
    MVSTC00    TARGET    ZOSR06.MVSTC00.TARGET.CSI
    MVSTC10    TARGET    ZOSR06.MVSTC00.TARGET.CSI
    MVSTC11    TARGET    ZOSR06.MVSTC00.TARGET.CSI
    MVSTC12    TARGET    ZOSR06.MVSTC00.TARGET.CSI
    MVSTD00    TARGET    ZOSR06.MVSTD00.TARGET.CSI
    MVSTD10    TARGET    ZOSR06.MVSTD00.TARGET.CSI
    MVSTD11    TARGET    ZOSR06.MVSTD00.TARGET.CSI
    MVSTD12    TARGET    ZOSR06.MVSTD00.TARGET.CSI
    MVSTE00    TARGET    ZOSR06.MVSTE00.TARGET.CSI
    MVSTE10    TARGET    ZOSR06.MVSTE00.TARGET.CSI
    MVSTE11    TARGET    ZOSR06.MVSTE00.TARGET.CSI
    MVSTE12    TARGET    ZOSR06.MVSTE00.TARGET.CSI
```

Figure 5-80 Command generation - Select zone

### Command generation - Select zone

Select the CSI data set that you want to use; in this example, we selected the GLOBAL CSI.

## 5.43 Command generation - LIST command

```
COMMAND GENERATION - LIST COMMAND

===>

Specify the entry type and name to be listed:

ENTRY TYPE    ===> sysmod    Entry type to be listed.
                                To display a selection list of all
                                valid entry types or to specify
                                multiple entry types, leave ENTRY TYPE
                                and ENTRY NAME blank.

ENTRY NAME    ===> all_      Entry name to be listed.
                                ALL to list all entries.
                                Blank to specify multiple entry names.

Specify additional LIST options:

ALL ZONES     ===> NO        YES to list all zones

XREF OPTION   ===> NO        YES for the XREF option

To continue with LIST request, press ENTER.
To cancel this request, enter END.
```

Figure 5-81 Command generation - LIST command

### Command generation - LIST command

As shown in Figure 5-81, we selected ENTRY TYPE=SYSMOD and ENTRY NAME=ALL.

**Note:** If you leave ENTRY TYPE and ENTRY NAME blank, SMP/E will display a panel with all entry types and will ask you to make a selection.

## 5.44 Command generation - LIST global zone SYSMOD options

```
====>          COMMAND GENERATION - LIST GLOBAL ZONE SYSMOD OPTIONS

Enter YES to limit the SYSMODs or MCS to be processed:
FUNCTIONS      ====> NO    FUNCTION type SYSMODs
PTFS           ====> yes   PTF      type SYSMODs
APARS          ====> NO    APAR     type SYSMODs
USERMODS       ====> NO    USERMOD  type SYSMODs

ERROR          ====> NO    SYSMODs marked in ERROR

HOLDERROR      ====> NO    SYSMODs held for HOLDERROR
HOLDSYSTEM     ====> NO    SYSMODs held for HOLDSYSTEM
HOLDUSER       ====> NO    SYSMODs held for HOLDUSER

SOURCEID       ====> NO    SYSMODs for selected SOURCE-IDs
EXSRCID        ====> NO    SYSMODs without selected SOURCE-IDs
FORMID         ====> yes   _SYSMODs for selected FMIDs

NOACCEPT       ====> NO    NO or DLIB zone name
NOAPPLY        ====> NO    NO or TARGET zone name

To ignore the data entered above and return to previous
panel, enter END .
```

Figure 5-82 Command generation - LIST global zone SYSMOD options

### Command generation - LIST global zone SYSMOD options

In the List Global Zone Option, we selected PTFs by entering YES (because we want to list all PTFs that are received) and FORMID as YES (because we want to restrict the search for only two FMIDs).

## 5.45 Command generation - List FORFMID

```
====>          COMMAND GENERATION - LIST FORFMID          Row 1 to 10 of 10
   SCROLL ==> PAGE

The FORFMID operand of the LIST command indicates that
only those SYSMODS applicable to the FMIDs or FMIDSETs
specified in the FORFMID list are eligible for selection.

Enter the names of FMIDs or FMIDSETs to be included.
When the list is complete, enter END .

      FMIDs/FMIDSETs
' ' ' ' HCS7708_
' ' ' ' JWSZ203_
' ' ' ' _____
' ' ' ' _____
' ' ' ' _____
' ' ' ' _____
' ' ' ' _____
' ' ' ' _____
' ' ' ' _____
' ' ' ' _____
```

Figure 5-83 Command generation - List FORFMID

### Command generation - List FORFMID

You can select one or more FMIDs in the panel shown in Figure 5-83. When you complete the list of names of FMIDs or FMIDSETs, enter END on the command line.



## 5.46 Command generation - Selection menu

```
====> end_

                                COMMAND GENERATION SELECTION MENU

Select one of the following:
 10 RECEIVE      20 RESETRC      30 LIST BACKUP    40 ZONECOPY
 11 APPLY        21 JCLIN        31 LIST LOG        41 ZONEEDIT
 12 ACCEPT       22 UCLIN        32 LIST             42 ZONEDELETE
 13 REJECT       23 CLEANUP      33 UNLOAD          43 ZONEEXPORT
 14 RESTORE      24 GENERATE      34 REPORT          44 ZONEIMPORT
 15 LINK         25 LOG           35 BUILD MCS        45 ZONEMERGE
                  26 UPGRADE                        46 ZONERENAME
  47 GZONEMERGE

Enter or verify the following:
ZONE NAME          ===>          (required)
OPTIONS NAME       ===>          OPTIONS name or
                               blank
SMP/E PROCESS PARAMETER ===> WAIT WAIT or END

To make additional selections enter selection and press ENTER
To EDIT, BROWSE, or SUBMIT generated jobs enter the END command
To leave without submitting any job enter the CANCEL command

The LIST command was created based on your input
```

Figure 5-84 Command generation - Selection menu

### Command generation - Selection menu

After you finish your selection, SMP/E will issue the message The LIST command was created based on your input, as shown in Figure 5-84.

Now, the job that you want is generated by SMP/E. If you want any other service of SMP/E, you can go to the panel again and select another command. The second command will be generated in the same job as the previous one. When you do not have any other changes to make, enter END on the command line.

## 5.47 Command generation - SUBMIT

```
                                COMMAND GENERATION - SUBMIT

===> e_

Select one of the following:
  E - EDIT the job that was generated
  B - BROWSE the job that was generated
  S - SUBMIT the job that was generated

Enter or modify the JOB statement.
===> //GRECCOJ JOB (ACCOUNT),'NAME'
===> /**
===> /**
===> /**

To end this dialog without submitting the job,
enter END . ( NOTE: The job stream is not saved.)
```

Figure 5-85 Command generation - SUBMIT

### Command generation - SUBMIT

After you issue the END command, the panel shown in Figure 5-85 is displayed.

In this panel you can *edit the job*, option **E**, or *submit the job*, option **S**. When you submit the job, the job executes and you can then browse the output to obtain the information you requested.

## 5.48 The generated job

```
File Edit Edit_Settings Menu Utilities Compilers Test Help
EDIT GRECC0.SC55.SPFTEMP1.CNTL Columns 00001 00072
Command ==> Scroll ==> PAGE
***** ***** Top of Data *****
000001 //S1 EXEC PGM=GIMSMP,
000002 // PARM='PROCESS=WAIT',
000003 // DYNAMNBR=120
000004 //*
000005 //* NOTE: THIS JCL CREATED BY THE COMMAND GENERATION DIALOGS.
000006 //*
000007 //* SMP ZONE-RELATED FILES ARE DYNAMICALLY ALLOCATED,
000008 //* THIS INCLUDES THE SMPPTS, SMPLOG, AND SMPDLIB DATA SETS,
000009 //* IF APPLICABLE.
000010 //*
000011 //* SMP FILES
000012 //*
000013 //SMPCSI DD DISP=SHR,DSN=ZOSR06.GLOBAL.CSI
000014 //*
000015 //*
000016 //SMPCNTL DD *
000017 SET BOUNDARY (GLOBAL)
000018 .
000019 LIST PTFS
000020 FORMID
000021 (
000022 HCS7708
000023 JWSZ203
000024 )
000025 SYSMOD .
***** ***** Bottom of Data *****
```

Figure 5-86 The generated job

### The generated job

Figure 5-86 shows the job generated by SMP/E; you can submit it and it will execute the commands that you chose.

This is the end of our LIST command example.

## 5.49 Java archive (JAR) file support

- ❑ SMP/E will provide support for both
  - **JAR replacement files**
    - complete replacement for a JAR file
    - constructed in JAR file format
    - treated very much like all other ++hfs elements
    - ++JAR
  - **JAR update files**
    - contain only new and changed component files
    - constructed in JAR file format
    - use jar command to 'extract' the component files from a JAR update file and to 'update' a JAR file with the component files
    - ++JARUPD

Figure 5-87 Java ARchive (JAR) file support

### Java archive (JAR) file support

SMP/E Version 3 Release 2 or z/OS V1R5 has introduced new element types to describe JAR files and SMP/E will replace and update such files. The ++JAR element type is used to add and replace entire JAR files and the ++JARUPD element type is used to provide an update for a JAR file.

To update a JAR file, SMP/E will add or replace component files within a Java Archive, rather than replacing the entire Java Archive file. This allows for much smaller and more granular PTFs for products that use Java and JAR files. JAR files reside in directories within the UNIX file system, and SMP/E will use the Java **jar** command to update a JAR file when processing a ++JARUPD element.

The JAR file format is based on the popular ZIP file format and is used for aggregating many files into one. Although JAR can be used as a general archiving tool, the primary motivation for its development was so that Java applets and their requisite components (.class files, images, and sounds) can be downloaded to a browser in a single HTTP transaction, rather than opening a new connection for each piece. This greatly improves the speed with which an applet can be loaded onto a Web page and begin functioning. The JAR format also supports compression, which reduces the size of the file and improves download time still further. These attributes make the JAR file format the preferred method to bundle Java applets and applications.

Fundamental operations on a JAR file are performed using the Java Archive Tool provided as part of the Java Development Kit (JDK™). The Java Archive Tool is invoked using the **jar**

command. The **jar** command is used to create JAR files, view and extract the contents of JAR files, and update the contents of JAR files. To update the contents of a JAR file means to replace a subset of the files within the archive, add additional files to the archive, or both.

**Note:** IBM Developer Kit for OS/390, Java 2 Technology Edition is required for SMP/E to perform JARUPD processing. See Java 2 on the OS/390 and z/OS Platforms for more information.

## 5.50 GIMZIP and GIMUNZIP service routines

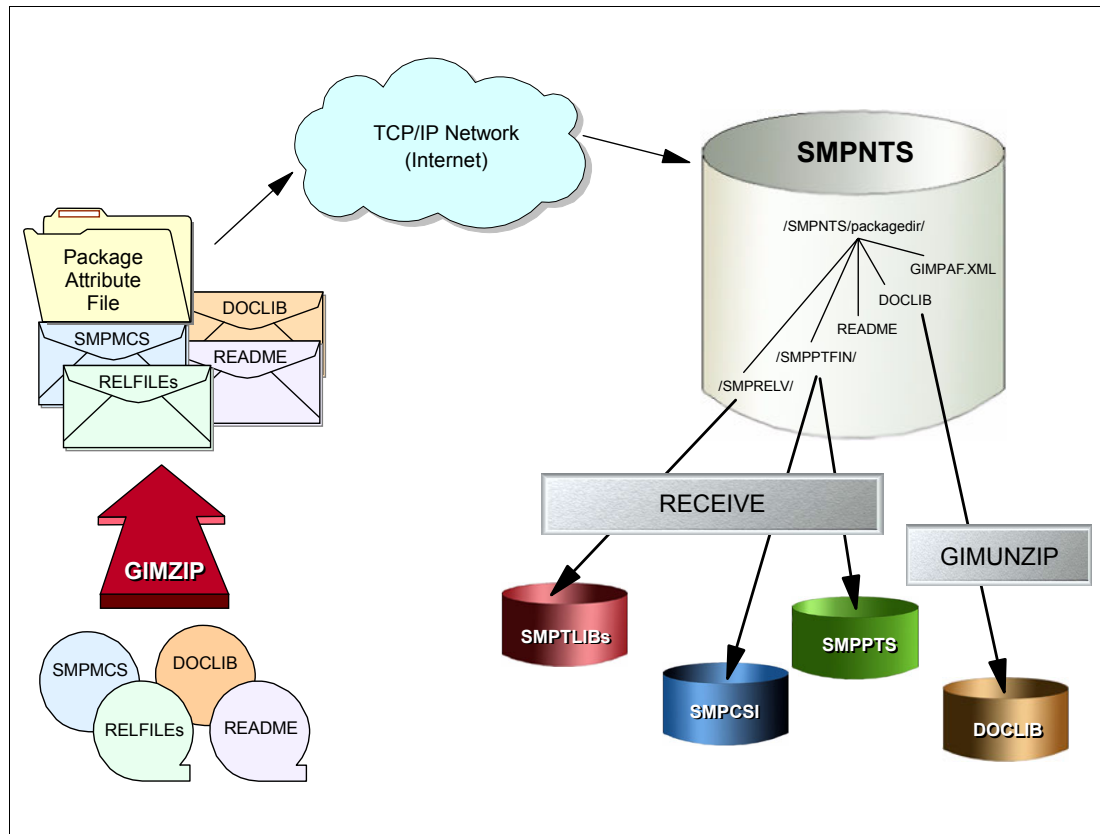


Figure 5-88 GIMZIP and GIMUNZIP service routines

### GIMZIP packaging service routine

The GIMZIP service routine creates portable packages of software and associated materials. Typically the packages will contain SYSMODs, RELFILE data sets, HOLDDATA, and associated materials such as documentation, samples, and text files. These GIMZIP packages can be transported through a network, processed by the GIMUNZIP service routine, and then processed by the SMP/E RECEIVE command. A single GIMZIP package typically consists of several archive files.

More specifically, a GIMZIP package consists of a single package definition file, a set of archive files, and text files. The package definition file describes the total package and identifies the archive files and text files contained in the package. The GIMZIP package file produced is stored in the package directory in a UNIX file system. The package directory is specified by the SMPDIR DD statement. The GIMZIP service routine processes the following data structures as input to create GIMZIP packages:

- ▶ A portable image of any of the following:
  - Sequential data set
  - Partitioned data set
  - VSAM data set
  - A file in the UNIX file system
  - A directory in the UNIX file system
- ▶ The information necessary to reload the data from the portable image

**Note:**

- ▶ GIMZIP/GIMUNZIP are separate load modules residing in the MIGLIB library and runs independently from the rest of SMP/E processing.
- ▶ Introduced into SMP/E Version 3 Release 3 or z/OS V1R6, the GIMZIP/GIMUNZIP service routines have been enhanced to allow packages to also contain VSAM ESDS, KSDS, LDS, and RRDS data sets, and UNIX files and directories.

GIMZIP collects various attributes for the input data sets and files specified in the package control tags. These attributes, along with portable images of the input data sets, are temporarily stored in a UNIX file system. These images are then archived and compressed to create archive files. Each GIMZIP package will contain one or more archive files.

For each package there are two additional files:

- ▶ The *package attribute file*, which describes the contents of the package itself. The package attribute file identifies the archives for a package and contains a hash value for each archive within the package. The *hash value* for each archive is used for data integrity purposes and can be checked when the original data set is re-created from the archive using the GIMUNZIP service routine.
- ▶ An *extensible stylesheet language (XSL) file*, which describes how to format the information found in the package attribute file. The XSL file is used for rendering (displaying on a browser) the package attribute file.

## Archive files

The actual software and associated materials for a package are stored in archive files. GIMZIP creates an archive file for each input data set or file specified by a <FILEDEF> tag. An archive file consists of a portable image of the original data set or file and attributes of the original data set or file needed to reload the data from the portable image.

The attributes of the original data set or file are recorded in a file attribute file in the UNIX file system called GIMFAF.XML. For data sets, the portable image of the original data set is stored temporarily as a file in the UNIX file system (this step is not required for UNIX files and directories). The name of this temporary file is the type attribute value for the archive (SMPPTFIN, SMPHOLD, SMPRELF) or the name MVSFILE if the archive has no type attribute specified. An archive file is then created by using the UNIX System Services **pax** command to combine the GIMFAF.XML file and the original data into an archive. Although a UNIX file is not first stored as a temporary file with a name matching its type attribute value, its name within the archive will be its type attribute value, or the name MVSFILE, if the archive has no type attribute specified. The GIMUNZIP service routine and the RECEIVE command both expect an archive file to contain the GIMFAF.XML component file, and the following, depending on the original content:

- ▶ For data sets, a file named SMPPTFIN, SMPHOLD, SMPRELF, or MVSFILE.
- ▶ For UNIX, files named SMPPTFIN, SMPHOLD, or MVSFILE.
- ▶ For UNIX directories, the directory and its contents are stored in the archive using the original names.

Data sets and files with a type of README are an exception to the previously described archive processing. README data sets and files have no file attribute file and are not archived using **pax**, but rather are stored unchanged in a file in the package directory. Likewise, README data sets and files are not subject to archive segmentation.

The absolute names for archive and README files in the package directory use the following format: /package\_directory/subdir/Snnnn.original\_\_name.pax.Z

The absolute names for archive segment files in the package directory use the following format: /package\_directory/subdir/date\_timeofday.nofm

- ▶ *package\_directory* indicates the package directory specified on the SMPDIR DD statement.
- ▶ *subdir* indicates the subdirectory into which the archive file is stored. GIMZIP creates this subdirectory based on the filetype or the subdir attributes specified on the <FILEDEF> tag for the data set. If neither attribute is specified, then no subdirectory is used. If the file type is README, the subdirectory is created only if the subdir attribute has been specified.

**Note:** This option is available in SMP/E Version 3 Release 2 or z/OS V1R5.

- ▶ *Snnnn* specifies the sequence indicator for the archive file. The sequence indicator is necessary so the correct order can be determined for the SMPHOLD and SMPPTFIN files when eventually processed by the SMP/E RECEIVE command. GIMZIP processes the input data sets or files in the sequence in which they are specified in the package control tags and the indicator is assigned accordingly. All archive files, except those specified with a file type of SMPRELF, are assigned a sequence indicator. If the file type is README and an archid tag was specified, then no sequence number will be assigned.
- ▶ *original\_name* specifies the original name of the data set, file, or directory, as indicated on the <FILEDEF> tag. This could also be the archid for UNIX directories and README data sets or files, or the date and time for UNIX directories, if the archid is not specified.
- ▶ *pax.Z* is the file extension for all archive files, except those with a file type of README. The pax.Z extension indicates a file that has been processed by the UNIX System Services **pax** command with the compress option.
- ▶ *date\_timeofday* is the date and time of day value, unique for each archive, but the same for each segment file of a particular archive.
- ▶ *n* is the segment number.
- ▶ *m* is the total number of segments for the archive.

**Note:** Since SMP/E Version 3 Release 2, large archive files within GIMZIP packages may be divided into smaller segments. The GIMZIP Service Routine has a new SEGMENT option to allow you to specify the maximum desired size of archive segment files in megabytes. GIMZIP will then divide large archives into multiple smaller physical files, thus enabling more efficient network transport and retry operations on the packages. GIMUNZIP service routines have been updated to support processing of packages with segmented archive files.

## File Attribute File

The File Attribute File (FAF) is included in the archive file along with the data set, file, or directory when it is archived. The FAF contains control tags that:

- ▶ Describe the attributes of the original data set, file, or directory
- ▶ Provide information needed by GIMUNZIP to process the archive



### Example of using GIMZIP

Suppose a GIMZIP package is to be created containing the following data sets, files, and directories:

1. /SAMPLE/ORDER123/readme.html - a README file
2. /SAMPLE/ORDER123/SMPMCS - an SMP/E MCS file
3. SAMPLE.IBM.FMID001.F1 - an SMP/E PDS relative file
4. SAMPLE.IBM.FMID001.F2 - an SMP/E PDSE relative file
5. SAMPLE.ORDER123.DOCLIB - a PDS containing documents
6. SAMPLE.ORDER123.RIMLIB - a PDS containing related installation materials
7. SAMPLE.ORDER123.MVS.GLOBAL.CSI - a VSAM cluster
8. /SAMPLE/ORDER123/RootHFS/ - the root directory

The job stream on Figure 5-89 can be used to create such a GIMZIP package. This package directory resides in the UNIX file system and it specified by the SMPDIR DD statement. The package directory name is:

```
' /u/smpe/GIMZIP/ORDER123/ '
```

```

//JOBx      JOB ...
//STEP1     EXEC PGM=GIMZIP,PARM='SEGMENT=12M'
//SMPDIR    DD PATH='/u/smpe/GIMZIP/ORDER123/',PATHDISP=KEEP
//SMPDOUT   DD SYSOUT=*
//SYSPRINT  DD SYSOUT=*
//SYSUT2    DD UNIT=SYSALLDA,SPACE=(CYL,(200,20))
//SYSUT3    DD UNIT=SYSALLDA,SPACE=(CYL,(50,10))
//SYSUT4    DD UNIT=SYSALLDA,SPACE=(CYL,(25,5))
//SYSIN     DD *
<GIMZIP description="This is a sample software package.">
<FILEDEF name="/SAMPLE/ORDER123/readme.html"
archid="readme.html"
description="This is the README file for the package."
subdir="README"
type="README">
</FILEDEF>
<FILEDEF name="/SAMPLE/ORDER123/SMPMCS"
description="This is the SMPMCS file for ORDER123."
archid="SMPMCS"
type="SMPPTFIN">
</FILEDEF>
<FILEDEF name="SAMPLE.IBM.FMID001.F1"
archid="FMID001.F1"
description="This is SMP/E RELFILE 1 for FMID001."
type="SMPRELF">
</FILEDEF>
<FILEDEF name="SAMPLE.IBM.FMID001.F2"
archid="FMID001.F2"
description="This is SMP/E RELFILE 2 for FMID001."
type="SMPRELF">
</FILEDEF>
<FILEDEF name="SAMPLE.ORDER123.DOCLIB"
subdir="DOCLIB"
description="This is the Document Library for the package.">
</FILEDEF>
<FILEDEF name="SAMPLE.ORDER123.RIMLIB"
subdir="RIMLIB"
description="This is the Related Installation Materials
Library for the package.">
</FILEDEF>
<FILEDEF name="SAMPLE.ORDER123.MVS.GLOBAL.CSI"
archid="GLOBAL"
description="This is a sample VSAM cluster.">
</FILEDEF>
<FILEDEF name="/SAMPLE/ORDER123/RootHFS/"
description="This is the entire root directory.">
</FILEDEF>
</GIMZIP>
/*

```

Figure 5-89 GIMZIP sample job

**Note:** For more information about the GIMZIP service routine, see *SMP/E Reference*, SA22-7772.

## GIMUNZIP file extraction service routine

The GIMUNZIP service routine is used to extract data sets, files, and directories from archive files in GIMZIP packages created by the GIMZIP service routine. These packages typically contain software and associated materials in the form of SYSMODs, RELFILE data sets, HOLDDATA, and other materials such as documentation, samples, and text files. These GIMZIP packages may be transported through a network, processed by the GIMUNZIP service routine, and then processed by the SMP/E RECEIVE command.

More specifically, the GIMUNZIP service routine extracts data sets, files, and directories from the archive files that compose the GIMZIP package. An archive file consists of a portable image of a sequential, partitioned, or VSAM data set, or a file or directory in a UNIX file system, and the information needed to create that data set, file, or directory from the portable image. The data set, file, or directory into which the archive file is to be extracted can already exist or GIMUNZIP can create a new one of the appropriate type. New sequential and partitioned data sets created by GIMUNZIP are always catalogued.

**Note:** Formerly the GIMUNZIP service routine would only extract data from an archive file within a GIMZIP package into a new data set allocated directly by GIMUNZIP. In SMP/E Version 3 Release 3 the GIMUNZIP has been enhanced to support extracting archive file data into existing data sets (new <ARCHDEF> tag option *REPLACE*).

GIMUNZIP uses the UNIX System Services **pax** command to expand the following component files from an archive file temporarily into a UNIX file system:

- ▶ The portable image of the original data set or file
- ▶ The file attribute file that contains the information necessary to reload the data from the archived data set or file

GIMUNZIP then reads the file attribute file and uses the information recorded there, along with information specified on the <ARCHDEF> tag, such as volume, newname, or prefix, to allocate a new or existing data set or file. The portable image of the original data set or file is stored in the data set or file just allocated for that purpose.

When GIMUNZIP encounters a package that contains archive segments, GIMUNZIP reassembles the archive segments into the original archive form, optionally verifies the integrity of the archive, and recreates the original data.

If the HASH=YES option was specified, then GIMUNZIP performs SHA-1 hash checking for the archive files. Specifically, GIMUNZIP reads the package attribute file for the package. The package attribute file, GIMPAF.XML, is found in the package directory, and contains the known hash values for the archives. These hash values were recorded by GIMZIP when the package and archives were created. GIMUNZIP then uses ICSF services to compute the current hash values for the archive files, and compares this value to the known hash values for the archive files.

If the computed hash value matches the known hash value, then the integrity of the archive file is ensured. However, if the computed hash value does not match the known hash value, GIMUNZIP processing stops. This condition indicates the archive file has been corrupted since it was produced by GIMZIP when the package was created.

## Example of using GIMUNZIP

Suppose a GIMZIP package contains a set of archive files and you want to extract the data sets and files from those archive files. In addition, you wish to verify the integrity of the archive files and rename the destination data sets and files in the process. The job stream in Figure 5-90 on page 388 can be used to perform such an operation.

```

//JOBx      JOB ...
//STEP1     EXEC PGM=GIMUNZIP,PARM="HASH=YES"
//SMPDIR    DD PATH='/u/smpe/ORDER123/',PATHDISP=KEEP
//SMPDOUT   DD SYSOUT=*
//SYSPRINT  DD SYSOUT=*
//SYSUT3    DD UNIT=SYSALLDA,SPACE=(CYL,(50,10))
//SYSUT4    DD UNIT=SYSALLDA,SPACE=(CYL,(25,5))
//SYSIN     DD *
<GIMUNZIP>
<ARCHDEF archid="SMPMCS"
replace="YES"
preserveid="YES"
newname="/userid/IBM/D0000029/SMPMCS">
</ARCHDEF>
<ARCHDEF archid="FMID001.F1"
newname="userid.IBM.FMID001.F1">
</ARCHDEF>
<ARCHDEF archid="FMID002.F2"
newname="userid.IBM.FMID002.F2">
</ARCHDEF>
<ARCHDEF archid="GLOBAL"
prefix="userid.IBM">
</ARCHDEF>
<ARCHDEF name="S0006.2003030335435443234.pax.Z"
replace="YES"
newname="/userid/IBM/D0000029/RootHFS/">
</ARCHDEF>
</GIMUNZIP>
/*

```

Figure 5-90 GIMUNZIP sample job

**Note:** For more information about the GIMUNZIP service routine see *SMP/E Reference*, SA22-7772. For more information about the **pax** command see *z/OS UNIX System Services Command Reference*, SA22-7802.

## Network delivery of SMP/E input

SMP/E Version 3 Release 1 can receive input from a network server, in addition to tape and DASD. This enables the delivery of SMP/E-installable products and service over the internet or an intranet. By installing software directly from a network source, SMP/E enables a more seamless integration of electronic software delivery and installation. This reduces the tasks and time required to install software delivered electronically.

SMP/E provides the GIMZIP and GIMUNZIP service routines to construct, and then later unwrap, network transportable packages of software. Specifically, the GIMZIP service routine will create a network transportable package as output. Once a package is made accessible on an FTP server, you can use the SMP/E RECEIVE command to transfer the package through a TCP/IP network directly into an SMP/E environment.

The RECEIVE command has been extended with new DELETEPKG, FROMNETWORK, and FROMNTS operands to process these network transportable packages. In SMP/E Version 3 Release 2 or z/OS V1R5, the RECEIVE FROMNETWORK and FROMNTS commands have been updated to support processing of packages with segmented archive files and subdirectories that may be produced by GIMZIP. SMP/E releases prior to SMP/E V3R2 cannot process GIMZIP output that contains segmented archive files. A toleration PTF will

issue an error message should a user try to process GIMZIP output that contains segmented archive files on a release of SMP/E prior to V3R2. For more information on the RECEIVE command changes, see SMP/E Commands, SA22-7771.

New CLIENT and SERVER data sets and SMPDIR and SMPNTS directories have been created to support this new processing. For more information on CLIENT, SERVER, SMPDIR, and SMPNTS, see SMP/E Reference, SA22-7772.

Figure 5-91 is a sample RECEIVE job that would process the SMPMCS and RELFILE data sets and files extracted by GIMUNZIP in the example shown in Figure 5-90.

```
//JOBx      JOB ...
//STEP1     EXEC PGM=GIMSMP
//SMPCSI    DD DSN=SMPE.GLOBAL.CSI,DISP=SHR
//SMPPTFIN  DD DSN=USERID.FMID001.SMPMCS,DISP=SHR
//SMPCTL    DD *
SET BDY(GLOBAL).
RECEIVE SYSMODS
          RFPREFIX(USERID).
/*
```

*Figure 5-91 Sample RECEIVE job for GIMUNZIP*

## 5.51 GIMXSID service routine

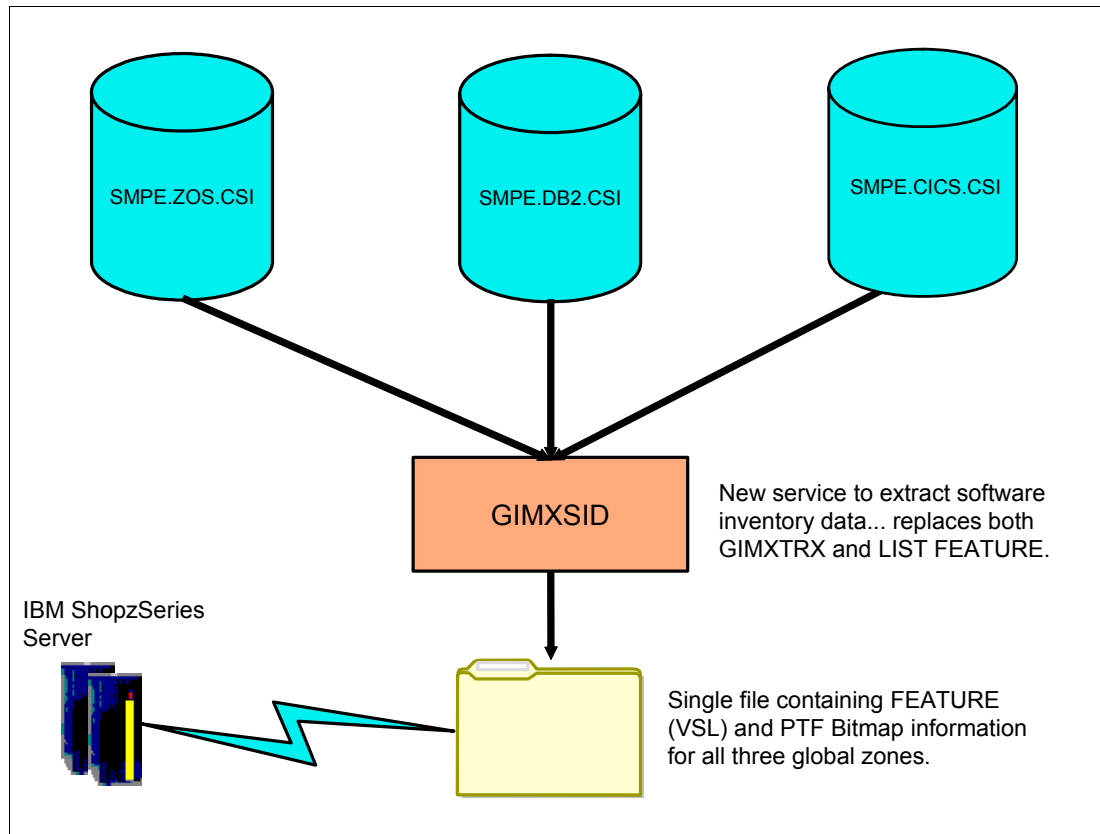


Figure 5-92 GIMXSID service routine

### GIMXSID service routine

GIMXSID is an SMP/E service routine to be used as part of the ShopzSeries offering. GIMXSID creates a single data source required by ShopzSeries to place customized software product and service orders. The data source created by GIMXSID, the software inventory data, is a composite of three kinds of information as follows:

- ▶ Feature List

List of FEATURES found in the SMPCSI data sets. The Feature List is used by ShopzSeries to perform product requisite checking and also to prime the order checklist when ordering a ServerPac.

- ▶ FMID List

A list of the FMIDs found in the SMPCSI data sets. The FMID List is used by ShopzSeries to scope service orders to the PTFs applicable solely to the user's desired configuration of target and global zones.

- ▶ PTF Bitmap

Bitmap representation of the PTFs found in the specified target zones and global zones. The PTF Bitmap is used by ShopzSeries to produce service packages that do not contain PTFs that are already present in the user's configuration.

GIMXSID is an independent load module residing in the MIGLIB library. GIMXSID runs independently from the rest of SMP/E processing.

SMP/E Version 3 Release 2 provides a new service routine, GIMXSID, to simplify and consolidate the data collection tasks necessary to place ShopzSeries software product and service orders. GIMXSID can be used to collect the software inventory product (FEATURE) and service (FMID and PTF) information for specified target zones, and in one or more global zones.

ShopzSeries provides a self-service planning and ordering capability on the Web. You can order system/product upgrades for either ServerPac or CBPDO. ShopzSeries includes the following significant functions:

- ▶ Profiling of the checklist. ShopzSeries will analyze and prefill the order checklist when you submit your existing installed inventory to the tool.
- ▶ Full automation to process orders that do not require license changes. Orders that include new or changed licenses will be routed to the IBM Sales Center.
- ▶ Access to software inventory reports through the Customer Analysis Tool.
- ▶ Ordering of service-only CBPDOs.

ShopzSeries support for maintenance includes ordering corrective service by PTFs or APARs and Recommended Service Upgrade (RSU) preventive service. In addition, service orders can be received on traditional media or via the Internet. For an RSU order, you can upload your SMP/E CSI installed service inventory, which is used to customize your order and eliminate duplicate service.

Figure 5-93 shows a sample of how to use GIMXSID. In this example, the ZOS14 and JES314 target zones are processed from the global zone in the SMPE.ZOSREAL.GLOBAL.CSI data set. In addition, all target zones from the global zone in the SMPE.DB2REAL.GLOBAL.CSI data set are processed. The software inventory output can be written to a data set or to a file in the UNIX file system.

```
//job      JOB ...
//BITMAP   EXEC PGM=GIMXSID
//SMPOUT   DD SYSOUT=*
//SMPXTOUT DD DSN=userid.GIMXSID.OUTPUT,DISP=(NEW,CATLG),
//          SPACE=(TRK,(1,1)),UNIT=SYSALLDA
//SYSIN    DD *
CSI=SMPE.ZOSREAL.GLOBAL.CSI
TARGET=ZOS14,JES314
CSI=SMPE.DB2REAL.GLOBAL.CSI
/*
```

Figure 5-93 GIMXSID sample JCL

## 5.52 SMP/E release levels

New releases of SMP/E must sometimes make changes to SMP/E data sets that cannot be properly processed by prior SMP/E releases. SMP/E usually makes incompatible changes only when necessary to provide new and improved capabilities. For example, a new type of element requires a new entry type in SMPCSI data sets and these new entry types are typically not understood or processed correctly by SMP/E levels that have not been specifically updated to do so.

The SMP/E element of z/OS is usually not updated with every release of z/OS. For example, the level of SMP/E that was shipped with z/OS Release 1 is identical to that shipped with OS/390 V2R7.

However, it is important to be aware of which release is being installed due to the changes implemented from release SMP/E V3R1, which generate incompatibilities with zones generated in previous releases. See *SMP/E User's Guide*, SA22-7773 for details.

### SMP/E Version 3 Release 1

The following is a summary of the enhancements and changes that have been incorporated in SMP/E 3.1. Detailed information can be found in the SMP/E manuals.

- ▶ SMPPTS spill data set support

SMP/E RECEIVE processing can use SMPPTS spill data sets, if defined, to store SYSMODs when the primary SMPPTS data set is full. Up to 99 spill data sets, named SMPPTS1 through SMPPTS99, can be defined with DD statements or DDDEFs. By eliminating the tasks involved when recovering from an overflowing SMPPTS data set, the use of SMPPTS spill data sets can reduce the amount of manual intervention and data set management required to install software service.

- ▶ RECEIVE from a network location

SMP/E can now receive input from a network server, in addition to tape and DASD. This enables the delivery of SMP/E-installable products and service over the internet or an intranet. It provides the GIMZIP and GIMUNZIP service routines to construct, and then later unwrap, network-transportable packages of software.

- ▶ HOLDDATA reports during APPLY and ACCEPT

SMP/E now provides additional HOLDDATA reports for APPLY and ACCEPT processing.

- ▶ Hierarchical File Input/Output support

SMP/E now allows the data sets SMPCNTL, SMPDEBUG, SMPHOLD, SMPJCLIN, SMPLIST, SMPOUT, SMPPTFIN, SMPPUNCH and SMPRPT to reside in the hierarchical file system.

- ▶ HFS data set identification

The SMP/E File Allocation Report and SMP/E library change file records have been enhanced to identify the physical HFS data sets where files in the hierarchical file system reside.

- ▶ Support for longer HFS LINK values

SMP/E has increased the maximum length allowed for a hierarchical file system element LINK value from 64 characters to 1023 characters. SMP/E has also increased the maximum length allowed for the alias values on ALIAS link-edit control statements from 64 characters to 1023 characters.



- Conditional JCLIN processing

SMP/E now allows a packager to use special JCL comments in the JCLIN input to cause SMP/E to skip over parts of the JCLIN input based on the installation environment. The parts of the JCLIN input that are skipped are not processed by the JCLIN command and do not contribute to the structure information derived from JCLIN processing.

- QUERY dialogs

The QUERY dialogs have been extended to include more information. Specifically, subentry fields have been added on various panels for LEPARM, RMIDASM, LASTUPD, LASTUPDTYPE, FESN, and JCLIN present indicators.

- RECEIVE CBPDO dialogs

The RECEIVE CBPDO dialogs have been enhanced to allow an unlimited number of additional CBPDO tape volumes to be specified. The IPOPRINT program used to print DOCLIB and PGMDIR files has been replaced with TSO PRINTDS. Note that TSO PRINTDS does not support asterisk (\*) as a valid SYSOUT class.

- ++PRODXML data element

A new data element type, ++PRODXML, has been added to SMP/E. The ++PRODXML data element is used for XML documents that describe a PRODUCT. This new data type is processed like any other element by SMP/E.

- Removal of function to create backup IEANUC01 (load modules)

The ability to save the target system's nucleus load module (IEANUC01) during APPLY, LINK, and RESTORE command processing has been removed from SMP/E.

- Replacements for GIMMPDFT and GIMMPUXD

If you previously used module GIMMPDFT to define allocation information for temporary data sets, you must now use member GIMDDALC in the SMPPARM data set; module GIMMPDFT no longer exists.

If you previously used module GIMMPUXD to define SMP/E exit routines, you must now use member GIMEXITS in the SMPPARM data set; module GIMMPUXD no longer exists.

## **SMP/E Version 3 Release 2**

The following is a summary of the enhancements and changes that have been incorporated in SMP/E V3R2. Detailed information can be found in the SMP/E manuals.

- A smaller SMPLTS data set

The SMPLTS data set in prior releases of SMP/E is very large and is used to manage all load modules that have a CALLLIBS subentry (load modules that exploit the link edit autocall facility).

In this version, the SMPLTS data set is used only for load modules that contain cross-zone modules and use CALLLIBS. If a load module uses CALLLIBS but does not contain any cross-zone modules, SMP/E rebuilds the load module from scratch during link edit operations, rather than saving a version of the load module in the SMPLTS data set. The result is a much smaller, or possibly empty, SMPLTS data set.

SMP/E will delete unneeded load modules from the SMPLTS data set during APPLY, RESTORE, and CLEANUP command processing. However, since such an SMPLTS data set will not be compatible with prior releases of SMP/E, this cleanup of existing load modules will occur only after you use the new UPGRADE command to indicate your desire to exploit new functions.

- ▶ **REPORT CALLLIBS replaced by LINK LMODS**

The LINK LMODS command replaces the REPORT CALLLIBS command. The LINK LMODS command provides all of the functions of the REPORT CALLLIBS command with some additional benefits. The LINK LMODS command refreshes the content of load modules by rebuilding them from scratch, and any load module can be specified by name or by reference to specific CALLLIBS ddnames. The LINK LMODS command will perform the link edit operations directly for the load modules instead of producing a JCL job to perform the link edit operations. In addition, SMP/E will attempt recovery, if requested, from out-of-space conditions encountered during those link edit operations.

- ▶ **The UPGRADE command**

The UPGRADE command allows you to make the trade-off between fully exploiting new SMP/E functions and preserving compatibility with prior SMP/E releases. The UPGRADE command authorizes SMP/E to exploit the new functions that make incompatible changes to SMP/E data sets, and you must use the UPGRADE command before exploiting such functions.

New SMP/E functions must sometimes make changes to SMP/E data sets that cannot be properly processed by prior SMP/E releases. For example, a new type of element requires a new entry type in SMPCSI data sets and these new entry types are typically not understood or processed correctly by SMP/E levels that have not been specifically updated to do so. Rather than arbitrarily making such incompatible changes on its own, SMP/E will continue to use processing that is compatible with prior releases until you use the UPGRADE command.

- ▶ **Java Archive (JAR) update support**

This version has introduced new element types to describe Java Archive (JAR) files, and SMP/E will replace and update such files. The ++JAR element type is used to add and replace JAR files, and the ++JARUPD element type is used to provide an update for a JAR file.

To update a JAR file means to add or replace component files within a Java Archive, rather than replacing the entire Java Archive file. This allows much smaller and more granular PTFs for products that use Java and JAR files.

JAR files reside in directories within the hierarchical file system and SMP/E will use the Java `jar` command to update a JAR file when processing a ++JARUPD element.

Note: Java 2 Technology Edition is required for SMP/E to perform JARUPD processing.

- ▶ **New SETTINGS option in SMP/E dialogs**

The new SETTINGS option provides an interactive method to configure settings for a single user in the SMP/E dialogs. These settings affect how JCL jobs are generated and how temporary and permanent data sets are allocated by the dialogs. The values entered or changed are saved permanently, for use later by other SMP/E dialog processes.

The SETTINGS option is a replacement for non-display panel GIM@UPRM. If you previously used panel GIM@UPRM to customize these settings, you must now use the SETTINGS option.

- ▶ **Side Deck Library DUMMY support**

SMP/E Version 3 Release 2 allows product developers to use a DUMMY data set specification for the SIDE DECK LIBRARY (SYSDEFSD) of DLL load modules.

During link edit operations, the SIDE DECK LIBRARY defines the data set where the binder will write IMPORT control statements for each symbol exported by a DLL load module. If the resultant file of IMPORT statements should not be retained, the product

developer can now use a DUMMY data set specification for the SIDE DECK LIBRARY, rather than defining a permanent target library to contain the unused side deck file.

The SIDE DECK LIBRARY is defined by the product developer in the JCLIN link edit step that defines the DLL load module.

- Edit step that defines the DLL load module

The GIMZIP Service Routine creates portable packages of software and related materials. In SMP/E Version 3 Release 2, users have more control over the structure of GIMZIP packages.

Large archive files within GIMZIP packages can be divided into smaller segments. The GIMZIP Service Routine optionally accepts, as an execution parameter, the maximum desired size of archive segment files in megabytes. GIMZIP will then divide large archives into multiple smaller physical files, thus enabling more efficient network transport and retry operations on the packages.

In addition, subdirectories can be specified in which to group and store documentation, samples, readme files, and other types of data within GIMZIP packages. The subdirectory is created in the hierarchical file system, within the package directory specified on the SMPDIR DD statement.

The RECEIVE FROMNETWORK and FROMNTS commands, as well as the GIMUNZIP Service Routine, have been updated to support processing of packages with segmented archive files and subdirectories that may be produced by GIMZIP.

- New data collection routine for ShopzSeries (GIMXSID)

SMP/E Version 3 Release 2 provides a new service routine, GIMXSID, to simplify and consolidate the data collection tasks necessary to place ShopzSeries software product and service orders. GIMXSID can be used to collect the software inventory product (FEATURE) and service (FMID and PTF) information for specified target zones, and in one or more global zones.

Figure 5-94 is an example of how to use GIMXSID. The software inventory output can be written to a data set or to a file in the hierarchical file system.

```
//job      JOB ...
//BITMAP   EXEC PGM=GIMXSID
//SMPOUT   DD SYSOUT=*
//SMPXTOUT DD DSN=userid.GIMXSID.OUTPUT,DISP=(NEW,CATLG),
//          SPACE=(TRK,(1,1)),UNIT=SYSALLDA
//SYSIN    DD *
CSI=data.set.name
TARGET=tzone1, tzone2, ...
/*
```

Figure 5-94 GIMXSID sample JCL

- SYSLIB allocated only when needed for assemblies

The SYSLIB concatenation is used by the assembler during APPLY, ACCEPT, and RESTORE command processing to find macros for assembly operations. In prior releases of SMP/E the data sets in the SYSLIB concatenation were allocated by SMP/E unconditionally during APPLY, ACCEPT, and RESTORE command processing, even if no assembly operations were performed.

In SMP/E Version 3 Release 2 the data sets in the SYSLIB concatenation will be allocated only if an assembly operation will be performed during the APPLY, ACCEPT, and RESTORE commands. This reduces SMP/E processing to allocate the data sets and eliminates possible contention for the data sets.

► Migration and the UPGRADE command

The new UPGRADE command is key for migrating to the latest release of SMP/E. The UPGRADE command allows you to make the trade-off between fully exploiting new SMP/E functions and preserving compatibility with prior SMP/E releases.

If you do not use the UPGRADE command, then your SMP/E zones and data sets will remain compatible with prior SMP/E releases. However, once you use the UPGRADE command, you authorize SMP/E to exploit new functions and to make incompatible changes to SMP/E zones and data sets, such as deleting unneeded load modules from the SMPLTS data set. Once such changes are made, those zones and data sets may not be properly processed by SMP/E releases that have not been specifically updated to do so.

► Removal of GIMUTTBL, GIMDFUT and GIM@UPRM

If you previously used macro GIMDFUT or module GIMUTTBL to restrict the utility programs that SMP/E could use and you want to maintain those restrictions, you must now use the z/OS Security Server (RACF) by creating a profile for the program in the PROGRAM general resource class. Macro GIMDFUT and module GIMUTTBL no longer exist in SMP/E Version 3 Release 2. Defining profiles for programs in the PROGRAM class controls who (which userids) can execute the programs.

If you previously used panel GIM@UPRM to customize settings for the SMP/E dialogs and you wish to keep those customized settings, you must now use the new SETTINGS option from the SMP/E Primary Option Menu. Panel GIM@UPRM no longer exists in SMP/E Version 3 Release 2. Using the new SETTINGS option means the information is persistent and that you do not need to update panel GIM@UPRM every time a new release of SMP/E is installed.

### SMP/E Version 3 Release 3

The following is a summary of the enhancements and changes that have been incorporated in SMP/E V3R3. Detailed information can be found in the SMP/E manuals.

► RECEIVE FROMNETWORK command has been changed.

The RECEIVE FROMNETWORK command has been changed to use the z/OS Communications Server FTP client. This allows SMP/E to:

- Provide secure transfer between the client and FTP server of the user ID and password, as well as the file data. A *secure transfer* refers to encryption and authentication using Transport Layer Security (TLS).
- Access remote FTP servers through a SOCKS firewall server.
- Access FTP servers using IPv6 addressing.
- Use the FTP.DATA configuration data set to specify local site parameters for the FTP client. The FTP.DATA configuration data set is optional, but must be used to specify the options for secure transfer and SOCKS firewall navigation. See the z/OS Communications Server IP User's Guide and Commands book for details.

Note: z/OS Communications Server is required in order to enable secure transfers, SOCKS firewall navigation, and IPv6 addressing.

► GIMZIP packages contents.

GIMZIP packages now can contain UNIX directories and files and VSAM data sets. Formerly GIMZIP packages could contain archives only for sequential and partitioned data sets. The GIMZIP and GIMUNZIP service routines have been enhanced to allow

packages to also contain VSAM ESDS, KSDS, LDS, and RRDS data sets, and UNIX files and directories.

Additionally, when building a package using GIMZIP, a unique ID value can be assigned to each archive. The ID value can then be used to identify an archive that is to be processed by GIMUNZIP, as opposed to using the archive's file name.

- GIMUNZIP can extract archives into existing data sets.

Formerly the GIMUNZIP service routine would only extract data from an archive file within a GIMZIP package into a new data set allocated directly by GIMUNZIP. GIMUNZIP has been enhanced to support extracting archive file data into existing data sets.

GIMUNZIP will now determine if the output data set already exists. If the data set already exists, GIMUNZIP copies the data from the archive file into the existing data set. If the data set does not already exist, GIMUNZIP allocates a new data set and then copies the data from the archive file into that new data set.

- SMP/E assigns RECEIVE command SOURCEID value to existing SYSMODs.

The RECEIVE command has been enhanced to assign the source ID specified on the SOURCEID operand of the command to SYSMODs found in the SMPPTFIN input stream, even if they are already received. Formerly, the source ID was not assigned to SYSMODs that are already received.

- SMP/E uses COPYMOD to copy load modules.

SMP/E now uses the COPYMOD control statement in conjunction with the SPCLCMOD and CMWA copy execution parameters when the copy utility is invoked to copy load modules. This results in the following:

- Any load module that is reblockable will be reblocked.
- Any load modules that cannot be reblocked, but can be copied without causing fat blocks, will be copied without complaint (RC=0).
- Any load modules that cannot be reblocked and cannot be copied without causing fat blocks will not be copied and the copy operation will fail with RC=8.

The result is that more SYSMODs should be applied and accepted without problems and with better space utilization of load module data sets. Additionally, the installation of load modules that are likely to cause a problem in the future is stopped until the user takes a corrective action (such as increasing the block size of the destination library).

- CSI Query allows entry name wildcard.

The CSI QUERY dialog allows a wildcard (pattern) for the entry name specification. A selection list of all entry names that match the specified pattern will be displayed when using a wildcard.

- Patterns of the form ABC\* or \*ABC may be specified, where ABC is a string from 0 to 7 characters long.
- An entry type specification is required when using an entry name wildcard.
- Only one wildcard character (\*) is allowed in the entry name specification.

- REJECT command accepts CHECK operand.

The CHECK operand is now allowed on the REJECT command. The CHECK operand indicates whether REJECT should perform a trial run of the command without actually updating any zones or data sets. This provides a way to test for errors that might occur during actual processing and to receive reports on the changes that would be made.

- SMP/E provides a new service routine to transfer GIMZIP packages.

A new service routine, GIMGTPKG, has been created to get GIMZIP packages independently of the RECEIVE FROMNETWORK command. Using GIMGTPKG, you can transport a GIMZIP package from a remote FTP server to a local z/OS system. The files in the GIMZIP package are stored in the SMPNTS directory for use later by the RECEIVE FROMNTS command or other applications and offerings.

Figure 5-95 shows a sample of how to use GIMGTPKG. Refer to the *SMP/E Reference*, SA22-7772, for more details.

```
//job      JOB ...
//GTPKG    EXEC PGM=GIMGTPKG
//SMPOUT   DD SYSOUT=*
//SYSPRINT DD SYSOUT=*
//SMPNTS   DD PATH='/u/userid/smpnts/',PATHDISP=KEEP
//SMPSRVR  DD *
           required FTP server and package information
/*
//SMPCLNT  DD *
           optional z/OS client information
/*
```

Figure 5-95 GIMGTPKG sample JOB

- RECEIVE FROMNETWORK command may require updates to CLIENT data set.

SMP/E now uses the z/OS Communications Server FTP client to perform package transfers and therefore, no longer directly controls all aspects of those operations as once specified in the RECEIVE FROMNETWORK CLIENT data set.

- pasv and keepalive attributes

If you previously used the *pasv* or *keepalive* attributes in the CLIENT data set for RECEIVE FROMNETWORK operations, you must now use the FWFriendly and FTPKEEPALIVE options in the FTP.DATA data set. If the pasv or keepalive attributes are found in the CLIENT data set they will be ignored.

If required, the FWFriendly and FTPKEEPALIVE options must be specified in the FTP.DATA data set instead. See the *z/OS Communications Server IP User's Guide and Commands* for more information about all the options available in the FTP.DATA data set.

- Firewall commands

If you previously specified FTP commands to navigate your local firewall using the <FIRECMD> tag of the CLIENT data set, then you may need to update them. Specifically, subcommands such as USER, PASS, and ACCT should no longer be specified in <FIRECMD>.

The commands you specify in the <FIRECMD> tags should be the same as those you use with the z/OS Communications Server FTP client, and should contain only the actual values (or the appropriate substitution variables) for user ID, password, and account.

## 5.53 SMP/E hints

Following are some SMP/E hints we learned in everyday experience:

- ▶ SMP/E is a product for management and documentation of software installation. SMP/E is a powerful tool, with many options for searches and with many ways of being used. Therefore, it is extremely important that before initiating the installation of any product with SMP/E, you should consult the specific manual about product installation and follow the instructions that they recommend. Suggestions offered in this publication are based in IBM software. Other software manufacturers may have other recommendations.
- ▶ When you are updating an SMP/E zone, be sure you have executed the backup procedures of the product data sets and also the SMP/E zones.
- ▶ When the product that you are installing or applying maintenance on uses HFS, pay close attention to the recommendations in *z/OS UNIX System Services Planning*, GA22-7800.







# Language Environment

This chapter introduces you to the Language Environment architecture, a system of constructs and interfaces that provides a common run-time environment and run-time services for all Language Environment-conforming programming language products (those products that adhere to Language Environment's common interface).

This chapter contains an overview of Language Environment, descriptions of Language Environment's full program model, callable services, storage management model, and debug information.

Language Environment provides a common run-time 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 run-time environment. POSIX-conforming C applications can use all Language Environment services.

Language Environment combines essential and commonly used run-time services, such as routines for run-time 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 run-time 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 (LE)

- ❑ Assembler and high level languages (HLL) concepts
- ❑ IBM compiler products and LE standards
- ❑ Components
- ❑ Run time environment
- ❑ Callable services
- ❑ Program management terms and terminology
- ❑ Program management model
- ❑ Condition handling model and terminology
- ❑ How conditions are represented
- ❑ Condition handling stack frame
- ❑ Condition handling signaling and responses
- ❑ Calling a LE service from assembler program
- ❑ Run time options customization
- ❑ User exits

*Figure 6-1 LE topics*

### Language environment

Today, enterprises need efficient, consistent, and less complex ways to develop quality applications 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 “High-level language and programs” on page 406.

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.

Figure 6-1 lists the topics covered in this chapter.

## 6.2 Assembler language and programs

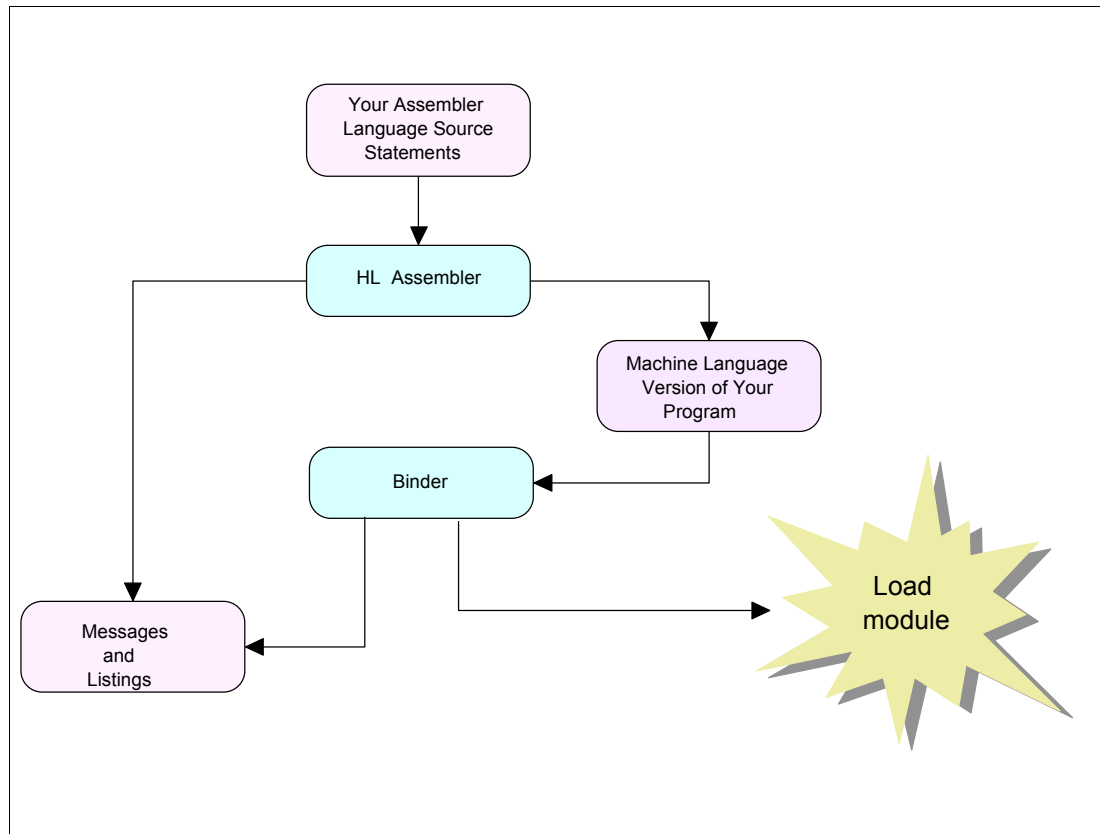


Figure 6-2 Assembler language

### Assembler language

A computer can understand and interpret only machine language. Machine language is in binary form and, thus, is very difficult to write. The assembler language is a symbolic programming language that you can use to code instructions instead of coding in machine language.

Because the 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, you can make your coding easier to read, understand, and change. Assembler is also enriched by a huge set of macros that make it a very powerful language; however, its major quality is the performance of its object code at run time.

One of the key aspects of a language is its readability and capacity to document itself. However, certain modern languages like C do not follow this rule. Then, we can say that if you write an assembler program using all the documenting capability of its features, chances are that you can have more readable code than some HLLs permit.

The assembler must translate the symbolic assembler language into machine language before the computer can run your program. The specific procedures followed to do this may vary according to the system you are using. However, the method is basically the same for all systems and consists of the following:

Your program, written in the assembler language, becomes the *source module* that is input to the assembler. 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 symbolic 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 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 if they were part of the original input in the source module.

IBM supplies macro definitions for input/output, 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 high level assembler operates under the z/OS operating system. This operating system provides the assembler with services for:

- ▶ Assembling a source module

- ▶ Running the assembled object module as a program

In writing a source module, you must include instructions that request any required service functions from the operating system.

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 very difficult to write an assembler language program using only machine instructions. The assembler provides additional functions, not discussed here, that make this task easier.

## 6.3 High-level language and programs

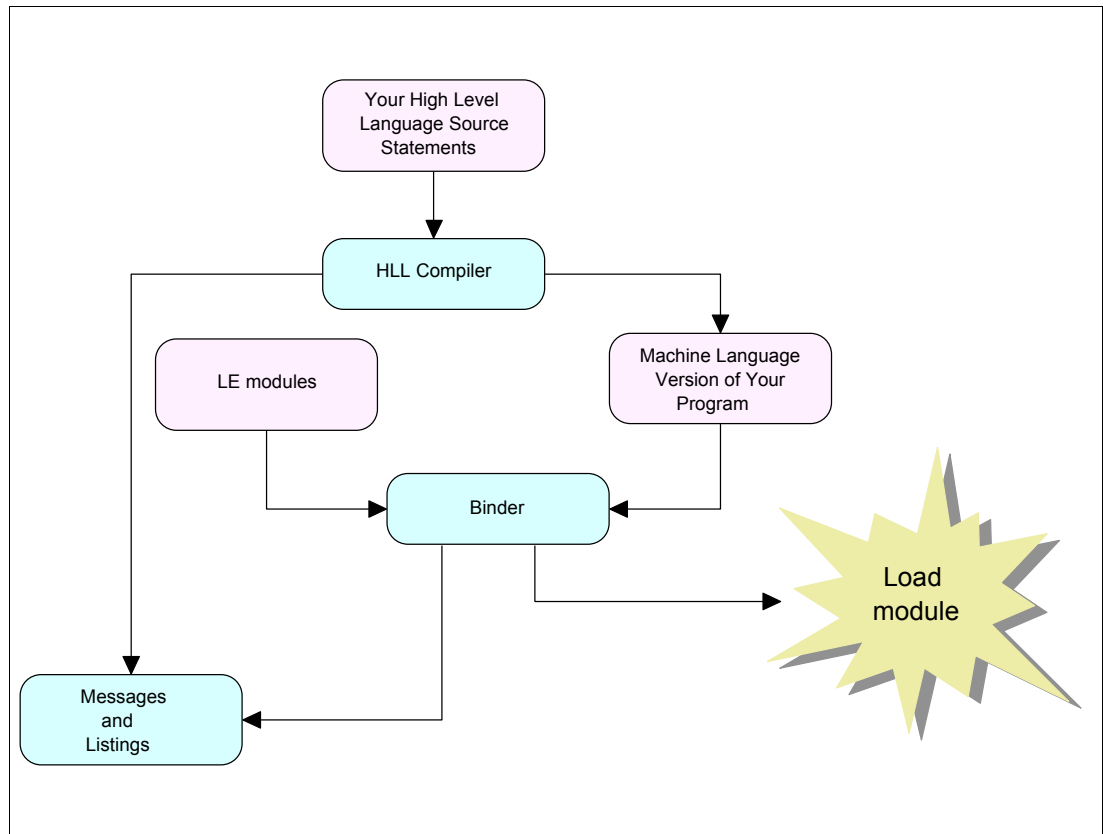


Figure 6-3 High-level language

### High-level language

A high-level language (HLL) is a *programming language* above the level of assembler language and below that of program generators and query languages. 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 high-level language in this chapter, we are referring to any of the following:

- ▶ C or C++
- ▶ COBOL
- ▶ FORTRAN
- ▶ PL/I

## 6.4 IBM compiler products and LE

- ☐ z/OS C/C++
- ☐ C/C++ Compiler for MVS/ESA(TM)
- ☐ AD/Cycle® C/370(TM) Compiler
- ☐ VisualAge for Java, Enterprise Edition for OS/390
- ☐ Enterprise COBOL for z/OS and OS/390
- ☐ COBOL for OS/390 & VM
- ☐ COBOL for MVS & VM (formerly COBOL/370)
- ☐ Enterprise PL/I for z/OS and OS/390
- ☐ VisualAge PL/I for OS/390
- ☐ PL/I for MVS & VM
- ☐ AD/Cycle PL/I for MVS & VM
- ☐ VS FORTRAN and FORTRAN IV (in compatibility mode)

Figure 6-4 IBM compiler products and LE

### IBM compiler products and LE

z/OS Language Environment is the prerequisite run-time environment for applications generated with the IBM compiler products shown in Figure 6-4.

IBM Enterprise COBOL for z/OS is IBM's strategic COBOL compiler for the zSeries platform and System 390. Enterprise COBOL is comprised 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. Enterprise COBOL, as well as IBM COBOL and VS COBOL II, supports the COBOL 85 Standard. Some features, such as the CMPR2 compiler option and SOM-based object-oriented COBOL syntax that IBM COBOL supported are no longer available with Enterprise COBOL.

Language Environment provides a single language run-time 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 run-time 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.

## 6.5 LE standards

- ❑ UNIX is not an open system
- ❑ C is the HLL of the open world
- ❑ Standards committees and products:
  - IEEE with Portable Operating System Interface (POSIX)
  - XPG, a set of companies for computer standards
  - ISO/IEC
- ❑ z/OS and Language Environment conforms to:
  - POSIX 1003.1
  - XPG4.2 SPEC 1170
  - ISO/IEC 9945

Figure 6-5 LE standards

### LE standards

z/OS UNIX is not a real “open system” because it has many different implementations that are incompatible with one another (more than 70 UNIX variants populate the market). Also, the source code was delivered together with the product, which makes each copy potentially unique in itself.

The UNIX market is very competitive, so each software house adds other functions to the kernel. This creates “UNIX proprietary” code. To address the problem, standards like Portable Operating System Interface (POSIX) were introduced to define interfaces based on UNIX.

The IEEE POSIX standard is a series of industry standards for code and user interface portability. POSIX support allows applications written for a UNIX-like operating system to be run on z/OS. C language programmers can access operating system services through a set of standard language bindings. C language programmers who install z/OS UNIX System Services (z/OS UNIX) and z/OS Language Environment can call C language functions defined in the POSIX standard from their C applications and can run applications that conform to ISO/IEC 9945-1:1990. Through C interfaces, Language Environment functions conform to XPG4.2 specifications and are branded by X/Open.

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.



## 6.6 LE components

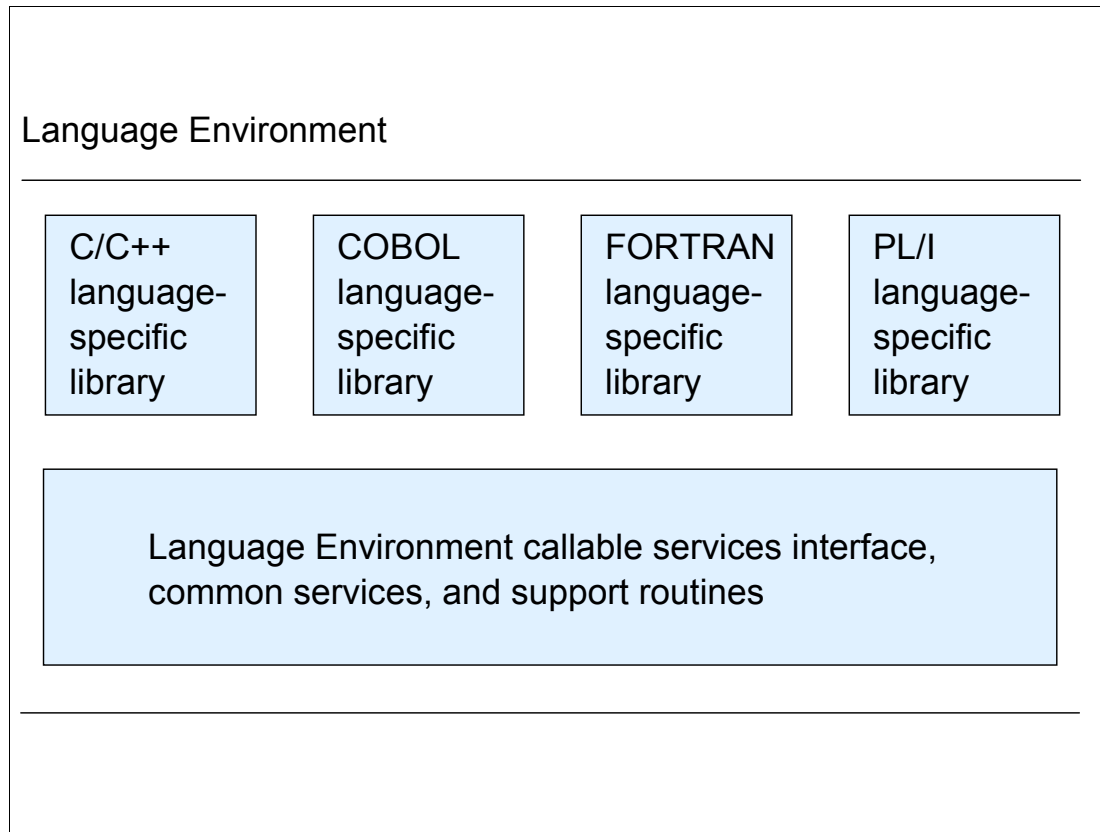


Figure 6-6 LE components

### LE components

Figure 6-6 shows the separate components that make up Language Environment.

Language Environment consists of:

- ▶ **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 run-time library** because many language-specific routines call Language Environment services. However, behavior is consistent across languages.

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 run-time library is available.

## 6.7 LE common run-time environment

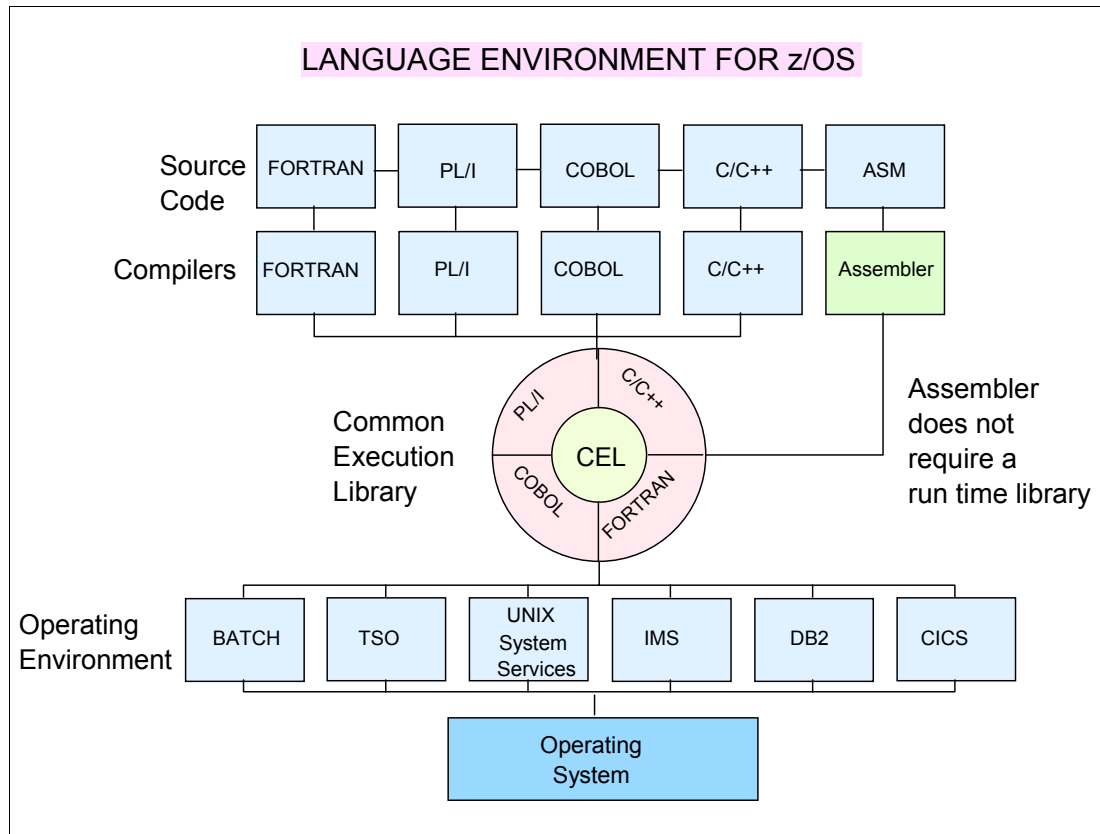


Figure 6-7 LE common run-time environment

### LE run-time environment

Figure 6-7 illustrates the common environment that Language Environment creates. It also shows that each HLL has its specific run-time and SYSLIB libraries, and shares with other HLLs a Common Execution Library (CEL). Figure 6-7 further shows that the load modules produced in this way can be executed in different operating environments under z/OS or VM/ESA.

## 6.8 LE run-time environment for AMODE 64

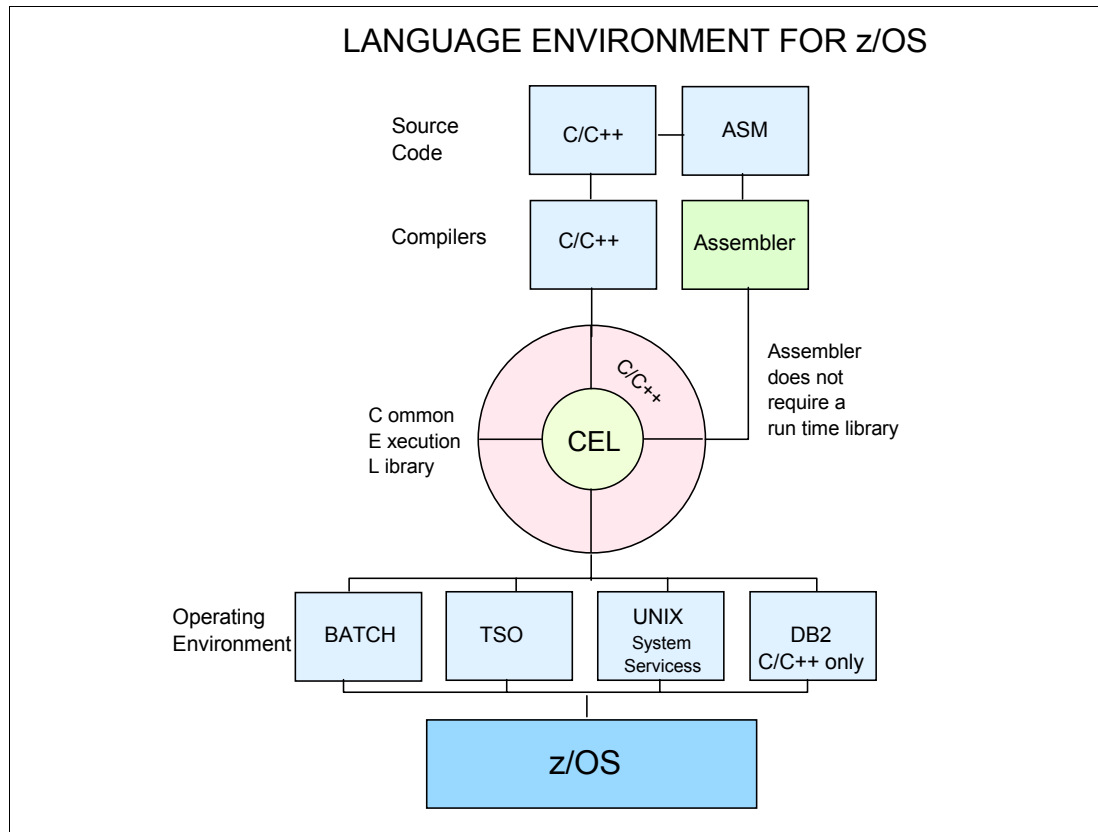


Figure 6-8 LE common run-time environment for AMODE 64

### Run-time for AMODE 64

Figure 6-8 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.

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. While this is an extremely high address, there are a few very important facts to consider:

- ▶ 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 run-time options to support AMODE 64 applications. The new run-time options primarily support the new stack and heap storage located above the bar. All other existing run-time 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.

**Note:** Language Environment-conforming AMODE 64 applications cannot run on any release prior to z/OS Version 1 Release 6.

## 6.9 Object code and LE

- ❑ The object code is included in the load module in three different times in z/OS:
  - At compile time
  - At binder time
  - At execution time
- ❑ LE run time environment
  - Libraries that contain code
- ❑ HLL and LE
  - Common run time environment
  - Common set of SYSLIB libraries

*Figure 6-9 Object code and LE*

### Object code and LE

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 run-time environment. Language Environment establishes a common run-time 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.

## 6.10 Callable services

- ❑ **LE callable services**
  - **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**

*Figure 6-10 Language environment callable services*

### 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.

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. 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 "CEEFMDT" 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 Figure 6-10 are for AMODE 31 only. For AMODE 64, none of the application writer interfaces (AWIs) will be supported in their present form. There may be C functions that provide similar functionality for some of the AWIs. A few non-standard 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.

## 6.11 LE program management terms and terminology

- ❑ General programming terms
  - Application program
  - Environment
- ❑ Terms and their HLL equivalents
  - Routine
  - Enclave
  - Process
  - Thread
- ❑ Terminology for data
  - Automatic
  - External
  - Local

Figure 6-11 LE program management terms and terminology

### Program management

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, run-time 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

Following are some 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 run-time environment of HLLs at the enclave level.



## LE terms and HLL equivalents

Following are some LE terms and HLL equivalents:

- ▶ Routine

In Language Environment, this refers to either a procedure, function, or subroutine.

Equivalent 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 run-time path within the Language Environment program management model, and is dispatched by the system with its own run-time stack, instruction counter, and registers. Threads may exist concurrently with other threads.

## Terminology for data

The following terminology describes the types of data used in an LE 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.

## 6.12 LE program management model

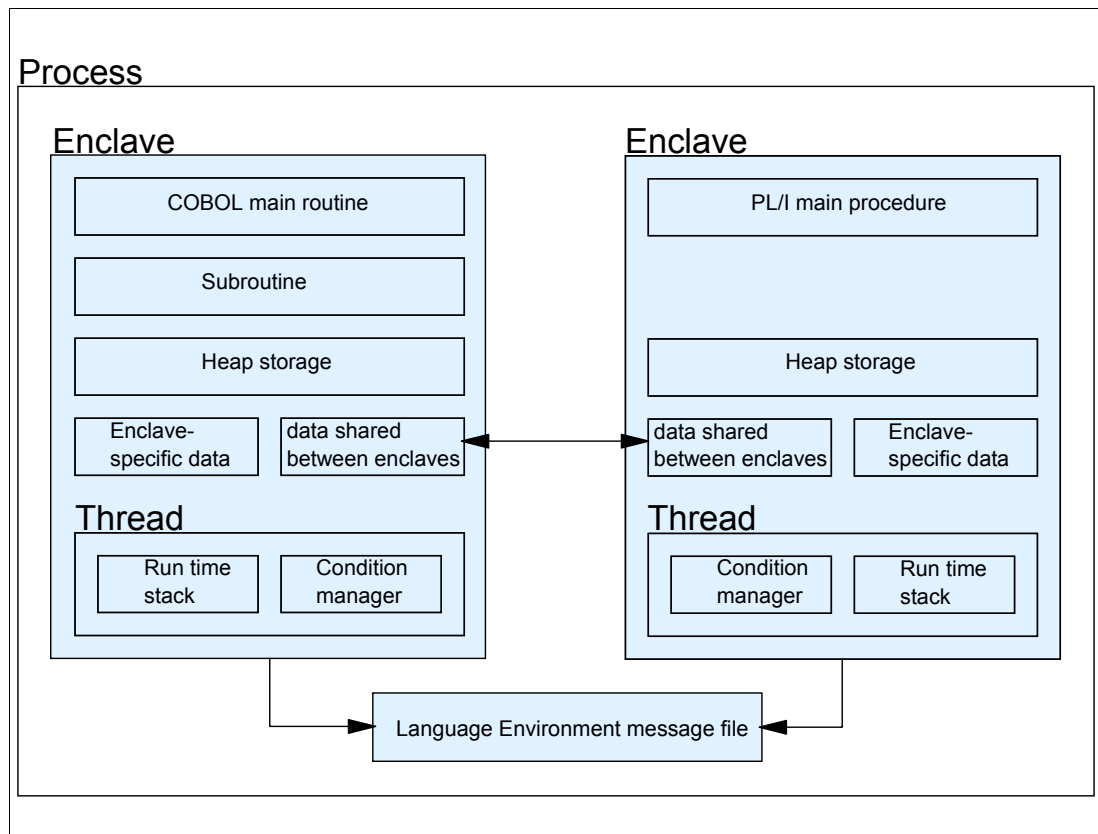


Figure 6-12 LE program management

### Program management model

Three entities — process, enclave, and thread — are at the core of the Language Environment program management model.

Figure 6-12 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.

### 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 since 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 run-time 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 run-time stacks, they can run concurrently within an enclave and allocate and free their own storage. Because they may execute concurrently, threads can be used to implement parallel processing applications and event-driven applications.

## 6.13 LE program management full model

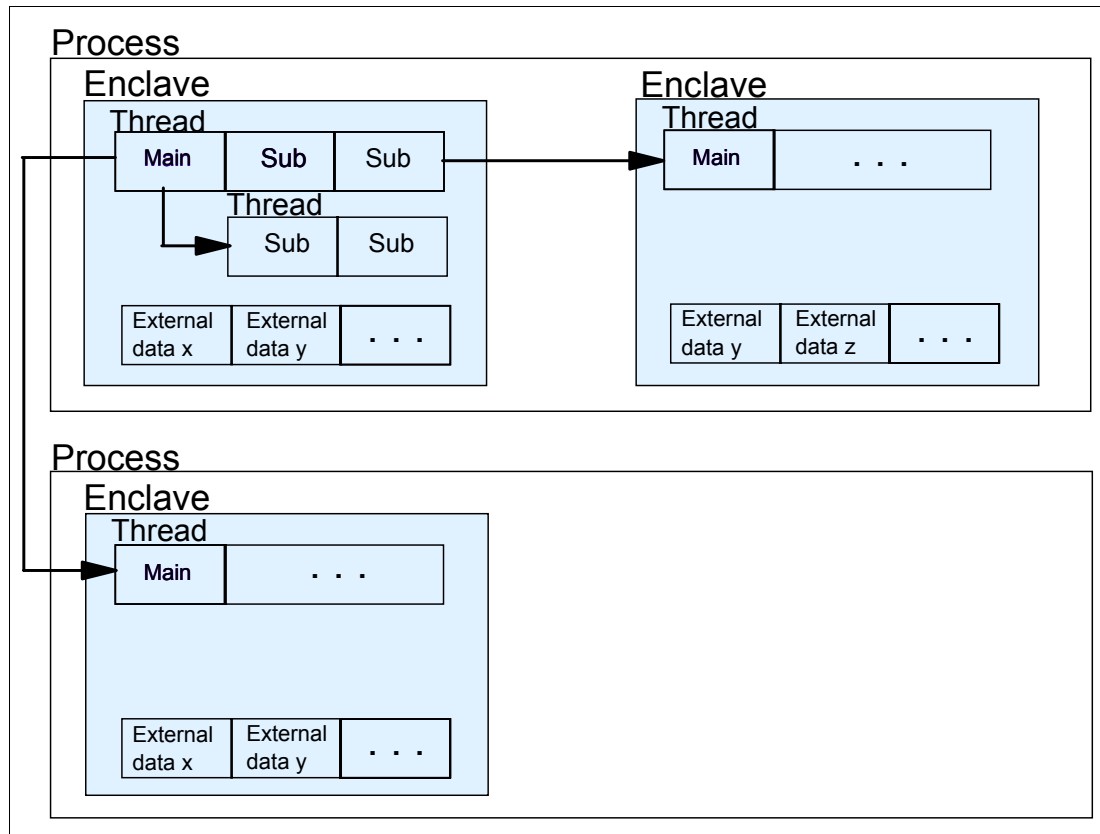


Figure 6-13 Program management full model

### Program management full model

Figure 6-13 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.

External data is available only within the enclave where it resides; notice that even though the external data may 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.

## 6.14 LE condition handling model description and terminology

- ❑ **Condition**
  - An exception that has been recognized by LE and thus is eligible to activate user and language condition handlers
- ❑ **Condition handler**
  - A routine invoked by LE that responds to conditions in an application.
- ❑ **Condition token**
  - Information about condition
- ❑ **Feedback code**
  - A 12 byte code returned from calls to LE routines
- ❑ **Resume cursor**
  - Contains the address where execution resumes after a condition is handled.
- ❑ **Stack frame**
  - The physical representation of the activation of a routine

Figure 6-14 LE condition handling terminology

### Condition handling model

For single- and mixed-language applications, the Language Environment run-time 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 on 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.

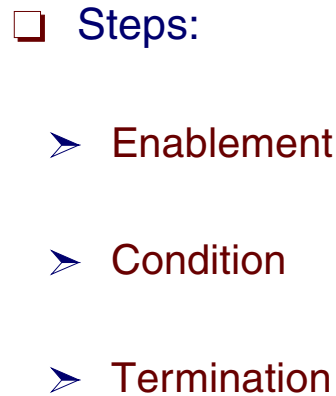
A unique feature of Language Environment condition handling is the condition token. The token is a 12-byte (AMODE 24/31) or 16-byte (AMODE 64) data type that contains an accumulation of information about each condition. The information can be returned to the user as a feedback code when calling Language Environment callable services. It can also be used as a communication vehicle within the run-time environment.

## Condition handling terminology

The following terminology is used in application programs:

|                          |                                                                                                                                                                                                                                                                                                                                                                            |
|--------------------------|----------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| <b>Condition</b>         | 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.                                                                                                                                                                       |
| <b>Condition handler</b> | 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.                                                                                                                          |
| <b>Condition token</b>   | 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.                                                                                                                     |
| <b>Feedback code</b>     | A condition token value used to communicate information when using the Language Environment callable services.                                                                                                                                                                                                                                                             |
| <b>Resume cursor</b>     | 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.                                                                                                                                                                |
| <b>Stack frame</b>       | <p>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 on program linkage and condition handling, and other information.</p> <p>A stack frame is conceptually equivalent to a dynamic save area (DSA) in PL/I, or a save area in assembler.</p> |

## 6.15 LE condition handling steps

- 
- Steps:
    - Enablement
    - Condition
    - Termination

*Figure 6-15 LE condition handling steps*

### Condition handling steps

Language Environment condition handling is performed in three distinct steps: the enablement, condition, and termination 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 run-time 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.

## 6.16 LE condition handling: Condition token

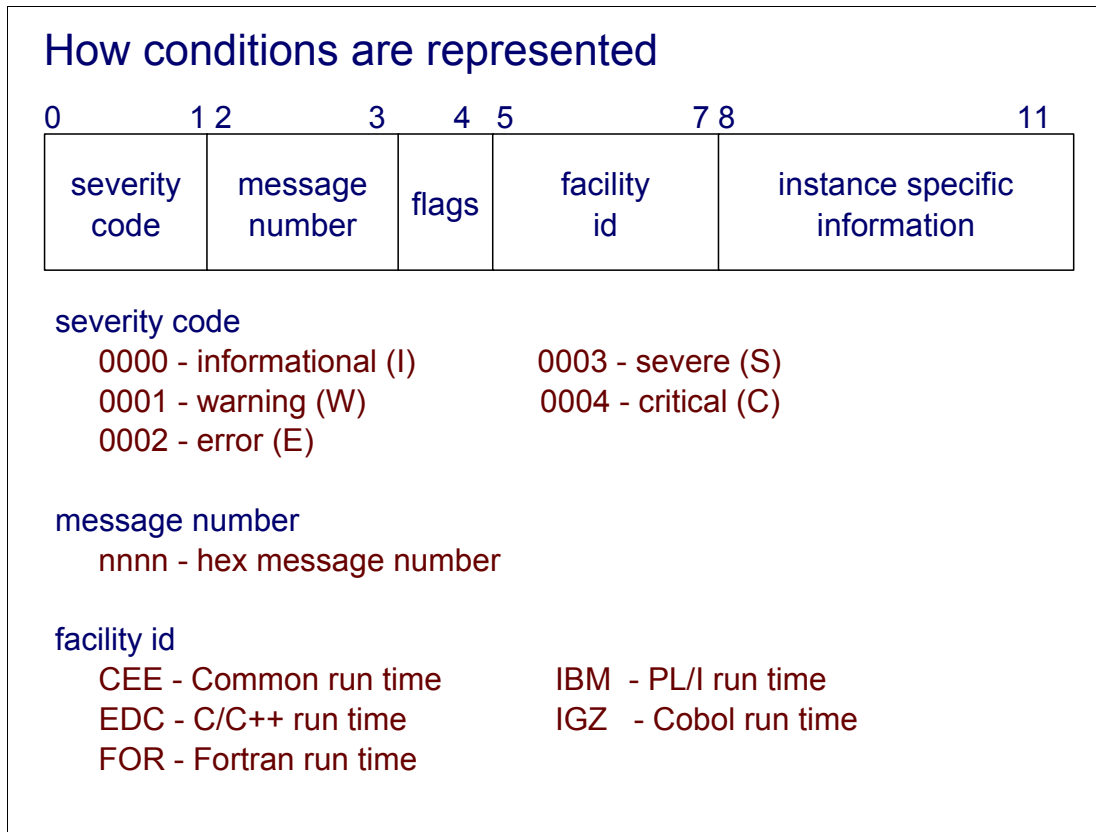


Figure 6-16 Condition token

### Condition token

A *condition token* is used to communicate information about a condition to Language Environment, message services, callable services, and routines. The token is a 12-byte data type with fields that indicate the following information about a condition:

- Severity of a condition.
- Associated message number: identifies the message associated with the condition. The combination of Facility\_ID and Msg\_No uniquely identifies a condition.
- Facility ID identifies the owner of the condition (Language Environment, Language Environment component, or user-specified). It is also used to identify a file containing message text that is unique for the condition.
- Instance-Specific Information (ISI) is a field that is created if the condition requires that data or text be inserted into a message, for example, a variable name. This field also contains qualifying data, which can be used to specify data (input or output) to be used when a routine resumes processing after a condition occurs.

**Note:** Under Language Environment AMODE 64, the condition token is 16-bytes long and the Instance-Specific Information is an 8-byte field.



## 6.17 LE condition handling stack frame

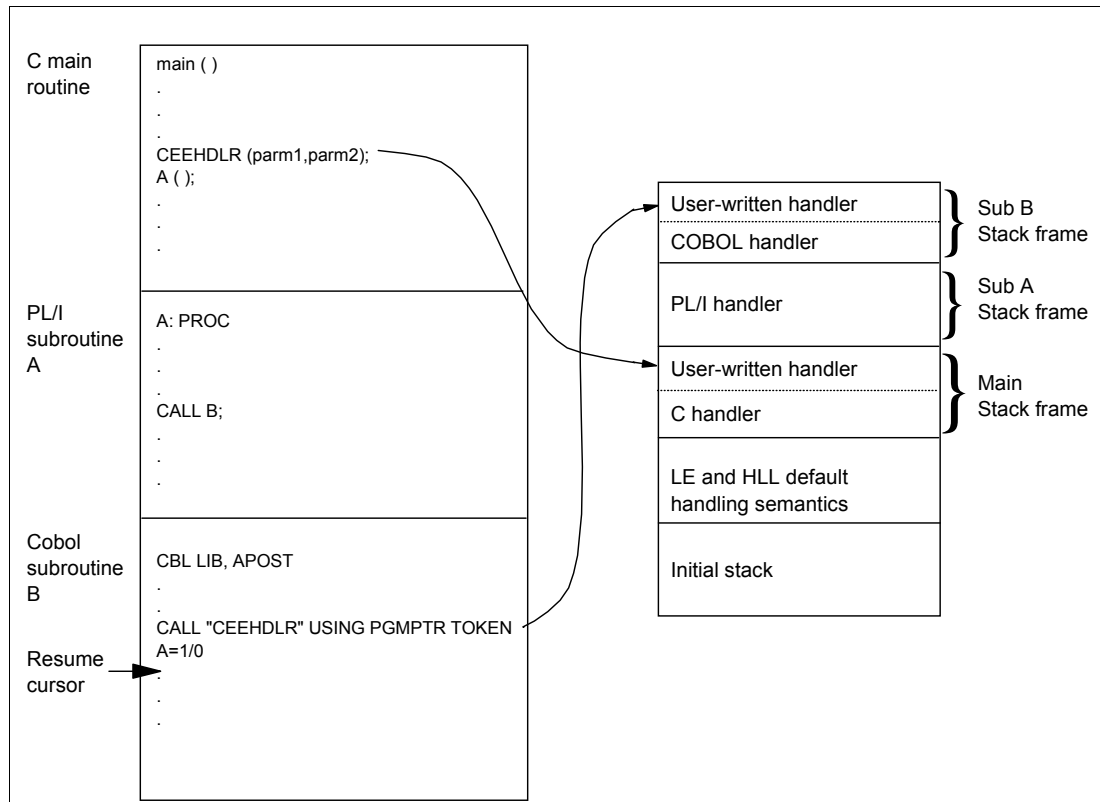


Figure 6-17 LE condition handling stack configuration

### Stack frame

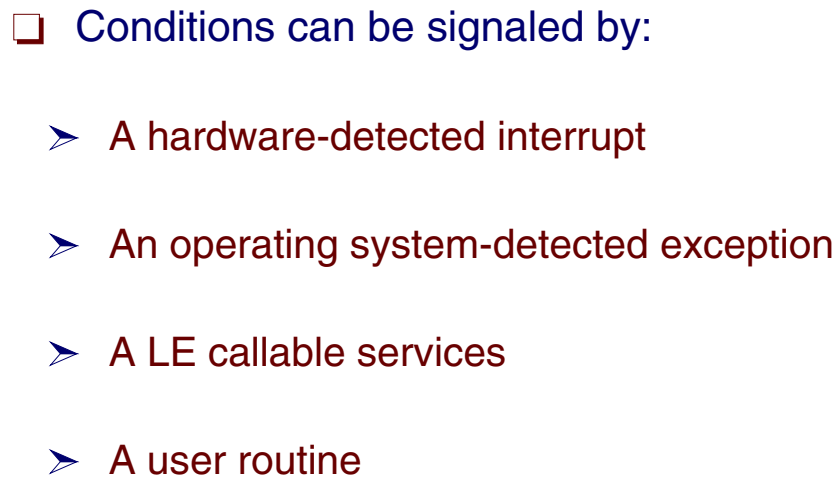
A stack frame is an area of storage that can contain automatic variables, information on program linkage and condition handling, and other information. It is created through any of the following:

- ▶ A function call in C or C++
- ▶ Entry into a compile unit in COBOL
- ▶ Entry into a procedure or begin block in PL/I
- ▶ Entry into an ON-unit in PL/I

Each routine adds a unique stack frame, in a LIFO manner, to the Language Environment storage. User-written condition handlers (registered through CEEHDLR) are associated with each stack frame. In addition, HLL handling semantics can affect the processing conditions at each stack frame. For an illustration of the Language Environment run-time stack and its divisions into stack frames, see Figure 6-17.

Each Language Environment user condition handler is explicitly registered through the callable service CEEHDLR or through the USRHDLR run-time option. Language-defined handling mechanisms are registered through language-provided constructs, such as the PL/I ON statement or the C signal() function. When a routine returns to its caller, its stack frame is removed from the stack and the associated handlers are automatically unregistered. Semantics associated with a routine are honored; for example, PL/I semantics on a return specify that any ON-units within a routine will be unregistered. If the USRHDLR run-time option is used, the user-written condition handler is registered at stack frame 0.

## 6.18 LE condition handling signaling

- 
- ❑ Conditions can be signaled by:
    - A hardware-detected interrupt
    - An operating system-detected exception
    - A LE callable services
    - A user routine

*Figure 6-18 Condition handling signaling*

### 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 may be signaled by user-written code through a call to the service CEESGL 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 may 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.

## 6.19 LE condition handling responses

□ Conditions can be responded in these ways:

- Resume
- Percolate
- Promote

Figure 6-19 Condition handling responses

### Responses

Conditions are responded to in one of the following ways:

- ▶ **Resume** terminates condition handling and transfers control, usually to the location immediately following the point where the condition occurred.  
A resume cursor points to the place where a routine should resume; it can be moved by the callable service CEEMRCR to point to another resume point.
- ▶ **Percolate** defers condition handling for an unchanged condition because a condition handler declined to handle it. Condition handling continues at the next condition handler.
- ▶ **Promote** is similar to percolate in that it passes the condition on to the next condition handler; however, it transforms a condition to another condition, one with a new meaning. Condition handling then continues, this time with a new type of condition.

**Note:** Promote is not supported in AMODE 64.

## 6.20 LE storage management model

### □ Storage management terminology

- Stack
- Heap
- Heap element
- Heap increment
- Heap pool
- Heap segment

*Figure 6-20 LE storage management terminology*

### Storage management model

Common storage management services are provided for all LE-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:

|                       |                                                                                                                                                                                             |
|-----------------------|---------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| <b>Stack</b>          | An area of storage in which stack frames can be allocated.                                                                                                                                  |
| <b>Heap</b>           | An area of storage used by Language Environment routines. The heap consists of the initial heap segment and zero or more increments.                                                        |
| <b>Heap element</b>   | A contiguous area of storage allocated by a call to the CEEGTST service. Heap elements are always allocated within a single heap segment.                                                   |
| <b>Heap increment</b> | Additional heap segments allocated when the initial heap segment does not have enough free storage to satisfy a request for heap storage.                                                   |
| <b>Heap pool</b>      | 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. |

**Heap segment**     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 run-time options and callable services.

For z/OS Language Environment, heap storage is made up of one or more heap segments comprised of an initial heap segment, and, as needed, one or more heap increments, allocated as additional storage is required. The initial heap may or may 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.

## 6.21 Debug Tool in the user environment

- ❑ Language Environment supports the IBM Debug Tool
- ❑ Use the Debug Tool as follows:
  - Monitor and interrupt program flow
  - Single step through an application
  - Add, remove, enable, or disable breakpoints
  - Display and change variables at breakpoints
- ❑ Run-time options for debugging routines
- ❑ Language-specific compiler options for debugging

Figure 6-21 Debug tool in the user environment

### IBM Debug Tool

Language Environment supports the IBM Debug Tool, a robust, interactive, interlanguage source-level debugging tool. Debug Tool helps you to examine, monitor, and control the execution of programs written in C/C++, COBOL, and PL/I. Debug Tool supports interactive and batch debugging of mixed-language applications.

Using the Debug Tool, you can:

- ▶ Monitor and interrupt the flow of a program to identify errors easily and correct them quickly
- ▶ Single step through your application or dynamically invoke Debug Tool when an error condition occurs
- ▶ Add, remove, enable, or disable breakpoints dynamically
- ▶ Display and change variables at breakpoints or monitor program variable changes as the program runs

Besides the language-specific compiler options important to debugging routines in Language Environment, there are also several language environment run-time options that affect debugging. When debugging AMODE 64 applications, the developer must be concerned about specific LE run time and language-specific compiler options to debug this kind of applications. The use of some compiler options (such as TEST or DEBUG) can affect the performance of your routine. See *z/OS Language Environment Debugging Guide*, GA22-7560, for a description of run-time options.

## 6.22 Sample assembler routine

```

MAIN      CEEENTRY PPA=MAINPPA
*
          LA      1,PARMLIST
          L       15,=V(CEEMOUT)
          BALR    14,15
*
*  Terminate the Language Environment environment
*
          CEETERM  RC=0,MODIFIER=0
* =====
*  CONSTANTS AND WORKAREAS
* =====
PARMLIST  DC      AL4(String)
          DC      AL4(DEST)
          DC      X'80000000'    Omitted feedback code
String    DC      AL2(STRLEN)
STRBEGIN  DC      CL19'In the main routine'
STRLEN    EQU     *-STRBEGIN
DEST      DC      F'2'
MAINPPA   CEEPPA
          CEEDSA
          CEECAA
          END      MAIN          Constants describing the code block
                                   Mapping of the dynamic save area
                                   Mapping of the common anchor area
                                   Nominate MAIN as the entry point

```

Figure 6-22 Sample assembler routine

### Assembler routine

Figure 6-22 shows a simple main assembler routine (source code) that brings up the environment, returns with a return code of 0, modifier of 0, and prints a message in the main routine.

## 6.23 LE run-time options customization

| Set defaults for                        | Sample Job | Required member |
|-----------------------------------------|------------|-----------------|
| Installation-wide z/OS batch            | CEEWDOPT   | CEEDOPT         |
| Installation-wide CICS                  | CEEWCOPT   | CEECOPT         |
| Region-specific CICS                    | CEEWROPT   | CEECOPT         |
| Region-specific IMS/LRR                 | CEEWROPT   | CEEDOPT         |
| Application-specific                    | CEEWUOPT   | CEEUOPT         |
| Installation-wide z/OS batch (AMODE 64) | CEEWQDOP   | CELQDOPT        |
| Application-specific (AMODE 64)         | CEEWQUOP   | CELQUOPT        |

Figure 6-23 LE run-time options customization

### LE run-time options customization

Figure 6-23 lists the sample jobs that are members of Language Environment sample library SCEESAMP. These sample jobs require other members of the SCEESAMP data set.

When you run the sample jobs, they create the CEEDOPT CSECT, an options control block which establishes the defaults for the options. The jobs invoke CEEXOPT during the assembly of the CEEDOPT module. When you modify the CEEXOPT macro invocation to change installation-wide defaults, you must specify each run-time option as either OVR or NONOVR.

The options in CEEUOPT override the default options in CEEDOPT or CEECOPT, unless NONOVR was specified for the option when CEEDOPT or CEECOPT was created.

The following example shows the IBM-supplied version of the CEEDOPT CSECT contained within the CEEDOPT member, with the default suboption values for each of the options.

```
CEEDOPT CSECT
CEEDOPT AMODE ANY
CEEDOPT RMODE ANY
      CEEXOPT ABPERC=((NONE),OVR),
              ABTERMENC=((ABEND),OVR),
              AIXBLD=((OFF),OVR),
              ALL31=((ON),OVR),
              ANYHEAP=((16K,8K,ANYWHERE,FREE),OVR),
```

```
x
x
x
x
x
```



```

BELOWHEAP=((8K,4K,FREE),OVR),           X
CBLOPTS=((ON),OVR),                     X
CBLPSHPOP=((ON),OVR),                   X
CBLQDA=((OFF),OVR),                     X
CHECK=((ON),OVR),                       X
COUNTRY=((US),OVR),                     X
DEBUG=((OFF),OVR),                      X
DEPTHCONDLMT=((10),OVR),                X
ENVAR=('',OVR),                         X
ERRCOUNT=((0),OVR),                    X
ERRUNIT=((6),OVR),                      X
FILEHIST=((ON),OVR),                    X
FILETAG=((NOAUTOCVT,NOAUTOTAG),OVR),    X
HEAP=((32K,32K,ANYWHERE,KEEP,8K,4K),OVR), X
HEAPCHK=((OFF,1,0,0,0),OVR),            X
HEAPPOLS=((OFF,8,10,32,10,128,10,256,10,1024,10,2048, X
10,0,10,0,10,0,10,0,10,0,10,0,10),OVR), X
INFMSGFILTER=((OFF,,,),OVR),            X
INQPCOPN=((ON),OVR),                   X
INTERRUPT=((OFF),OVR),                  X
LIBRARY=((SYSCEE),OVR),                 X
LIBSTACK=((4K,4K,FREE),OVR),            X
MSGFILE=((SYSOUT,FBA,121,0,NOENQ),OVR), X
MSGQ=((15),OVR),                       X
NATLANG=((ENU),OVR),                   X
NOAUTOTASK=(OVR),                      X
NOTEST=((ALL,*,PROMPT,INSPPREF),OVR),   X
NOUSRHDLR=(( ),OVR),                   X
OCSTATUS=((ON),OVR),                   X
PC=((OFF),OVR),                        X
PLITASKCOUNT=((20),OVR),               X
POSIX=((OFF),OVR),                     X
PROFILE=((OFF,''),OVR),                 X
PRTUNIT=((6),OVR),                     X
PUNUNIT=((7),OVR),                     X
RDRUNIT=((5),OVR),                     X
RECPAD=((OFF),OVR),                     X
RPTOPTS=((OFF),OVR),                    X
RPTSTG=((OFF),OVR),                     X
RTEREUS=((OFF),OVR),                    X
RTLS=((OFF),OVR),                       X
SIMVRD=((OFF),OVR),                     X
STACK=((128K,128K,ANYWHERE,KEEP,512K,128K),OVR), X
STORAGE=((NONE,NONE,NONE,OK),OVR),      X
TERMTHDACT=((TRACE,,96),OVR),           X
THREADHEAP=((4K,4K,ANYWHERE,KEEP),OVR), X
THREADSTACK=((OFF,4K,4K,ANYWHERE,KEEP,128K,128K),OVR), X
TRACE=((OFF,4K,DUMP,LE=0),OVR),          X
TRAP=((ON,SPIE),OVR),                   X
UPSI=((00000000),OVR),                  X
VCTRSVE=((OFF),OVR),                    X
VERSION=('',OVR),                       X
XUFLOW=((AUTO),OVR)                     X
END

```

**Note:** In z/OS V1R6, Language Environment run-time options LIBRARY, RTLS and VERSION are no longer supported.

## 6.24 LE user exits

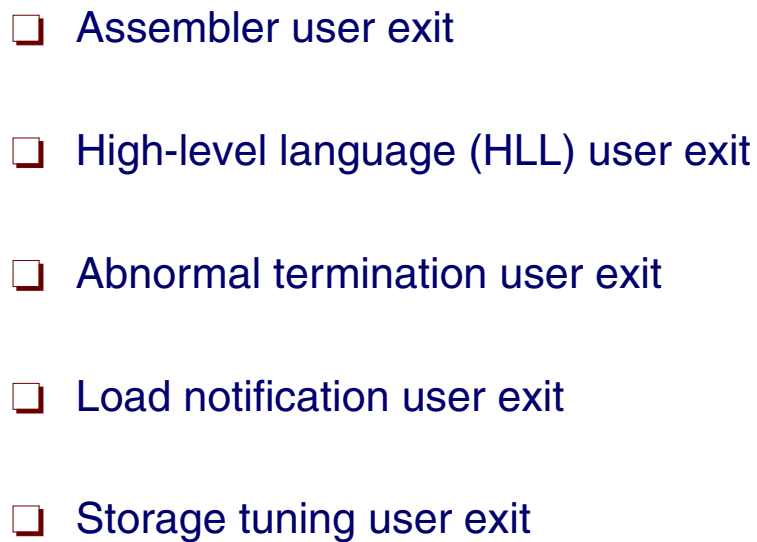
- 
- ❑ Assembler user exit
  - ❑ High-level language (HLL) user exit
  - ❑ Abnormal termination user exit
  - ❑ Load notification user exit
  - ❑ Storage tuning user exit

Figure 6-24 LE user exits

### User exits

Language Environment provides support for the following user exits:

- ▶ Assembler user exit

The assembler user exit can be used to perform functions for enclave initialization, normal and abnormal enclave termination, and process termination.
- ▶ High-level language (HLL) user exit

The HLL user exit can be used to perform functions for enclave initialization.
- ▶ Abnormal termination user exit

The abnormal termination user exit can be used to collect problem determination data when Language Environment is terminating an enclave due to a not handled condition.
- ▶ Load notification user exit

The load notification user exit can be used to improve performance by preventing frequently used modules from being loaded and deleted with each use. The load notification user exit is only available when Library Routine Retention (LRR) is used.
- ▶ Storage tuning user exit

The storage tuning user exit provides a programming interface that allows you to collect Language Environment storage tuning information and to set the Language Environment run-time option values for STACK, LIBSTACK, HEAP, ANYHEAP and BELOWHEAP. The storage tuning user exit is available for CICS, and for non-CICS environments when LRR is used.

# Related publications

The publications listed in this section are considered particularly suitable for a more detailed discussion of the topics covered in this redbook.

## IBM Redbooks

For information on ordering these publications, see “How to get IBM Redbooks” on page 436. Note that some of the documents referenced here may be available in softcopy only.

- ▶ *z/OS Version 1 Release 3 and 4 Implementation*, SG24-6581
- ▶ *z/OS Version 1 Release 2 Implementation*, SG24-6235
- ▶ *z/OS V1R3 DFSMS Technical Guide*, SG24-6569
- ▶ *OS/390 Version 2 Release 10 Implementation*, SG24-5976
- ▶ *JES3 in a Parallel Sysplex*, SG24-4776
- ▶ *MVS/ESA SP-JES3 Version 5 Implementation Guide Release 5.1.1, 5.2.1*, SG24-4582
- ▶ *MVS/ESA SP-JES2 Version 5 Implementation Guide Release 5.1.0, 5.2.0*, SG24-4583
- ▶ *OS/390 Software Management Cookbook*, SG24-4775

## Other publications

These publications are also relevant as further information sources:

- ▶ *z/OS MVS JCL Reference*, SA22-7597
- ▶ *z/OS MVS Interactive Problem Control System (IPCS) User's Guide*, SA22-7596
- ▶ *z/OS MVS Initialization and Tuning Reference*, SA22-7592
- ▶ *z/OS MVS Interactive Problem Control System (IPCS) Commands*, SA22-7594
- ▶ *z/OS MVS Planning: Operations*, SA22-7601
- ▶ *z/OS DFSMS: Using Data Sets*, SC26-7410
- ▶ *z/OS MVS System Data Set Definition*, SA22-7629
- ▶ *z/OS and z/OS.e Planning for Installation*, GA22-7504
- ▶ *z/OS MVS System Data Set Definition*, SA22-7629
- ▶ *z/OS Hardware Configuration Definition User's Guide*, SC33-7988
- ▶ *z/OS Hardware Configuration Definition Planning*, GA22-7525
- ▶ *Device Support Facilities User's Guide and Reference Release 17*, GC35-0033
- ▶ *z/OS MVS System Commands*, SA22-7627
- ▶ *z/OS MVS System Management Facilities (SMF)*, SA22-7630
- ▶ *z/OS MVS Installation Exits*, SA22-7593
- ▶ *Environmental Record Editing and Printing Program (EREP) Reference*, GC35-0152
- ▶ *z/OS JES2 Commands*, SA22-7526
- ▶ *z/OS JES2 Initialization and Tuning Reference*, SA22-7533

- ▶ *z/OS JES2 Initialization and Tuning Guide*, SA22-7532
- ▶ *z/OS TSO/E User's Guide*, SA22-77942
- ▶ *z/OS DFSMSdss Storage Administration Reference*, SC35-0424
- ▶ *z/OS DFSMS Access Method Services for Catalogs*, SC26-7394
- ▶ *z/OS DFSMS: Managing Catalogs*, SC26-7409
- ▶ *z/OS DFSMSdfp Storage Administration Reference*, SC26-7402
- ▶ *z/OS DFSMSHsm Implementation and Customization Guide*, SC35-0418
- ▶ *z/OS DFSMS: Using the Interactive Storage Management Facility*, SC26-7411
- ▶ *z/OS Packaging Rules*, SC23-3695
- ▶ *SMP/E Commands*, SA22-7771
- ▶ *SMP/E Reference*, SA22-7772
- ▶ *SMP/E User's Guide*, SA22-7773
- ▶ *z/OS UNIX System Services Command Reference*, SA22-7802
- ▶ *z/OS Communications Server IP User's Guide and Commands* (p. 398, 400)
- ▶ *z/OS UNIX System Services Planning*, GA22-7800
- ▶ *z/OS Language Environment Writing ILC Applications*, SA22-7563
- ▶ *z/OS Language Environment Programming Guide*, SA22-7561
- ▶ *z/OS C/C++ Run-Time Library Reference*, SA22-7821
- ▶ *Language Environment Programming Guide for 64-bit Virtual Addressing Mode*, SA22-7569
- ▶ *z/OS Language Environment Debugging Guide*, GA22-7560

## How to get IBM Redbooks

You can search for, view, or download Redbooks, Redpapers, Hints and Tips, draft publications and Additional materials, as well as order hardcopy Redbooks or CD-ROMs, at this Web site:

[ibm.com/redbooks](http://ibm.com/redbooks)



## ABCs of z/OS System Programming Volume 2

(0.5" spine)  
0.475" <-> 0.873"  
250 <-> 459 pages







**Redbooks**

# ABCs of z/OS System Programming

## Volume 2

**z/OS implementation and maintenance**

**Job management, JES2, JES3, SSI, LPA**

**LNKLST, SMP/E, LE**

The ABCs of z/OS System Programming is a eleven 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 would like 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 will serve as a powerful technical tool.

The contents of the volumes are:

- ▶ Volume 1: Introduction to z/OS and storage concepts, TSO/E, ISPF, JCL, SDSF, and z/OS delivery and installation
- ▶ Volume 2: z/OS implementation and daily maintenance, defining subsystems, JES2 and JES3, LPA, LNKLST, authorized libraries, Language Environment, and SMP/E
- ▶ Volume 3: Introduction to DFSMS, data set basics, storage management hardware and software, VSAM, System-managed storage, catalogs, and DFSMSStvs
- ▶ Volume 4: Communication Server, TCP/IP and VTAM
- ▶ Volume 5: Base and Parallel Sysplex, System Logger, Resource Recovery Services (RRS), global resource serialization (GRS), z/OS system operations, automatic restart management (ARM), Geographically Dispersed Parallel Sysplex (GDPS), availability in the zSeries environment
- ▶ Volume 6: Introduction to security, RACF, Digital certificates and PKI, Kerberos, cryptography and z990 integrated cryptography, zSeries firewall technologies, LDAP, Enterprise identity mapping (EIM), and firewall technologies
- ▶ 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 z/Architecture, zSeries processor design, zSeries connectivity, LPAR concepts, HCD, and HMC
- ▶ Volume 11: Capacity planning, performance management, WLM, RM, and SMF

### INTERNATIONAL TECHNICAL SUPPORT ORGANIZATION

### BUILDING TECHNICAL INFORMATION BASED ON PRACTICAL EXPERIENCE

IBM Redbooks are developed by the IBM International Technical Support Organization. Experts from IBM, Customers and Partners from around the world create timely technical information based on realistic scenarios. Specific recommendations are provided to help you implement IT solutions more effectively in your environment.

**For more information:**  
**[ibm.com/redbooks](http://ibm.com/redbooks)**

SG24-6982-01

ISBN 0738492965