IBM z/VSE
Version 6 Release 2

Guide to System Functions

SC34-2705-01

Note!

Before using this information and the product it supports, be sure to read the general information under “Notices” on page 197.

This edition applies to Version 6 Release 2 of IBM® z/Virtual Storage Extended (z/VSE®), Program Number 5686-VS6 and to all subsequent releases and modifications until otherwise indicated in new editions.

This edition replaces SC34-2705-00.

Order publications through your IBM representative or the IBM branch office serving your locality. Publications are not stocked at the addresses given below.

A form for readers' comments is provided at the back of this publication. If the form has been removed, address your comments to:

```
IBM Deutschland Entwicklung GmbH
Department 3282
Schoenaicher Strasse 220
D-71032 Boeblingen
Federal Republic of Germany
```

You may also send your comments by FAX or via the Internet:

```
Internet: s390id@de.ibm.com
FAX (Germany): 07031-16-3456
FAX (other countries): (+49)+7031-16-3456
```

When you send information to IBM, you grant IBM a non-exclusive right to use or distribute the information in any way it believes appropriate without incurring any obligation to you.

Contents

Figures.. vii

Tables.. xi

About This Publication.. xiii
Who Should Use This Publication.. xiii
How to Use This Publication..xiii
Where to Find More Information..xiii

Summary of Changes.. xv

Chapter 1. Storage Management.. 1
Virtual Storage Concept.. 1
Page Management.. 2
Relate Virtual Storage to Locations in Processor Storage.. 3
Address Space Layout..4
Virtual Address Spaces versus Real Address Space.. 4
Layout of a Virtual Address Space.. 5
Partition Allocation and Program Size Considerations..6
GETVIS Areas..9
Layout of a Static Address Space and its GETVIS Areas..10
Layout of a Dynamic Address Space and its GETVIS Areas.. 10
Defining Real Storage..11
Real Storage Layout..11
Data Spaces..12
64-Bit Address Space.. 13
Executing Programs in Virtual and Real Mode.. 13
Processor Storage Allocation for Real Mode Execution.. 14
Fixing Pages in Processor Storage..14

Chapter 2. Starting the System.. 15
ASI Procedures.. 15
Contents of an ASI IPL Procedure.. 15
Contents of ASI JCL Procedures..17
Naming Conventions for ASI Procedures.. 18
Starting Up the System..18
The ASI Master Procedure ($ASIPROC)..19
Cataloging an ASI Master Procedure..20
Establishing the Communication Device for IPL..21
Console Selection for Initial Installation.. 21
Console Selection when Performing a Normal IPL.. 21
IPL Communication Device List.. 22
Interrupt IPL Processing for Modifications.. 23
The IPL Load Parameter..23
Interrupt and Restart the IPL Process..24
Modifying IPL Parameters..26
Loading Phases into the SVA.. 28
Automatic SVA Loading During System Startup.. 29
Loading Single Phases or Using a Load List..29

Creating an SVA Load List....30
Notes on the SVA Command....31
Notes on Using the SET SDL Command....31
Replacing Phases Stored in the SVA....31
User-Defined Processing after IPL....32

Chapter 3. Controlling Jobs....33

Relating Files to Your Program....33
Processing of File Labels....34
The Label Information Area....36
Defining a Job....37
The JOB Statement....37
The End-of-Job (/&) Statement....38
Job Streams....38
Job Control for Device Assignments....39
Logical Units....39
Types of Device Assignments....40
Device Assignments....41
Shared Assignments....42
Additional Assignment Considerations....42
Job Control for Label Information....43
Label Information for Files on Disk Devices....43
Label Information for Files on Magnetic Tape....45
Storing Label Information....45
Adding and Deleting Label Information....47
Label Information Search Order....47
Controlling Magnetic Tape....48
Controlling Printed Output....49
Controlling Printed Output on an IBM 3800 Printing Subsystem....50
Processing a Program....50
Executing Cataloged Programs....52
Defining Options for Program Execution....53
Communicating with Application Programs via Job Control....54
Run in Virtual or Real Mode....54
Handling of System Input and Output....57
Using Conditional Job Control....59
Return Code Setting by Job Control....60
Statements for Conditional Job Control....62
Abnormal Termination of a Job Stream....65
Using Cataloged Procedures....66
SYSIPT Data in Cataloged Procedures....66
Cataloging Partition-Related Procedures....67
Several Job Steps in One Procedure....68
Using Symbolic Parameters....69
Using Nested Procedures....73

Chapter 4. Using VSE Libraries....79

Introducing the VSE Library Concept....79
Library Structure....79
VSE Library Types....80
Defining a Library, Sublibrary, or a SYSRES File....82
Defining Sublibraries....84
Defining Additional SYSRES Files....84
Establishing a Library Access Definition....85
Cataloging Members of Type PHASE....85
Cataloging Members of Type DUMP....85
Library Chaining....85

Permanent versus Temporary Library Access Definitions........ 86
The Search Sequence for Phases........ 86
Resetting a Library Access Definition........ 87
Displaying Library Access Definitions........ 88
Accessing Members Controlled by MSHP........ 88
The Librarian Program........ 88
Return Codes........ 88
Examples of Conditional Command Execution........ 89
Interactive Execution........ 90
Accessing Sublibraries........ 90
Generic Notation........ 90
Librarian Commands........ 91
Backup a SYSRES File, Library, Sublibrary, or Member........ 91
Catalog a Member........ 97
Change the Reuse Attribute of a Sublibrary........ 102
Compare Libraries, Sublibraries, or Members........ 102
Copy or Move a Library, Sublibrary or Member........ 103
Define a Library, Sublibrary, or a SYSRES File........ 105
Delete a Library, Sublibrary, or a Member........ 105
List Library, Sublibrary, or Member Information........ 107
Lock a Member........ 114
Punch and Re-Catalog a Member........ 117
Release Space for a Library or Sublibrary........ 118
Rename a Sublibrary or a Member........ 118
Restore a SYSRES File, Library, Sublibrary, or a Member........ 120
Search for Members........ 125
Test a Library or Sublibrary........ 127
Unlock a Member........ 127
Update a Member........ 127
Library Access for Application Programs........ 128
Accessing Member Data........ 129
Retrieving Status Information........ 130
Return Code Conventions........ 130
Example of a STATE Member Request........ 131
Example of OPEN/GET/CLOSE Requests........ 135
Processing Macros with the ESERV Program........ 138
High Level Assembler Considerations........ 139
Using the High Level Assembler Library Exit for Processing E-Decks........ 139
Function Description of Phase IPKVX........ 142

Chapter 5. Linking Programs........ 145
Structure of a Program........ 145
Source Books........ 146
Object Modules........ 146
Phases........ 147
Year Representation........ 148
Basic Applications of the Linkage Editor........ 148
Cataloging Phases into a Sublibrary........ 148
Link-Edit and Execute........ 149
Assemble (or Compile), Link-Edit, and Execute........ 150
Processing Requirements for the Linkage Editor........ 151
Preparing Input for the Linkage Editor........ 152
Naming a Phase........ 152
Defining a Load Address for a Phase........ 153
Linkage Editor Input - Source and Sequence........ 154
Linkage Editor Storage Requirements........ 154
Using the AUTOLINK Function........ 155

Suppressing the AUTOLINK Feature.. 155
Specifying Linkage Editor Helps.. 156
Designing an Overlay Program.. 156
Relating Control Sections to Phases.. 156
Using LOAD and FETCH Macros.. 158
Pseudo-Register Support.. 158
Coding Example.. 159
Support of Named Common Control Sections.. 159
How External References are Resolved.. 159
Examples of Linkage Editor Applications.. 161
Link-Edit and Execute Example.. 162
Compile and Execute Example.. 163

Chapter 6. Using VSE Facilities and Options.. 165
User-Written Exit Routines.. 165
Program Exit Routines.. 165
Writing an IPL Exit Routine.. 167
Writing a Job Control Exit Routine.. 168
Multiple Job Control Exit Routines.. 171
Writing a Job Accounting Interface Routine.. 175
Job Accounting Information.. 175
Programming Considerations.. 176
Tailoring the Program.. 177
Checkpointing Facility.. 177
Restarting a Program from a Checkpoint.. 178
Using Timer Services.. 179
DASD Sharing with Multiple VSE Systems.. 179
Reserving Devices for Exclusive Use.. 180
Resource Locking.. 180
Lock Communication File.. 181
How to Initialize a Shared VSE Environment.. 182
DASD Sharing under VM.. 183
Special Considerations for Shared Libraries.. 183
Recorder, Hardcopy, and History Files in a DASD Sharing Environment.. 183
An Example of a Two-System Installation.. 183
Error Recovery after System Breakdown.. 186
Designing Programs for Virtual Mode Execution.. 186
Programming Hints for Reducing Page Faults.. 186
Using Virtual Storage Macros.. 188
Coding for the Shared Virtual Area.. 191

Appendix A. Understanding Syntax Diagrams.. 193
Frequent Abbreviations.. 195
Continuation of Commands and Statements.. 195

Notices.. 197
Programming Interface Information.. 198
Trademarks.. 198
Terms and Conditions for Product Documentation.. 198

Accessibility.. 201
Using Assistive Technologies.. 201
Documentation Format.. 201
Glossary.. 203

Index.. 239

Figures

1. Virtual Storage and Processor (Real) Storage....1

2. Page Management Concept.... 2

3. Running a Program in Virtual Storage....3

4. Page Data Set Usage.... 4

5. Single and Multiple Address Spaces....4

6. z/VSE Storage Layout (Virtual Address Spaces/Real Address Space)....5

7. Virtual Address Space Layout (Size >16 MB)....6

8. Partition Allocation Examples....7

9. Partition Size and Program Area of a Dynamic Partition.... 8

10. Static Address Space, Partition Layout, and GETVIS Areas....10

11. Dynamic Address Space, Partition Layout, and GETVIS Areas....11

12. z/VSE Real Storage Layout....12

13. 64-Bit Virtual Address Space Layout....13

14. Example of an IPL Procedure ($IPLESA) after Initial Installation Complete....17

15. Job Stream Example for Creating a CDL....22

16. IPL Load Parameter Format....24

17. Loading Single SVA Phases....30

18. Loading SVA Phases Through a Load List....30

19. Creating an SVA Load List.... 30

20. File Label Processing.... 35

21. Control Statements Defining a Job Consisting of Two Job Steps....37

22. Example of a Job Stream....38

23. Possible Device Assignments....42

24. Sample Job for Label Checking of a File on Magnetic Tape..45

25. Job Control Statements to Assemble, Link-Edit, and Execute a Program in one Job Stream................50

26. Storage Layout of a Partition with Default GETVIS Area..56

27. Storage Layout of a Partition after a SIZE Command was Given..57

28. Program Execution with the SIZE Operand..57

29. Creation of SYSIN on Tape...58

30. Standard Return Codes for Conditional Job Control...60

31. Processing Flow of Nested Procedures (4 Levels).. 74

32. Tape Positioning at End of a Selective Restore for a Labeled Online Tape.....................................96

33. Output Format of a Member (Phase) Display..107

34. Output Format of a Member (Procedure) Display...108

35. Output Format of a Member (Procedure) Display (Format=HEX).. 108

36. Output Format of a Library Display (OUTPUT=STATUS)... 109

37. Output Format of a Library Display (OUTPUT=NORMAL).. 110

38. Output Format of a Sublibrary Display (OUTPUT=FULL)... 111

39. Output Format of an SDL Display... 112

40. Output Format of an SDL Display for a single Phase (O=NORMAL)...113

41. Output Format of an SDL Display (O=SHORT).. 113

42. Output Format of Locked Members..114

43. Example of a Stand-Alone Restore (Unlabeled Input Tape)..124

44. Output Format of Searched Members in Libraries (OUTPUT=NORMAL).. 125

45. Output Format of Searched Members in CONNECT Libraries (OUTPUT=FULL)................................126

46. Output Format of Search for a Single Phase (OUTPUT=FULL)... 126

47. Output Format of Search for Locked Members (OUTPUT=FULL).. 126

48. Code Example for a Librarian STATE Member Request.. 131

49. Code Example for Librarian OPEN/GET/CLOSE Requests.... 135

50. Record Types of an Object Module.... 146

51. Job Stream to Catalog a Program Permanently into a Sublibrary.... 148

52. Job to Link-Edit and Store a Program Temporarily for Immediate Execution.... 150

53. Job to Assemble, Link-Edit, and Execute a Program Stored Temporarily.... 150

54. Naming Multiphase Programs.... 152

55. Overlay Tree Structure.... 157

56. Link-Editing an Overlay Program.... 157

57. Summary of Program Exit Conditions (STXIT Macro).... 165

58. IPL Exit Routine Example.... 168

59. Job Control Exit Routine Example.... 170

60. Part of Content of JOBEXIT Sample.... 172

61. Creating an SVA Load List for JCL Exit Routines.... 172

62. JOBEXIT Sample with Several JCL Exit Routines.... 173

63. Example of a RESTART Job.... 178

64. Example of a DASD Sharing Configuration.... 184

65. Example of IPL Procedures for a DASD Sharing Environment.... 185

66. PFIX and PFREE Example.... 190

67. Example of Conventions for SVA Coding.... 192

Tables

1. STOP Points of IPL Commands.......... 27

2. Summary of Label Option Functions..........48

3. Sample of a Job to Assemble, Link-Edit and Execute a Program.......... 51

4. Partition Size and Related Number of Phases and ESD Items.......... 154

5. Job Accounting Table..........176

About This Publication

This publication describes system functions provided by IBM z/Virtual Storage Extended (z/VSE) Version 6 Release 2. It provides information helping you to use and understand these functions.

Who Should Use This Publication

The publication is intended for those users who have to understand and use z/VSE system functions. Some information is of importance for both, system administrators and programmers.

How to Use This Publication

This publication provides information about the following topics:

Storage Management
System Startup
Job Control
Librarian
Linkage Editor
Facilities and Options

Where to Find More Information

For an overview on the functions new with z/VSE 6.2 refer to the z/VSE Release Guide and z/VSE Planning.

z/VSE IBM Documentation

IBM Documentation is the new home for IBM's technical information. The z/VSE IBM Documentation can be found here:

https://www.ibm.com/docs/en/zvse/6.2

You can also find VSE user examples (in zipped format) at

https://public.dhe.ibm.com/eserver/zseries/zos/vse/pdf3/zVSE_Samples.pdf

Summary of Changes

This publication has been updated to reflect enhancements and changes that are implemented with z/VSE 6.2. It also includes terminology, maintenance, and editorial changes.

These are the enhancements that have been made available with z/VSE 6.2

- New facility to have return code setting by Job Control "Return Code Setting by Job Control" on page 60.

Chapter 1. Storage Management

Virtual Storage Concept

Programs are loaded and run in virtual storage (in a partition that is allocated in an address space). In Figure 1 on page 1, a virtual address space with the possible maximum of 2 GB is contrasted with a processor real storage size of 64 MB. z/VSE requires at least 64 MB processor storage. It supports and uses a maximum of 32 GB processor storage.

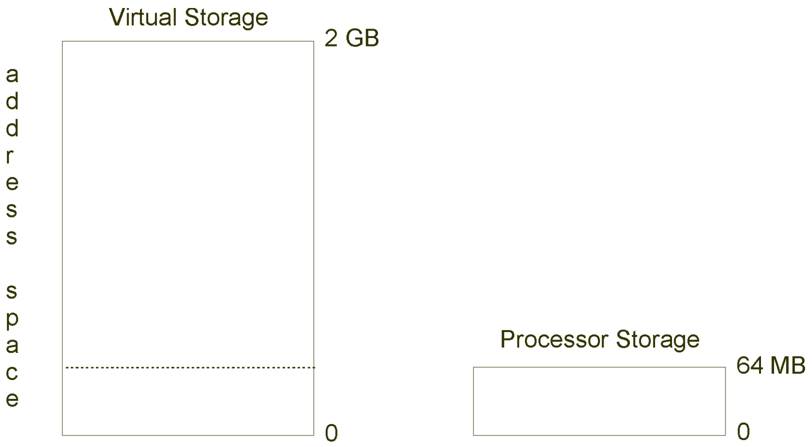


Figure 1. Virtual Storage and Processor (Real) Storage

Each instruction of a program must be in processor storage when the instruction is executed, and so must the data, which this instruction manipulates. The other instructions and data of that program in virtual storage do not have to be in processor storage at that same moment; they can reside on auxiliary storage until needed. The file that is used for this purpose is called the page data set.

It would be inefficient, however, to bring every instruction and its associated data into processor storage individually. Therefore, virtual storage is organized and manipulated in sections of 4 KB, called pages. Processor storage is also divided into 4 KB sections, called page frames. Page frames accommodate pages of a program during execution.

Note: z/VSE can be defined to run without a page data set (NOPDS). This is possible, if the available processor storage or the virtual machine size (if running under VM) is at least 64 MB. For details refer to z/VSE Planning.

Assume a system with a page data set. When a program is loaded from a library into virtual storage, and there are not enough page frames available to contain all the pages of a program, the system writes the contents of some page frames to the page data set. See Figure 2 on page 2 for an overview of this page management concept.

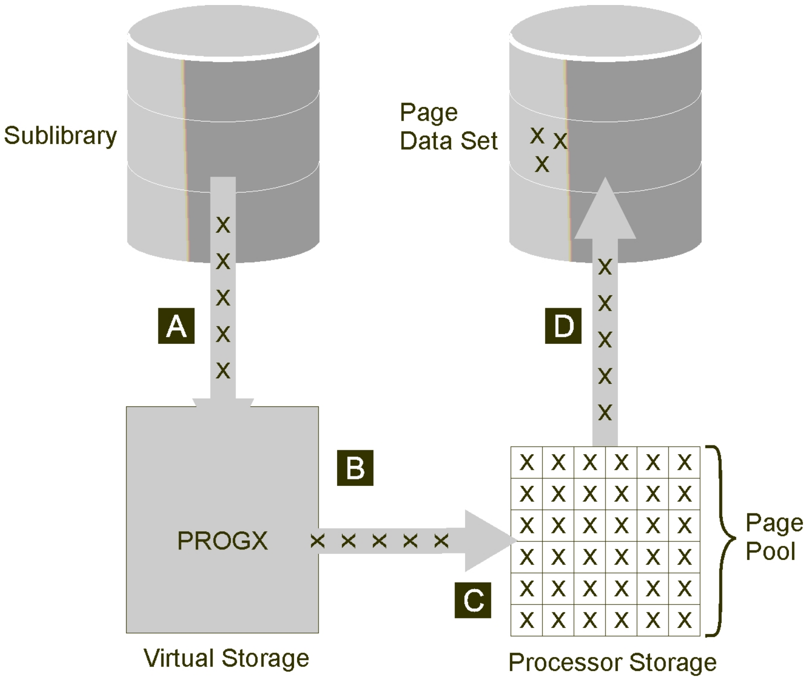


Figure 2. Page Management Concept

A program named PROGX A is conceptually loaded into virtual storage B. The supervisor finds page frames in the page pool of processor storage C. If there are not enough page frames available to accommodate all of PROGX, the supervisor stores the contents of some page frames on the page data set D.

Page Management

The section explains the concept of page management as shown in Figure 2 on page 2.

When a program is loaded for execution, it can be loaded in noncontiguous page frames of processor storage. The supervisor knows what processor storage locations pages of a given program occupy. If the program should cancel, due to an error, the storage dump produced by the system reflects the virtual addresses where the program was conceptually running. In Figure 3 on page 3, a 16 KB program that is named INVEN, is conceptually loaded at the virtual storage location 1024 KB. As shown, the system selected four page frames of processor storage, which are not contiguous. If the program ends abnormally and a storage dump is produced, the INVEN program would be shown as occupying addresses 1024 KB through 1040 KB minus 1.

Note: The values that are used in this figure are for demonstration only - they do not reflect real system values.

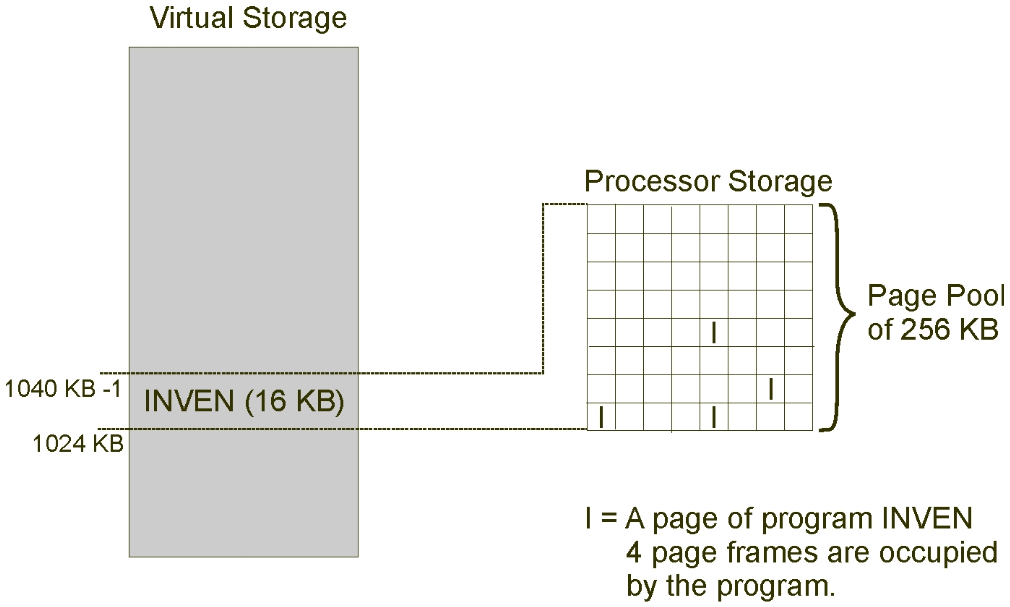


Figure 3. Running a Program in Virtual Storage

All of the information pertaining to the virtual storage and page frames is maintained within the system in a series of tables. These tables describe the virtual storage. Entries in these tables reflect the status of a given page of virtual storage.

Relate Virtual Storage to Locations in Processor Storage

The system does not anticipate where in processor storage a page is loaded. It translates the virtual addresses into real addresses when required for execution.

If an entire program fits into processor storage, none of the program's pages are placed on the page data set.

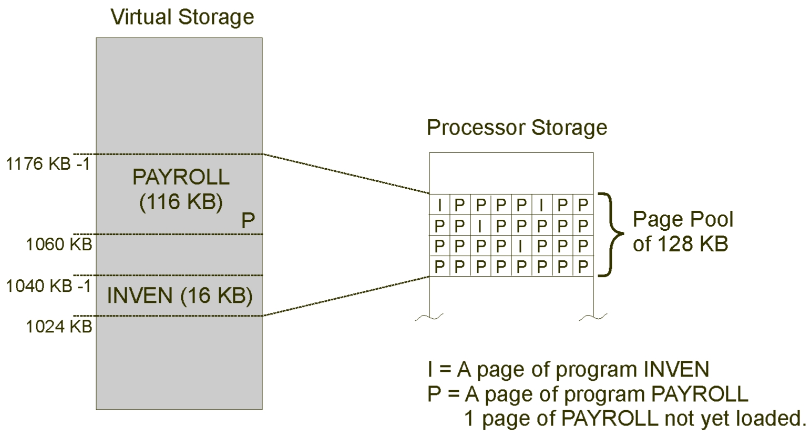
In the example shown in Figure 3 on page 3, no page of INVEN is paged out as long as the demand on processor storage does not exceed the number of available page frames.

If a second program is executed (multiprogramming) and this program together with INVEN is larger in size than the number of available frames in the page pool, the system would store as many (currently unused) pages as necessary on the page data set to keep both programs running.

In Figure 4 on page 4, a program that is called PAYROLL is being executed as well as INVEN. PAYROLL is a 116 KB program and conceptually loaded at virtual storage location 1060 KB. As the page pool in this example is only 128 KB, the total demand (INVEN + PAYROLL) of 132 KB exceeds the processor storage resource by 4 KB or one page frame.

The program PAYROLL does not start until all of its pages are loaded into processor storage. After 112 KB of program PAYROLL are loaded, the supervisor must make one page frame available for that program. It does this by selecting the least recently used page of program INVEN and storing it on the page data set. Once the page is saved on the page data set, the related page frame is available for the last page of program PAYROLL.

Note: The values that are used in this figure are for demonstration only - they do not reflect real system values.



During execution, whenever a required instruction or some data is not present in processor storage, execution is interrupted by a page fault. The required page must then be read into processor storage from the page data set.

Figure 4. Page Data Set Usage

Address Space Layout

While a virtual address space can have a maximum size of up to 2 GB the virtual storage concept makes it possible that processor (real) storage is much smaller. The minimum required is 64 MB.

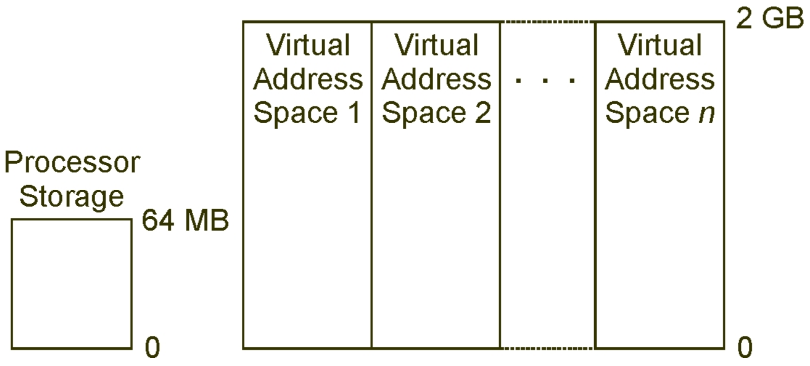


Figure 5. Single and Multiple Address Spaces

A virtual address space is divided into the:

- Private area for one or more private partitions.
- Shared areas 24-bit and shared areas 31-bit.

z/VSE Planning describes the storage and address space layout of the predefined environments that can be selected for initial installation.

Virtual Address Spaces versus Real Address Space

As shown in Figure 6 on page 5, z/VSE distinguishes virtual address spaces (identified in the Figure by 0 through X1) where X1 reflects a dynamic partition and a real address space (identified in the Figure by R).

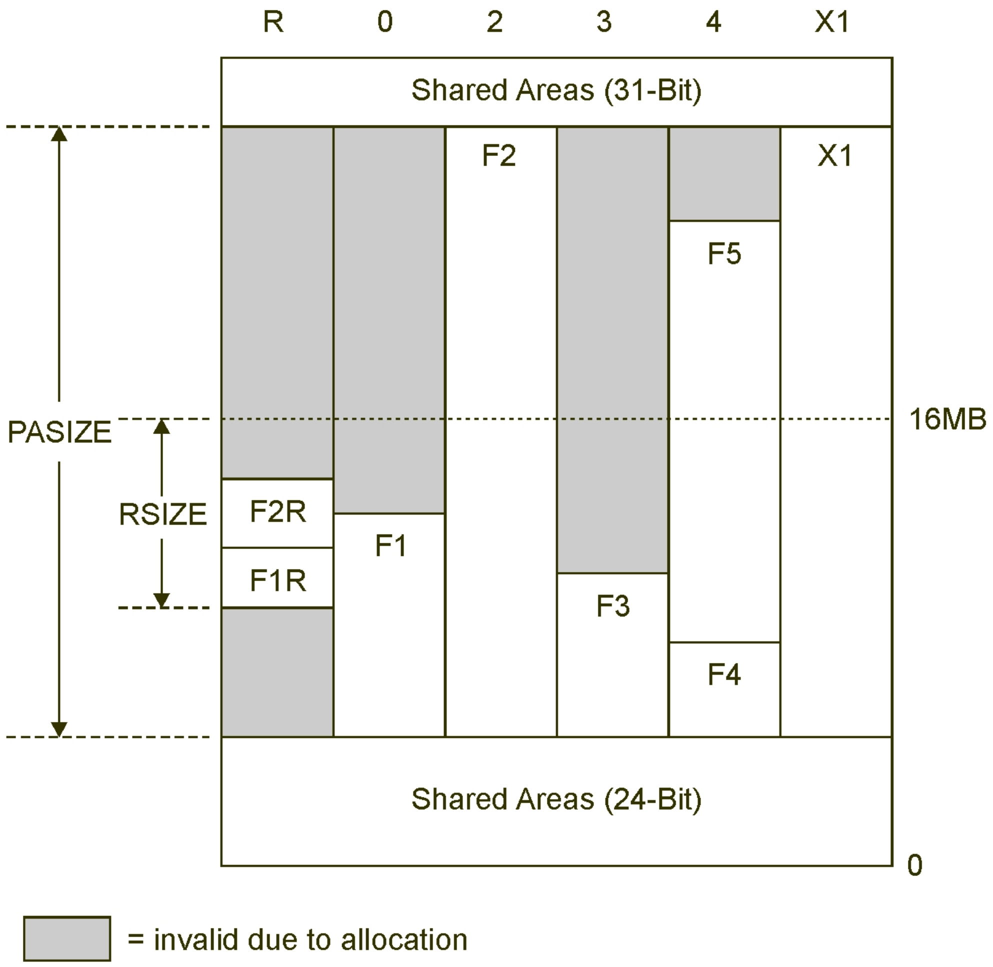


Figure 6. z/VSE Storage Layout (Virtual Address Spaces/Real Address Space)

Fx partitions reside in virtual address spaces. FxR are real partitions allocated with **ALLOC R** (where **RSIZE** defines the available storage for all **ALLOC R** definitions). Section "Defining Real Storage" on page 11 provides further details.

PASIZE is the maximum size available for private partition allocations within a virtual address space.

Layout of a Virtual Address Space

The address space layout that is shown (size >16 MB) applies also if the address space size is 16 MB, which is the required minimum.

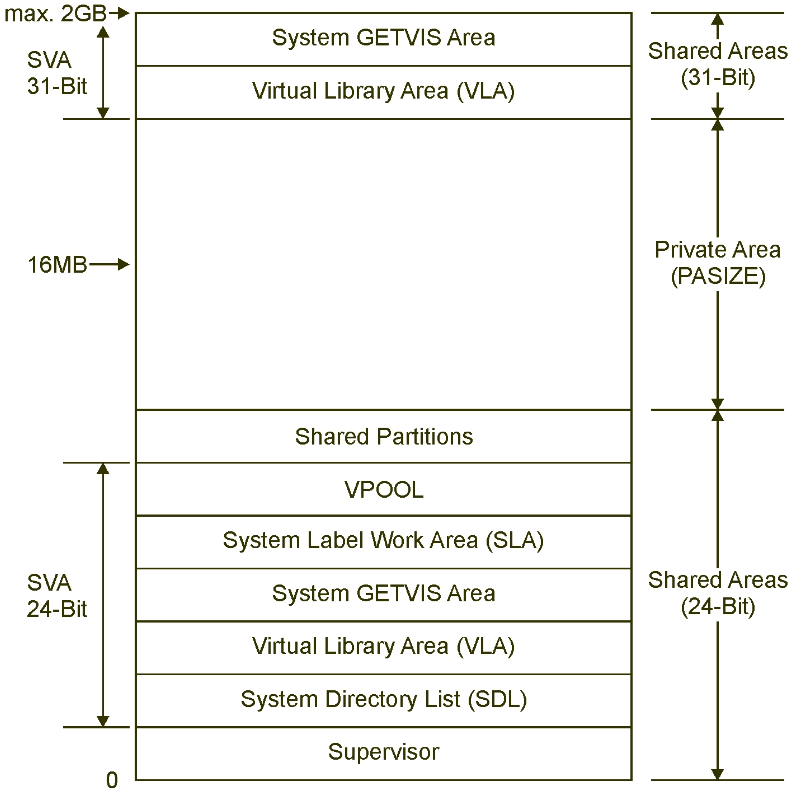


Figure 7. Virtual Address Space Layout (Size >16 MB)

The possible maximum size of this address space is 2 GB. For a running system, the actual size of an address space is to be calculated as follows:

```
PASIZE + size of shared areas 24-bit and 31-bit
```

The SVA (24-Bit) includes the following:

- The **VPOOL** area is needed to exchange data with the VIO (virtual I/O area).
- The **SLA** is the area that is used by z/VSE to store and maintain system and user label information.
- The system **GETVIS** area is an area of virtual storage that is reserved for use by the system.
- The **VLA** is the area in which phases resident in the SVA are stored.
- The **SDL** is the directory of the phases to be loaded into the SVA during system start.

The SVA (31-Bit) includes the following:

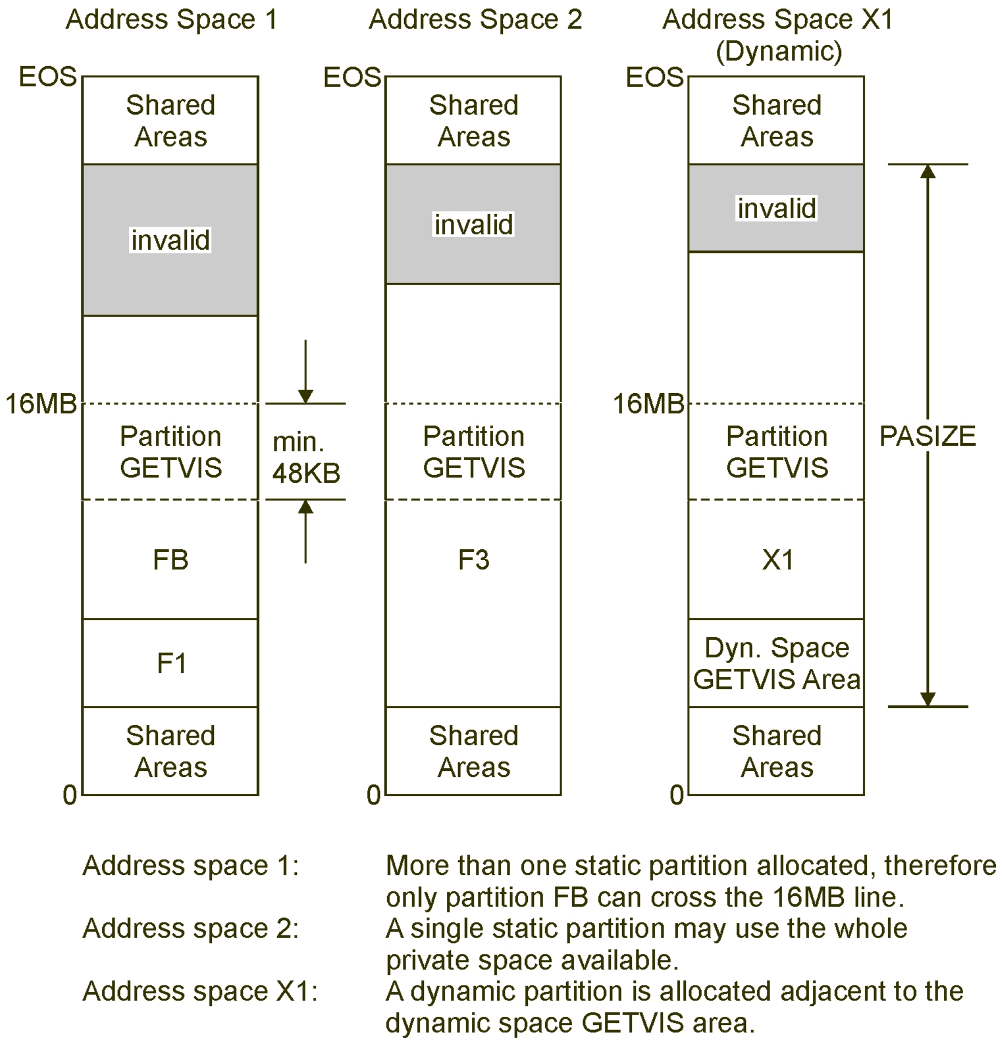
- The system **GETVIS** area is an area of virtual storage that is reserved for use by the system.
- The **VLA** is the area in which phases resident in the SVA are stored.

Note:

1. The shared areas (31-Bit) can start below 16 MB, dependent on the PASIZE specification and the shared areas (24-Bit), and can cross the 16 MB line.
2. SDL, SLA, VPOOL, and shared partitions are only available in the shared areas (24-Bit).
3. The private area must start at least 1 MB below 16 MB.

Partition Allocation and Program Size Considerations

Figure 8 on page 7 shows possible partition allocations for static partitions and a dynamic partition.



Address space 1: More than one static partition allocated, therefore only partition FB can cross the 16MB line.

Address space 2: A single static partition may use the whole private space available.

Address space X1: A dynamic partition is allocated adjacent to the dynamic space GETVIS area.

Figure 8. Partition Allocation Examples

Notes to Figure 8 on page 7:

1. The space marked as invalid cannot be used due to the values used for allocation (set through the ALLOC command).
2. EOS means "end of virtual storage" and is the sum of PASIZE plus the size of the shared areas. The resulting value must be <=2 GB (2048 MB).

Each partition requires a minimum program area of 80 KB and a minimum partition GETVIS area of 48 KB below 16 MB.

The size of a partition is defined by the allocation value. The maximum allocation value is determined by PASIZE (a parameter of the IPL SYS command). For a static partition, the allocation value is defined in the ALLOC command (included in the ALLOC procedure). For a dynamic partition, the allocation value is set in the dynamic class table (DTR$DYNC) and includes the size of the dynamic space GETVIS area.

Figure 9 on page 8 shows the layout for a dynamic partition. The layout for a static partition is identical except that it does not have a dynamic space GETVIS area.

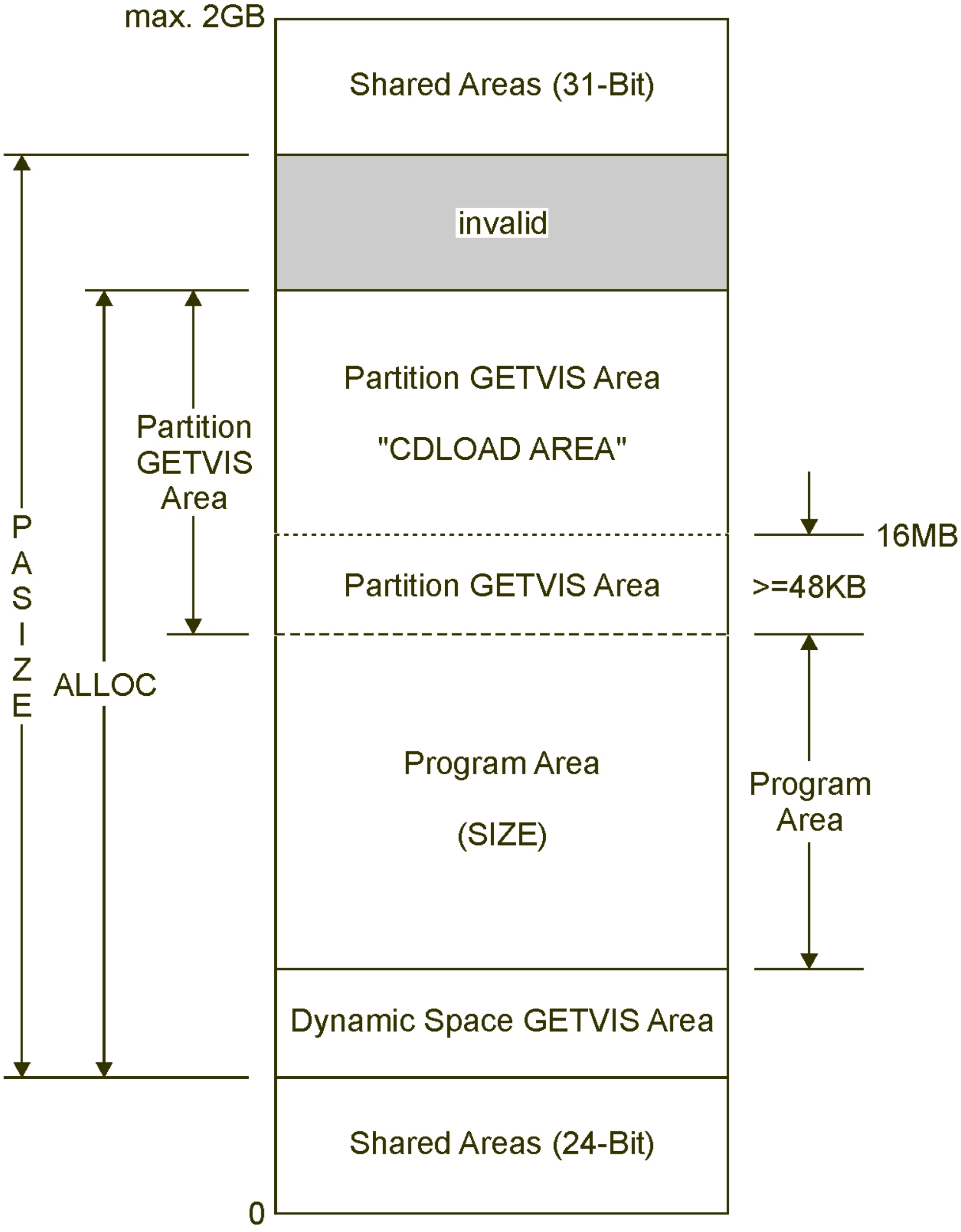


Figure 9. Partition Size and Program Area of a Dynamic Partition

A partition is divided into a program area and a partition GETVIS area. By use of the EXEC statement a program is loaded into the program area. The size of a program that can be loaded is restricted by the size of the program area. The size of the program area is either determined via the SIZE parameter in the EXEC statement, or the SIZE value in the dynamic class table (for a dynamic partition), or the SIZE command (for a static partition). The size of the remaining partition GETVIS area can then be calculated as follows:

Dynamic Partition

```
Partition GETVIS Area = ALLOC - SIZE - Dynamic Space GETVIS Area
```

Static Partition

```
Partition GETVIS Area = ALLOC - SIZE
```

The program area is always located completely below 16 MB. Its maximum size can be calculated as follows:

1. If a partition is <= 16 MB:
   - Dynamic Partition:

     ```
     Maximum Program Area = ALLOC - 48 KB - Dynamic Space GETVIS Area
     ```

   - Static Partition:

     ```
     Maximum Program Area = ALLOC - 48 KB
     ```

2. If a partition is > 16 MB:
   - Dynamic Partition:

     ```
     Maximum Program Area = 16 MB - (48 KB + Shared Area 24-Bit + Dynamic Space
                                     GETVIS Area)
     ```

   - Static Partition:

     ```
     Maximum Program Area = 16 MB - (48 KB + Shared Area 24-Bit)
     ```

Once a program is loaded into the program area, it can load additional phases with the CDLOAD macro into the partition GETVIS area above 16 MB (identified in Figure 9 on page 8 for better understanding as "CDLOAD Area").

Using the Virtual Storage Map

Before you reallocate virtual storage of an existing partition or initialize a new partition by allocating storage to it, you may first obtain a virtual storage map for getting the current storage and partition values of your system. You can obtain such a map using the:

- *Display Storage Layout* dialog

  For a detailed description of this dialog refer to z/VSE Administration.
- MAP command

  For a detailed description of this command refer to z/VSE System Control Statements.

GETVIS Areas

Certain functions need to acquire virtual storage dynamically during program execution. The GETVIS areas are used for this purpose. A program requests GETVIS space via the GETVIS macro. For a detailed description of this macro, refer to z/VSE System Macros Reference.

z/VSE maintains two GETVIS areas for a static address space and three for a dynamic address space:

- **Partition GETVIS area**

  Each partition has its own partition GETVIS area which may cross the 16 MB line depending on the allocation value. The minimum is 48 KB and must be below 16 MB as shown in Figure 8 on page 7, for example.
- **System GETVIS area**

  As the name implies, this area is reserved for system use. It is permanently assigned and belongs to the shared areas of virtual storage.

  z/VSE supports a 24-bit and a 31-bit system GETVIS area located in the corresponding shared areas. The 31-bit area may reside partly or completely below 16 MB.
- **Dynamic space GETVIS area**

  For dynamic partitions, z/VSE supports a 24-bit dynamic space GETVIS area. It can be considered as an extension of the system GETVIS area and is defined via the dynamic class table. The area exists from dynamic partition initialization until partition deactivation. The size of the dynamic space GETVIS area (the minimum is 128 KB) also influences the maximum partition size of a dynamic partition. The maximum size of a dynamic partition is:

```
PASIZE - Dynamic Space GETVIS Area
```

The following figures (Figure 10 on page 10 and Figure 11 on page 11) show in detail the layout of the GETVIS areas within an address space.

Layout of a Static Address Space and its GETVIS Areas

Figure 10 on page 10 shows the layout of an address space with a static partition below 16 MB and the System GETVIS Area (31-Bit) located partly below 16 MB.

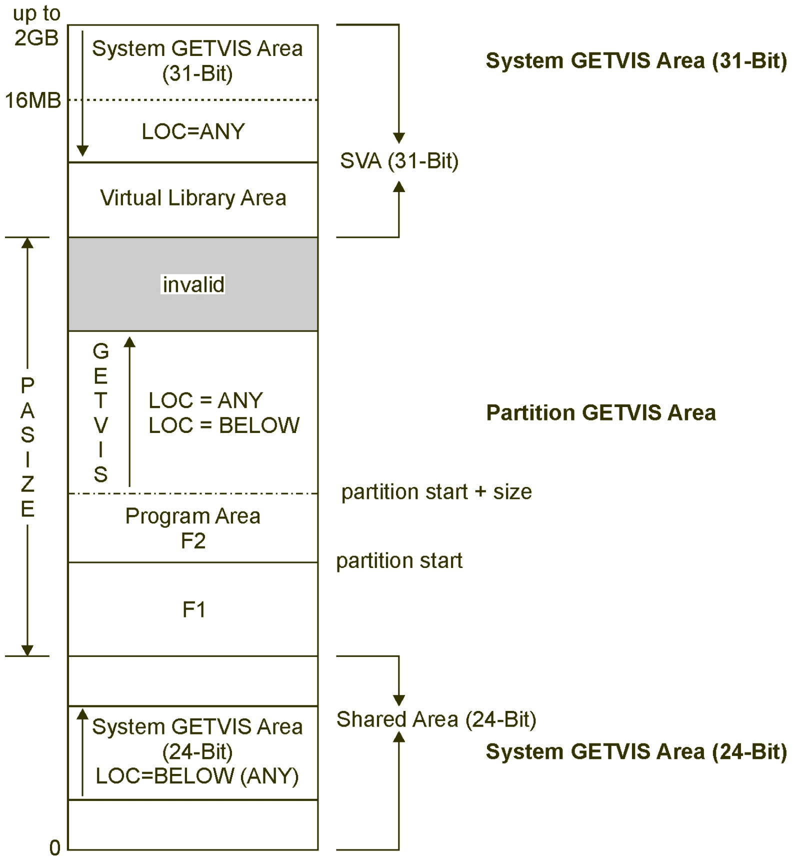


Figure 10. Static Address Space, Partition Layout, and GETVIS Areas

For a description of the GETVIS macro and its parameters, refer to .

Layout of a Dynamic Address Space and its GETVIS Areas

Figure 11 on page 11 shows the layout of an address space with a dynamic partition where the System GETVIS Area (31-Bit) is located above 16 MB.

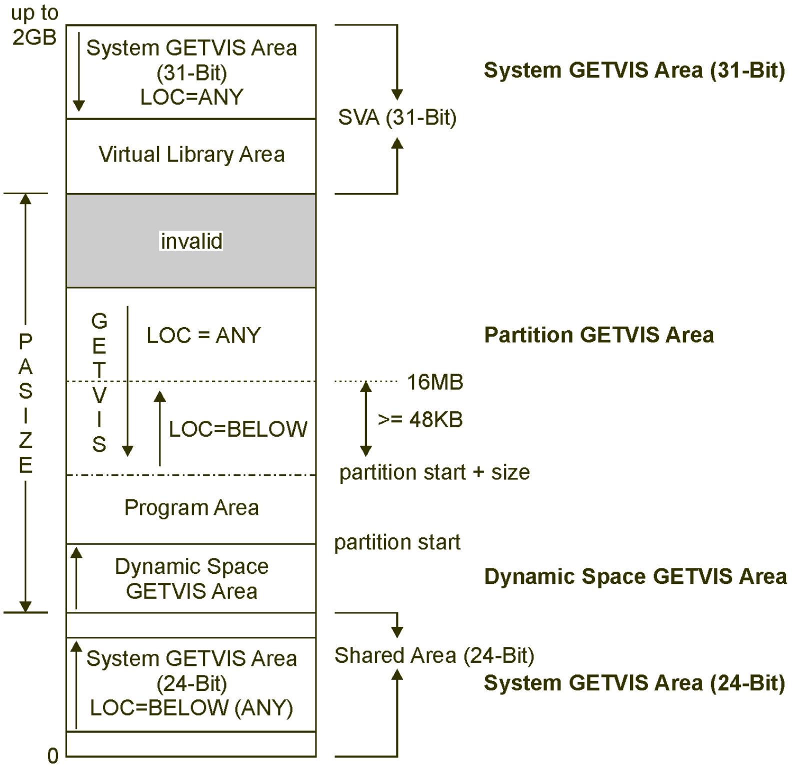

up to 2GB
System GETVIS Area (31-Bit) LOC=ANY
Virtual Library Area
invalid
GETVIS
LOC = ANY
LOC=BELOW
Program Area
Dynamic Space GETVIS Area
System GETVIS Area (24-Bit) LOC=BELOW (ANY)
0
PASIZE
SVA (31-Bit)
16MB
>= 48KB
partition start + size
partition start
Shared Area (24-Bit)
System GETVIS Area (31-Bit)
Partition GETVIS Area
Dynamic Space GETVIS Area
System GETVIS Area (24-Bit)

Figure 11. Dynamic Address Space, Partition Layout, and GETVIS Areas

For a description of the GETVIS macro and its parameters, refer to .

Defining Real Storage

For a program that is to be executed in real mode (EXEC ...,REAL), a real partition has to be specified with the ALLOC R command. Real partitions can only be specified for static partitions and are always located below 16 MB. Real partitions can also be used for fixing pages with the PFIX macro.

The total size which is needed for all ALLOC R has to be specified during IPL with the RSIZE operand of the IPL SYS command.

If only the fixing of pages (PFIX macro) is needed, no ALLOC R is necessary. Instead, the PFIX limits should be specified with the JCL command SETPFIX which can be used for both, static and dynamic partitions. PFIX limits can be specified for a BELOW and an ANY area. The BELOW area is located completely below 16 MB, whereas the ANY area may cross the 16 MB line.

The distinction between PFIX and ALLOC R areas results in a layout of real storage as shown in Figure 12 on page 12.

Note:

1. If a BELOW area for PFIX is reserved for a partition, no ALLOC R can be given for that partition and vice versa.
2. The number of page frames currently not fixed in the PFIX BELOW and ANY area are made available for system use.

For details about the PFIX macro refer to z/VSE System Macros Reference. For details about the JCL SETPFIX statement refer to z/VSE System Control Statements.

Real Storage Layout

Real storage means the processor storage physically available.

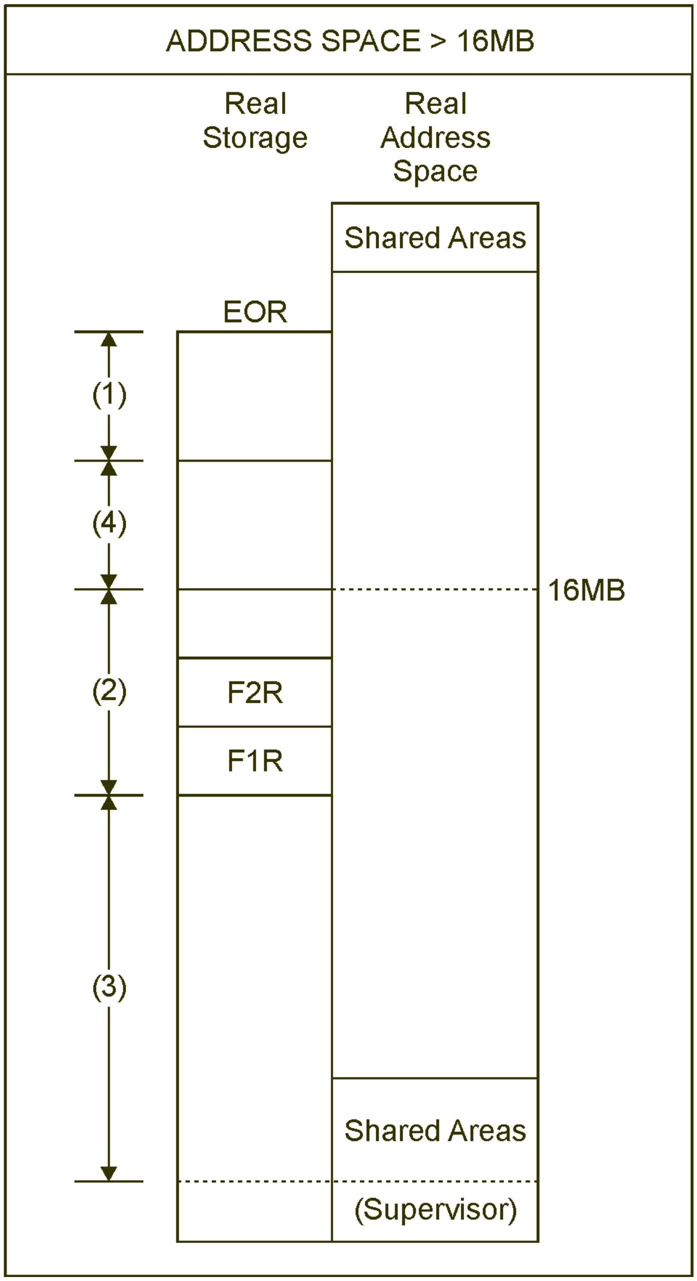


Figure 12. z/VSE Real Storage Layout

The area numbers have the following meaning:

(1) - Reserved for system use (EOR = end of real storage)
(2) - Available for EXEC REAL and PFIX (24-Bit) - reserved by RSIZE
(3) - Available for PFIX (24-Bit) - reserved by SETPFIX statement
(4) - Available for PFIX (31-Bit) - reserved by SETPFIX statement

The number of page frames currently not reserved in area (3) are made available for system PFIX (24-Bit). The number of page frames currently not reserved in area (4) are made available for system PFIX (31-Bit).

Data Spaces

A data space is an area in virtual storage similar to an address space. However, it contains data only. A data space can have a maximum size of 2 GB.

A typical use of a data space is a virtual disk for holding temporary files.

Refer to z/VSE Planning for details on using data spaces and virtual disks.

64-Bit Address Space

A 64-bit address space is a virtual address space that is supported by 64-bit addresses. It provides a vastly increased amount of virtual storage, compared to the first 2 GB of virtual storage previously available.

Storage in 64-bit address space above the 2 GB address, called the "bar", can be used to hold memory objects. Memory objects can be allocated for data only. Programs cannot run in the virtual storage above the bar. The size of a 64-bit address space is limited by the maximum value of VSIZE, which is currently 90 GB.

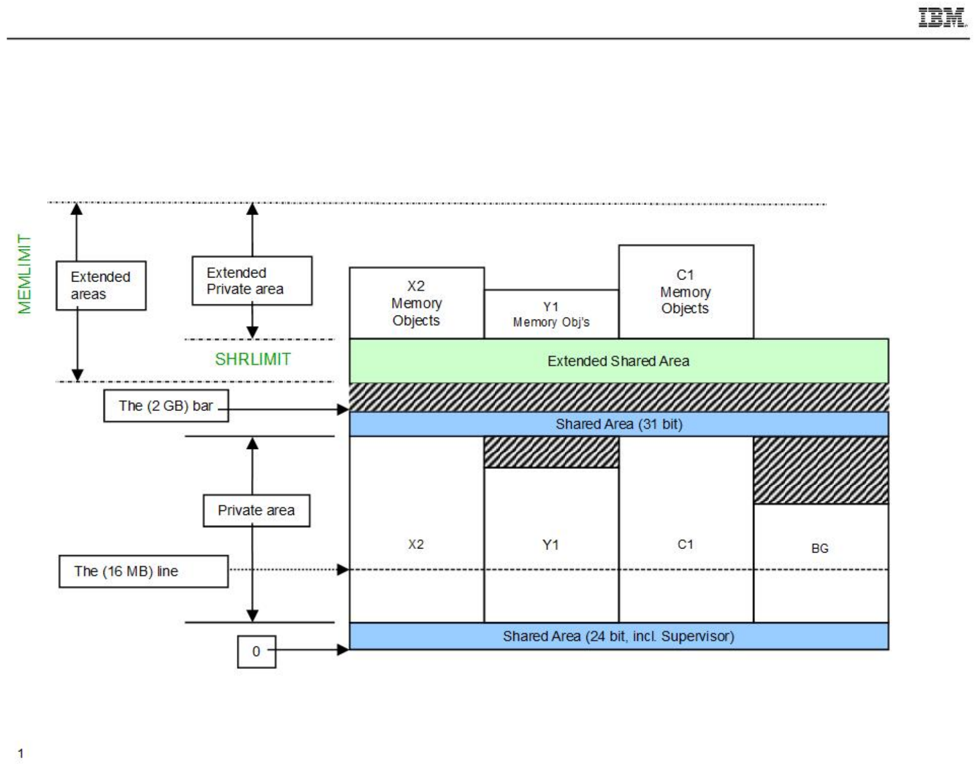


Figure 13. 64-Bit Virtual Address Space Layout

Refer to z/VSE Extended Addressability for details on 64-bit addressing support.

Executing Programs in Virtual and Real Mode

All programs when executing are conceptually running in the address space associated with a partition. The system selects page frames from the page pool for the pages of the executing programs. Execution can be in one of two modes:

Execution in Virtual Mode

The page frames occupied by pages of programs running in virtual mode continue to be part of the page pool. The system manages the processor storage, placing some pages on the page data set, when necessary, and retrieving pages as required. Programs running in virtual mode are pageable.

Execution in Real Mode

Note: Real mode execution is possible in a static but not in a dynamic partition.

The page frames occupied (previously reserved via ALLOC R) by pages of programs running in real mode are taken out of the page pool for the duration of that program's execution; these page frames will not be selected for another program. The program is fixed in processor storage and is non-pageable.

To have a program executed in real mode, an amount of processor storage must be allocated to the partition in which that program is to run. However, this processor storage remains part of the page pool until real-mode execution begins; it becomes part of the page pool again when real-mode execution ends. Certain programs, such as those with critical time dependencies, may have to run in real mode.

A partition may execute in only one mode at a given point in time; for example, the BG partition cannot initiate both real and virtual execution at the same time.

Processor Storage Allocation for Real Mode Execution

Refer to "Defining Real Storage" on page 11 for an introduction to the RSIZE operand of the IPL SYS command and the JCL command ALLOC R.

A specific number of page frames of processor storage can be allocated to any static partition for real mode execution. The allocation can be done at any time with the ALLOC R command.

Submitting

```
    ALLOC R,F7=40K,F8=24K
```

for example, causes the following:

> 40 KB and 24 KB of real address space are allocated to partitions F7 and F8, respectively. When real mode execution takes place, the processor storage addresses used by the system are the same as the addresses within the allocated real address space.

With the above ALLOC R command the largest program that can be executed real in the two partitions are 40 KB in F7 and 24 KB in F8.

Note: Allocated pages are PFIXed, if an *EXEC program,REAL* is issued for a partition. Whenever a page in the allocated area has been fixed by another partition, this page cannot be PFIXed for real execution until the other partition frees this page. You should be aware that an outstanding reply or a pause can keep a page fixed in the allocated real area.

Fixing Pages in Processor Storage

Note that the fixing of pages is only possible if a JCL SETPFIX or an ALLOC R command was given before. Refer also to "Defining Real Storage" on page 11 for additional details.

Allocated page frames are used not only for programs running in real mode, they may also be used for programs running in virtual mode. For example, for instructions or data that must be in processor storage and, therefore, cannot tolerate paging. The pages containing such code or data can be fixed via the PFIX (page fix) macro, and freed immediately after use via the PFREE (page free) macro.

When pages of a program running in a given partition are fixed in response to the PFIX macro, they are fixed in the page frames allocated to the partition. If a PFIX macro is issued and not enough storage has been allocated, the pages are not fixed, and a completion code indicating this is returned to the program.

Fixing pages in processor storage means that fewer page frames are available to other programs running in virtual mode thus potentially degrading total system performance. If you have programs with large I/O areas (fixed by the system for I/O operations), it reduces the initial size of the page pool and may degrade performance. Consider this effect carefully before allowing in addition the use of the PFIX macro at your installation.

For details about the PFIX and PFREE macros, refer to z/VSE System Macros Reference, and z/VSE System Macros User's Guide.

Chapter 2. Starting the System

The process of system startup consists of IPL (Initial Program Load) and the subsequent initialization of static partitions. It includes steps such as loading the supervisor into storage, defining the virtual storage layout, and setting partition parameters and values.

To allow an almost automated system startup with a minimum of operator intervention, z/VSE provides the Automated System Initialization (ASI) support. ASI has all the information that is required for system startup that is stored as cataloged procedures in the system sublibrary IJSYSRS.SYSLIB.

Note: Startup should always be performed via ASI procedures. It is not recommended to perform IPL interactively, entering each command manually at the system console (SYSLOG) since this is a cumbersome and time-consuming method to start your system. When using ASI you can interrupt IPL processing and modify IPL parameters as described under "Interrupt IPL Processing for Modifications" on page 23.

In the following discussion, the term "system startup" always implies that startup is performed through ASI procedures.

Predefined Startup Support

z/VSE includes, as shipped, for each predefined environment all necessary ASI IPL and JCL startup procedures. This includes procedures for partition allocations, library search definitions, label definitions, and others.

The startup sequence, the procedures that are involved, and how to modify them is described in detail in z/VSE Administration.

This section provides background information about system startup and its related ASI procedures and provides specific details about:

- ASI master procedure ($ASIPROC)
- Interrupting IPL processing for modifications
- Loading phases into the shared virtual area (SVA)

A further topic that is related to system startup is documented under "Writing an IPL Exit Routine" on page 167.

ASI Procedures

System startup always begins with the ASI IPL procedure and continues by processing the ASI JCL Procedures. ASI requires one procedure for IPL (ASI IPL procedure), and one job control procedure per partition (ASI JCL procedures).

Contents of an ASI IPL Procedure

IPL commands set or change various characteristics of your system. In addition to the supervisor parameters command (always the first command of an IPL procedure), the following IPL commands are available:

I/O configuration
ADD and DEL commands

Lock (communication) file
DLF command

System date and time
SET command

Activate XPCC/APPC/VM support
SET XPCC command

Daylight saving support
SET ZONEDEF command

Daylight saving support
SET ZONEBDY command

SCSI device connection definitions
DEF SCSI command

System disk file assignments
DEF command

Page data set definitions
DPD command

Supervisor parameters
SYS command

Shared virtual area definitions
SVA command

ADD and DEL commands precede all other commands. The DLF command (if any) must immediately follow all ADD/DEL commands. The SVA command is the last command to be submitted.

z/VSE provides a set of predefined IPL procedures for initial installation. z/VSE selects and modifies one of these procedures during initial installation according to the actual installation environment and renames it to $IPLESA. Figure 14 on page 17 shows as an example a $IPLESA procedure and how it might look like after initial installation has been completed. The example shown is for predefined environment B and DOSRES resides on an IBM FBA (SCSI) disk device.

z/VSE provides the *Tailor IPL Procedure* dialog to modify a current IPL procedure ($IPLESA) at any time if required. The dialog is described in detail in z/VSE Administration.

```
009,$$A$SUPI,VSIZE=264M,VIO=512K,VPOOL=64K,LOG
ADD 009,3277
ADD 00C,2540R
ADD 00D,2540P
ADD 00E,1403
ADD 02E,PRT1
ADD 180,3490E,08
ADD 181,3490E
ADD 188,3420T9,D0
ADD 189,TPA
ADD 18A,TPA,08
ADD 190:191,ECKD,DVCDN
ADD 19D:19E,ECKD,DVCDN
ADD 230:232,ECKD,DVCDN
ADD 300:309,3277
ADD 310:311,FBA
ADD 500:502,OSAX
ADD 888:889,3745,01
ADD 999,3277
ADD C02,FCP
ADD D00,FCP
ADD FDF,FBAV
ADD FEC,3505            POWER DUMMY READER, DO NOT DELETE
ADD FED,2520B2          POWER DUMMY PUNCH, DO NOT DELETE
ADD FEE,PRT1            POWER DUMMY PRINTER, DO NOT DELETE
ADD FEF,PRT1            POWER DUMMY PRINTER, DO NOT DELETE
ADD FFA,3505            ICCF INTERNAL READER, DO NOT DELETE
ADD FFC,3505            ICCF DUMMY READER, DO NOT DELETE
ADD FFD,2520B2          ICCF DUMMY PUNCH,  DO NOT DELETE
ADD FFE,PRT1            ICCF DUMMY PRINTER, DO NOT DELETE
ADD FFF,CONS            DUMMY CONSOLE, DO NOT DELETE
DEF SCSI,FBA=310,FCP=C02,WWPN=5005076300C69A76,LUN=560A000000000000
DEF SCSI,FBA=311,FCP=D00,WWPN=5005076300CE9A76,LUN=560B000000000000
SET ZONE=WEST/00/00
DEF SYSCAT=DOSRES
DEF SYSREC=SYSWK1
SYS BUFSIZE=1500
SYS NPARTS=60
SYS DASDFP=YES
SYS SEC=NO
SYS PASIZE=70M
SYS SPSIZE=0K
SYS BUFLD=YES
SYS SERVPART=FB
DPD VOLID=DOSRES,BLK=315392,NBLK=50176,TYPE=N,DSF=N
DPD VOLID=DOSRES,BLK=365568,TYPE=N,DSF=N
SVA SDL=700,GETVIS=(768K,6M),PSIZE=(652K,6M)
/+
/*
```

Figure 14. Example of an IPL Procedure ($IPLESA) after Initial Installation Complete

Contents of ASI JCL Procedures

ASI Background Procedure

This procedure must contain all job control statements and commands necessary to initialize the BG partition and the system as a whole:

- ALLOC commands to allocate virtual and real storage to the foreground partitions you intend to start.
- All permanent library definitions or assignments of logical units needed in the BG partition.
- The // SETPFIX statement (or SETPFIX command) for setting PFIX limits.
- The SIZE command for defining the maximum program size.
- // STDOPT statement for the definition of standard (permanent) options.
- // OPTION STDLABEL, together with label information, to set up the system standard label subarea if it was not set up during a previous system initialization.
- // OPTION PARSTD, together with label information, to set up (background or foreground) partition standard label subareas if they were not set up during a previous system initialization.

- // OPTION CLASSTD, together with label information to set up class standard label subareas for dynamic partitions if they were not set up during a previous system initialization.
- // JOB jobname for the initialization of the system recorder file and the hardcopy file.
- The START Fn command for each foreground partition to be started from this BG partition.
- The STOP command if the BG partition is to be spooled by VSE/POWER. The STOP command should immediately follow the START command for the VSE/POWER partition.

ASI Foreground Procedures

Such a procedure must include job control statements and commands necessary to initialize a particular foreground partition:

- // OPTION PARSTD, followed by label information, to set up the foreground partition standard label subarea if it was not set up during a previous system initialization or from the background partition.
- All permanent library definitions or assignments of logical units needed in the particular foreground partition.

Note: For how to initialize dynamic partitions, refer to z/VSE Planning and z/VSE Administration.

Naming Conventions for ASI Procedures

z/VSE assigns certain default names during initial installation. The defaults are:

```
IPL:   $IPLESA
```

```
JCL:   $0JCL
       $1JCL
       $2JCL
       .
       .
       .
```

When you catalog your own ASI JCL procedures, you must observe the same naming rule as when you catalog a partition-related procedure. The first character must be a $. The second character identifies the partition: 0 for the BG-partition, 1 for the F1-partition etc. The remaining characters must be identical for all procedures belonging to one set.

You might want to use names different from the default names. For example, the initialization of your system during the day deviates from that of the night shift. The day shift runs a full z/VSE system (including VSE/POWER, VTAM, CICS) whereas the night shift runs only simple batch jobs. In this case, you might prefer to use procedure names as follows: $IPLD, $0JCLD, $1JCLD, $2JCLD, ... for the day shift, and $IPLN, $0JCLN, $1JCLN, $2JCLN ... for the night shift. This applies only to static partitions; for dynamic partitions the procedure name is to be specified in the dynamic class table (as profile name).

Starting Up the System

For a formal step-by-step description of how to perform IPL, which is very much determined by your processor and hardware configuration, consult z/VSE Operation.

After the operator has initiated system startup (through performing IPL), the system searches for the IPL and JCL procedure names in the following sequence:

1. Retrieves the names from $ASIPROC, if a $ASIPROC master procedure exists. For a detailed description of creating a master procedure refer to “The ASI Master Procedure ($ASIPROC)” on page 19.
2. Uses the default names: $IPLESA and $$JCL (where the second $ is the partition placeholder).

In addition, z/VSE allows you to interrupt startup processing as described under “Interrupt IPL Processing for Modifications” on page 23 and enter IPL and JCL procedure names if required.

The ASI Master Procedure ($ASIPROC)

Note: As shipped, z/VSE includes only a $ASIPROC master procedure for initial installation (TYPE=INSTALL). The following information helps you implement your own $ASIPROC at your installation.

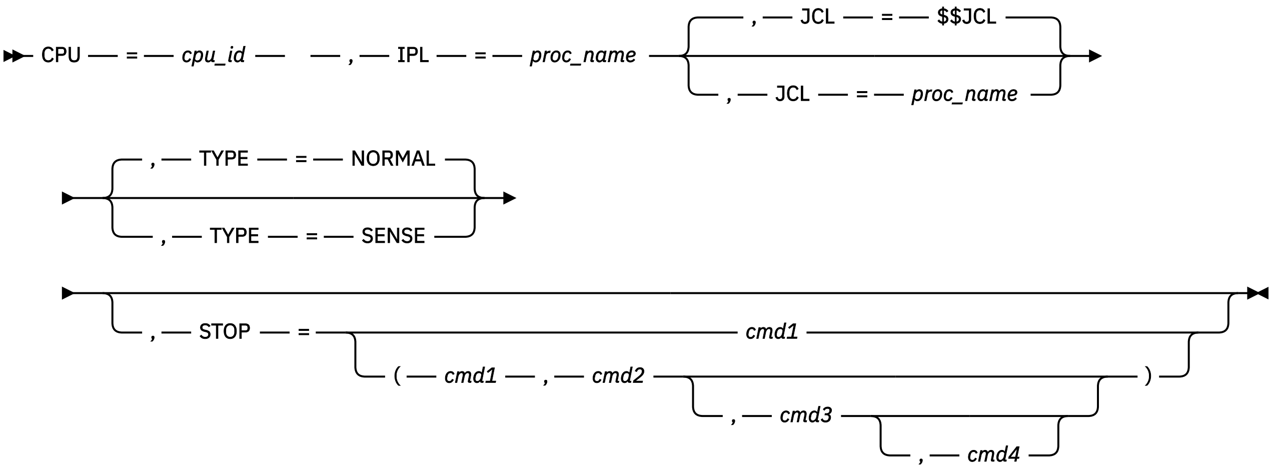
A typical example of using an ASI master procedure is an environment with multiple z/VSE systems sharing the DOSRES (system residence) disk device or an environment where two or more z/VSE systems run as guest systems under VM. An ASI master procedure is also useful

- if you have only one procedure set, but want to use other than default IPL and JCL procedure names, or
- if you plan to use the STOP facility as described in detail under "STOP Processing" on page 27; for example, when you are still "debugging" your ASI procedures.

The STOP facility allows you to specify, via the STOP parameter (see below), up to four different IPL commands. Upon encountering the first of a particular command type, the automatic IPL process stops, and gives the operator a chance to enter or update IPL commands at the console.

To build the master procedure, provide one (CPU) statement for each system. The statement allows you to specify the following parameters where the parameters CPU and IPL are mandatory. The syntax looks as follows:

Format



Parameteres

CPU=*cpu_id*
Specifies 12 hexadecimal digits to identify the CPU on which an ASI procedure is to be run. The *cpu_id* should be taken from message 0I04I which is issued automatically during IPL. If z/VSE runs as a guest system under VM, the first two digits of the *cpu_id* are ignored.

IPL=proc_name
Specifies the name of the ASI IPL procedure.

JCL=*proc_name*
Specifies the name of the ASI JCL procedure set; the name must start with $$.

Default: $$JCL

TYPE=*type*
For *type* you can specify one of the following:

NORMAL
This is the default and causes each ADD command of the IPL procedure to be checked for the correct device type.

SENSE

Causes IPL to sense devices and generate corresponding device information which is stored in PUB (physical unit block) entries. This is done for all devices defined in the IOCDS (Input/Output Configuration Data Set). Working in this way, the sense function ensures that you do not have to ADD any device unless:

- A device is not operational.
- A device does not support device sensing.
- A device requires special options to be defined (SHR, for example).
- Dummy devices are to be added.

Note: Parameter INSTALL is reserved for the system and used for initial installation of z/VSE only. It is identical to SENSE but in addition some installation-related processing is performed.

STOP=*command(s)*

A list of up to four different IPL commands, in arbitrary sequence. If more than one is specified, the commands must be enclosed within parentheses and separated by commas. The first of a specified command type that is encountered during IPL causes an interruption before the command is processed. This enables the operator to modify IPL parameters and commands. Refer also to "Modifying IPL Parameters" on page 26.

Cataloging an ASI Master Procedure

Following is a job stream example of how to catalog the master procedure:

```
// JOB CATALOG $ASIPROC
// EXEC LIBR
ACCESS SUBLIB=IJSYSRS.SYSLIB
CATALOG $ASIPROC.PROC
CPU=FFxxxxxx2094,IPL=$IPLX,TYPE=SENSE
CPU=FFxxxxxx2094,IPL=$IPLY,JCL=$$JCLY,STOP=(DEF,DLF,DPD)
/+
/*
/&
```

The FF in the CPU statements indicates that z/VSE is running under VM. For a z/VSE that runs natively, these two characters would show the model number of a processor. If the CPU identification of a virtual machine is equal (except for the first two characters) to another CPU identification in the master procedure, the entry for the virtual machine must come first. The CPU identification should be taken from message 0I04I which is issued automatically during IPL. If running under VM, replace xxxxxx with the CPU identification defined in the VM directory of your z/VSE guest system or specified with the VM CP command SET CPUID.

For procedure $IPLX, the system will use device sensing to add devices. As no TYPE operand is specified with $IPLY, the system will merely check the validity of the ADD commands in this procedure.

If you want devices added by device sensing in most of the IPL procedures addressed by the ASI master procedure, enter TYPE=SENSE as a command rather than an operand. It must be the first command in the master procedure. The IPL procedures which are not to use automatic adding must then be specified with the operand TYPE=NORMAL. You would get the same effect as in the example above by cataloging the following ASI master procedure:

```
 .
 .
TYPE=SENSE
CPU=FFxxxxxx2094,IPL=$IPLX
CPU=FFxxxxxx2094,IPL=$IPLY,JCL=$$JCLY,STOP=(DEF,DLF,DPD),TYPE=NORMAL
 .
 .
```

Note: The TYPE operand controls device sensing for one IPL procedure, the TYPE command sets the default value for all IPL procedures listed in the master procedure. In the example, TYPE=NORMAL overrides TYPE=SENSE for the second CPU statement.

Establishing the Communication Device for IPL

z/VSE supports the following types of system consoles for communication during IPL:

- **Integrated Console**

  This is the integrated console of an IBM S/390 service processor used to control and maintain the processor's configuration.

- **CRT Console**

  This is the system console support based on a local non-SNA 3270 terminal.

- **Line Mode Console**

  This is the system console support based on a virtual 3215 printer keyboard if z/VSE runs as a guest under VM.

Console Selection for Initial Installation

For initial installation either the Integrated Console may be used as system console (by specifying the appropriate load parameter) or a local terminal which is established as system console via interrupt. The IPL procedure shipped with z/VSE for initial installation contains an ADD command for a dummy system console:

```
ADD  FFF,CONS
```

If the Integrated Console is selected as system console for initial installation, it has the device number FFF.

During the installation process, z/VSE does device sensing and updates the IPL procedure with ADD statements for the devices attached. This is also true for the system console. Figure 14 on page 17 shows two ADD statements for the system console. The **ADD** *FFF,CONS* statement defines the Integrated Console, which can be used as system console.

```
ADD 009,3277
```

was added later after the device was recognized through device sensing. In this example for initial installation, the device was established as system console via interrupt and the console address 009 added to the first (supervisor command) statement defining it as the console for subsequent IPLs.

Console Selection when Performing a Normal IPL

During IPL, z/VSE selects the system console (SYSLOG) according to the following criteria:

1. IPL load parameter.
2. Device availability.
3. Console device specification in the ASI IPL procedure.
4. I/O interrupt received from a console device.

The basic selection rules are:

1. If a console type is specified in the IPL load parameter, then the system will route the messages to that device. This can be the Integrated Console or a local console (which is the default). Local console means CRT or Line Mode console.

   a. If a local console is requested or the Integrated Console is not available, the system will route messages to the local console specified in the ASI IPL procedure.

   b. If that device is not available or not operational, the system will wait for an interrupt from a local console.

2. If a system is IPLed without specification of a console type in the IPL load parameter, the system will route messages to a local console (CRT or Line Mode console).

a. The system selects first the console specified in the ASI IPL procedure.

b. If this device is not operational, the system waits for an interrupt from a local console.

Details about the IPL load parameter are provided under "The IPL Load Parameter" on page 23.

IPL Communication Device List

Any interrupt (on a first-come basis) establishes the issuing device as the IPL communication device (system console). It is advisable that terminal-oriented installations with locally attached terminals (such as the IBM 3277), install the IPL-phase $$A$CDL0, which defines communication device addresses valid for IPL.

Note: You cannot perform IPL from a device, which is not included in $$A$CDL0, once $$A$CDL0 has been installed, unless the system console is the Integrated Console.

$$A$CDL0 is not needed, if the Integrated Console is chosen as system console.

To build a restrictive pool of IPL communication devices, you assemble an IPL communication device list (CDL) and catalog the list under the phase name $$A$CDL0 in the system sublibrary IJSYSRS.SYSLIB. During IPL, this phase (if present) is loaded into storage. When the system enters the wait state and an interrupt occurs, the CDL is searched for the address of the device issuing the interrupt. If the address is listed, the interrupting device is accepted as an IPL communication device and processing continues. If the address is not found, the system remains in the wait state. Installation of the CDL is optional.

For IPL to be successful once $$A$CDL0 is installed, the SYSLOG (system console) device address must be present in the CDL. If you intend to submit IPL commands from card reader, you must enter their addresses in the CDL as well. To ensure backup in case of hardware errors during IPL, consider stand-by devices, such as another card reader or even an additional SYSLOG device in the CDL.

The CDL can have up to 8 entries, each of which is 4 bytes long:

Bytes
  Explanation

0 - 1
  Reserved

2 - 3
  cuu (device number)

You create the CDL by submitting a job that catalogs $$A$CDL0 into the system sublibrary IJSYSRS.SYSLIB as shown in Figure 15 on page 22.

```
// JOB CATALOG CDL
// LIBDEF PHASE,CATALOG=IJSYSRS.SYSLIB
// OPTION CATAL,NODECK
   PHASE $$A$CDL0,*
// EXEC ASMA90....
$$A$CDL0 CSECT
          DC  XL4'01F'     SYSLOG                (System Console)
          DC  XL4'0BD'     3277 Display Station
          .
          .
          END
/*
// EXEC LNKEDT
/&
```

Figure 15. Job Stream Example for Creating a CDL

Note: The statement

```
// EXEC ASMA90...
```

calls the High Level Assembler. Refer to "High Level Assembler Considerations" on page 139 for further details.

Once phase $$A$CDL0 has been cataloged, the CDL addresses remain effective for subsequent IPLs. However, you may:

- Replace the phase by another one. Either by assembling and link editing a new phase or by using the RENAME function of the librarian program to rename an already cataloged CDL that has a name other than $$A$CDL0.
- Override any CDL entry by manual intervention. This is the suggested approach if an erroneous CDL is cataloged in the system library. Use the MSHP CORRECT function that is described in z/VSE System Control Statements.

  The IPL procedure might loop indefinitely if the CDL is specified incorrectly. If this happens, it can be helpful to enter manually the device address for the system console and/or communication device (into low core locations X'10' to X'13' in hexadecimal format X'00000cuu') when IPL enters the WAIT state. If the *cuu* of the system console and the communication device are the same, it must be entered once. If they are different, it must be entered twice (X'00000cuu00000cuu'). To resume processing, press ENTER at the system console.

  Note: Manual intervention does not work for device number 000.

Interrupt IPL Processing for Modifications

If necessary, you can interrupt IPL processing and change currently defined procedure names or other IPL parameters. There are **three methods** to interrupt IPL processing and it depends on your processor which method you can use. These methods are described in detail under "Interrupt and Restart the IPL Process" on page 24 below.

Once interrupted, you can enter and modify IPL parameters at the system console as shown under "Modifying IPL Parameters" on page 26.

Restrictions when Using the Integrated Console

When your system console is an **Integrated Console**, you cannot interrupt IPL by pressing the ENTER key at the console. Also, you cannot restart IPL by an external interrupt.

If you want to change IPL parameters, you have to request prompting via the **IPL prompting code** in the IPL load parameter. This is the only method to be used with an Integrated Console and described under "Method 1: Interrupt IPL via the Load Parameter Facility" on page 24.

Restrictions when Using List-Directed IPL

When your system resides on a SCSI disk it can only be initialized via LD-IPL.

With this IPL architecture, the only method that can be used for restarting the IPL process is the load parameter. It is described under "Method 1: Interrupt IPL via the Load Parameter Facility" on page 24.

If you want to change IPL parameters, you have to request prompting via the IPL prompting code in the IPL load parameter. This is the only method to be used with an Integrated Console and described under "Method 1: Interrupt IPL via the Load Parameter Facility" on page 24.

The IPL Load Parameter

The IPL load parameter allows the specification of:

- Console type
- IPL message suppression code
- IPL prompting code
- Startup mode prompting code

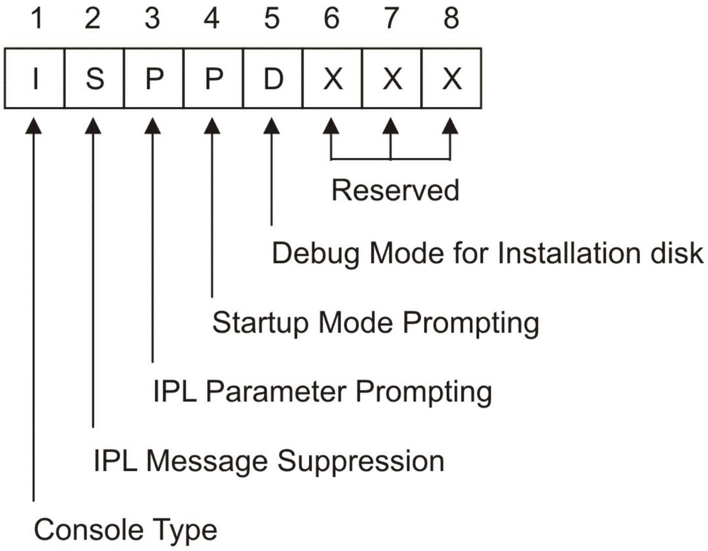


Figure 16. IPL Load Parameter Format

1. The console type specifies whether the messages are to be routed to the Integrated Console (I) or to a local console, which is the default
2. The IPL message suppression code can be used to request suppression of messages and command logging (S) during IPL.
3. The IPL prompting code can be used to request a prompting (P) for IPL parameters. IPL processing is then interrupted allowing to change or add IPL parameters. This support is identical to the IPLSTOP function of former releases and further discussed below.
4. The startup mode prompting code can be used to request prompting (P) for a startup mode such as BASIC, MINI, or COLD.
5. The debug mode for installation disk can be used to run an initial installation from installation disk with debug enabled.

For a detailed description of the operands of the IPL load parameter refer to z/VSE System Control Statements.

Interrupt and Restart the IPL Process

Method 1: Interrupt IPL via the Load Parameter Facility

This method can be used for all three supported IPL console types:

Integrated Console
CRT Console
Line Mode Console

It is the only method, however, that can be used with the Integrated Console or if the system has been installed on SCSI.

On the program load panel of the system console you can specify that you want IPL to stop to be able to supply or modify IPL parameters. The normal search sequence for ASI procedures is then interrupted. Proceed as follows:

- In the IPL load parameter field of the program load panel enter "P" for the IPL prompting code. Refer also to “The IPL Load Parameter” on page 23.
- If z/VSE runs under VM/ESA, enter, for example:

```
I cuu LOADPARM ..P
```

- Wait for message

```
0I03D ENTER SUPERVISOR PARAMETERS OR ASI PARAMETERS
```

- Enter or modify IPL parameters as required and press ENTER. Refer to “Modifying IPL Parameters” on page 26 for details.

Note: You have to purge explicitly the load parameter field on the hardware load panel when you re-IPL the system. Otherwise, IPL stops again waiting with message 0I03D for parameters to be entered.

Method 2: Interrupt IPL via External Interrupt

Note: This method can be used with a CRT or Line Mode console but not with the Integrated Console.

It cannot be used if the system has been installed on SCSI.

You can interrupt IPL and restart it in the following way:

- Interrupt IPL processing by creating an external interrupt before message

  ```
  0J10I IPL RESTART POINT BYPASSED
  ```

  appears on the screen.
- When the wait indicator is on, press the REQUEST or ENTER key at the system console.

  You may switch to a system console device different from the one specified in the IPL procedure by pressing the REQUEST or ENTER key at the device to be used as system console.
- Wait for message

  ```
  0I03D ENTER SUPERVISOR PARAMETERS OR ASI PARAMETERS
  ```
- Enter or modify IPL parameters as required and press ENTER. Refer to “Modifying IPL Parameters” on page 26 for details.

Method 3: Interrupt IPL via ENTER Key

Note: This method can be used with a CRT or Line Mode console but not with the Integrated Console. Use this method if Method 1 or 2 are not successful.

Method 3 cannot be used if the system has been installed on SCSI.

Proceed as follows:

- Press ENTER at the system console before message

  ```
  0J10I IPL RESTART POINT BYPASSED
  ```

  appears on the screen.
- Wait for message

  ```
  0J05D ASI STOP....
  ```
- Restart IPL by typing **0 IPL** (where 0 is the reply ID of the BG partition) and press ENTER.
- Press ENTER at the device you want to use as system console.
- Wait for message

  ```
  0I03D ENTER SUPERVISOR PARAMETERS OR ASI PARAMETERS
  ```
- Enter or modify IPL parameters as required and press ENTER. Refer to “Modifying IPL Parameters” on page 26 for details.

Note: The reply IPL is valid only as a response to message 0J05D if the following applies:

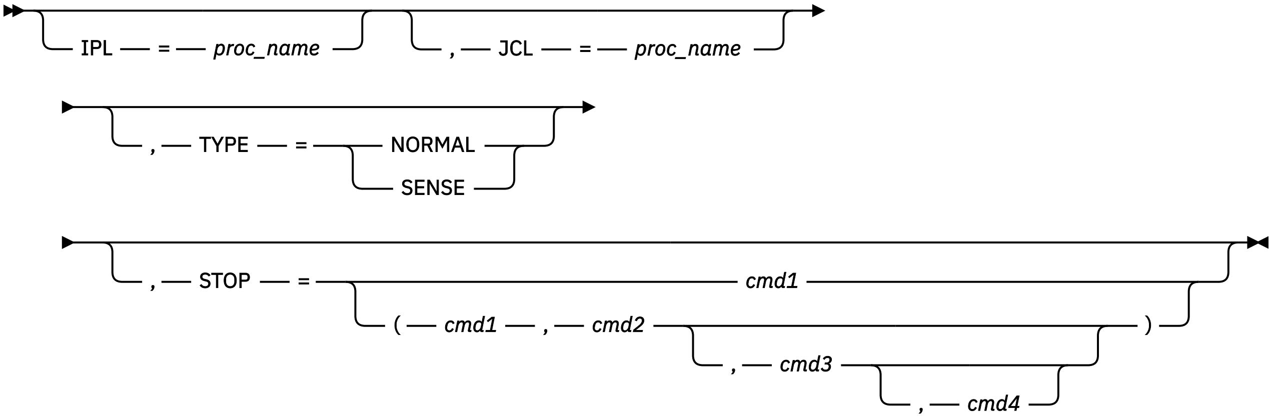
1. IPL is performed through an ASI IPL procedure.
2. The IPL restart point has not been passed yet.

In all other cases, the command will be rejected with an error message.

Modifying IPL Parameters

When IPL has been interrupted by one of the three methods described before, then, as a response to message **0I03D**, the IPL parameters shown in the syntax diagram can be entered. If a $ASIPROC exists, the defaults are taken from $ASIPROC. Otherwise, system defaults are taken.

Format



Parameters

IPL=*proc_name*
This is the name of the IPL procedure to be used to IPL the system. If omitted, ASI searches for the procedure name in the sequence shown under "Starting Up the System" on page 18.

JCL=*proc_name*
This is the name of the set of JCL procedures to be used for partition startup. If omitted, ASI searches for the procedure names in the sequence shown under "Starting Up the System" on page 18.

TYPE=NORMAL|SENSE
Specifies whether to verify the ADD commands in the chosen IPL procedure or whether device sensing is used to add devices automatically during IPL. Refer to "The ASI Master Procedure ($ASIPROC)" on page 19 for further details about the TYPE parameter.

STOP=*command(s)*
The IPL procedure stops at the specified command, and any IPL command valid at this point of processing can be entered and modified at the system console. If you press the ENTER key without any input, the interrupted IPL procedure resumes processing.

One or up to four IPL commands may be specified. For example:

```
STOP=SUP
STOP=(SUP,DEF,SVA)
```

The IPL commands that can be selected are SUP, ADD, DEL, SET, DEF, DLF, DPD, SVA.

Note: Do not specify a DPD command if your system is running without a page data set (NOPDS).

With the **SUP** specification you select the IPL supervisor parameters command for changing one or more of its parameters without creating and cataloging a new IPL procedure. The physical address of the console device (cuu) cannot be respecified, since the console device is already known.

Other Input:

If you enter a supervisor name or the supervisor parameters command (SUP) after message 0I03D, the system enters the conversational IPL mode for further input. Proceed as follows: enter "name" or "SUP" and press ENTER, then press ENTER again.

STOP Processing

IPL processing stops after reading the specified command (Method 1) or after reading the next command after the ENTER key was pressed or after an external interrupt was created (Method 2 and 3). The command is read but not processed.

The system displays the command stopped at, issues message `0J05D`, and waits for an IPL command to be entered at the system console. The operator may then enter one or more IPL commands valid at this point of IPL processing. To change a command, it is necessary to enter the complete command.

Entering `0` and pressing the ENTER key without any input causes IPL to continue with the IPL command displayed but not processed yet.

It depends on the IPL command, whether a STOP can be used to change the command at which IPL stops, or whether you can change previously processed commands only. Table 1 on page 27 shows for the various IPL commands where a STOP can be set to change information, or replace a command or add a command. You may also consult Figure 14 on page 17 for easier evaluation of the commands.

Table 1. STOP Points of IPL Commands

Command	Where to STOP	Comment
ADD DEF DEL	STOP=next IPL command in procedure behind command or group of commands if of same type	Case A: The command(s) entered after message 0J05D may override parameters previously specified in one of the commands of this command group. A new command may also be specified. Pressing ENTER without any input causes ASI to resume processing with the command stopped at.

DLF
DPD
	STOP=DLF
	STOP=DPD
		Case B:
		The complete DLF or DPD command may be entered after message 0J05D. In case of DPD be sure to enter all page data set extents. Next, the DLF or DPD command read from the procedure is interpreted, but rejected with message 0I36D since the newly entered command is now valid. When pressing ENTER, ASI proceeds with reading the next command from the procedure.

SET	STOP=next IPL command in procedure	Case C: The parameters of the SET command entered after message 0J05D override the parameters of the original SET command. The parameters not respecified keep their original value.
SET XPCC	STOP=next IPL command in procedure behind command or group of commands if of same type	Case A applies (see above)
SVA	STOP=SVA	Case D: The command entered after message 0J05D is processed, and the original command from the procedure is ignored.
SYS	STOP=next IPL command in procedure behind command or group of commands if of same type	Case A applies (see above)

Table 1. STOP Points of IPL Commands (continued)

Command	Where to STOP	Comment
Supervisor parameters command	STOP=SUP	Case D applies (see above) **Note:** The supervisor parameters command is the only valid command at this point of IPL processing that can be changed. In addition, you can use this stop to restart IPL processing by typing IPL as a response to message 0J05D.

Example for Changing the PASIZE

The PASIZE is a parameter of the IPL SYS command. As STOP point the SVA command is chosen since it follows in the command sequence after the SYS command (it is in fact the last command of an IPL procedure). To change the PASIZE, proceed as follows:

1. Interrupt IPL processing as described under "Interrupt and Restart the IPL Process" on page 24.
2. As a response to message 0I03D enter STOP=SVA and press ENTER.
3. ASI continues IPL processing up to the SVA command which it reads but does not process.

   As a response to message 0J05D enter the SYS command with the new PASIZE value (which overrides the original value).
4. Press the ENTER key to continue IPL processing with the SVA command.

Example for Changing the VSIZE

The VSIZE is a parameter of the supervisor parameters command, the first command in any IPL procedure. To change the VSIZE, proceed as follows:

1. Interrupt IPL processing as described under "Interrupt and Restart the IPL Process" on page 24.
2. As a response to message 0I03D enter STOP=SUP and press ENTER.
3. ASI stops before the supervisor parameters command is processed.

   As a response to message 0J05D enter the complete supervisor parameters command with the new VSIZE value.
4. Press the ENTER key to continue IPL processing.

Loading Phases into the SVA

Programs residing in the shared virtual area (SVA) can be used concurrently by programs running in different partitions. Such programs must be relocatable and re-enterable. Certain system phases must reside in the SVA but user programs, if required, can also be loaded into the SVA. Consider to place only those user phases into the SVA which are effectively used in a reentrant way by several tasks and partitions. "Coding for the Shared Virtual Area" on page 191 provides additional information important when writing programs that are to reside in the SVA.

A phase that is to be loaded into the SVA must be SVA eligible, that is, it must first be cataloged with the SVA parameter specified in the in the linkage editor PHASE statement. Refer to "Link-Editing for Execution in Any Partition" on page 153 for further details.

During IPL, z/VSE loads system phases from IJSYSRS.SYSLIB into the SVA (according to internal load lists).

In addition, during system startup z/VSE loads SVA-eligible phases through the **SET SDL** command into the SVA. This command maintains the SDL (system directory list) which includes the names of the phases that are to be loaded into the SVA.

The operator can use the **SET SDL** command (from the BG partition) at any time after system startup to load phases into the SVA.

SVA (24-Bit) and SVA (31-Bit)

As shown in Figure 7 on page 6, z/VSE includes an SVA (24-Bit) and an SVA (31-Bit) area. Programs that are loaded into the SVA are stored in the virtual library area (VLA) of the respective SVA. Even if there are two SVAs in a system there is always one SDL only. It resides in the SVA (24-Bit) and also addresses phases in the "high" VLA of the SVA (31-Bit).

SET SDL tries to load phases with the attribute **RMODE=ANY** into the "high" VLA first. If the space there is not sufficient, z/VSE stores it in the VLA of the SVA (24-Bit). z/VSE Extended Addressability provides details about the RMODE (residency mode) attribute and other items related to 31-bit addressing.

Automatic SVA Loading During System Startup

During system startup, z/VSE loads those system phases into the SVA which are required there. Phases that reside in IJSYSRS.SYSLIB are loaded during IPL, phases from PRD1.BASE are loaded later by the BG startup procedure $0JCL. If you want to add SVA phases to the system and want to have them loaded during startup, it is advisable to create a private load book and modify $0JCL as described in the following paragraphs.

It may, however, be necessary to have private SVA phases loaded before JCL becomes active (JCL exit routines, for example). At IPL time, no private load books can be specified since IPL SVA load books have predefined names. However, a name is reserved for private use: $SVA0000. Load book $SVA0000 is initially shipped as an empty phase. If you want to include the names of your own phases, you have to assemble and catalog this load book into IJSYSRS.SYSLIB as shown in Figure 19 on page 30. The name of the load book must be $SVA0000.

The following paragraphs describe how the system loads SVA phases from PRD1.BASE, and what has to be done to have private SVA phases loaded from private libraries automatically during startup.

Notes on the Startup Procedure $0JCL for the BG Partition

As shipped, startup procedure $0JCL includes the following statements for loading SVA phases for the components VTAM, CICS, REXX/VSE and the High Level Assembler:

```
// EXEC PROC=LIBSDL
SET SDL
LIST=$SVAVTAM
LIST=$SVACICS
LIST=$SVAREXX
LIST=$SVAASMA
/*
```

Procedure LIBSDL establishes the LIBDEF chain including library PRD1.BASE in which VTAM, CICS, REXX/VSE, and the High Level Assembler reside. The load lists $SVAVTAM, $SVACICS, $SVAREXX, and $SVAASMA identify the phases to be loaded from PRD1.BASE (and not IJSYSRS.SYSLIB) into the SVA.

To load SVA phases of z/VSE optional programs, other IBM licensed programs, or of your own programs modify procedure $0JCL through skeleton SKJCL0. The skeleton is described in z/VSE Administration.

If you install a program that includes SVA eligible phases, you can load them phase by phase or catalog a load list that identifies these phases. In each case, you must ensure that the sublibrary in which these programs and their SVA phases reside is included in the LIBDEF chain for the BG partition.

Loading Single Phases or Using a Load List

You can use the SET SDL command to load single phases or to specify a load list as shown by the job stream examples below. If a SET SDL command is processed, z/VSE searches for requested phases always in the active library chain (// LIBDEF PHASE,SEARCH=...) and in the system sublibrary IJSYSRS.SYSLIB. In case of a load list, z/VSE starts the search sequence with IJSYSRS.SYSLIB.

For automatic SVA loading, you can include such statements as shown in the examples below in procedure $0JCL (except for the statements // JOB and /&).

Job Stream Example for Loading Single Phases

```
// JOB LOAD INTO SVA
// DLBL library,'libraryname'
// EXTENT ,volid
// LIBDEF PHASE,SEARCH=library.sublibrary
SET SDL
PROGAA1,SVA
PROGAA2,SVA
PROGAA3,SVA
PROGAAB1,SVA
PROGAAB2,SVA
/*
/&
```

Figure 17. Loading Single SVA Phases

Note: Without the operand SVA, z/VSE only creates an entry in the SDL and doesn't load the phase into the SVA.

Job Stream Example for Loading Phases through a Load List

```
// JOB LOAD INTO SVA
// DLBL library,'libraryname'
// EXTENT ,volid
// LIBDEF PHASE,SEARCH=library.sublibrary
SET SDL
LIST=$SVAPROG
/*
/&
```

Figure 18. Loading SVA Phases Through a Load List

It is assumed, that load list $SVAPROG includes the entries of the programs listed in Figure 17 on page 30.

Creating an SVA Load List

To create a load list of your own, proceed as follows:

1. Submit a job as the one shown below. This gives you a listing of the names of the phases that the system loads into the SVA from a specific library. Such listings help you ensure that you do not specify the names of phases that are already included in existing load lists.

   ```
   // JOB PRINT LOAD LIST CONTENTS
   // EXEC LIBR
   ACCESS SUBLIB=library.sublibrary
   LIST $SVA*.PHASE
   /*
   /&
   ```

2. Assemble and catalog your own load list by using macro **SVALLIST**. To do this, use a job similar to the one shown in Figure 19 on page 30.

   ```
   // JOB BUILD LOAD LIST
   LIBDEF PHASE,CATALOG=library.sublibrary,SEARCH=library.sublibrary
   // OPTION CATAL
   // EXEC ASMA90....
           SPACE
   SIPL    TITLE '$SVAPROG PRIVATE SVA LOAD LIST'
           SPACE
           SVALLIST $SVAPROG,(phase01),(phase02),(phase03),               C
                    (phase04),(phase05),(phase06),(phase07),              C
                    (phase08),    ...   ,(phasenn)
           END
   /*
   // EXEC LNKEDT
   /&
   ```

   Figure 19. Creating an SVA Load List

For macro SVALLIST you have first to provide the name chosen for the load list ($SVAPROG) and then the names of the phases as shown. In theory, the number of phases that you can specify is unlimited. However, the more phases your load list includes the more virtual storage is required by the SVA.

Note: The statement

```
// EXEC ASMA90....
```

calls the High Level Assembler. Refer to "High Level Assembler Considerations" on page 139 for further details.

Notes on the SVA Command

There is a need to adjust the SVA storage requirements when loading additional phases. Use the SVA command with its operands SDL, PSIZE, and GETVIS to increase the SVA size beyond the size set by the system during IPL. The operands add space for:

- System directory list entries (SDL).
- Phases that are to be loaded into the SVA after system startup (PSIZE).
- The system GETVIS area (GETVIS).

As shipped, z/VSE reserves SVA space for its basic system phases and functions only.

For phases you want to have additionally loaded into SVA, increase the values specified in the SVA command for SDL and PSIZE. An increase of GETVIS may also be needed to reflect the system GETVIS requirements of your own programs.

For syntax and parameter details of the SVA command, refer to z/VSE System Control Statements.

Notes on Using the SET SDL Command

The SET SDL command is available for building SDL entries and loading phases into the SVA. Processing of the SET SDL command involves, for each specified phase, a search through one or more directories of the sublibraries that you have chained via the // LIBDEF job control statement to the BG partition.

If a search chain is not defined, only the system library IJSYSRS.SYSLIB is searched.

Other important points to note:

- A user who wants to build an SDL entry and load a phase into the SVA from an access-control protected sublibrary must have at least read access to the phase.
- With z/VSE you can also PFIX phases in the SVA. Refer to the description of the PHASE statement in*z/VSE System Control Statements*.
- Note that a fresh copy of the phase is loaded each time a SET SDL command for that phase is issued; multiple specifications may thus lead to an "SVA full" condition.
- It is recommended that you run the librarian program with the LISTDIR command after a SET SDL job stream to be certain that the entries are included as intended.

Replacing Phases Stored in the SVA

The following discussion applies to a system for which no re-IPL is performed.

Occasionally, a phase stored in the SVA needs to be changed; that is, it must be replaced by an updated version. To replace a phase in the SVA, you can link edit the updated version of the phase to the system sublibrary (IJSYSRS.SYSLIB), if possible. Link editing to a sublibrary other than the system sublibrary does not cause an immediate update in the SVA (the same applies to a deletion or a renaming of a phase).

The old version of the phase remains in the SVA, but its address is no longer available in the SDL (system directory list).

The change or resetting of a search chain that was used for the processing of a SET SDL command has no effect on the SVA. Therefore, phases loaded from a chained sublibrary stay in the SVA even after this chain is dropped.

User-Defined Processing after IPL

At large VSE installations, it may be desirable to perform certain processing at the end of an IPL procedure. It may, for instance, be important to know who performed the procedure, whether the right system pack was mounted, and whether the correct date was entered for the new work session. If you work with labeled data files it is important that they bear the correct creation date, so as to guarantee that data files are protected until their expiration date.

After the IPL procedure has been completed, control can be passed to a user exit routine (phase name = $SYSOPEN) that you may include for the purpose of checking system security and integrity. This routine is called once after every IPL procedure. The z/VSE distribution tape contains a dummy phase $SYSOPEN in the system sublibrary. If you do not use the facility, this phase has no effect on your system. Conventions for writing this kind of exit routine, together with an example, are discussed under "Writing an IPL Exit Routine" on page 167.

Chapter 3. Controlling Jobs

Introduction

The unit of work that is submitted to the system for execution is called a job. A job, and the environment in which it is to run, must be defined to the system through job control statements and commands. These job control statements and commands are processed by the job control program which is automatically loaded into storage as required.

The job control program provides automatic job-to-job transition. In other words, an unlimited number of jobs can be submitted to the system in one batch, and job control processes one job after the other without requiring intervention by the operator. The job or jobs submitted are referred to as a job stream.

The normal input source for the job control program is the logical unit SYSRDR. However, it accepts commands submitted through the console (SYSLOG).

When system startup has been completed, the job control program is ready to read any job control statements or commands that you submit. Normally, these are entered from the unit assigned to SYSRDR, occasionally also from the console.

The job control program runs in virtual mode in any partition. It is active only between jobs and job steps, and is not present in the partition while a program is being executed.

After each job control statement is read, control can be given to a user exit routine, which can examine and alter the input before it is processed by the system. For a description of this facility, refer to Chapter 6, "Using VSE Facilities and Options," on page 165.

.

Whenever applicable, this section shows whether a particular function can be performed using statements, commands, or both. For a detailed description of the formats and operands of statements and commands, refer to z/VSE System Control Statements.

Relating Files to Your Program

Most programs process files that are stored on auxiliary storage devices. Before such files can be processed, certain information about them must be provided to the system. This information includes:

- The address of the I/O device on which each of the files resides.
- For files on disk storage devices: the exact location of the file on the storage medium.
- For files on disk, or on labeled magnetic tape: a description of the file label, which is used for checking and protection purposes.

The above information, specified in job control label-information statements, is stored in the system by the job control program for use by the data management routines. How this is done is described below.

I/O Assignments

Whenever a program needs access to a file on auxiliary storage that program need not specify an actual device address but only a symbolic name, which refers to a logical rather than a physical unit. Before the program is executed, that logical unit must be associated with an actual device. This is done by the job control program when it processes an ASSGN statement or command which specifies the symbolic name of the logical unit and one of the following:

- A general device class or specific device type, with or without volume serial number.
- The physical address (channel and unit number) of the I/O device.
- A list of physical addresses.

- Another logical unit.

The example illustrates of some of these combinations.

Assignments are effective only for the partition in which they are issued.

```
Processing Program
            .
            .
            .
         DEVADDR=SYS002      in DTFxx macro
            .
            .
            .
```

```
Job Control

// ASSGN SYS002,(130,131)    [1]
// ASSGN SYS003,VOL=000003,ECKD    [2]
// ASSGN SYS004,CARTRIDGE    [3]


[1] Device list - if drive 130 is unassigned SYS002 will be assigned
    to it, if it is assigned the operating system tries 131.
[2] Device Type - the operating system searches for the device type
    (ECKD in this case) that is available and has the volume-id 000003.
[3] Device class - the operating system searches for an available CARTRIDGE device.
```

Processing of File Labels

The operating system relates physical devices to logical names, which are used in programs, via the ASSGN job control statement (or command). Certain device types (magnetic tape and disks) have removable volumes. It is important to ensure that the volumes containing the files to be processed are present on the assigned devices.

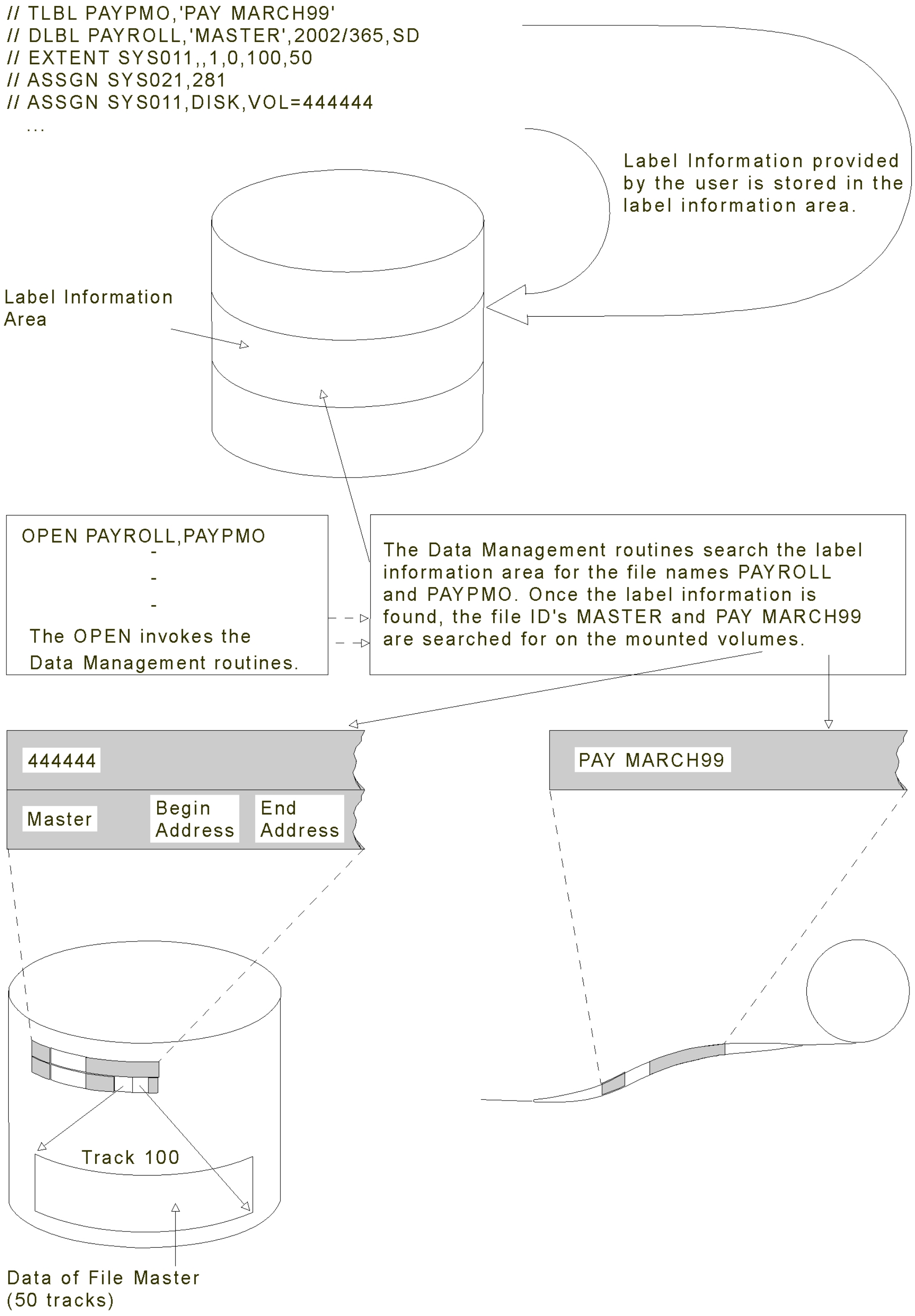
Labels are records that are stored on magnetic tape and disk volumes. It is through these labels that the system makes sure that the wanted volume of data is mounted. Labels are processed by the data management routines. Magnetic tape file labels are optional, although desirable for reasons of data integrity.

Note: z/VSE System Macros User's Guide describes tape and disk labels in detail and shows their layout.

File labels are written by the system when a file is created, based on label information that is submitted through job control statements. Libraries are treated in the same way and are considered as files.

To write a file label on magnetic tape, job control uses the information that is supplied in the TLBL statement. This label is written immediately preceding the associated file.

To write a file label on disk, job control uses the information that is supplied in the DLBL and EXTENT statements. When a labeled file is to be processed, the required // TLBL, or // DLBL and // EXTENT information must be available, so that job control can perform the wanted label checking on your existing file. Figure 20 on page 35 shows the relationship of label information that you provide by the above mentioned statements.



```
// TLBL PAYPMO,'PAY MARCH99'
// DLBL PAYROLL,'MASTER',2002/365,SD
// EXTENT SYS011,,1,0,100,50
// ASSGN SYS021,281
// ASSGN SYS011,DISK,VOL=444444
   ...
```

Label Information provided by the user is stored in the label information area.

```
OPEN PAYROLL,PAYPMO
            -
            -
            -
```

The OPEN invokes the Data Management routines.

The Data Management routines search the label information area for the file names PAYROLL and PAYPMO. Once the label information is found, the file ID's MASTER and PAY MARCH99 are searched for on the mounted volumes.

Figure 20. File Label Processing

When the program that processes the file is executed, the data management routines access the label information

- to write the appropriate labels onto the storage volume, and to check that no unexpired files are overwritten, if the file is to be created, or
- to check the contents of the label information area against the file label, if an existing file is to be processed. This ensures, for example, that the correct volume is mounted.

The TLBL, or DLBL and EXTENT job control statements can be submitted with each execution of a given program that processes labeled files. Job control temporarily stores these statements in the label information area. A recommended alternative for frequently accessed files is to store the label information permanently in the label information area.

The first two parameters of both the TLBL and DLBL statements are the same:

```
// TLBL filename,'file-id'
// DLBL filename,'file-id'
```

You code a `'filename'` in your program to identify a file. For example:

- In assembler language, it is the name that is assigned to the DTFxx macro (DTF=Define the File).
- In COBOL it is the name that is specified in the SELECT clause.
- In PL/I it is the identifier (with the FILE attribute) in the DECLARE statement.
- In RPG it is the name that is given for file name.
- In FORTRAN it is the file name that is associated with the data set reference number.

The system uses the file name from your program as a search argument in searching for label information in the label information area; therefore, you must code a matching file name in your // TLBL and // DLBL statements.

After the DLBL or TLBL statement has been located (based on `filename`), the file-id is used by the system to:

- Create a label for an output file.
- Locate and check the labels of an input file.

The Label Information Area

The label information area resides on the virtual disk with the address FDF (as defined in $0JCL.PROC, see VDISK...USAGE=DLA). As soon as the VDISK...USAGE=DLA command in $0JCL.PROC has been processed, the label information area on virtual disk is used exclusively for the sake of performance. Skeleton SKJCL0 in library 59 contains the following statements:

```
// VDISK UNIT=FDF,BLKS=2880,VOLID=VDIDLA,USAGE=DLA
*  VDISK UNIT=CUU,BLKS=81920,VOLID=VDIWRK
// EXEC PROC=STDLABEL   CALLS ALSO STDLABUP AND STDLABUS LOAD VDISK
```

Label information supplied within a job is automatically kept for the duration of that job, that is, temporarily. To make label information available for all the following jobs in a partition, enter the job control statement:

```
// OPTION PARSTD=ADD
```

followed by the TLBL, DLBL and EXTENT statements.

To make label information available for all the following jobs in any dynamic partition of class 'class', enter the following in the BG partition:

```
// OPTION CLASSTD=(class,ADD)
```

followed by the TLBL, DLBL and EXTENT statements.

To make label information available throughout the system (for all the following jobs in any partition), enter the following in the BG partition:

```
// OPTION STDLABEL=ADD
```

followed by the TLBL, DLBL and EXTENT statements.

Label information can be selectively deleted by using the commands:

```
// OPTION PARSTD=DELETE            (for partition permanent labels)
// OPTION CLASSTD=(class, DELETE)  (for class standard labels, from BG partition)
// OPTION STDLABEL=DELETE          (for system standard labels, from BG
                                    partition)
```

For further details about the OPTION statement, refer to z/VSE System Control Statements.

Standard Label Procedures

z/VSE includes standard label procedures named STDLABEL, STDLABUP, and STDLABUS, details are provided in z/VSE Planning.

Defining a Job

A program to be executed in a job is requested through an // EXEC statement. The occurrence of an // EXEC statement is called a job step. Each job can consist of one or several job steps. The beginning and end of a job are defined by the // JOB and /& (end-of-job) statements.

Figure 21 on page 37 shows an example of a multi-step job.

```
   (1) // JOB PAYCHEX
        .
   (2) additional job control statements
        .
   (3) // EXEC PAYROLL
        .
   (3) // EXEC CHECK
        .
   (4) /&
```

1. The job control program defines the beginning of a job. For jobname, you can specify a name of your own choice.
2. Additional job control statements such as DLBL, EXTENT or ASSGN if required.
3. The two job steps. The job control program is reloaded into storage at the end of each job step, enabling the reading of subsequent job control statements.
4. At the end of the CHECK program's execution, the job control program is reloaded and reads the end-of-job indicator.

Figure 21. Control Statements Defining a Job Consisting of Two Job Steps

You can include as many job steps in a job as you wish. However, you should not execute in one job several programs that are completely independent of one another. If one step terminates abnormally, the job control program ignores the remaining job steps up to the next /& or // JOB statement, unless you specify some other action in an ON statement. See the section “Using Conditional Job Control” on page 59 .

Following are some additional details about the job and end-of-job (/&) statements.

The JOB Statement

The JOB statement indicates the beginning of control information for a job. The specified job name is used, for example, by job accounting and to identify listings produced during the execution of the job. A JOB statement without a job name is rejected by job control as an invalid statement.

If the JOB statement is omitted, the system uses NO NAME as the job name. The JOB statement, however, should not be omitted, since many functions assume its presence. If, for example, the operator cancels a job using the attention routine CANCEL command, the job control program normally bypasses all statements on SYSRDR until it encounters a /&. However, if the job in question was submitted without a JOB statement, no statements in the job stream are bypassed even though job NO NAME was canceled. Also, no conditional job control and no symbolic parameters can be specified without a preceding JOB statement.

Having JOB statements with specific job names is useful when you issue the MAP command. The MAP command displays on SYSLOG the storage allocations for each partition, together with the name of a job that is currently active in the corresponding partition.

The JOB statement is always printed, together with the time of day, on SYSLST and SYSLOG. The JOB statement causes a skip to a new page before printing is started on SYSLST.

The End-of-Job (/&) Statement

This statement is the last one for each job (not job step); if it is omitted, the next JOB statement will cause control to be transferred to the end-of-job routine to simulate the /& statement.

When a /& statement is encountered, the job control program performs operations such as the following:

- Resetting all job control options for the partition to standard: either as established by the STDOPT command, or the system default if the particular option was not set through a STDOPT command.
- Resetting the PFIX limits to 0 if not requested otherwise in the SETPFIX statement.
- Resetting all temporary system and programmer logical unit assignments for the partition to the permanent assignment established by job control commands.
- Resetting conditional job control information to default values and resetting symbolic parameter definitions to undefined, if required.
- Deactivating all temporary library chains for the partition.
- Resetting the date from the DATE statement to the system date.
- Setting the user area and the UPSI byte to zero.
- Displaying an end-of-job (EOJ) message on SYSLST and SYSLOG.
- Ensuring that end-of-file has been reached on SYSIPT.
- Deleting the temporary partition's labels in the label information area. For more information on label processing, refer to "Storing Label Information" on page 45.
- Releasing temporary LIBSERV MOUNT requests.
- Enforcing end-of-procedure.
- Resetting user identification acquired by an ID statement if security is active.

Job Streams

The job control program provides automatic job-to-job transition. You can submit an unlimited number of jobs to a partition of the system in one batch, and the system processes one job after the other without requiring intervention by the operator (except replying to messages). The job or jobs submitted are referred to as a job stream. Figure 22 on page 38 shows an example of a job stream. It includes two jobs, named PAY1 and STOCK, each with two steps (each occurrence of an EXEC statement with a program name is a job step).

```
// JOB PAY1
// ASSGN SYSLST,00E
// ASSGN SYS001,160
// DLBL FILEP,'PAYFILE',....
// EXTENT SYS001,....
// EXEC PAYRUN
// PAUSE LOAD PAYCHECKS
// EXEC PAYCHK
/&
// JOB STOCK
// ASSGN SYSLST,00E
// ASSGN SYS003,162
// DLBL FILEP,'STKFILE',....
// EXTENT SYS003,....
// EXEC STKRUN
// EXEC STKLIST
/&
```

Figure 22. Example of a Job Stream

When setting up a job stream for a partition, you should bear in mind that all jobs will get the priority of that partition. Therefore, careful selection of the jobs for a particular partition or a dynamic class can help to improve the efficiency of your installation. For example, jobs which have a relatively low CPU usage and a relatively high rate of I/O activity and which, therefore, spend most of their time waiting for the completion of I/O operations, should run in a high-priority partition. Conversely, CPU-bound jobs should be in a partition with a lower priority.

The operator may want to interrupt the processing of a job stream in any partition, for example, to make last-minute changes to one of the jobs or to squeeze in a special rush job. He does this by using the PAUSE statement or command.

PAUSE Statement

The statement can be included anywhere among the job control statements of a job stream. It becomes effective at the point where it was inserted. Processing is suspended in the affected partition, waiting for operator input. The statement can contain instructions to the operator, and is always displayed on SYSLOG.

PAUSE Command

The command can be entered either at the operator console or within a job stream together with the job control statements for a job. When it encounters a PAUSE command, the system passes control of the specified partition to the operator console the next time it encounters end of job or end of job step.

Job Control for Device Assignments

When a program is to access a file the system needs the following information on the particular file:

- The address of the physical device where the file resides.
- For files on disk devices, the exact location of the file (extent information).
- For files on disk, and labeled tapes, information required for checking and protection purposes (part of the label information).

To associate a logical unit with an actual physical device, you have to use the ASSGN statement in the partition in which the program is to run.

Following is a discussion of the logical units that can be used.

Logical Units

There are two types of logical units: system logical units, primarily used by the system programs, and programmer logical units, primarily used by the user-written programs. The following list shows the names of the logical units and the I/O devices that each of these logical units can represent. In the case of disk devices, the logical unit is not assigned to the entire volume mounted on the device but only to the referenced extent(s).

Logical unit name
  Type of I/O device

SYSRDR
  Card reader, magnetic tape unit (single volume), or disk; used as input unit for job control statements or commands.

SYSIPT
  Card reader, magnetic tape unit (single volume), or disk; used as input unit for programs.

SYSPCH
  Card punch, magnetic tape unit, or disk; used as the unit for punched output.

SYSLST
  Printer, magnetic tape unit, or disk; used as the unit for printed output.

SYSLOG
Operator console; used for communication between the system and the operator.

SYSLNK
Disk; used as input to the linkage editor.

SYSRES
System library extent on a disk volume (DOSRES).

SYSREC
One disk extent; used to store error records collected by the error recovery and recording function in the system.

SYSCAT
Disk; used to hold the VSE/VSAM master catalog.

SYSCTL
For system use.

SYSnnn
(where nnn = 000 .. 254). Format for naming programmer logical units, which are discussed later in this section.

System Logical Units

All of the above logical unit names, except SYSnnn, represent system logical units (as opposed to programmer logical units). Of these system logical units, user-written programs may use SYSIPT and SYSRDR for input, SYSLST and SYSPCH for output, and SYSLOG for communication with the operator. No other system logical units may be used within user-written programs (or label information statements, which are discussed later in this section).

Two additional symbolic names, SYSIN and SYSOUT, are used under certain conditions:

SYSIN
Can be used if you want to assign SYSRDR and SYSIPT to the same card reader or magnetic tape unit. Must be used, if you want to assign SYSRDR and SYSIPT to the same disk extent.

SYSOUT
Must be used if you want to assign SYSPCH and SYSLST to the same magnetic tape unit. SYSOUT cannot be used to assign SYSPCH and SYSLST to disk, because these two units must refer to separate extents.

SYSIN and SYSOUT are valid only for the job control program, and cannot be referenced in a user-written program. Examples for the use of SYSIN and SYSOUT are given in "Handling of System Input and Output" on page 57.

Programmer Logical Units

Programmer logical units can be assigned to any I/O device installed on the system. Each partition must have at least 10 programmer logical units, and can have at most 255 (SYS000 - SYS254). The number of programmer logical units available for a given static partition can be defined with the NPGR command, see z/VSE System Control Statements for details.

Types of Device Assignments

Device assignments are either permanent or temporary, depending on the type of ASSGN statement or command used. Device assignments are set up between jobs or job steps any time after IPL by the ASSGN job control command (no //) or the // ASSGN job control statement.

Permanent Device Assignments

A permanent assignment may look as follows:

```
// ASSGN SYS009,130,PERM
```

It is valid as long as a static or dynamic partition exists, unless superseded by another ASSGN job control command. A permanent assignment can be changed for the duration of a job or job step by a // ASSGN statement, or by an ASSGN command with the TEMP operand.

Temporary Device Assignments

A temporary assignment may look as follows:

```
// ASSGN SYS008,130,TEMP
```

It is valid for a single job only, unless superseded by another temporary or permanent assignment. Temporary assignments are reset by:

- a /& or JOB statement, whichever occurs first, or by
- a RESET job control statement or command.

The permanent assignment which was valid before the temporary assignment was made becomes valid again.

Restrictions

1. If one of the system logical units SYSRDR, SYSIPT, SYSLST, or SYSPCH is assigned to a disk extent, the assignment must be permanent.
2. If SYSIN is assigned to a disk extent, this assignment must be permanent.
3. SYSOUT, if used, must be a permanent assignment.
4. Before a tape unit is assigned to SYSLST, SYSPCH, or SYSOUT, all previous assignments to this tape unit must be permanently unassigned. This can be done by using a DVCDN command.

Device Assignments

Each partition has its own set of system logical units. For example, the BG partition has a SYSRDR, SYSLST, SYSIPT etc. as do all the other generated partitions. As each partition is started, assignments must be made for the system logical units. Some assignments need be made only in one partition and are valid for all partitions. These are logical units that service the system. The following units fall into this category:

Logical name	How assigned
SYSLOG	ASSGN job control command (permanent)
SYSREC	IPL DEF command
SYSRES	Disk address entered at IPL
SYSCAT	IPL DEF command, if VSE/VSAM is used.

All of the other system logical unit assignments must be made for each individual partition.

Each partition also has its own set of programmer logical units (SYS000 through SYSnnn) where nnn is the number of programmer logical units specified for the partition minus 1.

You must make assignments of the programmer logical units as needed by the programs running in each partition. Certain IBM-supplied programs require specific programmer logical unit assignments. For example the linkage editor requires SYS001 and the assembler requires SYS001, SYS002, and SYS003.

Shared Assignments

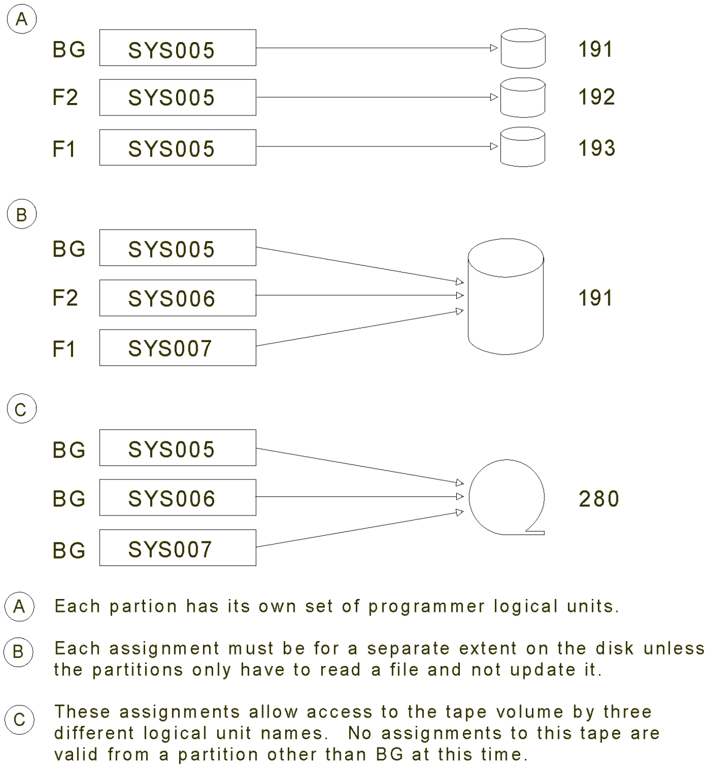
Within the same partition, different logical units can be assigned to the same physical device. For example:

```
// ASSGN SYSLST,00E
// ASSGN SYS007,00E
```

Both logical names SYSLST and SYS007 are assigned to the device at address 00E.

Normally it is not possible to share physical devices (except disk devices) between partitions. For example, if you have a tape drive assigned to the BG partition, but not used by that partition, you must first unassign it in BG before attempting to assign it in F2. However, if you use a spooling program, such as VSE/POWER, you can share unit record devices (printer or card reader, for example) between partitions.

With disk devices this problem does not exist, because each extent on a disk is treated as a separate device. Also, support is available that allows an assignment to a disk extent to be shared by two or more programs.



A Each partion has its own set of programmer logical units.

B Each assignment must be for a separate extent on the disk unless the partitions only have to read a file and not update it.

C These assignments allow access to the tape volume by three different logical unit names. No assignments to this tape are valid from a partition other than BG at this time.

Figure 23. Possible Device Assignments

Additional Assignment Considerations

This section discusses statements and commands that can be used in context with logical unit assignments.

The LISTIO Statement/Command

The LISTIO statement or command displays a listing of current I/O assignments. For details refer to z/VSE System Control Statements.

The RESET Command

The RESET command resets temporary I/O assignments to their permanent value. For details refer to z/VSE System Control Statements.

Job Control for Label Information

The following section discusses label information for files on disk devices and on magnetic tape.

Label Information for Files on Disk Devices

After you have informed the system, via the **ASSGN** job control statement or command, which volume or physical device is to be used, you must supply the following information to allow the creation (output files) or checking (input files) of labels:

1. A description of the characteristics of the file. You specify this in the DLBL job control statement.
2. For non-VSAM files only: the exact location of the file on the storage medium. You specify this in one or more **EXTENT** job control statements.

The label information you supply in the **DLBL** job control statement may include the following:

- The name of the file. This name must be identical to the corresponding file name specified in your program. For programs written in assembler language this would be the name of the DTF.
- An identification of the file, which may include generation and version numbers of the file. This name is the one contained in the file label on the storage device. It is associated with the file name via the DLBL statement.
- The expiration date or retention period of the file.
- The type of access method used to process the file.
- An indication of whether or not a data secured file is to be created.
- The block size (BLKSIZE) if your file is a sequential disk file and resides on a CKD device.
- The control interval size (CISIZE) if your file is a sequential disk file and resides on an FBA device.

A disk file can consist of one or more data areas on one or more volumes. For each of these data areas, called extents, you supply the following information on an EXTENT job control statement:

- The symbolic name of the device on which the volume containing the file extent is mounted.
- The serial number of this volume.
- The type of the extent. An indexed sequential file, for instance, can consist of data areas, index areas, and overflow areas. For each of these areas an extent must be defined, and its type (data, index, or overflow) must be specified.
- The sequence number of the extent within the file.
- For CKD devices:

  The number of the track (relative to zero) on which the file extent begins.

  The amount of space (in tracks) the file occupies.
- For FBA devices:

  The block number on which the file extent begins.

  The amount of space (in blocks) the file occupies.

Examples of Submitting Label Information for Disk Files

Here are a number of examples of how to code the job control statements required to create or access the labels for the various types and organizations of disk files. It is helpful, if you are familiar with the formats of the DLBL and EXTENT job control statements as described in z/VSE System Control Statements.

Sequentially Organized Disk Files (Single Volume, Single Extent)

In the following example, the program CREATE creates a sequential disk file named SALES. The file comprises one extent of 190 tracks on a ECKD device, starting on relative track number 1320. The cuu with the volume serial number 111111 is assigned to the symbolic device name SYS005:

```
// JOB EXAMPLE
// ASSGN SYS005,ECKD,VOL=111111,SHR
// DLBL SALES,'ANNUAL SALES RECORDS',2018/365,SD
// EXTENT SYS005,111111,1,0,1320,190
// EXEC CREATE
/&
```

The job control program checks the DLBL and EXTENT statements for correctness and stores the supplied information in the label information area for the duration of the job or job step.

Sequentially Organized Disk Files (Multiple Volume, Multiple Extents)

Assume that program PROG100 needs a sequential disk file located on three different disk volumes. The file consists of four extents on ECKD devices: two on the volume with serial number 000020, one on volume 000100 and one on volume 000006. The following job stream shows the label statements required:

```
   // JOB SAMLABEL
   // ASSGN SYS005,ECKD,VOL=000020,SHR
   // ASSGN SYS006,ECKD,VOL=000100,SHR
   // ASSGN SYS007,ECKD,VOL=000006,SHR
1  // DLBL FILNAME,'FILE ID',2018/365,SD
   // EXTENT SYS005,000020,1,0,10,2010
   // EXTENT SYS005,000020,1,1,4000,1510
   // EXTENT SYS006,000100,1,2,64,1300
   // EXTENT SYS007,000006,1,3,50,636
2  // EXEC PROG100
3  /&
```

```
Explanation:

1  Only one DLBL statement is required. For each
   extent, one EXTENT statement must be supplied
   in the sequence in which the extents are
   processed.

2  Logical IOCS in PROG100 opens the first extent
   using the file name and file ID in the DLBL
   statement, and the logical unit and volume
   serial number in the first EXTENT statement
   to locate the actual label on the disk pack.
   After PROG100 has processed the first extent,
   logical IOCS opens the second extent, based
   on the extent sequence number.

3  The /& statement causes the label information stor-
   ed in the label information area to be cleared.
   Thus, if the next job requires the same file, the
   label statements must be resubmitted (see "Storing
   Label Information" later in this chapter).
```

Direct Access Files

The program PROG101 processes a direct access file consisting of four extents contained on three disk devices. The following job stream shows the label statements required to process the file:

```
   // JOB DALABEL
   // ASSGN SYS005,DISK,VOL=000065,SHR
   // ASSGN SYS006,DISK,VOL=000025,SHR
   // ASSGN SYS007,DISK,VOL=000002,SHR
1  // DLBL FILNAME,'FILE ID',2018/365,DA
   // EXTENT SYS005,000065,1,0,10,2010
   // EXTENT SYS005,000065,1,1,4000,1510
   // EXTENT SYS006,000025,1,2,64,1300
   // EXTENT SYS007,000002,1,3,50,636
   // EXEC PROG101
   /&

Explanation:

1  The label statements follow the same pattern
   as for sequential files (described in the
   preceding examples) except that the DLBL
   statement must specify DA to indicate direct
   access.
```

As mentioned before, label information is stored by the system in the label information area which is described under “Storing Label Information” on page 45.

Label Information for Files on Magnetic Tape

Files on magnetic tape can be processed with or without labels. For tape files with IBM standard labels, the label information must be submitted through the TLBL job control statement. (A tape file can also have standard-user or non-standard labels; for these labels no job control statements are required).

The standard label information submitted in the TLBL statement may include the following:

- The name of the file. This name must be identical to the corresponding filename (DTF name) specified in your program.
- An identification of the file.
- Creation date for input and expiration date (or retention period) for output files.
- The volume serial number of the tape reel or cartridge that contains the file.
- For files that extend over more than one volume, the sequence number of the volume.
- For volumes that contain more than one file, sequence number of the file.
- The version and modification number of the file.

As with disk files, the label information you supply in the TLBL job control statement is checked and stored in the label information area. Refer also to “Storing Label Information” on page 45.

Following is an example of job control statements for label checking of a file on magnetic tape:

```
// JOB UPDATE
// ASSGN SYS008,280
*  PLEASE MOUNT CURRENT ACCOUNTS RECEIVABLE TAPE
// PAUSE
// TLBL ACCT,'ACCTS.REC.FILE'
// EXEC UPDATE
   ...
   data cards
   ...
/*
// MTC REW,SYS008
// ASSGN SYS010,280
// TLBL ARFILE,'ACCTS.REC.FILE'
// EXEC ARREPORT
/&
```

Figure 24. Sample Job for Label Checking of a File on Magnetic Tape

The programs UPDATE and ARREPORT access the same file 'ACCTS.REC.FILE', yet they use different file names and different programmer logical units.

UPDATE opens a file named ACCT on logical unit SYS008 and ARREPORT opens a file named ARFILE on SYS010. In both cases, the file accessed is 'ACCTS.REC.FILE'. If the two programs had used the same file name and programmer logical units, one ASSGN statement and one // TLBL statement would have been sufficient.

Storing Label Information

Job control stores label information in the label information area either temporarily (for the duration of one job or job step) or permanently. There are different label groups for storing the label information:

- a partition temporary label group for each partition
- a partition permanent label group for each partition
- a class standard label group for each dynamic class
- a system standard label group

Label information stored in a partition label group can be accessed only from the associated partition. Label information stored in the system label group can be accessed from all partitions; stored in the class label group it can be accessed from each dynamic partition of the associated class.

The USRLABEL, PARSTD, CLASSTD=class, and STDLABEL operands of the OPTION job control statement determine in which label group the label information provided in DLBL or TLBL statements is to be stored.

USRLABEL

Causes all label information to be stored for the scope of one job or job step. Label information submitted between job steps overlays the label information submitted for a preceding job step. Therefore each job step (that is each EXEC statement) should be preceded by the label information statements it requires. For example, in Figure 24 on page 45 both job steps UPDATE and ARREPORT are preceded by a TLBL statement. Note that the label information for file ACCT is not available for the second job step (ARREPORT).

The label information is written into the temporary label group of the associated partition.

If no other label related option is specified, or if the OPTION statement is omitted, USRLABEL is assumed.

PARSTD

Causes label information to be stored permanently for all subsequent jobs in the same partition. The label information is written to the partition's permanent label group.

Partition permanent labels can be submitted in the partition to which they belong. For foreground partitions they can also be submitted through a job running in the background partition. The job stream must contain one of the following statements:

// OPTION PARSTD=Fn
// OPTION PARSTD=(Fn,ADD)

All label information following this statement is put into the partition permanent label group of partition Fn (n is the number of the foreground partition). The above statement can be given only when partition Fn is inactive.

CLASSTD=class

For **dynamic** partitions only.

Causes label information to be stored permanently for all subsequent jobs in any dynamic partition of class 'class'. The label information is written to the class standard label group. Class standard labels can be submitted from the BG partition only. The job running in BG must contain one of the following statements:

```
// OPTION CLASSTD=class
// OPTION CLASSTD=(class,ADD)
```

All label information following this statement is written into the class standard label group of class 'class'. The above statement can be given only if the stated 'class' is disabled and no partition of this 'class' is active.

STDLABEL

Causes label information to be stored permanently for all subsequent jobs in any partition. The label information is written to the system standard label group and can be submitted only in the background partition.

For details of the OPTION statement, refer to z/VSE System Control Statements.

Terminate your // OPTION PARSTD, // OPTION STDLABEL, or // OPTION CLASSTD=class job stream with a // OPTION USRLABEL statement. This ensures that any subsequent label information is written only to the partition temporary label group, and does not fill up the standard label label groups (system, partition, class). The OPTION statement with USRLABEL specified also indicates to the system that no further partition or system standard labels will follow; a /&, // JOB, or // EXEC statement has the same effect.

Adding and Deleting Label Information

When PARSTD, CLASSTD or STDLABEL is given without the specification ADD or DELETE, any label information currently in the respective label group is completely overwritten by the newly supplied data. If you want to keep the old label information and only add more labels to it, code one of the following:

```
  // OPTION PARSTD=ADD
  // OPTION STDLABEL=ADD
  // OPTION PARSTD=(Fn,ADD)
  // OPTION CLASSTD=(class,ADD)
```

Specifying

```
  // OPTION PARSTD=DELETE
  // OPTION STDLABEL=DELETE
  // OPTION PARSTD=(Fn,DELETE)
  // OPTION CLASSTD=(class,DELETE)
```

causes labels to be deleted from the respective label group. Such a statement must be followed by one or more statements of the form

```
  filename
```

where filename (of the DLBL statement) indicates which label is to be deleted. The last filename statement must be followed by a /*.

Label Information Search Order

During program execution, the data management routines search the label information area in the sequence outlined below.

- For a static partition the search sequence is as follows:
  1. Partition temporary label group (USRLABEL).
  2. Partition permanent label group (PARSTD).
  3. System standard label group (STDLABEL).

  This sequence also applies for a dynamic partition if the PARSTD option is used.
- If a program runs in a dynamic partition and the PARSTD option is not used, the search sequence is as follows:
  1. Partition temporary label group (USRLABEL).
  2. Class standard label group (CLASSTD).
  3. System standard label group (STDLABEL).

The summary of label options given in Table 2 on page 48 indicates the conditions under which a label option remains in effect and the conditions that govern the retention of the label data in the label information area.

Table 2. Summary of Label Option Functions

Option in search sequence	Type of label information	Option in effect until	Label information retained	For
USRLABEL	temporary	STDLABEL, PARSTD, or CLASSTD is specified	for one job or job step. The /& statement causes the temporary label area to be cleared. Additional label information from a subsequent job step will overlay previous label information.	the partition in which the option was specified.
PARSTD	permanent	• end-of-job step • end of job • USRLABEL, STDLABEL, or CLASSTD is specified	for all subsequent jobs until deleted. (1)	the partition in which the option was specified, or as specified in PARSTD=Fn.
CLASSTD=class	permanent	• end-of-job step • end-of-job • USRLABEL, PARSTD, or STDLABEL is specified	for all subsequent jobs until deleted. (1)	dynamic partitions of a class
STDLABEL	permanent	• end-of-job step • end of job • USRLABEL, PARSTD, or CLASSTD is specified	for all subsequent jobs until deleted. (1)	all partitions

(1) Either explicitly deleted (=DELETE) or by giving the option without an operand.

Stored label information can be displayed using program LSERV as follows:

```
// JOB
// EXEC LSERV
/*
/&
```

If you specify no parameter, all label groups are displayed or printed. For a description of the parameters available, refer to z/VSE System Utilities.

Controlling Magnetic Tape

Note: The term magnetic tape in this context means both, tapes and cartridges.

The MTC job control statement or command controls operations such as:

- Spacing the tape backward or forward to the required file.
- Spacing the tape backward or forward a specified number of physical records.
- Rewinding the tape to the beginning.

- Writing a tapemark to indicate the end of a file.

In the following example, program PROGA creates a labeled tape file named RATES on tape volume 222222. At the end of the first job step, an MTC job control statement requests the system to rewind (REW) the tape to the beginning of the volume so that the newly created file can be processed by PROGB. After PROGB, the tape is rewound and unloaded (RUN).

```
// JOB TAPE
// ASSGN SYS004,CARTRIDGE,VOL=222222
// TLBL RATES,'MASTER',2018/365,222222
// EXEC PROGA
// MTC REW,SYS004
// EXEC PROGB
// MTC RUN,SYS004
/&
```

For details of the operations which can be carried out by the MTC statement or command, refer to z/VSE System Control Statements.

Controlling Printed Output

Most of the printers use a forms control buffer (FCB) to control the skipping of forms. In addition, printers may have a universal character set buffer (UCB). Examples of printers equipped with both of these buffers are the 3203 and 3211 printers.

The buffers of these printers must be loaded during or immediately after IPL, and they may have to be reloaded later between job steps or, occasionally, while a job step using the printer is being executed.

Following is a description of the methods available for loading the buffers.

Loading the FCB

- Automatic loading during IPL
- Using the SYSBUFLD program between job steps or immediately after IPL
- Using the LFCB command
- Using the LFCB macro in the problem program
- Using the FCB parameter in the VSE/POWER * $$ LST statement.

Loading the UCB

- Automatic loading during IPL (applies to PRT1 printers)
- Using the SYSBUFLD program between job steps or immediately after IPL
- Using the LUCB command
- Using the UCS command (applies only to a 1403U printer).

Using the SYSBUFLD program offers the advantage that hardly any operator activity is involved; on the other hand, loading the buffers by using the LFCB or LUCB command does not require the operator to wait for a partition to finish processing.

When the contents of an FCB or a UCB are replaced by a new buffer image, the system uses this new image to control printed output until the buffer is reloaded or until the next IPL.

None of the above methods provides automatic resetting of the buffer load to the original contents. However, if you use the FCB parameter in the VSE/POWER statement * $$ LST, the FCB is automatically reset during processing of the * $$ EOJ statement. For more information on loading the FCB under VSE/POWER, see the VSE/POWER documentation.

It may be necessary to reset the buffer to the original contents before taking a storage dump, to ensure that the dump is printed in the correct format, without any part of it being left out.

Details on how to load the FCB and UCB are contained in z/VSE System Control Statements.

Controlling Printed Output on an IBM 3800 Printing Subsystem

The IBM 3800 Printing Subsystem uses an electrophotographic technique with a low-powered laser to print output. It provides more features than current impact printers.

The following methods of controlling the IBM 3800 are available:

- The SETDF attention routine command, which allows the operator to set and/or reset default control values for the IBM 3800. A SETDF command can set default control values for:
  - One character arrangement table
  - The forms control buffer
  - Copy modification
  - Paper forms identification
  - The forms overlay name
  - Bursting and trimming or continuous forms stacking.
- The SETPRT job control statement or command, which allows you to override the current default control values for the duration of one job. At the end of the job, these values are reset to those specified in the last SETDF command, or to the hardware defaults if SETDF was not specified.

Refer to *z/VSE System Control Statements* for a detailed description of SETDF and SETPRT.

Processing a Program

Three job steps are necessary to obtain output from an application program once the source program has been written:

1. Assembling or compiling of the source program into an object module.

   Object modules are discussed in Chapter 5, "Linking Programs," on page 145.
2. Link-editing of the object module to form an executable program, also called a phase.
3. Running the program.

Each of these steps is initiated by the job control program in response to an EXEC job control statement, which must be the last job control statement submitted for a job step. Figure 25 on page 50 gives an example of the job control statements needed to assemble, link edit, and process a source program. The example assumes that ASSGN statements for the assembler work files have been given in the partition.

```
// JOB INPUT
// OPTION LINK
// EXEC ASMA90.....
   .
   .
source program
   .
   .
/*
// EXEC LNKEDT
// EXEC
   .
   .
input data for user program
   .
   .
/*
/&
```

Figure 25. Job Control Statements to Assemble, Link-Edit, and Execute a Program in one Job Stream

The statement `// EXEC ASMA90....` calls the High Level Assembler. Refer to "High Level Assembler Considerations" on page 139 for further details.

IBM language translators (and many other IBM programs) read their input from SYSIPT. If SYSRDR and SYSIPT are assigned to the same device, the source statements of your program must follow the corresponding EXEC job control statement. In this example, the assembler language statements would have to follow the // EXEC ASMA90.... statement. The end of the input data submitted for one program must be indicated by a /* (end-of-data) statement.

Note: How the job shown in Figure 25 on page 50 is processed by the system is explained in Table 3 on page 51.

Instead of submitting three EXEC statements, you can invoke all three steps by one EXEC statement. Specifying the GO parameter in the EXEC statement which invokes the assembler (compiler) causes the linkage editor and your executable program to be run automatically once the assembly (compilation) is finished. The source program and any additional data required by your program must be submitted via SYSIPT.

This method is useful during the testing of a new program.

Table 3. Sample of a Job to Assemble, Link-Edit and Execute a Program

Control Statement or Data	Program in Control	Function
// JOB INPUT	Job Control	Stores the specified job name, INPUT.
// OPTION LINK	Job Control	Sets the LINK bit to indicate: • that link-editing is allowed in this job. • that the High Level Assembler makes the assembled object module available as input on SYSLNK for the linkage editor. • that the linkage editor is to store the executable program temporarily for execution in the same job.
// EXEC ASMA90....	Job Control	Passes job information such as the program name to the super-visor and transfers control to it.
	Supervisor	Loads the High Level Assembler which replaces job control in the partition.
... source program ...	High Level Assembler	Reads the source program (from SYSIPT, which is assigned to the same device as SYSDR -- see Note, below), assembles it, and stores the object module on disk (SYSLNK) -- as input for the linkage editor.
/*	Supervisor	High Level Assembler signals end-of-file to supervisor when a /* occurs.
	Supervisor	Loads job control, which replaces the High Level Assembler in the partition.

Table 3. Sample of a Job to Assemble, Link-Edit and Execute a Program (continued)

Control Statement or Data	Program in Control	Function
// EXEC LNKEDT	Job Control	Passes LNKEDT, the program name of the linkage editor, to the supervisor and transfers control to the latter.
	Supervisor	Loads the linkage editor, which replaces job control.
	Linkage Editor	• Reads and processes the object module generated and stored on SYSLNK by the High Level Assembler. • Stores the executable program phase in the virtual I/O area. • Transfers control to the supervisor.
	Supervisor	Loads job control, which replaces the linkage editor.
// EXEC	Job Control	• Passes NONAME to the supervisor because no name is specified in the // EXEC statement. • Transfers control to the supervisor.
	Supervisor	Loads the program last stored temporarily (in virtual I/O area) by the linkage editor; this program replaces job control.
... input data for user-program ...	User Program	Reads and processes the input data from SYSIPT.
/*	Supervisor	User program signals to supervisor end-of-file when a /* occurs.
	Supervisor	Loads job control, which replaces the unnamed user program.
/&	Job Control	Turns off the LINK bit. Reads the next statement from the device assigned to SYSRDR.
NOTE: The High Level Assembler, like any other IBM language translator, reads its control input from the device assigned to SYSIPT.		

Executing Cataloged Programs

Assembly or compilation, linking and execution in one job or in one GO step is possible only with single-phase programs.

Multiphase programs must be cataloged permanently in a sublibrary after they have been assembled and link edited. This method also saves assembling and link editing a program for every run. It can, of course, be used for single-phase programs, too.

The linkage editor catalogs programs as PHASE-type members in a sublibrary. This is discussed in detail in Chapter 5, "Linking Programs," on page 145.

To execute a cataloged program you use a LIBDEF PHASE,SEARCH... statement to specify in which sublibrary the program was cataloged, and an EXEC job control statement specifying the name under which the program was cataloged. For example, the job below executes a program that was cataloged in a sublibrary under the name PROGA; data cards are submitted on SYSIPT:

```
// JOB USERPROG
   ...
   (ASSGN, LIBDEF, and label
   statements as required,
   if not permanently valid
   in the partition)
   ...
// EXEC PROGA
   ...
   input data
   ...
/*
/&
```

Defining Options for Program Execution

You can find a complete list of the available options in z/VSE System Control Statements.

In the preceding section, it was shown how the OPTION job control statement can be used to specify the type of label information to be stored for a file (options USRLABEL, PARSTD, CLASSTD, and STDLABEL) and to define whether a program is to be link edited (LINK option).

There are a number of additional system functions which you can invoke through the OPTION job control statement. The most important ones are:

// OPTION CATAL Causes the linkage editor to catalog a phase into the sublibrary specified in the current LIBDEF PHASE,CATALOG... statement. Causes also the setting of the LINK option.

// OPTION LOG Logs all job control statements submitted to the system on SYSLST. This makes it easier to find errors in the job control statements.

// OPTION PARTDUMP Dumps the contents of the registers, a formatted portion of the supervisor area, and the current partition on SYSLST in case of abnormal program termination.

// OPTION DUMP Dumps the same information as is dumped by the PARTDUMP option, but provides an unformatted dump of the entire supervisor instead of just a formatted portion of it.

// OPTION DECK Causes the assembler (compiler) to write an object module to SYSPCH. The object module can then be combined with other object modules by the linkage editor to form one executable program, or it can be used as input to the librarian program to catalog it into a sublibrary.

// OPTION LIST, LISTX, SYM, XREF, ERRS Prints various listings produced by the language translators (compilers) on SYSLST. These listings include object code, symbol table, cross-reference, and error lists which are useful debugging aids during the test period of a program. SXREF may be specified instead of XREF to obtain a cross reference listing that includes only the referenced labels in the assembled program. Some of these options can be suppressed by specifying the prefix NO (for example, NOLIST, NODUMP).

Options may be set permanently using the STDOPT (standard option) command. Specified standard options become effective after the next /& statement or JOB statement.

Permanent (STDOPT) options are valid for all jobs unless overridden by an OPTION job control statement. Options specified in an OPTION statement remain in effect until

- A contrary option is specified, or

- A JOB or /& statement is encountered which resets the options to the permanent values.

You may display permanent and temporary options by means of the QUERY STDOPT and QUERY OPTION commands.

Communicating with Application Programs via Job Control

Program Switches (UPSI)

You can cause a program to take a specific path of action by setting program switches that can be tested by that program when it is executed.

You can set these program switches, called UPSI (user program switch indicator), "on" (1) or "off" (0) using the UPSI job control statement. The specific meaning attached to each bit in the UPSI byte depends on the design of the program. The statement

// UPSI 10000110

for example, sets bits 0, 5, and 6 of the UPSI byte to 1, and bits 1, 2, 3, 4, and 7 to zero. A program can inspect these switches and take a specific path based on their setting.

Passing a Parameter

Another way of passing information to a program before it starts execution is by way of the EXEC statement.

PARM parameters are useful, for example, in an accounting application that prepares reports of daily, weekly, and monthly accounts. Using the parameter, you can inform the application program when the daily, weekly, or monthly reports are due. In the PARM parameter, you may specify up to 100 bytes of information, enclosed by single quotes. The PARM parameter may be specified up to three times thus allowing for a string in storage of up to 300 bytes. The following two statements are equivalent:

```
// EXEC PROGA,SIZE=PROGA,PARM='TUESDAY WEDNESDAY'
// EXEC PROGA,SIZE=PROGA,PARM='TUESDAY',PARM=' WEDNESDAY'
```

When the problem program receives control, register 1 points to a fullword in virtual storage. This fullword has the following layout:

Byte 0
Reserved. Bit 0 is an indicator:

1= length of PARM value greater than 0

0= length of PARM value equal 0

Byte 1-3
Address

The high-order bit of the fullword indicates whether the PARM value is a null string or not. The rest of the high-order byte is reserved. The address in bytes 1, 2 and 3 of this fullword is that of a two-byte field containing the length of the PARM character string. The PARM character string itself starts at an offset of 2 from this address.

To test whether a PARM value has been passed, compare registers 1 and 15. If their contents are equal, a PARM value was not specified.

The PARM value is stored in the system GETVIS area.

Run in Virtual or Real Mode

All programs invoked through an EXEC job control statement are normally run in virtual mode. If there is a need to run a program in real mode, you specify the REAL operand in the EXEC statement.

Example:

```
// JOB NAME
   ...
   additional job control statements
   ...
// EXEC PROGA,REAL
/&
```

If, for the above example, this job is submitted in partition F8, then there must be enough processor storage allocated to the F8 partition by an ALLOC R command to hold the entire program PROGA.

If a program executing in real mode needs a real GETVIS area, use the SIZE operand of the EXEC statement. For example:

```
// EXEC PROGA,REAL,SIZE=32K
```

Assuming that the F8 partition has 52KB of processor storage allocated then the remaining 20KB of that partition are available to the program for dynamic allocation by a GETVIS macro request, which is discussed later in this chapter.

If you specify SIZE=AUTO or SIZE=phasename, job control automatically uses the information in the program's sublibrary directory entry to calculate the size of the program that is to be loaded. SIZE=AUTO requests the system to take the size from the largest phase of the program. With SIZE=phasename, the actual size of the named phase is used.

With a few exceptions, all IBM-supplied and user-written programs can be executed in virtual mode. The exceptions are:

- The IBM-supplied program OLTEP (Online Test Executive Program).
- User-written programs if they contain channel programs for devices that are not supported by z/VSE.
- User-written programs if they
  - contain routines with time-dependent code for execution of I/O requests.
  - contain channel programs that are modified during command execution.
  - contain I/O appendage routines causing page faults.

A program may request additional storage from the partition GETVIS area (this area is described in the following section, "Dynamic Allocation of Storage"). During real mode execution, this storage is obtained from the allocated processor storage which remains after the SIZE operand of the EXEC statement has been processed. The size of the real GETVIS area equals the size specified in the ALLOC R command minus the size specified in the SIZE operand of the EXEC statement. Specifying a SIZE value, therefore, allows you to issue GETVIS requests from a program running in real mode (contrary to execution in virtual mode, a default partition GETVIS area is not provided for real mode execution). For a program that is executed in real mode, allow 16K per open file, and allow additional processor storage if double buffering is used or if FBA files with large CI-sizes or VSE/VSAM files are opened. For most IBM-supplied programs that must run real, an allocation of 48K for GETVIS requests is sufficient.

Note that the FREEVIS macro releases GETVIS space which was obtained through a GETVIS macro; that space is again available for subsequent GETVIS requests. When issued from a program running in real mode, however, the space is not returned to the page pool until the execution of the particular job is finished.

Allocation of Partition GETVIS Storage

GETVIS areas are dynamic storage areas and are part of a static or dynamic address space as described under “GETVIS Areas” on page 9.

Each partition (static and dynamic) has an area that is called the partition GETVIS area. The minimum and default partition GETVIS area for a partition is 48 KB. If a partition extends beyond 16 MB, the partition GETVIS area has a minimum (and default) size of 48 KB plus the partition area that extends beyond 16 MB. The default of 48 KB is not applicable to real mode execution; in this case, you must reserve storage yourself (as described in the preceding section).

The partition GETVIS area is also used by certain system components for functions such as opening files or processing labels. When you no longer need the area that is acquired by a GETVIS macro, release it with a FREEVIS macro. For details about using these macros, refer to z/VSE System Macros Reference.

Figure 26 on page 56 shows the virtual storage layout of a 440 KB partition with a partition GETVIS area of 48 KB.

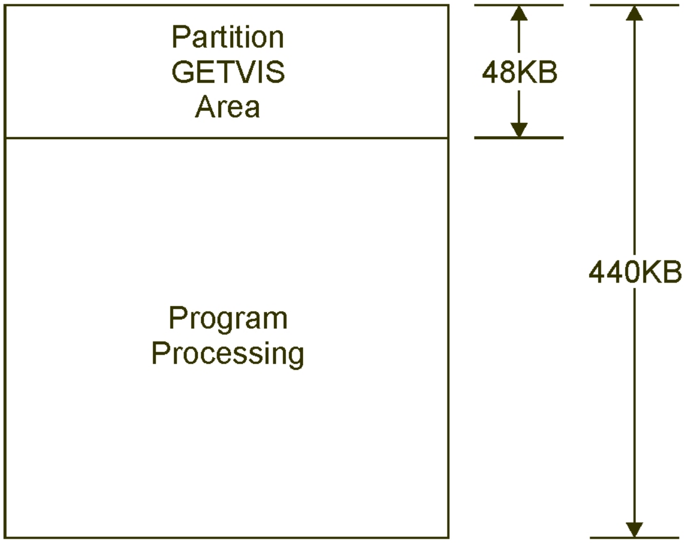


Figure 26. Storage Layout of a Partition with Default GETVIS Area

You can increase the size of a partition GETVIS area through

- the SIZE job control or attention routine command (static partitions only), or
- the SIZE parameter of the job control EXEC statement (static and dynamic partitions).

Note:

The SIZE command is not allowed for dynamic partitions. The permanent SIZE for dynamic partitions must be specified in the dynamic class table. Refer to z/VSE Planning for details about the dynamic class table parameters.

An EXEC statement with the SIZE operand works for a dynamic partition in the same way as for a static partition.

With the SIZE command or parameter, you specify the amount of virtual storage available for program execution in a partition. The balance of that partition's allocation is the partition GETVIS area. Given a SIZE of 380 KB, the result is a storage layout for the partition as shown in Figure 27 on page 57.

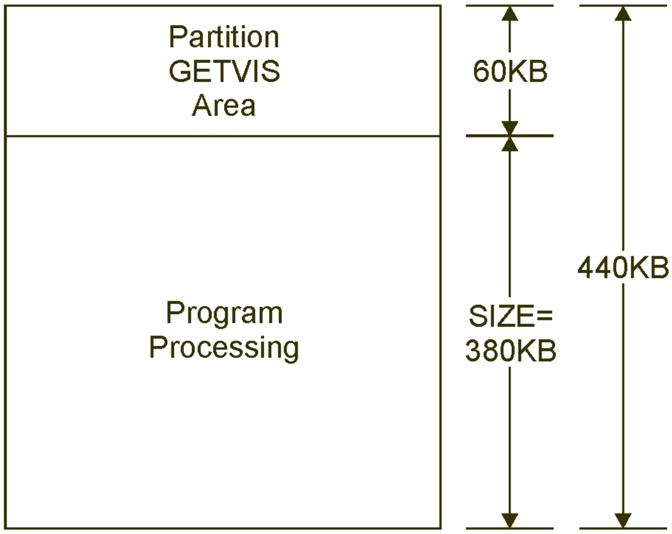


Figure 27. Storage Layout of a Partition after a SIZE Command was Given

The boundaries set by the SIZE command are valid until next IPL, or until another SIZE command is given, or until a partition allocation/deallocation takes place.

You can temporarily alter the partition GETVIS area by using the SIZE operand (as shown in Figure 28 on page 57 with a value of 360 KB) on the job control EXEC statement (also for dynamic partitions). The SIZE operand establishes boundaries in the same way as the SIZE command, except that the operand value holds only for one job step. At the end of the job step, the GETVIS size is set to the system default of 48 KB or to the previously established permanent value.

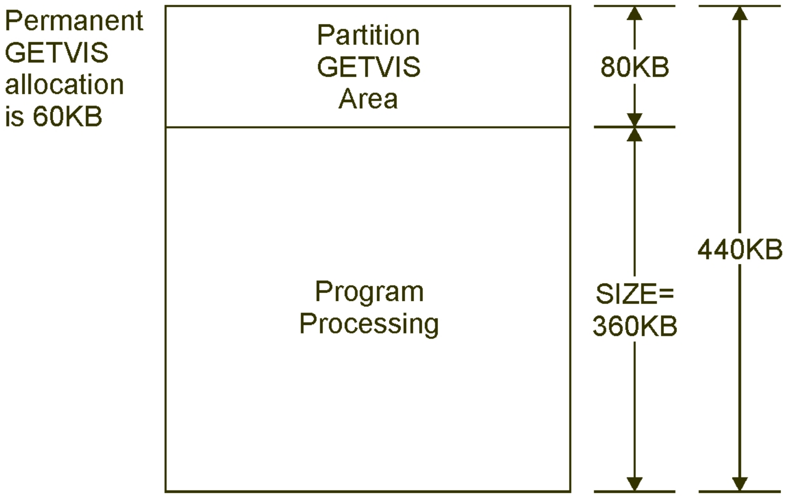


Figure 28. Program Execution with the SIZE Operand

Handling of System Input and Output

I/O devices (except disk devices) cannot be assigned to more than one partition at the same time. Using just one card reader for reading in jobs, for example, can lead to bottlenecks. The SYSRDR or SYSIPT job stream for one partition must be processed completely and the card unit unassigned before input for another partition can be read in. This also applies accordingly to the system output on SYSLST and SYSPCH if only one printer and one card punch are available.

Since this situation can cause a considerable decrease of system throughput, you may consider storing the input job streams and the system output on disk. This allows several partitions to read system input from or write system output to disk simultaneously. As a disk is a high-speed device, this increases system throughput.

VSE/POWER, a component of z/VSE, is available for this kind of input and output spooling. The spooling program stores the job streams on disk, transfers the jobs to the partitions for execution, and stores list and punch output on disk before it is finally printed or punched.

The devices used by dynamic partitions must be spooled by VSE/POWER.

System Files on Tape

If the system input units SYSRDR and SYSIPT are assigned to the same magnetic tape unit, they can (but need not) be referred to as SYSIN. If the system output units SYSLST and SYSPCH are assigned to the same magnetic tape, they must be referred to as SYSOUT. If SYSLST or SYSPCH is assigned to a standard label tape and no new label information is supplied, the old labels remain on the tape. SYSIPT and/or SYSRDR cannot be assigned to a multivolume tape file.

To store an input stream on magnetic tape, you must write your own program that transfers this job stream to the tape. Assume, as in the example in Figure 29 on page 58, that you have written such a program and cataloged it in a sublibrary under the name CDTAPE.

```
        // JOB BUILDIN          ┐
    (1) // ASSGN SYS004,00C     │      read from 00C as SYSRDR
    (2) // ASSGN SYS005,182     │
    (3) // EXEC CDTAPE          ┘
        // JOB A                ┐
           .                    │
           .                    │
        /&                      │
        // JOB B                │      job stream read from SYS004
           .                    │
           .                    │
        /&                      │
    (4) **                      ┘
        /&
```

1. SYS004 is assigned to the card reader from which CDTAPE reads the job stream.
2. SYS005 is assigned to the tape, which is to receive the job stream.
3. The CDTAPE program is executed and writes the job stream onto tape.
4. ** As defined in program CDTAPE to signal end of file on SYS004.

Figure 29. Creation of SYSIN on Tape

After completion of the job BUILDIN shown in Figure 29 on page 58, you can assign SYSIN to the tape containing the job stream; job control then reads and processes the jobs A and B from the tape just as it would have done from the card reader.

In the same way, you can direct the system output on SYSLST and SYSPCH to go on magnetic tape and then use your own or an IBM-supplied program to print or punch the contents of the tape on the printer or card punch.

After a system file on tape has been processed, it is recommended to use a CLOSE job control command (no //). This causes a tapemark to be written after the file. The second (optional) operand of the CLOSE command can be used to unassign a system logical unit or reassign it to another device. The following command closes the SYSIN file on tape and reassigns SYSIN to the card reader at address 00C:

```
CLOSE SYSIN,C
```

The CLOSE command can either be entered on SYSLOG or can be included at the end of the job stream on tape.

System Files on Disk

When both SYSRDR and SYSIPT are assigned to the same disk, they must refer to the same disk extent, and must be assigned as SYSIN. SYSOUT cannot be used if SYSLST and SYSPCH are assigned to disk. Only single extent system files are supported.

For system files on disk, you must provide the required label information by means of DLBL and EXTENT job control statements. In these statements, use the following predefined file names:

```
    IJSYSIN for SYSRDR, SYSIPT, SYSIN
    IJSYSPH for SYSPCH
    IJSYSLS for SYSLST
```

The assignment of a system file to a disk extent must always be permanent, and it must follow the DLBL and EXTENT statement.

Example for a SYSIN assignment to a CKD type disk device:

```
// DLBL IJSYSIN,'DISKINFILE'
// EXTENT SYSIN,123456,1,0,1260,30
   ASSGN SYSIN,131
```

After a system file on disk has been processed, it must be closed by a CLOSE job control command (no //). The second (optional) operand of the CLOSE command can be used to unassign a system logical unit or reassign it to another device. The CLOSE command can either be entered on SYSLOG by the operator or it can be included at the end of the job stream on disk. The following command closes a SYSIN file on disk and reassigns SYSIN to the device at address 00C:

```
  CLOSE SYSIN,00C
```

If SYSIPT is assigned to a disk extent, the CLOSE command must precede the /& statement. Multiple SYSIPT data files can be read via multiple job steps with one /& at the end of the job stream.

If an FBA disk device is to be used, you should be aware that the DTFSD support for system files on disk is limited to sequential GET or PUT for fixed unblocked records. (That is, the UPDATE=YES parameter is not supported.)

Record Formats of System Files

SYSLST records are 121 characters and SYSPCH records 81 characters long. From SYSRDR and SYSIPT, job control accepts either 80- or 81-character records.

The first character of the SYSLST and SYSPCH records is assumed to be an ASA carriage control or stacker selection character, respectively.

Using Conditional Job Control

By using conditional job control you can increase considerably the flexibility of jobs. It allows either skipping or execution of subsequent parts of the job, based on the return codes passed to job control by programs, or when abnormal termination occurs or the job is canceled.

In addition, you may set parameters within a job. A parameter may be a character string or a predefined return code, and can be checked during processing of the job to make decisions on skipping or executing particular job control statements.

In application programs in Assembler language, you can set return codes. These may range from 0 - 4095, and you may choose your own return codes. If you set a return code higher than 4095, it is treated as if it were 4095. There are several return codes used by IBM programs, such as the linkage editor, which have a standard meaning. These meanings are shown in Figure 30 on page 60.

Return Code
Meaning

0
The requested function has been executed successfully.

4
A problem has been encountered, but it was possible to continue and complete the function.

8
The requested function has been completed, but major parts were bypassed.

12
The requested function could not be performed.

16
A severe error occurred and the step was terminated.

Figure 30. Standard Return Codes for Conditional Job Control

If the job control program receives a return code greater than or equal to 16, it terminates the job unless an ON statement, specifying a different action for this return code, has been given.

Thus, when using the linkage editor, you can make further processing of the job dependent on the return codes provided by the linkage editor.

To use conditional job control, a program has to pass a return code in register 15 when returning control to the system at end of processing. This may be done in one of two ways:

1. When the system loads a program for execution it provides the return address in register 14. If the program returns control to the system via this address, the system considers the contents of register 15 as a return code.

   Passing the return code in this way requires the program to save the return address of register 14 for returning control to the system and load register 15 with the return code at the end of processing.

   Bytes 0 and 1 of register 15 are not part of the return code. However, bit 0 is used to indicate whether a dump is required or not. If bit 0 is off, job control passes control to the supervisor via the EOJ macro; if it is on, via the DUMP macro.

2. The return code can also be passed via an EOJ or DUMP macro issued directly by the program being processed. The program or main task which issues an EOJ macro indicates to the system that the job step is finished. A DUMP macro causes the system to terminate the job step and to produce a dump which is either printed on SYSLST or stored in the dump sublibrary.

   The return code can be specified directly with the macro or provided in a register with a corresponding indication in the macro specification. If a register is used, Register 15 is recommended. If the indication is omitted, a return code of zero is assumed. If a subtask issues a DUMP or EOJ macro with a return code, this return code is ignored.

The return codes passed to job control are tested by IF or ON statements. These statements are shortly described in the following paragraphs, together with the other statements provided for conditional job control. For a complete description and the syntax of each statement refer to z/VSE System Control Statements.

Return Code Setting by Job Control

As mentioned above, application programs may pass return codes to job control, which can then be tested by IF or ON statements.

It is also possible to have return codes set by job control itself.

The Job Control Command (JCC) SET provides the MRC=n and RC=n parameters as new last and/or maximum return code to be specified:

```
SET MRC=n          With n=0,...,4095
SET RC=n           With n=0,...,4095
```

Both parameters require a decimal value n in the range of 0 through 4095 (inclusive).

The SET parameter RC=n sets the last return code ($RC) to the specified decimal value. It also updates the maximum return code ($MRC) if it is not set or less than the specified value.

The SET parameter MRC=n sets the maximum return code ($MRC) to the specified decimal value. It does not affect the last return code ($RC) in any way.

You can also direct Job Control to set a return code in Cancel or Abnormal Termination situations (default-wise no return code is set in such a situation).

When one of the following OPTIONs is defined, Job Control sets the corresponding return code in the appropriate situation:

```
OPTION ABENDRC=n|NO         With n=0,...,4095 and default NO
OPTION CANCELRC=n|NO        With n=0,...,4095 and default NO
OPTION JCANCLRC=n|NO        With n=0,...,4095 and default NO
```

These OPTIONs may also be set as a standard option (STDOPT):

```
STDOPT ABENDRC=n|NO         With n=0,...,4095 and default NO
STDOPT CANCELRC=n|NO        With n=0,...,4095 and default NO
STDOPT JCANCLRC=n|NO        With n=0,...,4095 and default NO
```

The following table describes which OPTION (or STDOPT) is used in which situation:

STDOPT / OPTION	When used?	Correlates to Conditional JCL situation
ABENDRC	When a job step abnormally terminates (for example because of a program check or due to the CANCEL macro from the main task or CANCEL ALL from a subtask).	$ABEND situation
CANCELRC	When the OPERATOR CANCELs the job, that is when • The job is canceled with the AR CANCEL command. • Or the job is canceled by POWER (PCANCEL job or PFLUSH partition). • Or the job is canceled due to a JCL failure and simulated operator cancel is requested (that is, OPTION JCANCEL and SCANCEL are set).	$CANCEL situation
JCANCLRC	When the job is canceled • By the JCL CANCEL command. • Or because of a JCL error (and OPTION JCANCEL has been set but not SCANCEL).	n/a (neither $ABEND nor $CANCEL)

The output from the QUERY STDOPT and QUERY OPTION commands has been extended to show the currently defined STDOPTs / OPTIONs.

Refer to z/VSE System Control Statements for a description of Job Control Commands and Statements.

Statements for Conditional Job Control

To build jobs that take advantage of conditional job control, the following statements are provided:

- ON statement
- IF statement
- SETPARM statement
- GOTO statement
- LABEL statement

ON Statement

The ON statement is valid for the rest of the job. It specifies a condition which may arise, and the action to be taken when it arises. The condition is checked after each job step following the ON statement. An ON statement may look as follows:

```
ON $RC>30 OR $ABEND GOTO ERR10
```

This statement causes a skip to label ERR10 if the return code is greater 30 or an abnormal termination occurs. The following default ON conditions are in effect whenever a job is started:

```
ON $RC<16 CONTINUE
ON $RC>=16 GOTO $EOJ
ON $CANCEL GOTO $EOJ
ON $ABEND GOTO $EOJ
```

$EOJ means the end-of-job statement /&. $ABEND means an abnormal termination. $CANCEL means a job cancellation through the AR CANCEL or job control CANCEL command.

An ON condition specified within a procedure is in effect till the end of the procedure; an ON condition specified outside a procedure (from SYSRDR, for example) is in effect till end-of-job. ON conditions are checked in the sequence last in-first checked.

IF Statement

The IF statement is valid at the point in the job where it occurs. Like the ON statement, the IF statement specifies a condition. However, no action is specified. The next statement or command in the job stream may be regarded as the "action". If the condition is true, the statement following the IF is processed; if not, this statement is skipped.

When an IF statement is entered from the console (SYSLOG), the "following statement" is the next statement entered from SYSLOG, or, if no further statement is entered from SYSLOG and ENTER has been pressed, the next statement from SYSRDR. // JOB, /&, and /+ statements are processed independent of any skip condition. IF statements may look as follows:

1. IF $RC<=8 THEN

   This statement causes the following statement to be processed if the return code is less than or equal to 8. If the return code is greater than 8, the statement is skipped.

2. IF $MRC<=36 AND $RC<8 THEN

   If no return code passed by previous job steps exceeds the maximum of 36 and the return code of the last job step is less than 8, the following statement is executed. Otherwise, it is skipped.

3. IF PNAM=WEEK THEN

   If the parameter named PNAM has been defined before as WEEK, the following statement is executed. Otherwise, it is skipped. Such a parameter can be set with a SETPARM statement.

SETPARM Statement

With the SETPARM statement you can assign a value to a symbolic parameter in a job control procedure. The value can be:

a character string,
$RC, or
$MRC.

Whenever the symbolic parameter occurs in the job (after the SETPARM statement), it is processed as if it were the specified string or return code. This is called "substitution of the parameter". For example, after the statement:

```
SETPARM PNAM=WEEK
```

the symbolic parameter &PNAM is replaced by the character string WEEK. Specifying SETPARM RC1=$RC causes the symbolic parameter &RC1 to be replaced by the return code of the last job step; SETPARM RC2=$MRC causes the parameter &RC2 to be replaced by the highest return code up to this point in the job. The parameter values can be used to modify job control statements, or they can be checked by the IF statement.

The SETPARM statement allows you to specify symbolic parameters at different levels. These levels determine for how long a parameter is active during job processing.

- Symbolic parameters at level n
- Symbolic parameters at the VSE/POWER job level
- Symbolic parameters at system level

For details refer to z/VSE System Control Statements.

GOTO Statement

With the GOTO statement you can skip all following statements (except // JOB, /&, and /+ statements) up to the target label specified in the GOTO statement. For example, when you specify:

```
GOTO ACCB1
```

processing of the job continues after the statement:

```
/. ACCB1
```

If a /+ statement is encountered during searching, the rest of the job is skipped.

You can specify $EOJ as a label, indicating that all statements up to end-of-job are to be skipped.

If SYSLOG gets control for input (in case of an error, for example), skipping is suspended. If a GOTO statement is entered via SYSLOG, you can directly return to SYSRDR (by pressing ENTER) and have the GOTO statement executed. If you enter additional statements via SYSLOG, these statements are executed and the execution of the GOTO statement is suspended until control is given back to SYSRDR.

Label Statement

With the label statement you can define entry points (labels) within a job. Such a label can be specified in a GOTO statement, or in the GOTO action operand of an ON statement. The format of a label statement is shown below:

```
/. name
```

The name must be one to eight alphameric characters long. The first character must be alphabetic. The two characters '/.' identify the statement as a label.

A symbolic parameter is not allowed as a label.

The job example shows how the IF, ON, SETPARM, GOTO and LABEL statements are used to build a conditional job.

Assume that you have a job that performs some kind of calculations on either a weekly or monthly basis. Five programs are involved: CALP1, CALP1A, CALP2, CALP3, and CALERR. CALP3 is only to be performed for monthly calculations. The execution of CALP1A and CALERR depends on the return codes passed. CALP1A needs to know whether a weekly or monthly calculation is to be performed.

In the example a monthly calculation is assumed:

```
     // JOB CALCULA
(1)  ON $RC>20 GOTO CAERROR
(1)  ON $RC<=20 CONTINUE
(2)  SETPARM PNAM=MONTH
(2a) // EXEC CALP1
(3)  IF $RC¬=0 THEN
(4)  // EXEC CALP1A,PARM='&PNAM'
     // EXEC CALP2
(5)  IF PNAM=MONTH THEN
     // EXEC CALP3
(6)  GOTO $EOJ
(7)  /. CAERROR
     // EXEC CALERR
 (8)  /&
```

(1)
The ON statement sets a global condition that is valid for the whole job stream. After each job step (EXEC performed) it is checked whether the return code passed is greater 20. If so, all statements up to label CAERROR are skipped and the error program CALERR is called. Otherwise, processing continues with the next statement.

ON $RC<=20 CONTINUE overrides the default-ON condition: ON $RC>=16 $EOJ.

(2)
The SETPARM statement assigns to parameter PNAM the value (MONTH) which is used as input for program CALP1A and for determining whether program CALP3 must be run.

(2a)
If the return code passed is greater than 20, processing continues at label CAERROR, because the ON conditions are checked immediately after end-of-step, and the following IF statement (3) cannot be processed.

(3)
After program CALP1 has finished processing it is checked whether the return code is not equal to zero. If so, program CALP1A is performed, otherwise (that is, return code=0) processing continues with CALP2.

(4)
By specifying PARM='&PNAM', substitute the symbolic parameter &PNAM with MONTH and supply it as input for CALP1A.

PARM is an operand of the EXEC statement for passing information to the program to be executed. (Besides a symbolic parameter, as in this example, you can specify a value of up to 100 characters in length and enclosed in quotes.) Symbolic parameters are discussed in detail later in this section.

(5)
After program CALP2 has finished, a check is made for parameter PNAM. If the value is MONTH, CALP3 is performed. Value WEEK would cause program CALP3 to be skipped.

(6)
After the job stream has been successfully completed the GOTO statement causes a skip to end-of-job (/&).

(7)
Label CAERROR defines the entry for error processing performed by program CALERR.

(8)
If no error processing is required, you could specify in (1) ... GOTO $EOJ instead of GOTO CAERROR. This would cause a direct skip to end-of-job (/&).

When using conditional job control the following rules must be observed:

- The statements of a job stream can only be skipped in forward direction.
- If a label is not found before end-of-procedure or end-of-job, a skip to end-of-job is performed.
- No check for duplicate labels is performed. A skip is always performed up to the first matching label found.
- ON-conditions are checked first whenever a job step has been executed.
- If there are several ON statements defined in a job stream, the conditions and actions of these statements are stored and processed in a "last in-first checked" sequence.

Abnormal Termination of a Job Stream

Jobs that terminate abnormally are handled as follows:

Cancel Condition

If a job stream is canceled, either through the AR or job control CANCEL command, and an ON $CANCEL condition is in effect, the action specified is performed (for example, a skip to a label) and processing continues. Otherwise, the job is flushed.

ABEND Condition

An ON $ABEND condition is raised when a job step ends abnormally. If an ON $ABEND action has been specified, this action is performed (for example, a skip to a label) and processing continues. Otherwise, the job is flushed.

If you want to perform your own abnormal termination processing, you have to use the STXIT AB user exit. Such a routine may issue a return code and define in addition how the erroneous job step is to be terminated - via an EOJ or via a DUMP macro.

In the following example, two input files are to be merged and then processed. The contents of the input files is first verified and the subsequent processing steps are chosen according to the result of the verification step:

```
     // JOB EXAMPLE 1
(1)  ON $ABEND GOTO ERRLST
(2)  SETPARM INPPARM=''
     // EXEC VERXL1
(2a) SETPARM RC1=$RC
     // EXEC VERXL2
(3)  IF $MRC=0 THEN
     GOTO MERGE
(3a) IF $RC>0 AND RC1>0 THEN
     GOTO ERRLST
(4)  SETPARM INPPARM=INCOMPLETE
     // EXEC COPYXL
(5)  * Use alternate file for processing
     * incomplete input
     // DLBL NEWMFIL1,'NEW MASTER',10
     // EXTENT SYS010,,1,0,200,20
(5)  // PAUSE   PLEASE ENTER TEMPORARY ASSIGNMENT
     GOTO PROCESS
```

```
(6)  /. MERGE
     // EXEC MERGEXL
(7)  /. PROCESS
     // EXEC PROXL12
(8)  IF $RC>4 OR INPPARM=INCOMPLETE THEN
     GOTO ERRLST
(9)  ON $ABEND GOTO $EOJ
     // EXEC LISTXL
     GOTO $EOJ
(10) /. ERRLST
     // EXEC ERRLST12
     /&
```

(1)
An ON condition is set for the whole job stream. In case of an abnormal termination a skip to label ERRLST is performed.

(2)
The parameter named INPPARM is nullified.

(2a)
The return code of program VERXL1 is assigned to the parameter RC1.

(3)
After both verification programs (VERXL1 and VERXL2) have completed processing, a test is made whether return codes greater zero have occurred. If not, a skip to label MERGE is performed and both input files are merged and processed.

(3a)
If both programs, VERXL1 and VERXL2, passed return codes greater zero, a skip to label ERRLST is performed and program ERRLST12 is executed before finishing the job.

(4)
If only one program ended with a return code greater zero, parameter INPPARM is set to INCOMPLETE. Program COPYXL is performed to copy the incomplete input to the processing file.

(5)
An alternate file must be used to process the incomplete input and the // PAUSE statement allows the operator to enter // ASSGN SYS010, ... before a skip to label PROCESS is performed.

(6)
Label MERGE is the entry point for processing the correct input.

(7)
Label PROCESS is the entry point if no merge could be performed.

(8)
If the return code of program PROXL12 is greater than 4 or INCOMPLETE is set for parameter INPPARM, a skip to label ERRLST is performed and program ERRLST12 is executed.

(9)
The $ABEND condition is changed from GOTO ERRLIST (1) to the system default which is GOTO $EOJ. This means, if program LISTXL terminates abnormally, a skip to end-of-job is performed and no error list is produced.

(10)
Label ERRLST is the entry point in case of an error and if an error list is to be produced.

Using Cataloged Procedures

This section describes cataloged procedures and how to use them. It includes information on:

- Input data in procedures
- Partition-related procedures
- Retrieving cataloged procedures
- Multi-step procedures
- Using symbolic parameters
- Nested procedures

How a procedure is cataloged is discussed in Chapter 4, “Using VSE Libraries,” on page 79.

SYSIPT Data in Cataloged Procedures

Data read from the logical unit SYSIPT may be made part of your cataloged procedure. System service and utility programs, as well as language translators read their input from SYSIPT. Such input may be a data file (end indicator is a /* statement) or control information (submitted via statements) required for a particular program. Only those control statements are read from SYSIPT that follow the EXEC statement calling the program. Control statements that can be placed before the EXEC statement are read from SYSRDR as are all other statements of a job stream. The linkage editor statements (ACTION, ENTRY, INCLUDE, and PHASE) fall into this category.

To include SYSIPT inline data in cataloged procedures is useful mainly in case of control information for system programs.

Inline data in procedures can also be any data that is processed under control of the device independent IOCS (DTFDI) used by your program or IBM-supplied programs. Normally, though, you would not catalog source programs or data for your application programs as part of a procedure.

When cataloging a procedure containing SYSIPT data, you must use the operand **DATA=YES** in the librarian CATALOG command. When procedures are nested, either all or none of them must have been cataloged with DATA=YES. You cannot mix DATA=YES and DATA=NO procedures in one nesting.

Note:

1. Including SYSIPT data in a cataloged procedure requires that you include all data not only part of it.
2. SYSIPT data in a cataloged procedure cannot contain symbolic parameters.

Cataloging Partition-Related Procedures

Although a given procedure may be executed in any partition, a particular job may need a specific set of job control statements dependent on the partition in which it runs. For example, you may want to run a job to define search chains and assign logical units for each partition. Since each partition requires a different set of statements, you need a cataloged procedure for each of your partitions. Partition-related cataloged procedures then allow you to retrieve and execute the appropriate procedure with one version of the EXEC statement, no matter which partition you are running in. The ASI JCL procedures are examples of partition-related procedures.

To catalog partition related procedures, follow the naming conventions described below.

Naming Conventions for Static Partitions:

- $ as the first character of the procedure name.
- The partition indicator (0=BG, 1=F1, 2=F2, ..., A=FA, B=FB) as the second character.
- Any alphameric characters, including trailing blanks, as the third through eighth characters.

Naming Conventions for Dynamic Partitions:

- The class of the dynamic partition as the first character of the procedure name.
- $ as the second character of the procedure name.
- Any alphameric characters, including trailing blanks, as the third through eighth characters.

In the EXEC statement used to start the job, the first two characters of the procedure name must be $$, with the remaining characters identical to the last six characters of the cataloged name. For example, the statement:

```
// EXEC PROC=$$INIT
```

calls procedure $0INIT when entered in the BG partition, $1INIT in the F1 partition, $2INIT in the F2 partition, and so on. For a dynamic partition with class 'C', for example, the procedure name would be C$INIT.

Retrieving Cataloged Procedures

To retrieve a cataloged procedure from a sublibrary, you must use the PROC operand in the EXEC job control statement specifying the name of the cataloged procedure. Assume that a program called PAYROLL uses the following job control statements (in addition to the // JOB and /& statements) and that these statements have been cataloged in a sublibrary under the procedure name PAY.

```
   // ASSGN SYS017,SYSRDR
   // ASSGN SYS018,SYSPCH
```

```
// ASSGN SYS019,00E
// ASSGN SYS020,TAPE
// ASSGN SYS021,DISK,VOL=111111
// TLBL TAPFLE,'FILE-IN'
// DLBL DSKFLE,'FILE-OUT',2018/365,SD
// EXTENT SYS021,111111,1,0,200,400
// EXEC PAYROLL
/+
```

If the program PAYROLL is to be executed, the programmer (or operator) would simply prepare the following job control statements:

```
// JOB USER1
// EXEC PROC=PAY
/&
```

When job control reads the EXEC PROC=PAY statement in the input stream, it knows by the operand PROC that a cataloged procedure is to be inserted and retrieves the procedure from the sublibrary, which must have been specified in a

```
LIBDEF PROC,SEARCH=...
```

statement entered in the partition. When job control reads and processes the control statement

```
// EXEC PAYROLL
```

the program PAYROLL is loaded and given control. When the execution of that program is complete, job control reads the next statement from the sublibrary and, in this example, would find an end of procedure indicator (/+). This causes job control to end reading procedure statements from the sublibrary and to continue reading input from the device that is assigned to SYSRDR. Job control now finds the /& statement and performs end-of-job processing as usual.

Note: The listing of job control statements on SYSLOG and/or SYSLST shows the message `EOP  PAY` at the end of the inserted procedure (depending on the LOG/NOLOG option or the LOG/NOLOG command).

Several Job Steps in One Procedure

A cataloged procedure may contain more than one EXEC statement, that is, it may contain control statements for more than one job step (within the same job).

A program written in assembler language, for instance, requires three job steps to assemble, link edit, and execute the program. Using a cataloged procedure, your input stream for the entire job (on SYSIN for simplicity) would contain the following:

```
// JOB USER
// OPTION LINK
// EXEC ASMA90....
   ...
source deck of program to be assembled
   ...
/*
// EXEC LNKEDT
// EXEC
   ...
data for program to be executed
   ...
/*
/&
```

If the OPTION statement and the three EXEC statements were cataloged under the name ASDPROC, the input stream could be simplified as shown below:

```
Input from SYSIN:            Procedure ASDPROC:

// JOB USER
// EXEC PROC=ASDPROC         // OPTION LINK
  ...                        // EXEC ASMA90....
source statements of
program to be                // EXEC LNKEDT
assembled                    // EXEC
```

```
  ...
/*                            /+ (end indicator)
  ...
data to be
processed
  ...
/*
/&
```

Note:

1. The statement

   ```
   // EXEC ASMA90....
   ```

   calls the High Level Assembler. Refer to "High Level Assembler Considerations" on page 139 for further details.
2. Since all data input is to be read from SYSIPT, procedure ASDPROC must be cataloged with DATA=NO.

The same can be done for any number of job steps. The programs called within one job should logically belong together. A stock control program STOCK, for instance, may be run daily to compile statistical data that can be used to prepare the following lists:

1. An exception list that shows which items are low in stock. Required daily.
2. A list that shows the sales in currency for a certain item or group of items. Required weekly.
3. A list that shows the sales in number of units for each item or group of items. Required monthly.
4. An inventory list. Required half-yearly.

To simplify processing, four procedures may have been cataloged:

STKPR1 -
two job steps: the first to execute STOCK, the second to prepare list 1.

STKPR2 -
three job steps: the first two are the same as for STKPR1, the third to prepare list 2.

STKPR3 -
four job steps: the first three the same as for STKPR2, the fourth to prepare list 3.

STKPR4 -
five job steps: the first four the same as for STKPR3, the fifth to prepare list 4.

Which lists are printed after every run of STOCK depends then on which cataloged procedure is used.

Using Symbolic Parameters

You can further increase the flexibility of your jobs by using, in addition to conditional job control, symbolic parameters as described in the following paragraphs. You can specify symbolic parameters in any job or cataloged procedure. By doing so you are able to set up your job streams in advance and modify them at processing time as needed.

Defining Symbolic Parameters

A symbolic parameter is a character string of 1 to 7 alphameric characters preceded by an ampersand (&). The first character after & must be alphabetic. Parameters are referenced by a parameter name of 1 to 7 alphameric characters, where the first one must be alphabetic.

When a cataloged procedure is processed each symbolic parameter is either assigned a value or nullified. This can be done with the EXEC PROC (for defining or passing values), with the PROC (for defining default values), and with the SETPARM statement (for setting values during processing).

For a complete description and the syntax of each of these statements refer *z/VSE System Control Statements*. The following example shows how to define symbolic parameters:

Job Control Statements Submitted:

```
     // JOB TESTFILE
     // EXEC PROC=TESTFIL,FID=FILE2,SER=000002
     /&
```

Procedure Called (TESTFIL):

```
     // PROC PGM=PTAP1,FID=FILE1
     // TLBL TEST,'&FID',,&SER
     // EXEC &PGM
     /+
```

Job Generated:

```
     // JOB TESTFILE
(1)  // TLBL TEST,'FILE2',,000002
(2)  // EXEC PTAP1
     /&
```

(1)
The definition of the symbolic parameters &FID (file identification) and &SER (file serial number) are provided by the EXEC PROC statement.

(2)
Since EXEC PROC does not provide a substitution for &PGM, the default value PTAP1, assigned in the PROC statement, becomes effective.

When using symbolic parameters the following rules must be observed:

- The operation field of a statement cannot contain a symbolic parameter.
- Symbolic parameters are not allowed in a PROC or label statement.
- A symbolic parameter that is not defined causes an error message.
- If the value assigned to a symbolic parameter is a character string of alphameric characters only, no enclosing quotes are necessary. Otherwise, it has to be enclosed in quotes.
- The maximum length allowed for the value of a symbolic parameter is 50 characters.
- If the same parameter is defined in an EXEC PROC statement and in the PROC statement of the called procedure, the value assigned in the EXEC PROC statement is used. Values assigned in the PROC statement of the procedure are default values.
- If an ampersand (&) is to be part of the JCL statement or of the string to be assigned to a parameter, the following rules apply:
  - An & must be represented by two &;
  - A single & followed by an alphabetic character is interpreted as the beginning of a symbolic parameter.
  - A single & followed by a non-alphabetic character is an invalid combination, except a single & followed by a blank (which is left unchanged).
- For PROC and EXEC PROC statements, the following rules apply:
  - If you do not want to assign default values to parameters, you do not need a PROC statement.
  - If a cataloged procedure with symbolic parameters does not contain a PROC statement, PROC without operands is assumed.
  - If duplicate parameter names occur in an EXEC PROC or PROC statement an error message is issued.
- For EXEC PROC and PROC statements nine continuation lines are allowed.
- The recommended maximum number of parameters to be specified per statement is 36. Sufficient buffer space is available for that number. If you specify more, the buffer space available may not be sufficient.

SETPARM Statement

The values assigned by SETPARM are effective outside of procedures or for the procedure in which the SETPARM statement occurs. However, when a parameter is passed to a called procedure (as in nested procedures) it can be effective for the called procedure as well. Refer to "Using Nested Procedures" later in this section.

Symbolic parameters at POWER job level (SETPARM PWRJOB) or system level (SETPARM SYSTEM) are effective on any procedure level. Their contents can be displayed on the console with the QUERY SETPARM command. For details refer to z/VSE System Control Statements.

Defining Parameters and Passing Parameter Values

Parameter values can be passed from a calling procedure to a called procedure. The term "calling procedure" includes the job stream submitted. When a called procedure calls another cataloged procedure, the term "nested procedures" applies. Nested procedures are discussed in detail later in this chapter. The following discussion applies to the job submitted and to any procedure called by it.

When you call a procedure, you can define parameters and pass values in various ways. Assume the following procedure call issued by a job or a procedure named XYZ:

```
// EXEC PROC=CALL1,B=MAX,C='',D=,E,A=&A
```

B=MAX

The value (character string) MAX is assigned directly to parameter B for CALL1. This has no effect for XYZ.

C=''

Parameter C is nullified.

D=

Parameter D is nullified.

E

The value for E has to be defined in the calling procedure (XYZ). (For example, SETPARM E=400.) This value is in effect for XYZ and is passed to the called procedure (CALL1). but may be changed by CALL1, and the change is valid for both CALL1 and XYZ. For example, if CALL1 issues SETPARM E=500, the parameter value of 500 becomes valid for CALL1 and XYZ.

This allows you to pass back to the calling procedure a new value for a passed parameter, a return code for example.

A=&A

This assumes that both XYZ and CALL1 have a parameter named A. The current value of the parameter A in XYZ is passed to the parameter A in CALL1. A change of that value within CALL1 does not affect the value defined for XYZ.

Concatenating Symbolic Parameters

Symbolic parameters may be concatenated with a constant. If a symbolic parameter precedes a constant, a delimiter is required. A period (.) is used for that purpose. The period does not appear in the resulting value. If a period is required between the parameter and a following constant, two periods must be specified. For example:

```
Concatenated item:   Value of parameter:   Result:
```

```
&SIZE.(80)           SIZE=BLOCK            BLOCK(80)
&LIBNAME.LIB         LIBNAME=PRIV1         PRIV1LIB
SYS&NUM              NUM=001               SYS001
&A..B                A=X                   X.B
AB&C.D               C=''                  ABD
```

Also, a symbolic parameter can be concatenated with another symbolic parameter. They may be or may not be separated by a period, because the single '&' at the beginning of the second parameter also ends the first parameter.

```
Concatenated item:   Value of parameter:   Result:

&A.&B.
 or                  A=19,B=89             1989
&A&B.
```

In the following example, the execution of subsequent job steps, invoked by a cataloged procedure, is made dependent on the return code of the previously executed program. Job Control Statements Submitted:

```
// JOB SPROGRAM
ON $ABEND OR $CANCEL GOTO AB
ON $RC>7 CONTINUE
SETPARM CRC=0
// EXEC SPRGCTR1
SETPARM CRC=$RC,ABEND=NO
// EXEC PROC=SPROC123,CRC,PGM=SPRGE2
GOTO EVEN
/. AB
SETPARM ABEND=YES
// EXEC SPRGONLY
/. EVEN
IF CRC<12 OR ABEND=YES THEN
// EXEC SPRGEVEN
/&
```

Procedure Called (SPROC123):

```
IF CRC=0 THEN
// EXEC SPRGE1
IF CRC¬=8 THEN
// EXEC &PGM
IF CRC<=8 THEN
// EXEC SPRGE3
/+
```

If a cataloged procedure with symbolic parameters does not start with a // PROC statement, // PROC without parameters is assumed.

Sequence of Processed Statements:

```
    // JOB
(1) ON $ABEND OR $CANCEL GOTO AB
(1) ON $RC>7 CONTINUE
    // EXEC SPRGCTR1
(2) SETPARM CRC=$RC,ABEND=NO
    IF CRC=0 THEN
    // EXEC SPRGE1
    IF CRC¬=8 THEN
(3) // EXEC SPRGE2
    IF CRC<=8 THEN
    // EXEC SPRGE3
(4) GOTO EVEN
(5) /. AB
    SETPARM ABEND=YES
    // EXEC SPRGONLY
(6) /. EVEN
    IF CRC<=12 OR ABEND=YES THEN
    // EXEC SPRGEVEN
    /&
```

(1)
ON conditions that are in effect for the whole job are set.

(2)
The return code of program SPRGCTR1 is used for deciding which job steps are to be executed next (CRC=$RC).

(3)
Program name SPRGE2 has been defined by the EXEC PROC statement.

(4)
Normal processing; either program SPRGEVEN executed or a further skip to /& is performed.

(5)
Entry point for abnormal termination or a cancel condition occurred; programs SPRFONLY and SPRGEVEN (ABEND=YES) are executed.

(6)
Entry point for normal processing. Program SPRGEVEN is executed if the return code from program SPRGCTR1 is less than 12.

Using Nested Procedures

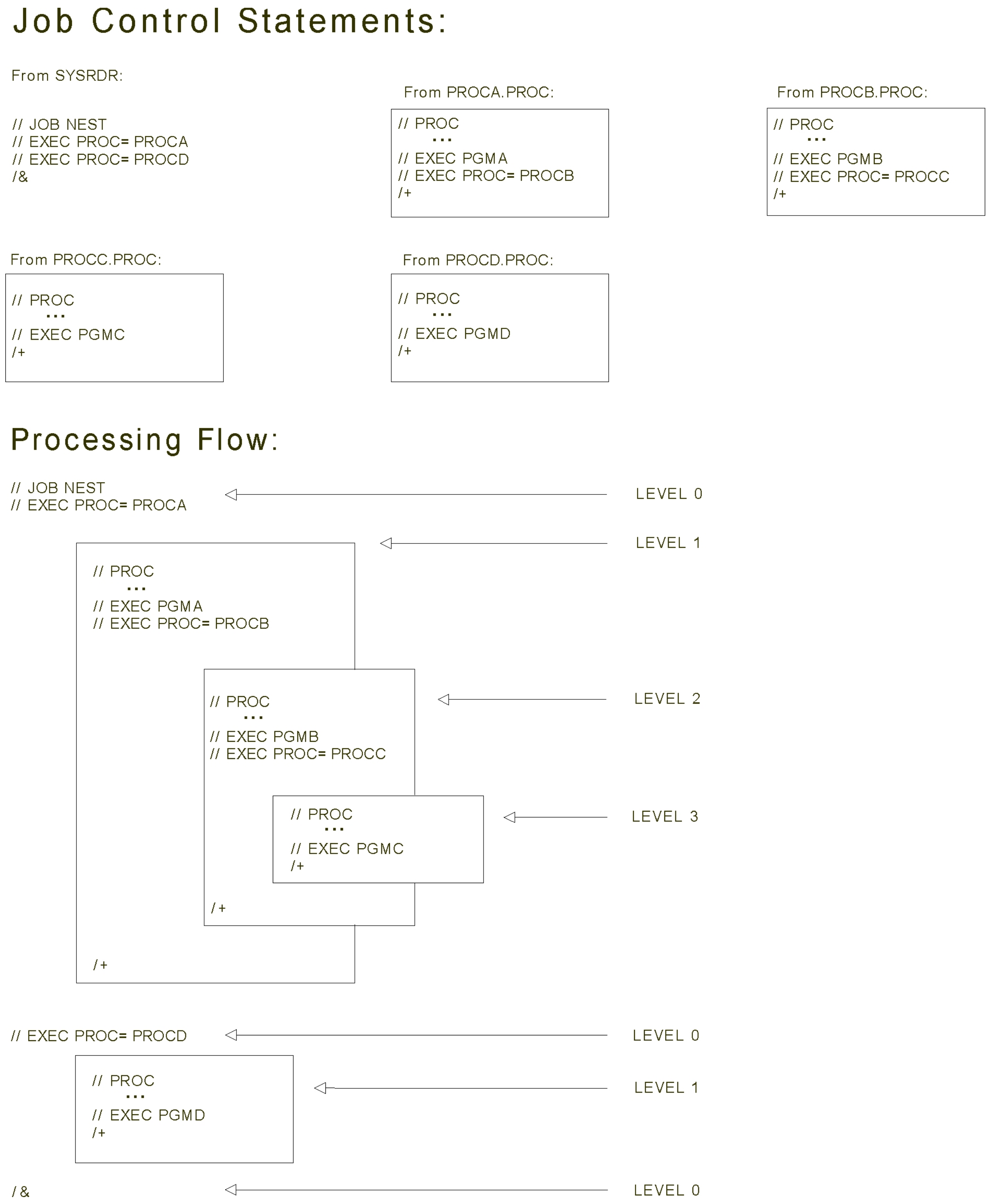
Cataloged procedures can call other cataloged procedures. This is referred to as nesting. Any cataloged procedure can be nested; such a procedure may or may not contain symbolic parameters and/or conditional job control statements. But by using nested procedures you can make full use of the advantages of conditional job control and symbolic parameters. Up to 16 nesting levels are allowed. Level 0 is the SYSRDR level, level 1 is a procedure call from level 0.

A procedure is said to contain another procedure when an EXEC PROC statement occurs in the procedure. For example, if procedure A has the statement // EXEC PROC=B in it, then procedure A contains procedure B (or, conversely, procedure B is contained in procedure A).

The following rules must be observed when using nested procedures:

- A cataloged procedure cannot call itself.
- A procedure cannot call a procedure in which it is contained.
- Either all or none of the procedures in one nesting must have been cataloged with DATA=YES.
- A GOTO statement and the target label have to be on the same level, that is, within a single procedure or within the job submitted. This is also true for an ON statement referencing a label.
- An ON condition is checked on the nesting level on which it is specified and in any procedure called from this level or below in the same nesting. However, the search for the target label is only performed on the level on which the ON statement was specified.
- The definition of a symbolic parameter in an EXEC PROC statement takes effect only for the called procedure.
- If a parameter is to be passed to a called procedure and back, it must first be defined on the calling level.
- A nested procedure must not include a LIBDEF statement.

Refer also to the processing flow of nested procedures shown in Figure 31 on page 74.



```
// JOB NEST
// EXEC PROC= PROCA
// EXEC PROC= PROCD
/&
```

```
// PROC
    ...
// EXEC PGMA
// EXEC PROC= PROCB
/+
```

```
// PROC
    ...
// EXEC PGMB
// EXEC PROC= PROCC
/+
```

```
// PROC
    ...
// EXEC PGMC
/+
```

```
// PROC
    ...
// EXEC PGMD
/+
```

Figure 31. Processing Flow of Nested Procedures (4 Levels)

The following example uses nested procedures, symbolic parameters, and conditional job control to process files either on a monthly or daily basis.

Job Control Statements Submitted:

```
// JOB NESTED AND CONDITIONAL
ON $RC>=8 GOTO LIST
SETPARM UPDPRM='DAILY'
SETPARM RCSTRG=
// EXEC PROC=FILUPDT,SER=338004,USERLBL=YES,
        RCSTRG,UPDPRM
/. LIST
```

```
IF UPDPRM='MONTHLY' THEN
// UPSI 1
EXEC LISTSUM,PARM='&RCSTRG'
/&
```

Procedure Called (FILUPDT):

```
     // PROC DEV=3380,USERLBL=NO
     // ASSGN SYS004,TAPE,VOL=TAPEIN
(1)  // EXEC PROC=LABEL,FN=CHANGES,FID='INPUT.FILE',
             RP=',0',LU=SYS005,TRBL=95,SER,USERLBL
     // SETPARM TBNO=95
(1)  // EXEC PROC=LABEL,FN=DAYFILE,FID='DAY.MASTER.
                  FILE',
             RP=',5',LU=SYS006,TRBL=190,TBNO,SER,
             USERLBL
(1)  // EXEC PROC=LABEL,FN=MONFILE,RP=',40',
             FID='MONTH.MASTER.FILE',LU=SYS007,
             TRBL=285,TBNO,SER,USERLBL
     // EXEC PGM=INPUT
     ON $RC>12 GOTO $EOJ
     SETPARM RETC1=$RC
     ON $ABEND GOTO RECOVERY
     // EXEC UPDATE,PARM='&UPDPRM'
     SETPARM RETC2=$RC
(1)  SETPARM RCSTRG='&RETC1,&RETC2'
     ON $ABEND GOTO $EOJ
     IF RETC2>6 THEN
     GOTO RECOVERY
     GOTO EOP
     /. RECOVERY
     // EXEC RECOVER,PARM='&RCSTRG'
     /. EOP
     /+
```

(1)
In these statements the specifications for parameters FID, RP, and RCSTRG have to be within single quotes because the special characters period (.) and comma (,) are used.

Procedure Called (LABEL):

```
// PROC RP=,SER=,TBNO=38,
        SCT=,DEV=3380,TP=TEMP
IF USERLBL=NO THEN
GOTO ASGNONLY
// DLBL &FN,'&FID'&RP
// EXTENT &LU,&SER,1,0,&TRBL,&TBNO&SCT
/. ASGNONLY
// ASSGN &LU,&DEV,&TP,VOL=&SER,SHR;
/+
```

Sequence of Processed Statements:

```
     // JOB NESTED AND CONDITIONAL
(1)  ON $RC>=8 GOTO LIST
     SETPARM UPDPRM='DAILY'
     SETPARM RCSTRG=
(2)  // ASSGN SYS004,TAPE,VOL=TAPEIN
```

```
(3)  IF USERLBL=NO THEN
     GOTO ASGNONLY                            (skipped)
(4)  // DLBL CHANGES,'INPUT.FILE',0
(4)  // EXTENT SYS005,338004,1,0,95,38
     /. ASGNONLY
(5)  // ASSGN SYS005,3380,TEMP,VOL=338004,SHR
(6)  // SETPARM TBNO=95
(7)  IF USERLBL=NO THEN
     GOTO ASGNONLY                            (skipped)
(8)  // DLBL DAYFILE,'DAY.MASTER.FILE',5
(8)  // EXTENT SYS006,338004,1,0,190,95
     /. ASGNONLY
(9)  // ASSGN SYS006,3380,TEMP,VOL=338004,SHR
(10) IF USERLBL=NO THEN
     GOTO ASGNONLY                            (skipped)
(11) // DLBL MONFILE,'MONTH.MASTER.FILE',40
(11) // EXTENT SYS007,338004,1,0,285,95
```

```
     /. ASGNONLY
(12) // ASSGN SYS007,3380,TEMP,VOL=338004,SHR
(13) // EXEC PGM=INPUT
(14) ON $RC>12 GOTO $EOJ
(15) SETPARM RETC1=$RC
(16) ON $ABEND GOTO RECOVERY
(17) // EXEC UPDATE,PARM='&UPDPRM'
(18) SETPARM RETC2=$RC
(19) SETPARM RCSTRG='&RETC1,&RETC2'
(20) IF RETC2>6 THEN
     GOTO RECOVERY
(21) ON $ABEND GOTO $EOJ
     GOTO EOP
     /. RECOVERY
(22) // EXEC RECOVER,PARM='&RCSTRG'
     /. EOP
(23) /. LIST
(24) IF UPDPRM='MONTHLY' THEN
     // UPSI 1
(25) // EXEC LISTSUM,PARM='&RCSTRG'
     /&
```

(1)

Initial ON conditions and values are set. 'DAILY' indicates that this is a daily update run as opposed to a monthly update run. RCSTRG is assigned a null string.

(2)

This is the first statement of procedure FILUPDT. The parameter values passed with // EXEC PROC=FILUPDT cause the following: USERLBL=YES overrides the default USERLBL=NO specified in the // PROC statement. For RCSTRG a null string and for UPDPRM 'DAILY', as specified at the beginning of the job stream, is passed.

(3)

This is the first statement of procedure LABEL, which is called for the first time here. With statement // EXEC PROC=LABEL parameter values are specified directly. In addition, the specification of SER and USERLBL causes the values for these parameters (specified with // EXEC PROC=FILUPDT) to be passed to procedure LABEL.

Procedure LABEL is called three times: (3), (6), (9). All three calls are shown so that label ASGNONLY appears three times. During processing only the label of that procedure call currently being processed is in effect. If USERLBL=NO were in effect, statements // DLBL and // EXTENT would be skipped and previously defined labels used instead.

(4)

The parameter values are defined as follows:

For the // DLBL statement the values (filename, file-id, and retention period) are specified in the // EXEC PROC=LABEL statement. This is also true for the values of the // EXTENT statement, except for the volume serial number (338004) which is defined in the EXEC PROC=FILUPDT statement and passed via the // EXEC PROC=LABEL statement, and the number of tracks (38) for which the default value of the // PROC statement is in effect.

(5)

The parameter values are passed as follows:

The logical unit (SYS005) is specified in // EXEC PROC=LABEL statement. The volume serial number (338004) is specified in the // EXEC PROC=FILUPDT statement and passed by specifying SER in the // EXEC PROC=LABEL statement. For the device type (3380) and the temporary assignment (TEMP) the default values as specified in the // PROC statement are in effect.

(6)

The SETPARM statement defines the number of tracks to be in effect for the subsequent two calls of procedure LABEL.

(7)

Second call of procedure LABEL.

(8)

Parameter values are passed.

(9)
For parameter value passing refer to (5).

(10)
Third call of procedure LABEL.

(11)
Parameter values are passed.

(12)
For parameter value passing refer to (5).

(13)
Read input tape (to disk file CHANGES) and prepare input for update.

(14)
The job is terminated if a job step ends with a return code > 12.

(15)
The return code passed by program INPUT is assigned to RETC1.

(16)
Another ON condition is set for the following job steps.

(17)
The symbolic parameter &UPDPRM is replaced by the value 'DAILY' serving as processing parameter for program UPDATE.

(18)
The return code passed by program UPDATE is assigned to RETC2.

(19)
Both return codes (passed by programs INPUT and UPDATE) are assigned to RCSTRG which was set to zero at the beginning of the job stream.

(20)
Depending on the return code of program UPDATE, either a skip to label RECOVERY or to label EOP is performed.

(21)
The system default is reassigned to the $ABEND condition.

(22)
Both return codes serve as processing parameter for program RECOVERY.

(23)
Label /. EOP is the last statement of procedure FILUPDT; with label /. LIST the processing of the job stream originally submitted is resumed.

(24)
A program switch for program LISTSUM may be set via the // UPSI statement depending on a 'DAILY' or 'MONTHLY' run.

(25)
Program LISTSUM prints a summary of the monthly or daily master files and explains the return codes which were passed back from procedure FILUPDT and which are used as input for LISTSUM.

Chapter 4. Using VSE Libraries

Introducing the VSE Library Concept

Libraries can be considered as files which are under the control of the Librarian program. They need space on disk devices to be allocated with either:

- **DLBL** and **EXTENT** statements, if residing in non-VSAM space, or with
- **DLBL** and **IDCAMS** control statements, if residing in VSAM-managed space.

The allocated disk file must then be defined as a library by specifying the file name through a Librarian DEFINE command. Refer to "Defining a Library, Sublibrary, or a SYSRES File" on page 82 for job stream examples. At least one sublibrary must be defined within a library before any type of library member can be cataloged.

Libraries in VSE/VSAM space can also be defined through the *Define a Library* dialog of the Interactive Interface as described in z/VSE Administration.

It is advisable to add the **DLBL** and **EXTENT** statements for libraries to the system standard label area, if you define them with the Librarian **DEFINE** command. In this way, you avoid having to submit these statements and commands each time you need to define access to a library.

Libraries are accessed by the following programs:

- Language translators to retrieve source-type members for inclusion in the object module (OBJ) to be created.
- The Linkage Editor to retrieve members of type OBJ for processing and to catalog members of type PHASE newly created.
- The supervisor fetch/load routine to fetch or to load members of type PHASE into storage for execution.
- Job control to retrieve members of type PROC called in **EXEC PROC** statements.
- VSE/POWER, to retrieve SLI members.
- Dump programs to catalog members of type DUMP.
- The Info Analysis program to work with members of type DUMP.
- The Librarian program to perform library service functions.
- User application programs, using the Librarian application programming interface (API).

Normally, you need a **LIBDEF** job control statement for all the programs except the Librarian, to set up library access definitions. "Establishing a Library Access Definition" on page 85 discusses the usage of the LIBDEF and the related LIBDROP and LIBLIST statements.

MSHP, the VSE installation and service tool, accesses your system's libraries by calling the Librarian or the linkage editor. How you can access members that are under MSHP control is discussed under "Accessing Members Controlled by MSHP" on page 88.

Library Structure

A library always consists of one or more sublibraries. Programs, procedures and dumps are stored as members in the sublibraries. Sublibraries vary in size. They are dynamically extended as required until the space assigned to the library as a whole is exhausted.

If library space is freed because a member has been deleted, this space is automatically available for reuse.

Sublibrary members are identified by member name and member type.

The Librarian program addresses libraries by their names. Sublibraries are addressed by library and sublibrary name, for example:

```
LIB1.SUBN
```

If the Librarian is to address a member or members, an ACCESS or CONNECT command must be given first to specify in which sublibrary. For example:

```
ACCESS SUBLIBRARY=LIB1.SUBN
```

For other programs, the LIBDEF statement specifies in which sublibrary a member is to be searched for, or in which sublibrary it is to be stored. Sublibraries of one or more libraries may be chained (concatenated). Such a search chain is always related to a partition.

VSE Library Types

A z/VSE system includes two VSE library types: the system library and additional private libraries. There is no difference in their logical structure.

System Library

The system library (IJSYSRS) contains at least the predefined system sublibrary (SYSLIB). The system library is also referred to as SYSRES file. After your system has been installed, IJSYSRS.SYSLIB contains all system programs needed for system startup and for the operation of your system. The system library occupies one extent only and resides on the disk volume used for performing IPL. This volume is referred to as DOSRES.

An alternate IJSYSRS.SYSLIB resides on the system volume SYSWK1. This is intended for use with system functions such as fast service upgrade and unattended node support.

Private Libraries

User-written programs are usually stored in private libraries. It is preferable to have them on disk volumes of their own to reduce the disk arm movement on the SYSRES volume and to provide faster access to the programs in general.

In addition, private libraries offer more flexibility with regard to size and organization. The following functions are supported for private libraries:

- A private library may be maintained in space managed by VSE/VSAM. This allows a dynamic extension of the library.
- A private library may occupy several extents. These extents may be located on more than one disk volume.
- Besides being chained, libraries and their sublibraries may be shared among partitions and across systems. This reduces the effort for handling the libraries and increases system availability in general.

Types of Library Members

Most of the members stored in your libraries belong to one of the predefined member types shown below:

source
(A one-character, alphameric type): Source books (source code to be processed by a language translator)

OBJ
Object modules (language translator output)

PHASE
Phases (machine-readable code processed by the Linkage Editor and ready for execution)

PROC
Procedures (sets of job control statements) or REXX programs

DUMP
Dumps (data collected by the dump routines of the system)

You may also define member types of your own. How to do this is discussed later in this section.

The maximum member size that the system can handle is about 2^{31} records for the fixed record format and about 2^{31} bytes (2GB) for the undefined format. Usually, this is only of interest for members of type DUMP (when dumping a 2GB partition).

Source Books

Source books contain source code, either in the form of source statements or macro definitions, which are to be processed by a language translator. To specify the member type of a source book one of the following is allowed: A through Z, 0 through 9, #, $, or @. The letters A through I, P, R, and Z are reserved and used for system components.

The sublibrary (or sublibraries) in which the language translator is to search for source books is (are) specified in the job control statement:

```
// LIBDEF SOURCE,SEARCH=library.sublibrary...
```

This statement applies to all source member types.

When the assembler, for example, encounters a macro definition, the assembler searches for the appropriate macro in the sublibraries specified (and in the system sublibrary) and replaces the statement with the source code found.

Similarly, when a compiler encounters a reference to a source program, it searches for a SOURCE-type member of the same name in the sublibraries specified in the LIBDEF SOURCE,SEARCH statement.

Object Modules

Language translators process source code and produce output in the form of object modules. These modules need to be processed by the Linkage Editor to become executable phases. During the link-editing of a module other modules may have to be included. If so, the Linkage Editor searches the sublibraries specified for the modules in question. In this way, sections of code that are used by a number of different programs need be written, translated, and cataloged in object format only once.

Refer to “Cataloging Object Modules” on page 98 for sample job streams which assemble and catalog object modules.

Phases

Phases are programs or sections of code that have been processed by the Linkage Editor and are ready to be loaded into storage for execution. Phases are cataloged by the Linkage Editor as members of the type PHASE in the sublibrary specified in the job control statement:

```
// LIBDEF PHASE,CATALOG=library.sublibrary
```

Normally, the Linkage Editor builds phases in relocatable format. For a relocatable phase, the loader of VSE/Advanced Functions modifies the addresses as required when the phase is being loaded. Such a phase can be loaded for execution into the address area of any partition.

Programs that have been written as self-relocatable are linked and cataloged by the Linkage Editor as self-relocatable phases. Any address relocation to be done for a self-relocatable phase must be handled within that phase itself after it has been loaded.

When a program is to be run, the phase name is specified in the job control EXEC statement. The loader searches for a member of the type PHASE with this name in the sublibrary (or sublibraries) specified in the job control statement:

```
// LIBDEF PHASE,SEARCH=library.sublibrary...
```

and in the system sublibrary.

Procedures

Procedures are frequently used sets of job control statements (and sometimes data) in card image format. These sets of control statements are referred to as cataloged procedures when they are stored in a sublibrary. You must store procedures as members of the type PROC, using the Librarian commands:

```
ACCESS SUBLIB=library.sublibrary
   (to specify the sublibrary to be used), and
CATALOG name.PROC...
   (to name the procedure,)
```

followed by the job control statements (and data, if required) to be included in the procedure.

For details of the Librarian commands, refer to z/VSE System Control Statements.

A job stream being processed may call cataloged procedures for inclusion into that job stream.

Cataloged procedures may be modified while they are being processed. For details, refer to "Using Cataloged Procedures" on page 66.

A cataloged procedure may contain inline data, that is, data which is read by the associated program from the device that is used for reading the job control statements (SYSIPT). A procedure containing inline data must be cataloged with the operand DATA=YES in the CATALOG statement. Further details about inline data are provided under "SYSIPT Data in Cataloged Procedures" on page 66.

Dumps

Dumps contain system relevant data and are created by the system's dump routines, for example, when an abnormal program termination occurs. The data is stored in the default dump library SYSDUMP:

```
// LIBDEF DUMP,CATALOG=SYSDUMP.sublibrary
```

The stored dumps can be analyzed later for problem determination with the Information/Analysis program. One dump sublibrary is available for each **static** partition with the partition identifier used as the sublibrary name: SYSDUMP.BG, SYSDUMP.F1, SYSDUMP.F2, and so on. Only one sublibrary is reserved for **dynamic** partitions: SYSDUMP.DYN.

All these sublibraries should be used for dumps only.

User-Defined Member Types

Besides the predefined member types discussed above, you may store any type of data in your libraries and assign your own member type to it.

You can also change a predefined member type into a member type of your own definition, using the Librarian RENAME command.

This allows you, for example, to maintain several versions of a cataloged procedure. You can achieve this by assigning the same member name but different member types (of your choice) to the different versions. One version, however, must always have the predefined member type for procedures, namely PROC. This version can be considered as activated, and when the procedure is called, it is always this version that is chosen. You can easily activate another version of this cataloged procedure by changing its member type to PROC. The same principle applies when you have different versions of programs, whether source programs or phases.

A sublibrary may contain any or all member types used at an installation. This allows you to store, in one sublibrary, all members that are owned by one programmer, or that belong to one application.

Defining a Library, Sublibrary, or a SYSRES File

A private library can reside in VSAM-managed or non-VSAM-managed space.

Private Libraries in Non-VSAM-Managed Space

The physical location and the size of the library is defined via DLBL and EXTENT statements. For example:

```
// JOB CREATE LIB
// OPTION PARSTD=ADD
// DLBL YOURLIB,'VSE.PRIV.LIB',99/365,SD
// EXTENT ,000888,1,0,100,190
// EXEC LIBR
DEFINE LIB=YOURLIB
/*
/&
```

The minimum size of the extent is 1 track for CKD devices or 10 blocks for FBA devices. The maximum number of extents that can be specified for a private library is 16. If these extents are located on different volumes, the device type must be the same.

Private Libraries in VSAM-Managed Space

To define a private library in VSAM-managed space use the VSE/VSAM IDCAMS utility program to:

- Create the master and a user catalog
- Create VSE/VSAM space
- Define a VSE/VSAM cluster for your library

Instead of working with IDCAMS directly, you may use the *Define a Library* dialog for this task. Refer to *z/VSE Administration* for details.

When the primary space allocation is exhausted VSE/VSAM automatically extends the library by using the secondary allocation specified.

The maximum number of extents is 32 if EXTents=MAX32 is specified in the DEFINE Library command. The maximum size of a library in VSAM-managed space is therefore

```
psize + 31 * ssize
```

where psize is the primary allocation value and ssize the secondary allocation value. VSE/VSAM always attempts to get an allocation in one single extent. If this is not possible, multiple extents are used for each allocation, and the possible size of the library becomes less than defined above.

The actual size of a library in VSAM-managed space can be displayed with the Librarian command LISTDIR or the VSE/VSAM command LISTCAT.

A job for creating a cluster is shown below. A primary allocation of 10 cylinders and a secondary allocation of 2 cylinders is used. It is assumed that the label information for the master and user catalogs has been stored in the label information area.

```
// JOB CLUSTER
// EXEC IDCAMS,SIZE=AUTO
   DEFINE CLUSTER -
   (FILE (YOURLIB) -
   NAME (VSE.PRIV.LIB) -
   NONINDEXED -
   NOREUSE -
   SHAREOPTIONS (3) -
   RECORDFORMAT (NOCIFORMAT) -
   VOLUMES (000999 000777) -
   CYLINDERS(10 2)) -
   CATALOG (USER.CATALOG.NO1)
/*
/&
```

Note: The parameters NONINDEXED, NOREUSE, SHAREOPTIONS (3) and RECORDFORMAT (NOCIFORMAT) must be used.

After running the cluster job stream you can create a private library, in this example with the name YOURLIB:

```
// JOB CREATE VSAMLIB
// DLBL YOURLIB,'VSE.PRIV.LIB',,VSAM,DISP=(OLD,KEEP)
// EXEC LIBR
DEFINE LIB=YOURLIB
/*
/&
```

The parameter DISP=(OLD,KEEP) must be used. Only a DLBL but no EXTENT statement is required.

A library in VSAM-managed space is deleted by deleting the library and by deleting the cluster using the IDCAMS utility program. Secondary extents can be given back to VSE/VSAM by running the following jobs:

Backup library
Delete library
Delete cluster
Define cluster
Define library
Restore library

Defining Sublibraries

There is no difference in creating sublibraries in non-VSAM or VSAM-managed space, or in the system library IJSYSRS.

Note: For details about the locking function in connection with the DEFINE sublibrary command, refer to "Locking Rules" on page 114 and "Librarian Handling of IGNLOCK" on page 115.

To create three sublibraries named YSUB1, YSUB2, and YSUB3 in library YOURLIB, the following job stream is required. It is assumed that the label information is in the partition standard label area.

```
// JOB CREATE
// EXEC LIBR
DEFINE SUBLIB=YOURLIB.YSUB1,YOURLIB.YSUB2,YOURLIB.YSUB3
/*
/&
```

Defining Additional SYSRES Files

Refer also to "VSE Library Types" on page 80.

Besides your SYSRES file for IPL (identical with system library IJSYSRS), there might be a need for additional SYSRES files that allow you to have system libraries with a different setup available. To process these additional SYSRES files, you can use the names IJSYSR1 through IJSYSR9.

To perform IPL with such a file, you must create a system sublibrary that is named SYSLIB within the file to contain the system programs and possibly user programs required. You can then perform IPL with this file, for example IJSYSR4, as with your original SYSRES file. The system interprets the SYSRES file that is used for IPL always as IJSYSRS.

For a SYSRES file you can define only one extent, which must be in non-VSAM-managed space. The extent starts on a fixed location on disk.

For CKD devices the extent starts on cylinder 0, track 0, but as a start address you must specify track 1 in the EXTENT statement. The system library itself starts on track 8 and has a minimum size of one track. Thus the minimum number of tracks to be specified in the EXTENT statement is 8.

For FBA devices the extent starts on block 0 but as a start address you must specify block 2 in the EXTENT statement. The system library itself starts on block 4096 and has a minimum size of 10 blocks. Thus the minimum number of blocks to be specified in the EXTENT statement is 4104 (block 2 - 4105).

A job for a CKD device might look as follows:

```
// JOB CREATE SYSRES
// DLBL IJSYSR4,...
// EXTENT ,000666,1,0,1,200
// EXEC LIBR
DEFINE LIB=IJSYSR4
DEFINE SUBLIB=IJSYSR4.SYSLIB
/*
/&
```

The SYSRES file that is created contains the proper IPL records according to the device type (CKD or FBA). The Librarian retrieves this information from the SYSRES file that is presently used for IPL.

Establishing a Library Access Definition

A LIBDEF statement either specifies a sublibrary in which members of type PHASE or DUMP are to be stored, or establishes a search chain that indicates the sequence of sublibraries to be searched when retrieving members of any predefined type.

Cataloging Members of Type PHASE

The following statement specifies a sublibrary in which the output of the Linkage Editor is to be cataloged:

```
// LIBDEF PHASE,CATALOG=MYLIB.MSUB1
```

Instead of PHASE you may specify the character asterisk (*). In connection with the CATALOG parameter, * can mean only PHASE. See also examples for library chaining.

Note: There is no default sublibrary for cataloging phases. A link-edit job, for example, is canceled if the required sublibrary information for cataloging a phase has not been provided through a // LIBDEF statement.

Cataloging Members of Type DUMP

The following statements specify each a sublibrary in which members of type DUMP created by the system are to be stored:

```
// LIBDEF DUMP,CATALOG=SYSDUMP.BG    (dump sublibrary for BG partition)
// LIBDEF DUMP,CATALOG=SYSDUMP.F1    (dump sublibrary for F1 partition)
            .
            .                        (dump sublibrary for Fn partition)
            .
// LIBDEF DUMP,CATALOG=SYSDUMP.DYN   (dump sublibrary for dynamic partitions)
```

SYSDUMP is the default name of the system dump library. Partition identifiers should be used as sublibrary names as shown, except for the dynamic partition dump sublibrary. The Information/Analysis program, which helps you to interpret dumps, uses these sublibrary names by default.

Library Chaining

When searching for a member you can either specify a single sublibrary or a chain of sublibraries. A search chain allows you to search several sublibraries when retrieving members of the predefined types PHASE, OBJ, SOURCE and PROC. The sublibraries specified in a search chain may be part of one or more libraries. Also, a particular sublibrary may appear in the search chains of any or all partitions.

A search chain is established through a list of sublibrary names, qualified by library names corresponding to file names in DLBL statements. DLBL and EXTENT information and Librarian DEFINE commands must be entered before the LIBDEF statement is processed. Refer to the following example:

```
// DLBL MYLIB,...
// EXTENT ,111111,...
// DLBL YOURLIB,...
// EXTENT ,222222,...
    .
    .
// EXEC LIBR
DEFINE LIB=MYLIB YOURLIB
DEFINE SUBLIB=MYLIB.MYSUB1 YOURLIB.YSUB1
/*
    .
```

```
   .
// LIBDEF PHASE,SEARCH=(MYLIB.MSUB1,YOURLIB.YSUB1)
```

When requested to fetch or load a phase, the system searches first sublibraries MYLIB.MSUB1 and YOURLIB.YSUB1 and finally the system sublibrary IJSYSRS.SYSLIB. The system always adds IJSYSRS.SYSLIB at the end of a search chain, unless you specify it explicitly somewhere else in the chain.

You can also establish a single search chain for all member types (PHASE, OBJ, SOURCE and PROC.) For this purpose, use the character * as member type specification. For example:

```
// LIBDEF *,SEARCH=(MYLIB.MSUB1,YOURLIB.YSUB1,OURLIB.OSUB1)
```

Again, if a member is not found in the sublibraries defined, IJSYSRS.SYSLIB is searched as well.

Permanent versus Temporary Library Access Definitions

A library access definition that is specified as TEMP is valid only for the job in which it is submitted. TEMP is also the default. A permanent library access definition that is specified as PERM is valid for the partition in which it is submitted until:

- that partition is deactivated via the UNBATCH command, or
- a new permanent LIBDEF statement overrides the existing one, or
- a LIBDROP statement is entered. In this case, the default search chain becomes active.

For example:

```
// LIBDEF OBJ,SEARCH=(YOURLIB.YSUB,OURLIB.OSUB1),PERM

would be overridden by;

// LIBDEF OBJ,SEARCH=(HISLIB.HSUB,HERLIB.HSUB1),PERM

The default search chain for OBJ-type members would be activated by:

// LIBDROP OBJ,SEARCH,PERM
```

If both temporary and permanent definitions exist, the following rules apply:

- For SEARCH, the TEMP search chain is logically placed before the PERM search chain, and both are searched until the member is found.
- For CATALOG, the TEMP library definition is used.

In any search chain, you can specify up to 32 sublibraries. For each partition, you can specify a permanent chain of up to 32 sublibraries and a temporary chain (per job) of up to 32 sublibraries. If you specify SDL and/or IJSYSRS.SYSLIB explicitly in a chain, this number is reduced to 30 or 31.

You can cancel a library access definition explicitly at any time by using the LIBDROP statement. This might be necessary, for example, if you want to delete a sublibrary from partition F1 when it is specified in a search chain in partition F2. In this case, you would enter in partition F2:

```
// LIBDROP *,SEARCH
```

The Search Sequence for Phases

A search chain for member type PHASE always includes the system directory list (SDL). The search sequence is different for $- and non-$ phases:

- non-$ phases: SDL → TEMP-chain → PERM-chain → IJSYSRS.SYSLIB
- $ phases: SDL → IJSYSRS.SYSLIB → TEMP-chain → PERM-chain

For further details, refer to the description of the LIBDEF statement in z/VSE System Control Statements.

The example shows a Linkage Editor job stream. Permanent and temporary library access definitions are used. It is assumed that DLBL, EXTENT, and DEFINE statements have been given for the specified libraries and sublibraries.

Permanent definitions in partition F1

```
LIBDEF PHASE,CATALOG=YOURLIB.YSUB1,PERM
LIBDEF OBJ,SEARCH=(MYLIB.MSUB1,MYLIB.MSUB2),PERM
```

Job stream submitted in partition F1

```
// JOB LINKEDIT
LIBDEF PHASE,CATALOG=HERLIB.HERSUB1,TEMP
LIBDEF OBJ,SEARCH=HISLIB.HISSUB1,TEMP
// OPTION CATAL
   PHASE....
   INCLUDE ....
// EXEC LNKEDT
/&
```

In this job stream, the temporary LIBDEF for cataloging overrides the permanent one, that is, the link-edited phase is cataloged in sublibrary HERSUB1.

The search chain for the object modules to be link-edited is extended by the temporary definition, which is put in front of the permanent definition. Thus, the search chain that is used looks as follows:

```
HISLIB.HISSUB1,MYLIB.MYSUB1,MYLIB.MYSUB2,IJSYSRS.SYSLIB
```

The system sublibrary is added by default. If your installation includes access control protected libraries or sublibraries that are to be included in search chains the following restrictions must be observed:

- Private libraries and sublibraries must be protected with a universal access right (UACC) if you want them included in a permanent search chain. In addition, they can be protected on an individual user basis (ACC).
- Private sublibraries that are to be included in a temporary chain can be protected with UACC or ACC or both.

All access rights are allowed.

Resetting a Library Access Definition

The LIBDROP statement entered in any partition cancels a library access definition that was established previously for the same partition by a LIBDEF statement. For example:

```
// LIBDEF OBJ,SEARCH=(YOURLIB.YSUB1,OURLIB.OSUB1),-
              CATALOG=OURLIB.OSUB2,PERM
```

is reset by

```
// LIBDROP OBJ,PERM
```

A library access definition is also reset when a new permanent LIBDEF statement overrides the existing definition.

If not reset explicitly, all temporary library definitions will be reset at end-of-job. A permanent library access definition is automatically canceled when the partition is deactivated using the UNBATCH command. If a HOLD command is given before UNBATCH, the permanent library definitions are not deactivated and are available again when the partition is restarted. For details on the commands refer to z/VSE System Control Statements.

Displaying Library Access Definitions

Using the LIBLIST statement, you can request a display of the currently active library access definitions for a particular member type. The display may show one or all static partitions. You can direct the display to SYSLOG or to SYSLST. For example:

```
// LIBLIST PHASE
```

causes the display of all permanent and temporary library definitions of type PHASE for the partitions in which the statement is executed.

You may display with a single LIBLIST statement all library access definitions, temporary and permanent, for all member types and for all static partitions. For example:

```
// LIBLIST *,*,SYSLST
```

In this example, the output is printed on SYSLST.

Accessing Members Controlled by MSHP

If a sublibrary member is cataloged under MSHP (Maintain System History Program) control, it remains under control of MSHP. This is also true for an MSHP-controlled member that you copy (by COPY, MOVE, or BACKUP/RESTORE) into another sublibrary.

Normally, a MSHP controlled member cannot be replaced, changed, locked, or unlocked by the Librarian program outside the control of MSHP. However, to cope with an emergency situation (failure of MSHP, for example), an MSHP bypass is available. To use this bypass, code the EXEC statement for the Librarian as follows:

```
  // EXEC LIBR,PARM='MSHP'
```

You may use this bypass to copy an MSHP-controlled member, or to replace an MSHP-controlled member in the target sublibrary. To manipulate the copied member in the target sublibrary, you must again specify *PARM='MSHP'* in your *// EXEC LIBR* statement.

The Librarian Program

The Librarian program provides all the functions needed to service your libraries. The Librarian is called using an // EXEC LIBR statement followed by one or more Librarian commands. The end of the command stream is indicated by a /* delimiter.

The Librarian program can run in any partition. To cancel the Librarian, cancel the partition in which it is running, as with any other program.

A CANCEL command (without the FORCE parameter) for the Librarian's partition may not take effect immediately. This is because of the "delayed cancel" function, which causes the Librarian to complete execution of the current command before it is terminated.

Return Codes

After each Librarian command is executed, the Librarian sets a return code. This indicates the success of the command,as follows:

RC:
Indicates that:

0
the command was completed successfully.

2
the command was completed successfully, and a particular result was reached (for example, TEST and COMPARE commands).

4

the command was completed, but an exceptional condition occurred, or the requested result already existed.

8

the command was only partly executed, but the Librarian program could continue processing.

16

a severe error occurred while processing the command. The Librarian program terminates.

The highest return code set during the Librarian session is passed to the job control program. Here, it can be tested by an IF or ON job control command.

Conditional Execution

The Librarian offers the possibility of processing or omitting certain commands in the command stream, dependent on the return codes set after each command. The ON command compares the return code with a specified value, and causes a branch to a label command if the result of the comparison is "true".

The /. LABEL command indicates a point in the command stream to which the GOTO action of an ON command may refer.

The GOTO command is used to skip unconditionally to a specified label statement.

For details on these commands, see z/VSE System Control Statements.

Examples of Conditional Command Execution

Depending on the return code set, you may want to break off the Librarian session, or have different sets of commands executed, depending on the result of a particular command.

For example, if a COMPARE command shows that two phases are identical, you may want to erase one of them. If they are different, they should both be kept. The COMPARE command issues a return code of 0 if the compared members are identical, and a code of 2 if they are different. You could use the following commands. (The shortest abbreviations accepted by the Librarian are used.)

```
// EXEC LIBR
ON $RC = 2 GOTO END
CON S=LIB1.SUB1 : LIB1.SUB2
COM PROGA.PHASE
A S=LIB1.SUB2
DEL PROGA.PHASE
/. END
/*
```

As another example, if a generic copy of all members from a sublibrary is only partly executed, a return code of 8 is set. In this case, it is advisable to list the directory of the "to" sublibrary to see which members have been copied and which have not.

The following command stream could be used:

```
// EXEC LIBR
ON $RC = 8 GOTO LIST1
CON S=LIB1.SUB1 : LIB1.SUB2
COP *.* R=YES
GOTO NEXT
/. LIST1
ON $RC = 8 GOTO LIST2
LISTD S=LIB1.SUB2
/. NEXT
CON S=LIB2.SUB3 : LIB2.SUB4
COP *.* R=YES
GOTO END
/. LIST2
LISTD S=LIB2.SUB4
/. END
/*
```

Interactive Execution

The Librarian can also be used interactively via SYSLOG. After the EXEC LIBR command is processed, the Librarian prompts the operator to enter Librarian commands. The end of a command sequence is indicated on SYSLOG by entering END. The ON, GOTO and /. LABEL commands are not accepted from SYSLOG.

When the Librarian is called from SYSLOG, it directs all list output (from LIST and LISTD commands) and all messages to SYSLOG by default. When you expect a lot of output, it is advisable to specify SYSLST in the UNIT operand of these commands. If, for any reason, you want to stop a listing, enter a CANCEL command for the partition in which the Librarian is running. This will stop the listing, but the Librarian itself will remain active in the partition.

For details on how to handle and analyze members of type DUMP refer to z/VSE Guide for Solving Problems.

Accessing Sublibraries

When using the Librarian you have to define the library or sublibrary to be accessed in one of the following ways:

- As an operand of a Librarian command.
- By using the ACCESS or CONNECT command.

Examples are given when discussing the various Librarian functions.

Generic Notation

For certain functions that handle library members you can specify member names and member types in generic notation by using the character '*'. For example:

ABC*.OBJ
is interpreted as all members of type OBJ whose names begin with ABC.

***.OBJ**
is interpreted as all members of type OBJ.

.
is interpreted as all members of all types.

The following Librarian commands allow generic notation:

BACKUP
COMPARE
COPY
DELETE
LIST
LISTDIR
MOVE
PUNCH
RENAME
RESTORE
SEARCH
TEST
UNLOCK

To separate the operands of a command you may use either blanks or commas. If a command does not fit on one input line, or if you want a certain operand on a line by itself for easier editing or update, use a hyphen (-) as a continuation character. For example:

```
DEFINE LIB=MYLIB-
  YOURLIB-
  PRODLIB
```

Librarian Commands

This section describes the Librarian commands as listed below. It provides examples to show their usage.

BACKUP
CATALOG
CHANGE
COMPARE
COPY
DEFINE
DELETE
INPUT (PUNCH)
LIST
LISTDIR
LOCK
MOVE (Merge)
PUNCH
RELEASE
RENAME
RESTORE
SEARCH
TEST
UNLOCK
UPDATE

For a complete description and the syntax of each Librarian command, refer to z/VSE System Control Statements.

Backup a SYSRES File, Library, Sublibrary, or Member

By using the BACKUP command you can backup libraries, sublibraries, members, and SYSRES files onto tape. Libraries, sublibraries and members remain unchanged after restore, apart from an internal re-organization which removes scattered free space and usually results in faster read access.

Note: The SYSRES file is identical to the system library IJSYSRS. IJSYSR1 through IJSYSR9 also define SYSRES files.

SYSRES files can be backed up either for an online or a stand-alone restore run, but private libraries, sublibraries, and members for an online restore run only. RESTORE=ONLINE is the default.

The backup command accepts unlabeled tapes or tapes with standard labels.

You can also use backup/restore runs to recover partially damaged libraries and sublibraries. Only the intact members are backed up and can then be restored.

Note: If a damaged member is too large for the input buffers of the backup routine, it is not skipped. In this case, the backup run is canceled by the system. The size of the input buffer depends on the size of the partition in which the Librarian is running, and may be up to 64K. Damaged members larger than 64K should be deleted or renamed before backup.

To backup a library or sublibrary to a disk device, you can use the COPY command. The COPY function also includes an internal reorganization.

Creating a Backup Tape for a Stand-Alone Restore

If RESTORE=STANDALONE is specified, the stand-alone programs and utilities required for stand-alone restore are retrieved from the system sublibrary (IJSYSRS.SYSLIB) of the first or only SYSRES file specified and included in the backup tape. IJSYSR1 through IJSYSR9 define SYSRES files, IJSYSRS your IPLed system. If no SYSRES file has been specified, the stand-alone programs and utilities are retrieved from the SYSRES from which IPL was performed.

An exception to this rule is the stand-alone DITTO/ESA for VSE program. This program is retrieved from sublibrary PRD1.BASE if not installed in SYSRES as the other stand-alone programs.

The specification of a library, sublibrary, or member will be optional for the case that the operand RESTORE=STANDALONE is specified. Thus, if no library, sublibrary, or member operand is present on the BACKUP command, only the stand-alone programs are written onto the output tape and the tape is positioned behind the stand-alone utilities file.

A stand-alone backup tape can be restored online. Only SYSRES files can be restored stand-alone. If a backup tape contains a SYSRES file for stand-alone restore and in addition private libraries or sublibraries, these private libraries or sublibraries must be restored online.

Note that a backup tape for a stand-alone restore need not include a SYSRES file; the SYSRES file can be on another backup tape.

Further Considerations for Creating Backup Tapes

A history file may also be part of a backup tape. However, a history file cannot be restored with the Librarian restore function. An MSHP INSTALL or RESTORE job is required for this. The INSTALL function can restore both history files and libraries.

For a backup run you can use unlabeled tapes or tapes with standard labels. If the TAPELABEL=filename operand is not specified for the BACKUP command, the Librarian program assumes an unlabeled tape. If TAPELABEL is specified, the "filename" must be identical to the seven character file name of the // TLBL statement of the output tape.

The layout of the output created by a single BACKUP command is described below. Each BACKUP command creates the same set of files. The files are separated by tapemarks.

A single BACKUP command may be submitted with a list of libraries, sublibraries, or members. The resulting backup file would contain all libraries, sublibraries, or members, as specified. The sequence in which the libraries, sublibraries and members are restored depends on their position in the backup file, not on the sequence in which they are specified in the RESTORE command.

If the backup files created by several BACKUP commands are on one tape, the RESTORE commands must reflect the sequence of the backup files.

The output from several BACKUP commands can be written to one tape. In this case, it is advisable for unlabeled tapes to use the ID=name operand of the BACKUP command. This makes the restore of the required data easier. If the ID operand is not used, the backup tape must be repositioned "manually", using MTC commands. With an ID operand, all you have to do at restore is to specify the appropriate name, and the Librarian finds the correct backup file automatically.

Note: The backup tape is not rewound automatically before or after a BACKUP command is executed.

However, if RESTORE=STANDALONE is specified in the BACKUP command, the Librarian rewinds the output tape **before** output begins. This is because the stand-alone programs must be at the beginning of the tape.

For details about tape positioning refer to "Tape Positioning" on page 95.

Layout of an Online Tape

An unlabeled online tape created with a single BACKUP command (RESTORE=ONLINE) contains the files described below. A labeled tape includes in addition a header and a trailer; refer also to Figure 32 on page 96.

File 1: This file is ignored for restore. It may be empty or contain user information. To be included on the backup tape, this information must be stored in a library member in card image format, that is, the record length must be 80 bytes. The library member is to be specified in the BACKUP command (with the HEADER= operand).

File 2: File 2 consists of a backup file identification and, optionally, the system history File.

The file identification simplifies later restoring of an unlabeled tape if more than one backup file is written on the tape. The identification must be alphanumeric and can be 1 to 16 characters long. If a backup of the system history file is requested it is also part of this file.

File 3 (Backup file): Contains the libraries, sublibraries, or members requested for backup.

If the space of a single tape volume is not sufficient, a multivolume backup tape is created; alternate tape drive assignment is supported.

Layout of a Stand-Alone Tape

The output produced by the Librarian BACKUP command with the RESTORE=STANDALONE parameter is a single-volume or multi-volume file consisting of 4 files separated by tapemarks. The first 2 files on the stand-alone tape are the *stand-alone IPL file* and the *stand-alone utility file*. These 2 files must reside completely on the first tape volume. The remaining 2 files are optional.

A labeled tape includes in addition a header and a trailer.

File 1: Contains a header file and stand-alone programs. The header file is optional and and may be used by IBM for system information. If you specify your own header file, it must contain IPL bootstrap records for tape. The stand-alone programs include:

- IPL bootstrap phases.
- The console support phase.
- The z/VSE supervisor.
- Further IPL programs.
- A load list of the SVA programs needed in the stand-alone environment.

File 2: Contains the *stand-alone utility file* including (in the order as specified in the SVA load list: $SVASA) the SVA programs required in the stand-alone environment. This file includes:

- The stand-alone RESTORE program (LIBSARE).
- The VSE/Fast Copy program (FCOPY).
- The ICKDSF (Device Support Facilities) program.
- The DITTO/ESA for VSE program.

The presence of these programs is optional, but the file must include at least one of them.

File 3: As file 2 of online tape.

File 4: As file 3 of online tape.

The output from a BACKUP command with RESTORE=STANDALONE must be at the start of a tape. The output of any following BACKUP commands with RESTORE=ONLINE may be written to the same tape. The unlabeled tape will then contain 4 files of a stand-alone backup, 3 files of an online backup, another 3 files of an online backup, and so on.

At the end of the backup tape, the Librarian writes a null file, a file containing an end-of-backup record, and another null file.

Note: If you create a labeled tape for RESTORE=STANDALONE, the program issues message L127I. The message warns you that standard header labels exist and must be skipped for initial program load (IPL) of the tape (they are skipped by repetitive IPLs until the restore program finds the first record of the load program).

You can avoid message L127I if you create a separate IPL tape containing only the stand-alone programs. To do this, perform a backup run with an unlabeled tape for stand-alone restore (`BACKUP ... RESTORE=STANDALONE ...`) and do not specify any library, sublibrary, or member.

You can use this tape later on for initial program load (IPL) of the restore program and have the program read a separate backup tape with a SYSRES file as input for the stand-alone restore run.

Examples of BACKUP Jobs

Example 1

The following job performs a backup of SYSRES file IJSYSR1, library YOURLIB, and sublibrary MYLIB.MSUB1, each preceded by the backup file-ID specified. The default RESTORE=ONLINE is in effect:

```
// JOB BACKUP
// EXEC LIBR
BACKUP LIB=IJSYSR1,TAPE=181,ID=BA1
BACKUP LIB=YOURLIB,TAPE=181,ID=BA3
BACKUP SUBLIB=MYLIB.MSUB1,TAPE=181,ID=BA4
/*
/&
```

For the corresponding restore job stream refer to "Restore a SYSRES file, Library, or Sublibrary" later in this section.

Example 2

Assume that you want to backup the IPLed SYSRES file, the history file, and restore the backup tape stand-alone. This would require the following job stream (for an unlabeled tape):

```
// JOB BACKUP
// EXEC LIBR
BACKUP LIB=IJSYSRS,TAPE=181,RESTORE=STANDALONE, -
           INCLUDE=HISTORY
/*
/&
```

The Librarian retrieves the programs and data that go into file 1 and 2 of the backup tape from IJSYSRS.SYSLIB; DITTO/ESA for VSE from PRD1.BASE if not found in IJSYSRS.SYSLIB.

This is followed by the history file which is stored as part of file 3 on the backup tape.

Finally, the system library IJSYSRS, the actual backup file (file 4), is copied to the backup tape.

Example 3

The following job stream creates a backup copy of library MYLIB on a labeled tape. TL1 is the file name assumed:

```
// JOB BACKUP
// TLBL TL1, ...
// MTC REW,181
// EXEC LIBR
BACKUP LIB=MYLIB TAPE=181 TAPELABEL=TL1
/*
/&
```

Note: The ID operand is not used in this example. The ID operand is mainly used for unlabeled tapes; for labeled tapes only if an additional check is to be performed.

Example 4

The following job stream creates backup copies of selected library members:

```
// JOB BACKUP
// EXEC LIBR
```

```
BACKUP MYLIB.MSUB1.M*.OBJ-
       MYLIB.MSUB1.M*.S* TAPE=181 ID=MB1
BACKUP MYLIB.MSUB1.MYPHASE.PHASE-
       TAPE=181 ID=MB2
/*
/&
```

The first BACKUP statement selects members for backup through generic member specification. The second BACKUP statement selects a single member for backup.

Multiple Backups on One Tape

When the last BACKUP command addressing a particular tape has been executed, the Librarian writes End-of-Backup-Tape records for both, labeled and unlabeled tapes. When these are reached during a restore run, the RESTORE function stops searching for the specified backup file-ID.

If the remainder of the same tape is to be used for further backups, these End-of-Backup-Tape records must be overwritten. This can be done for unlabeled tapes as follows:

- Mount the partly-used backup tape.
- Issue a RESTORE command, specifying this tape and a backup file ID which does not exist on the tape:

  ```
  RES * TAPE=cuu SCAN=YES ID=XXXX
  ```

  (SCAN=YES prevents a restore taking place if the specified ID does exist after all).
- The Librarian searches the tape for the specified file-ID, does not find it, and positions the tape behind the last existing file.
- Issue the desired BACKUP command or commands.

Tape Positioning

To a certain extent, tape positioning is done automatically. For a selective restore, additional tape positioning control on your part might be necessary, if you have labeled tapes and multiple backups on a single tape.

Positioning for an Online Tape

No tape rewind is done at the beginning or end of a BACKUP or RESTORE run.

- Positioning after BACKUP

  An unlabeled online tape is positioned behind the backup file, a labeled online tape behind the trailer label.

- Positioning after RESTORE

  After a non-selective restore, the tape is positioned immediately behind the trailer label (labeled tape). For an unlabeled tape or after a *selective restore*, the tape is positioned within the currently processed backup file, immediately behind the restored object.

Positioning for a Stand-Alone Tape

For the BACKUP run, a tape rewind is done at the beginning.

For the RESTORE run, a tape rewind is done at the beginning if either of the following is true:

1. The initial program-load address is different from the address of the tape.
2. The backup tape is labeled and the labels are to be processed.

Further positioning characteristics:

- Positioning after BACKUP

An unlabeled stand-alone tape is positioned behind the backup file, a labeled stand-alone tape behind the trailer label.

Note: If no history file and no backup file are written to tape, a stand-alone tape is positioned behind the *stand-alone utility file*. In this case, no trailer label is written for a labeled output tape. Thus, on completion of the Librarian BACKUP command, a VSE/Fast Copy dump may be written to the same tape.

- Positioning after stand-alone IPL

  On completion of IPL, the stand-alone tape is positioned behind the *stand-alone utility file* for further processing by a stand-alone Librarian RESTORE or stand-alone VSE/Fast Copy.

- Positioning after RESTORE

  A stand-alone Librarian RESTORE positions the tape, (except for a *selective restore*) behind the restored file (unlabeled tape) or behind the trailer label (labeled tape).

  After a *selective restore*, that is, if a backup file contains several IJSYSRx libraries and the last one of these libraries is not restored, the tape remains positioned behind the IJSYSRx that has been restored.

User-Controlled Positioning

You may have a need for this if you do a selective restore; for example, of one or more library objects out of a number of such objects stored on your labeled backup tape, or of a complete backup file if two or more such files are stored on the tape.

To properly control the positioning of a labeled tape, you should know how your backup data is stored on the tape. This is shown for an online tape in Figure 32 on page 96 for one backup file. Figure 32 on page 96 shows, in addition, where the tape is positioned at the end of a selective restore.

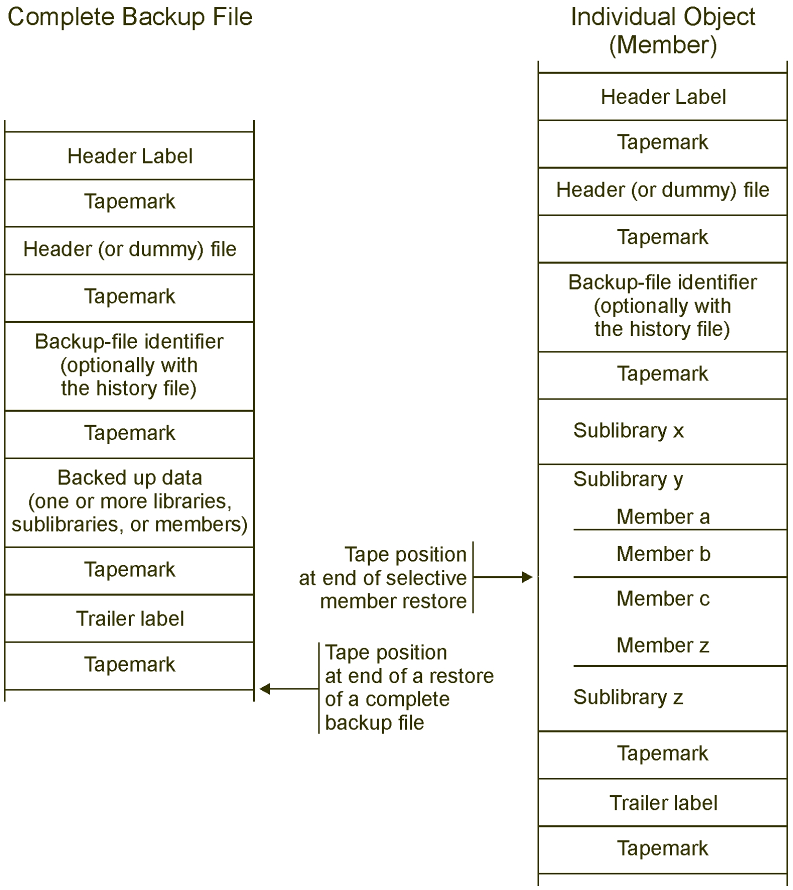

Complete Backup File
Header Label
Tapemark
Header (or dummy) file
Tapemark
Backup-file identifier (optionally with the history file)
Tapemark
Backed up data (one or more libraries, sublibraries, or members)
Tapemark
Trailer label
Tapemark
Tape position at end of selective member restore
Tape position at end of a restore of a complete backup file
Individual Object (Member)
Header Label
Tapemark
Header (or dummy) file
Tapemark
Backup-file identifier (optionally with the history file)
Tapemark
Sublibrary x
Sublibrary y
Member a
Member b
Member c
Member z
Sublibrary z
Tapemark
Trailer label
Tapemark

Figure 32. Tape Positioning at End of a Selective Restore for a Labeled Online Tape

Catalog a Member

With this function you can catalog members of any type (except of the types DUMP and PHASE), or of a predefined source-type and of any user-defined member type.

Note:

It is not possible to replace locked members through cataloging. You must unlock the member first (UNLOCK command) before you can replace it through a CATALOG command. Refer also to "Librarian Handling of IGNLOCK" on page 115.

A time stamp is used to indicate for each member when it was cataloged for the first time and also when it was replaced last. The time stamp information can be displayed with the LISTDIR command.

A member of type PHASE is cataloged by the Linkage Editor. For cataloging phases refer to the description of the Linkage Editor.

Cataloging Source Books

As member type for source books the following single characters are allowed: A through Z, 0 - 9, #, $, and @.

The following letters are reserved for IBM programs:

A --
is the member type for assembler source code and source macro definitions.

B --
is the member type for network definition source code for VTAM.

C --
is the member type for COBOL source code.

D --
is the member type for alternate assembler copy source code. It contains non-edited macros and copy books for programs that are to be executed in a telecommunications network control unit.

E --
is the member type for assembler macros. These can be IBM-supplied or user-written macro definitions in an edited (partially processed) format (also referred to as E-Deck).

Since the DOS/VSE Assembler has been replaced by the High Level Assembler, you can no longer create E-Decks. However, you can process existing E-Decks with the High Level Assembler (see also note below).

F --
is the member type for alternate assembler macros. IBM uses it to distribute edited macros for use by programs that are to be executed in a telecommunications network control unit.

P --
is the member type for PL/I source code.

R --
is the member type for RPG II source code.

Z --
is the member type for sample programs that are supplied by IBM.

Note: For member types A and E further details are provided under "Processing Macros with the ESERV Program" on page 138 and "Using the High Level Assembler Library Exit for Processing E-Decks" on page 139.

The remaining reserved characters (G, H, I) are used by IBM for future additions. You should avoid, wherever possible, using one of the reserved member types. If you must use such a member type, ensure that you do not use duplicate names.

Assume that you want to catalog source code into sublibrary YSUB1, which is part of library YOURLIB. The member name is YBOOK1. The member type has been defined as L. An existing member of the same name and type is to be deleted. The job stream then looks as follows:

```
     // JOB CATSOURCE
     // EXEC LIBR
(1)  ACCESS SUBL=YOURLIB.YSUB1
(2)  CATALOG YBOOK1.L REPLACE=YES
        ...
        ...
        source statements
        ...
        ...
     /+
     /*
     /&
```

(1)
The ACCESS command defines the sublibrary to be accessed.

(2)
The CATALOG statement specifies member name and member type, and that the old version of that member (same member name, same member type), is to be replaced.

(3)
/+ indicates the end of the source code.

Instead of the EOD specification BKEND statements can be used. Edited macro definitions that are to be cataloged with member type E can be preceded by a MACRO statement and followed by a MEND statement. The /+ delimiter is not needed when BKEND or MEND is used.

Cataloging Object Modules

To catalog an object module into a sublibrary (as input for the Linkage Editor) you must submit the object modules on SYSIPT immediately behind the CATALOG command, if the object modules are available as card deck. The following job catalogs two object modules, named YMOD13, and YMOD14 into sublibrary YOURLIB.YSUB2.

```
     // JOB CATOBJ
     // EXEC LIBR
(1)  ACCESS SUBLIB=YOURLIB.YSUB2
(2)  CATALOG YMOD13.OBJ
        ...
        ...
        object module YMOD13
        ...
        ...
(3)  /+
(2)  CATALOG YMOD14.OBJ
        ...
        ...
        object module YMOD14
        ...
        ...
(3)  /+
     /*
     /&
```

(1)
The ACCESS command defines the sublibrary in which the modules are to be stored.

(2)
The CATALOG command defines the names under which the object modules are to be stored in the sublibrary.

(3)
The EOD indication can, but need not, be specified for the module since the end is indicated by an END statement that is generated by the compiler.

You can compile or assemble a program and catalog the resulting object module in a sublibrary in the same job stream. For that purpose, you assign SYSPCH, which receives the output of the language translator, to a disk or tape and then use the object module on that device as input for your catalog run. An example using a magnetic tape for SYSPCH is shown below. To assign SYSPCH to a disk, you must in addition supply the necessary DLBL and EXTENT job control statements and the CLOSE job control command, because // RESET SYSPCH does not work on disk devices.

```
     // JOB SOURCE
     // OPTION DECK
(1)  // ASSGN SYSPCH,380
(2)  // MTC REW,SYSPCH
     // EXEC ASMA90....
(3)     PUNCH 'CATALOG YMOD13.OBJ REPLACE=YES'
        ...
        ...
(4)     source statements
        ...
        ...
     /*
(5)  // MTC WTM,SYSPCH,2
(6)  // MTC REW,SYSPCH
(7)  // RESET SYSPCH
(8)  // ASSGN SYSIPT,380
(9)  // EXEC LIBR,PARM='ACCESS SUBLIB=YOURLIB.YSUB2'
     /&
```

Note: The statement

```
// EXEC ASMA90....
```

calls the High Level Assembler. Refer to "High Level Assembler Considerations" on page 139 for further details.

(1)
A magnetic tape device is assigned to SYSPCH to receive the assembler output.

(2)
Rewinds the tape to its load point.

(3)
Causes the assembler to write a CATALOG statement in the requested format on SYSPCH in front of the object module.

(4)
The assembler processes the source statements that are submitted and writes the object module on SYSPCH.

(5)
Writes tape-marks on SYSPCH to indicate the end of the object module.

(6)
Rewinds the tape to its load point.

(7)
The tape is unassigned as SYSPCH.

(8)
The tape is assigned as SYSIPT to serve as input for the Librarian.

(9)
The Librarian catalogs the object module using SYSIPT as input. The Librarian reads as the first record the CATALOG command created by the assembler. The remaining information that is needed by the Librarian, the sublibrary name, is provided in the EXEC statement. Since the Librarian reads its input from SYSIPT, an ACCESS command (which is read from SYSRDR) cannot be used to specify the sublibrary name.

Cataloging Multiple Object Modules

You can catalog several object modules as one library member (a multiple-object file) into a sublibrary. The following job stream catalogs a multiple-object file as a single library member:

```
    * $$ JNM=CATALOG,CLASS=0
    // JOB CATALOG EXAMPLE
    // EXEC LIBR,PARM='MSHP'
    ACCESS S=TESTLIB.S1
    CATALOG EXAMPLE.OBJ REP=YES EOD=/+
     ESD            INLPCAT         INLPQNAM        INLPMSG
     ESD            INLPREAD        INLPCACK        INLPEXIT
     ESD            INLPGST         INLPGSTI        INLCCOMR
     TXT            å00  INLPCAT C550II412711637°Ö}   {     × &};
     TXT   ½          ¤   ¤Ú"
     TXT               µ
     RLD                         U     Y     Ö     0
     RLD                 Ü     \
(1)  END
     ESD            SVASL          ½
     TXT            °Ö}  { œ   }  ì   ì     K   A¡&     {&&   oØ
     TXT             qx%ì0 {ì01}ì00  Õ Lì\} q }   ÚASSEMBLE
     TXT            LIBDEF
(2)  END
(3) /+
    /*
    /&
    * $$* EOJ
```

(1)
End of first module

(2)
End of second module

(3)
End of multiple-object file

As shown in the job stream, you need one Librarian CATALOG command to catalog several object modules. This command recognizes the end of an object module by reading the END statement. The END statement must be the last statement in a single object module. The END statement processing recognizes that the "end of catalog" condition is true only, if the card following the END is not a valid object statement. Valid object cards are records starting with X'02' in the first column (ESD, TXT, RLD, and END).

Cataloging Procedures

A cataloged procedure can contain control statements and data.

The end of the control statements to be cataloged must be indicated by an end-of-data (EOD) delimiter.

The default value for the EOD delimiter is /+. If the procedure to be cataloged contains this combination, an alternate EOD delimiter must be defined to the Librarian program. This is defined in the EOD operand of the CATALOG command. The EOD delimiter must not contain commas, blanks, or /*, and must be specified in columns 1 and 2 of the input line following the last member record. Only the EOD characters indicate the end of the member. Any other input data is considered part of the member, including /* and /&; The following job stream catalogs a procedure with the name YPROC11 into sublibrary YSUB3:

```
     // JOB CATPROC
     // EXEC LIBR
     ACCESS SUBL=YOURLIB.YSUB3
(1)  CATALOG YPROC11.PROC
        ...
        ...
        control statements
        ...
        ...
(2)  /+
     /*
     /&
```

(1)
No EOD is explicitly specified since the default /+ is used. Since the REPLACE parameter is not specified explicitly, REPLACE=NO is assumed. This means, if a procedure of the same name exists in the sublibrary, the new procedure is not cataloged.

(2)
End-of-Data indication.

The name of a procedure can be related to the partition in which the procedure is intended to be run. For details, see "Cataloging Partition-Related Procedures" on page 67.

The presence of SYSIPT data must be indicated to the Librarian program by specifying DATA=YES in the CATALOG command. In addition, you must indicate the end of inline data by the /* statement. The following example catalogs a partition-related procedure for the BG partition consisting of control statements and SYSIPT data:

```
    // JOB CAT
    // EXEC LIBR
(1) ACCESS SUBL=YOURLIB.YSUB3
(2) CATALOG $0YPROC1.PROC DATA=YES
      ...
      ...
      control statements
      ...
    // EXEC PAYROLL
      ...
      data for program PAYROLL (SYSIPT data)
      ...
      ...
(3) /*
      ...
      ...
      control statements
      ...
      ...
(4) /+
    /*
    /&
```

(1)
Defines the sublibrary in which the procedure is to be cataloged.

(2)
The CATALOG command defines the procedure as BG procedure and indicates that SYSIPT data is included.

(3)
SYSIPT data usually follows an // EXEC statement in the procedure and is followed by /*.

(4)
The end of the procedure to be cataloged is indicated by /+.

Cataloging Members with a User-Defined Member Type

You can catalog any kind of user data and assign your own member type to it. This type of data can be handled and manipulated with the Librarian functions. For example, it can be copied, compared, punched, and listed.

Refer also to "Rename a Sublibrary or a Member" on page 118.

Restrictions when Cataloging Procedures

1. If the cataloged procedure, for example an ASI JCL procedure, includes the // JOB statement, there must be no other JOB statement active in the partition when you retrieve the procedure through the EXEC statement.
2. In a procedure, the job control program recognizes any /+ as an end-of-procedure, and terminates processing of the current procedure.

Change the Reuse Attribute of a Sublibrary

If a sublibrary is defined with REUSE=AUTOMATIC, or the REUSE operand is omitted, it is possible to switch to immediate space reclamation using the Librarian CHANGE command.

Remember that the CHANGE command and the REUSE operand of the DEFINE affect only sublibraries, whereas the RELEASE command affects libraries or sublibraries.

Note: This command must be used carefully. See also "Release Space for a Library or Sublibrary" on page 118.

Compare Libraries, Sublibraries, or Members

With the COMPARE command you can compare libraries, sublibraries, and members. The comparison can be based either on directory information or on the member contents. For example:

```
    // JOB COMPARE
    // EXEC LIBR
(1) COMPARE LIB=YOURLIB:MYLIB
(2) COMPARE SUBL=OURLIB.OSUB2:MYLIB.MSUB3,PUNCH=YES
(3) CONNECT SUBL=YOURLIB.YSUB3:MYLIB.MSUB1
(4) COMPARE AB*.OBJ PUNCH=YES
(5) CONNECT SUBL=YOURLIB.YSUB2:=.YSUB3
(6) COMPARE PRINTXA6.L,PRINTXA7.L,DATA=MEMBER
(7) CONNECT SUBL=YOURLIB.YSUB2:=.=
(8) COMPARE PRINTXAA.L:PRINTXBB.L DATA=MEMBER
    /*
    /&
```

(1)
Library YOURLIB is compared with library MYLIB. Since the DATA parameter is not explicitly specified the default DATA=DIRECTORY is in effect. As a result, the names and types of all members that reside in YOURLIB but not in MYLIB are printed on SYSLST.

(2)
Sublibrary OURLIB.OSUB2 is compared with sublibrary MYLIB.MSUB3. Because the DATA parameter is not explicitly specified, the default DATA=DIRECTORY is in effect. As a result, the names and types of all members that reside in OURLIB.OSUB2 but not in MYLIB.MSUB3 are printed on SYSLST. In addition, because PUNCH=YES is specified, a COPY command is created on SYSPCH with the names of the members printed on SYSLST as operands. An end-of-file indication completes the output on SYSPCH. The COPY command created can be used to copy the members missing in MYLIB.MSUB3 from OURLIB.OSUB2 into MYLIB.MSUB3.

PUNCH=YES can only be used in connection with DATA=DIRECTORY.

(3)
When comparing members, the sublibraries to be accessed, YOURLIB.YSUB3 and MYLIB.MSUB1, must be specified in a preceding CONNECT command.

(4)
Since the DATA parameter is not explicitly specified, the default DATA=DIRECTORY is in effect. As a result, the names of all object modules that begin with 'AB' and reside in YOURLIB.YSUB3 but not in MYLIB.MSUB1 are printed on SYSLST. For PUNCH=YES refer to (2).

(5)
Refer to (3). Character '=' can be used since the library is the same as in the first operand.

(6)
DATA=MEMBER causes the member contents to be compared, record by record. Source book PRINTXA6.L in YOURLIB.YSUB2 is compared with the source book of the same name in YOURLIIB.YSUB3. The same is done with source book PRINTXA7.L. After the first mismatch, comparing is stopped for that specific member and the records causing the mismatch are printed on SYSLST. Comparing continues with the next member.

(7)
Refer to (3). Character '=' can be used since the sublibrary is the same as in the first operand.

(8)

Source books PRINTXAA and PRINTXBB are compared. Both reside in sublibrary YSUB2. After the first mismatch, comparing is stopped for that specific member and the records causing the mismatch are printed on SYSLST.

Copy or Move a Library, Sublibrary or Member

With the COPY or MOVE command you can copy or move libraries, sublibraries and members. These operations can be performed between any supported disk devices.

Note: For details about the locking function in connection with the COPY and MOVE commands, refer also to "Locking Rules" on page 114 and "Librarian Handling of IGNLOCK" on page 115.

In any copy or move operation there are always two libraries or sublibraries involved which might be identical. Data is copied or moved from one library or sublibrary into another one. These are referred to as 'from-library' and 'to-library' or as 'from-sublibrary' and 'to-sublibrary'.

The functions of the MOVE and COPY commands are similar, except that the MOVE command causes the moved data to be deleted from the 'from-' library or sublibrary.

Following is a description of COPY operations.

Copying Libraries and Sublibraries

You can either copy complete libraries or select particular sublibraries for copying. For example:

```
    // JOB COPY LIB/SUBLIB
    // EXEC LIBR
(1) COPY LIB=YOURLIB:MYLIB REPLACE=YES LIST=YES
(2) COPY SUBL=YOURLIB.YSUB1:OURLIB.OSUB4 -
              MYLIB.MSUB1:OURLIB.=
    /*
    /&
```

(1)

All sublibraries of library YOURLIB are copied into library MYLIB, which must already exist.

REPLACE=YES indicates to the Librarian to replace sublibraries with the same name already existing in MYLIB with the copied version from YOURLIB. With REPLACE=NO, which is the default, a sublibrary is only copied if it does not already exist in the to-library.

Note: A sublibrary will be replaced only if it does not include locked members. In the example, TLOCK is not specified which means that the default TLOCK=NORMAL is active and the COPY command does not update the sublibrary if it includes locked members. See also the following example for copying members.

With LIST=YES a printout is produced on SYSLST which lists the names and types of all members copied together with the corresponding from/to libraries and sublibraries.

(2)

Sublibrary YSUB1 of library YOURLIB is copied into library OURLIB and named OURLIB.OSUB4, and sublibrary MSUB1 of library MYLIB is also copied into library OURLIB with the name OURLIB.MSUB1. If a library or sublibrary name is the same for the 'from' and 'to' specification you can use the character '=' to indicate that.

Since the default REPLACE=NO is in effect, copying is only done if a sublibrary of the same name does not already exist in the to-library.

Since the default LIST=NO is in effect, no printout is produced.

Copying Members

When copying members you have to use in addition the CONNECT command to specify the sublibraries to be accessed. For example:

```
    // JOB COPY MEMBERS
    // EXEC LIBR
(1) CONNECT SUBL=YOURLIB.YSUB1:=.YSUB6
(1) COPY VERFY8.PHASE,VERFY9.PHASE,VERFA1.OBJ, -
              REPLACE=YES
(2) CONNECT SUBL=YOURLIB.YSUB2:MYLIB.MSUB4
(2) COPY AB*.PHASE,BC*.*,LIST=YES
(3) CONNECT SUBL=YOURLIB.YSUB2:=.=
(3) COPY A.A:A.B TLOCK=COPY REPLACE=YES
    /*
    /&
```

(1)

The library members VERFY8.PHASE, VERFY9.PHASE and VERFA1.OBJ are copied from sublibrary YSUB1 into sublibrary YSUB6. Since YSUB6 is also part of library YOURLIB the character '=' can be used as library indication.

If library members of the same name and type do already exist in YOURLIB.YSUB6 they are replaced by the copies from YOURLIB.YSUB1 because REPLACE=YES is specified.

(2)

Copying is to be performed from sublibrary YOURLIB.YSUB2 into sublibrary MYLIB.MSUB4.

All phases whose names begin with 'AB' are copied from YSUB2 into MSUB4. In addition, all members of any type whose names begin with 'BC' are also copied from YOURLIB.YSUB2 into MYLIB.MSUB4.

Since REPLACE=NO is in effect no copying is performed if a member of the same name and type already exists in MYLIB.MSUB4.

Since LIST=YES is specified a printout is produced on SYSLST which lists the names and types of all members copied, together with the corresponding from/to libraries and sublibraries. Such a printout is especially useful when specifying the members to be copied in generic format as in this example.

(3)

In this copying example, the from-sublibrary and the to-sublibrary are identical. Member A.A will be copied to A.B. in YOURLIB.YSUB2 and replaces member A.B if it exists and even if it is locked. This is indicated by REPLACE=YES and TLOCK=COPY. If A.A is locked, the locking information will be copied to A.B, because TLOCK=COPY was specified. In this case, A.B will be locked with the same lockid as A.A after copying has been completed.

Moving Libraries, Sublibraries, and Members

The MOVE command, like the COPY command, places a copy of the specified sublibrary or member of the from-library or sublibrary into the to-library or sublibrary, respectively. However, the sublibrary or member which has been moved is deleted from the from-library. The same operands are used as for the COPY command. Running the above job streams with the MOVE command instead of the COPY command would yield the same result, except that the entities which are moved are deleted from the source.

If the from-member is locked, the MOVE operation will bypass this member.

Since a move operation always includes a delete operation, it is not performed if conditions exist such as the following:

- The from-sublibrary is defined in a LIBDEF statement.
- The from-sublibrary is being accessed by another VSE partition at the same time.
- The from-sublibrary contains locked members.

Merging Two Sublibraries

There is no MERGE command in the Librarian command set. To merge two sublibraries, use a CONNECT command followed by a COPY or MOVE command with generic member specification. Using the COPY command leaves the from-sublibrary as it was; the MOVE command causes members to be deleted from the from-sublibrary.

To merge, for example, two sublibraries named LIB1.SUBA and LIB2.SUBB, submit the following Librarian job:

```
// JOB MERGE
// EXEC LIBR
CONNECT LIB1.SUBA : LIB2.SUBB
COPY *.*
/*
/&
```

LIB2.SUBB will now contain all the members it contained before, plus any members of different names and types which were in LIB1.SUBA. LIB1.SUBA remains unchanged.

If members of the same name and type existed in both sublibraries before the copy (for example, LIB1.SUBA.PAY.PROC and LIB2.SUBB.PAY.PROC), the version already present in LIB2.SUBB is kept. If the LIB1.SUBA versions of duplicate members are to be kept in the merged sublibrary, simply turn the CONNECT command in the example around:

```
CONNECT LIB2.SUBB : LIB1.SUBA
```

However, you may want to have the LIB1.SUBA versions of the duplicate members in the merged sublibrary and keep LIB1.SUBA unchanged. In this case, you would submit the job:

```
// JOB MERGE
// EXEC LIBR
CONNECT LIB1.SUBA : LIB2.SUBB
COPY *.* REPLACE=YES
/*
/&
```

In the example job shown above, you can enter a MOVE command in place of the COPY. The command:

```
MOVE *.*
```

would leave only the members with duplicate names in LIB1.SUBA. The command:

```
MOVE *.* REPLACE=YES
```

would empty LIB1.SUBA, with its version of the duplicate members in LIB2.SUBB.

For details refer to z/VSE System Control Statements.

Define a Library, Sublibrary, or a SYSRES File

For details, refer to “Defining a Library, Sublibrary, or a SYSRES File” on page 82.

Delete a Library, Sublibrary, or a Member

With the DELETE command you can delete libraries, sublibraries, or members.

Note: For details about the locking function in connection with the DELETE command, refer also to “Librarian Handling of IGNLOCK” on page 115 and “Locking Rules” on page 114.

Deleting Libraries

If the library to be deleted is specified in a LIBDEF statement, that LIBDEF statement must be dropped (LIBDROP statement) or changed before the library is deleted. If a library contains locked members, these members must first be UNLOCKED before the library can be deleted. You may delete one or more libraries with a single DELETE command:

```
// JOB DELETE LIB
// EXEC LIBR
DELETE LIB=YOURLIB MYLIB
/*
/&
```

Libraries YOURLIB and MYLIB are deleted when this job stream is run. That is, the corresponding VTOC entries are deleted. IJSYSRS, the system library, cannot be deleted.

Note: For libraries in VSAM-managed space, the Librarian DELETE command removes the library from the control of the Librarian, but the VSE/VSAM cluster remains unchanged. This cluster can be redefined later as a library, or it can be deleted completely using VSE/VSAM Access Method Services.

Deleting Sublibraries

If the sublibrary to be deleted is specified in a LIBDEF statement, that LIBDEF statement must be dropped (LIBDROP statement) or changed before the sublibrary is deleted. A sublibrary can only be deleted if it is not used at the same time in a different partition of the system. If a sublibrary includes locked members, these members must be UNLOCKED first before the sublibrary can be deleted. You may delete one or more sublibraries with a single DELETE command:

```
// JOB DELETE SUBLIB
// EXEC LIBR
DELETE SUBL=YOURLIB.YSUB2 YOURLIB.YSUB3 MYLIB.MSUB4
/*
/&
```

When this job stream is run sublibraries YOURLIB.YSUB2, YOURLIB.YSUB3 and MYLIB.MSUB4 are deleted. The space is released and can be used again.

Deleting Members

You may delete one or more members with a single DELETE command. The sublibrary has to be specified in a preceding ACCESS command. Locked members must first be UNLOCKED before they can be deleted (refer also to “Locking Rules” on page 114). You may delete one or more members with a single DELETE command:

```
// JOB DELETE MEMBERS
// EXEC LIBR
ACCESS SUBL=YOURLIB.YSUB5
DELETE ACCOUN1B.PHASE ACCOUN2B.OBJ ACCOUN3B.PROC
/*
/&
```

Phase ACCOUN1B, object module ACCOUN2B, and procedure ACCOUN3B are deleted.

Releasing Freed Space

If the space freed by member deletion belongs to a library on a shared disk, or is accessed by another partition, the release of the freed space is delayed until a non-shared status exists, or the other access is dropped. This ensures, for example, that a read operation can continue although the same member is being deleted in parallel.

A shared status exists if the library is either shared across CPUs or between the partitions of a single VSE system (via LIBDEF statements for example).

The REUSE attribute, specified in the DEFINE SUBLIB command, controls the releasing of space. REUSE=AUTO (the default) delays release until the sublibrary is in a non-shared status. REUSE=IMMEDIATE causes immediate release of space in the sublibrary when a member is deleted.

Note: If any sublibrary in a given library contains sensitive members, all sublibraries in this library should have the attribute REUSE=AUTO.

The RELEASE command causes the immediate release of all member and directory space belonging to members deleted while REUSE=AUTO is in effect and the sublibrary is shared.

For further details refer to “Release Space for a Library or Sublibrary” on page 118.

Note: For detail on punching a member refer to “Punch and Re-Catalog a Member” on page 117

List Library, Sublibrary, or Member Information

For the listing function two commands are available, the LIST and the LISTDIR (LD) command.

The LIST Command

By using the LIST command you can display the contents of one or more members, either on SYSLST or SYSLOG.

Phases and dumps are listed in a combined hexadecimal and character string format. For all other member types you can specify FORMAT=HEX so that each member record is followed by a two-line hexadecimal translation. For example:

```
    // JOB LIST
    // EXEC LIBR
(1) ACCESS SUBL=IJSYSRS.SYSLIB
(2) LIST LVTOC.PHASE
(3) LIST $IPLVSE.PROC
(4) LIST $IPLVSE.PROC FORMAT=HEX
    /*
    /&
```

(1)
Defines the sublibrary to be accessed.

(2)
Phase LVTOC is to be displayed. Refer to Figure 33 on page 107 for the output format.

(3)
Procedure $IPLVSE is to be displayed. Refer to Figure 34 on page 108 for the output format.

(4)
Procedure $IPLVSE is to be displayed with FORMAT=HEX on. Refer to Figure 35 on page 108 for the output format.

Note that a HEX display is usually used for code that exists, for example, as a phase. In example (4), a procedure ($IPLVSE) is used for a HEX display. This is to allow a compare of the HEX display with the display of the same procedure created with the LIST command under (3).

Since no output device is explicitly specified, the default is taken. This is SYSLST, if the commands were entered from SYSIPT, or SYSLOG, if the commands were entered from SYSLOG.

```
MEMBER=LVTOC.PHASE       SUBLIBRARY=IJSYSRS.SYSLIB    DATE: 1999-01-24
                                                      TIME: 15:34
--------------------------------------------------------------------------------
000000  05F047F0 F0625CC9 ........ 60F3F0F1 40C3D6D7     * 0  00 *IJWLTV1.04 ...
000020  E8D9C9C7 C8E340C9 ........ F540D3C9 C3C5D5E2     *YRIGHT IBM CORP 19 ...
000040  C5C440D4 C1E3C5D9 ........ C5D9E3E8 40D6C640     *ED MATERIAL,PROGRA ...
000060  C9C2D400 059041A0 ........ AE2318DE D709B3A0     *IBM
          .
          .
          .
000640  B2FC5880 B3004C80 ........ 1FEE43E0 B337192E     *
000660  4770960E 1E785E70 ........ 5880B308 197847D0     *
000680  966C58E0 B2FC4C70 ........ 42201000 5070B28C     *
0006A0  1E7E5E70 B2CCD201 ........ 20005EE0 B28C5EE0     *
0006C0  B2D8BE6F E0005E80 ........ B31419F8 47709686     *
          .
          .
          .
```

Figure 33. Output Format of a Member (Phase) Display

```
MEMBER=$IPLVSE.PROC      SUBLIBRARY=IJSYSRS.SYSLIB    DATE: 1999-01-24
                                                      TIME: 15:34
--------------------------------------------------------------------------------
009,$$A$SUPX,VSIZE=120M,VIO=512K,VPOOL=64K,LOG
ADD 009,3277
ADD 00C,2540R
ADD 00D,2540P
      .
      .
      .
SYS NPARTS=44
SYS SEC=NO
SYS PASIZE=30M
SYS SPSIZE=0K
SYS BUFLD=YES
DPD VOLID=DOSRES,CYL=209,NCYL=12,TYPE=N,DSF=N
      .
      .
      .
SVA SDL=300,GETVIS=768K,PSIZE=(256K,2000K)
```

Figure 34. Output Format of a Member (Procedure) Display

```
MEMBER=$IPLVSE.PROC      SUBLIBRARY=IJSYSRS.SYSLIB    DATE: 1999-01-24
                                                      TIME: 15:34
--------------------------------------------------------------------------------
009,$$A$SUPX,VSIZE=120M,VIO=512K,VPOOL=64K,LOG
FFF655C5EEDE6EECEC7FFFD6ECD7FFFD6EDDDD7FFD6DDC4444444444444444444444444444444444
009BBB1B2477B52995E1204B596E5122B57663E642B3670000000000000000000000000000000000
ADD 009,3277
CCC4FFF6FFFF44444444444444444444444444444444444444444444444444444444444444444444
1440009B327700000000000000000000000000000000000000000000000000000000000000000000
ADD 00C,2540R
CCC4FFC6FFFFD4444444444444444444444444444444444444444444444444444444444444444444
1440003B254090000000000000000000000000000000000000000000000000000000000000000000
ADD 00D,2540P
CCC4FFC6FFFFD4444444444444444444444444444444444444444444444444444444444444444444
      .
      .
      .
SVA SDL=300,GETVIS=768K,PSIZE=(256K,2000K)
EEC4ECD7FFF6CCEECE7FFFD6DECEC74FFFD6FFFFD544444444444444444444444444444444444444
2510243E300B753592E7682B72995ED2562B20002D00000000000000000000000000000000000000
```

Figure 35. Output Format of a Member (Procedure) Display (Format=HEX)

The LISTDIR Command

By using the LISTDIR (LD) command you can display the contents (or part of the contents) of a library or sublibrary directory. You can also display the system directory list (SDL).

The output is either displayed on SYSLST or SYSLOG (depending on the input device), sorted alphanumerically. At sublibrary level, the primary sort field is the member type. You can control the type and amount of the output with the OUTPUT parameter. For example:

```
// JOB LIST
// EXEC LIBR
LISTDIR LIB=IJSYSRS OUTPUT=STATUS
/*
/&
```

This job stream displays the status information of library IJSYSRS on SYSLST. Refer to Figure 36 on page 109.

```
STATUS  DISPLAY      LIBRARY=IJSYSRS                DATE: 2014-10-07
                                                    TIME: 06:56
--------------------------------------------------------------------
FILE-ID        : (NOT DISPLAYED FOR IJSYSRS)
CREATION DATE  : 2014-08-11   06:51
SUBLIBRARIES   :        1
EXTENTS        : MAX16

LOCATION (BAM) : DEVICE=3390  VOLID=DOSRES CYL =     0.08 -    59.14

LIBRARY BLOCK  : SIZE=  1024 BYTES   DATA SPACE=   988 BYTES

TOTAL   SPACE  :   29436 LIBRARY BLOCKS   (100 %)
USED    SPACE  :   26850 LIBRARY BLOCKS   ( 91 %)
DELAYED SPACE  :       8 LIBRARY BLOCKS   (  0 %)
FREE    SPACE  :    2578 LIBRARY BLOCKS   (  9 %)

--------------------------------------------------------------------
SUBLIBRARY CREATION   SPACE     NO. OF      USED  DELAYED    % LIBR.
           DATE       REUSAGE   MEMBERS     LB'S  LB'S       SPACE
--------------------------------------------------------------------
SYSLIB     2014-08-11   AUTO       2711    26845        8      91 %
```

Figure 36. Output Format of a Library Display (OUTPUT=STATUS)

OUTPUT=STATUS is applicable for libraries and sublibraries. Other output formats are FULL, NORMAL, and SHORT to be used for libraries, sublibraries, and members.

The OUTPUT parameter is not applicable for displaying the SDL. The following job stream displays library, sublibrary, member, and SDL information on SYSLST:

```
     // JOB LIST
     // EXEC LIBR
(1)  LD L=IJSYSRS O=NORM
(2)  LD S=IJSYSRS.SYSLIB O=FULL
(3)  LD SDL
(4)  LD SDL PHASE=$IJBLBR
(5)  LD SDL PHASE=$$*.P* O=SHORT
(6)  LD S=PRD2.CONFIG LOCKID=*
     /*
     /&
```

(1)
Displays the contents of library IJSYSRS in the OUTPUT=NORMAL format. Refer to Figure 37 on page 110.

(2)
The LISTDIR command displays for OUTPUT=FULL also the locking information of a member. Refer to Figure 38 on page 111.

(3)
Displays the system directory list (SDL). Refer to Figure 39 on page 112.

(4)
Displays status and directory information for a single phase in OUTPUT=NORMAL format. Refer to Figure 40 on page 113.

(5)
Displays status and directory information for selected phases in OUTPUT=SHORT format. Refer to Figure 41 on page 113

(6)
Displays all locked members in sublibrary PRD2.CONFIG. Refer to Figure 42 on page 114.

```
STATUS  DISPLAY      LIBRARY=IJSYSRS                DATE: 2014-10-07
                                                    TIME: 07:00
---------------------------------------------------------------------
FILE-ID        : (NOT DISPLAYED FOR IJSYSRS)
CREATION DATE  : 2014-08-11   06:51
SUBLIBRARIES   :        1
EXTENTS        : MAX16

LOCATION (BAM) : DEVICE=3390  VOLID=DOSRES CYL =     0.08 -    59.14

LIBRARY BLOCK  : SIZE=  1024 BYTES   DATA SPACE=   988 BYTES

TOTAL   SPACE  :   29436 LIBRARY BLOCKS   (100 %)
USED    SPACE  :   26850 LIBRARY BLOCKS   ( 91 %)
DELAYED SPACE  :       8 LIBRARY BLOCKS   (  0 %)
FREE    SPACE  :    2578 LIBRARY BLOCKS   (  9 %)

---------------------------------------------------------------------
SUBLIBRARY CREATION   SPACE     NO. OF      USED  DELAYED    % LIBR.
           DATE       REUSAGE   MEMBERS     LB'S  LB'S       SPACE
---------------------------------------------------------------------
SYSLIB     2014-08-11   AUTO       2711    26845        8      91 %

DIRECTORY DISPLAY    SUBLIBRARY=IJSYSRS.SYSLIB      DATE: 2014-10-07
                                                    TIME: 07:00
---------------------------------------------------------------------
 M E M B E R      CREATION   LAST     BYTES    LIBR CONT SVA  A- R-
NAME     TYPE     DATE      UPDATE   RECORDS   BLKS STOR ELIG MODE
---------------------------------------------------------------------
APPLID   A        14-08-11   -  -      163 R       8 YES   -   -   -
BSSTIXE  A        14-08-11   -  -      104 R       5 YES   -   -   -
CICS     A        14-08-11   -  -      234 R      11 YES   -   -   -
DEVTYPE  A        14-08-11   -  -       23 R       1 YES   -   -   -
DFHDCTC2 A        14-08-11   -  -      123 R       8 YES   -   -   -
....
```

Figure 37. Output Format of a Library Display (OUTPUT=NORMAL)

The column 'CREATION DATE' indicates when that member was cataloged for the first time. The column 'LAST UPDATE' indicates when that member was recataloged or updated last.

The following applies to the BYTES/RECORDS column:

> If the size of a member exceeds a value of 8 digits, the notation of the size for such a member is in KB (kilobytes) or in KR (kilorecords). Members of type DUMP, for example, can be in this range.

The last two columns for A-MODE and R-MODE indicate whether the program (phase) is eligible for running in a 24-bit or 31-bit environment or both.

```
STATUS  DISPLAY      SUBLIBRARY=IJSYSRS.SYSLIB      DATE: 2014-10-07
                                                    TIME: 07:07
--------------------------------------------------------------------
CREATION DATE: 2014-08-11   06:51
MEMBERS      :     2711
SPACE REUSAGE: AUTOMATIC
USED    SPACE:    26845 LIBRARY BLOCKS    ( 91% OF LIBRARY SPACE)
DELAYED SPACE:        8 LIBRARY BLOCKS    (  0% OF LIBRARY SPACE)
--------------------------------------------------------------------

--------------------------------------------------------------------
DIRECTORY DISPLAY    SUBLIBRARY=IJSYSRS.SYSLIB      DATE: 2014-10-07
                                                    TIME: 07:07
--------------------------------------------------------------------
 M E M B E R
NAME     TYPE        M E M B E R    I N F O R M A T I O N
$IJBALE  PHASE       CREATION DATE      : 2014-08-11   06:51
                     LAST UPDATE        :     -  -

                     NUMBER OF RECORDS  :        1
                     LOGICAL RECORD SIZE:     8248
                     LIBRARY BLOCKS USED:          9
                     FIRST LIBRARY BLOCK:     7.09.05 (CYL.TRK.REC)
                                ON VOLID: DOSRES
                     LAST LIBRARY BLOCK :     7.09.13 (CYL.TRK.REC)
                                ON VOLID: DOSRES

                     CONTIGUOUSLY STORED: YES
                     MSHP CONTROLLED    : YES
                     MSHP BYPASS USED   : NO
                     SYSIPT DATA        : NO
                     LOCKED             : NO
                     LOCKID             : -

                     PHASE SIZE         :     8248   (HEX:  002038)
                     LOAD  ADDRESS (HEX): 400078
                     ENTRY ADDRESS (HEX): 400078
                     SVA ELIGIBLE       : YES
                     PFIX REQUESTED     : YES
                     RELOCATABLE        : YES
                     ADDRESSING MODE    : 31
                     RESIDENCY MODE     : ANY

$IJBALET PHASE       CREATION DATE      : 2014-08-11   06:51
                     LAST UPDATE        :     -  -

                     NUMBER OF RECORDS  :        1
                     LOGICAL RECORD SIZE:     2200
                     LIBRARY BLOCKS USED:          3
                     FIRST LIBRARY BLOCK:     7.09.14 (CYL.TRK.REC)
                                ON VOLID: DOSRES
                     LAST LIBRARY BLOCK :     7.09.16 (CYL.TRK.REC)
                                ON VOLID: DOSRES

                     CONTIGUOUSLY STORED: YES
                     MSHP CONTROLLED    : YES
                     MSHP BYPASS USED   : NO
                     SYSIPT DATA        : NO
                     LOCKED             : NO
                     LOCKID             : -
```

Figure 38. Output Format of a Sublibrary Display (OUTPUT=FULL)

```
STATUS DISPLAY       SDL  AND  SVA                  DATE: 2014-10-07
                                                    TIME: 07:11
--------------------------------------------------------------------
SDL       TOTAL ENTRIES :    908   (100%)
          USED  ENTRIES :    513   ( 56%)
          FREE  ENTRIES :    395   ( 44%)

SVA(24)   TOTAL SPACE   :   1292K  (100%)
          USED  SPACE   :    977K  ( 76%)
           - PFIXED AREA:     87K  (  7%)  START AT: 001FC108
          FREE  SPACE   :    315K  ( 24%)

SVA(31)   TOTAL SPACE   :  10664K  (100%)
          USED  SPACE   :   7846K  ( 74%)
           - PFIXED AREA:   1050K  ( 10%)  START AT: 0A3635A0
          FREE  SPACE   :   2818K  ( 26%)
--------------------------------------------------------------------

DIRECTORY DISPLAY    SDL SORTED BY PHASE NAME       DATE: 2014-10-07
                                                    TIME: 07:11
--------------------------------------------------------------------
 M E M B E R    ORIGIN SVA/MOVE  LOADED  PHASE  ADDRESS  ENTRY POINT
NAME     TYPE   SYSLIB   MODE   INTO SVA SIZE   IN SVA   IN SVA
--------------------------------------------------------------------
$$BACLOS PHASE    YES    MOVE      31      658  09B82CC8 09B82CC8
$$BATTNA PHASE    YES    MOVE      31     2216  09B82F60 09B82F60
$$BATTNK PHASE    YES    MOVE      31     1104  09B83808 09B83808
$$BATTNR PHASE    YES    MOVE      31      389  09B83C58 09B83C58
$$BCLOSE PHASE    YES    MOVE      31     1192  09B83DE0 09B83DE0
$$BCLOS2 PHASE    YES    MOVE      31      624  09B84288 09B84288
$$BCLOS5 PHASE    YES    MOVE      31     1056  09B844F8 09B844F8
$$BCLRPS PHASE    YES    MOVE      31      712  09B84918 09B84918
$$BCVSAM PHASE    YES    MOVE      31      776  09B84BE0 09B84BE0
$$BCVS02 PHASE    YES    MOVE      31      358  09B84EE8 09B84EE8
.....
$IJBAR   PHASE    YES     24       24    37288  00208E50 00208E50
$IJBAR31 PHASE    YES    ANY       31    82064  0A448998 0A448998
$IJBASGN PHASE    YES     24       24     2888  000CF000 000CF000
$IJBATTN PHASE    YES    ANY       31     2600  09A00000 09A00000
$IJBBPX  PHASE    YES    ANY       31    14648  09A00A28 09A00A28
$IJBCJC  PHASE    YES    ANY       31     3024  09A04360 09A04360
$IJBCPUB PHASE    YES    ANY       31     1248  0A4484B0 0A4484B0
$IJBCRT  PHASE    YES    ANY       31   136520  09A04F30 09A04F30
$IJBCSIO PHASE    YES    ANY       31   172120  0A41E450 0A41E6D0
$IJBCUIR PHASE    YES    ANY       31     8124  0A41C490 0A41C490
$IJBDCMD PHASE    YES     24       24    30768  000CFB48 000CFB48
$IJBDSP  PHASE    YES    ANY       31     6864  0A41A9B8 0A41A9B8
$IJBDSPA PHASE    YES    ANY       31      448  0A41A7F0 0A41A7F0
.....
PRB$AID  PHASE    YES     NO       NO     8648      -        -
PRB$FDM  PHASE    YES     NO       NO     6176      -        -
PRB$IDH  PHASE    YES     NO       NO    79264      -        -
....
```

Figure 39. Output Format of an SDL Display

Column SVA/MOVE MODE can contain MOVE, ANY, 24 or NO and has the following meaning:

- MOVE means that the phase is self-relocatable and is only valid for B- and C-transient phases. It indicates that the phase is loaded into the SVA (31-bit or 24-bit, if the free space in the 31-bit SVA was not large enough to hold the phase at load time) in order to be moved from there to the respective transient area when the phase is to be executed.
- ANY means that the phase can be loaded into the 31-bit SVA as well as into the 24-bit SVA.
- 24 means that the phase can only be loaded into the 24-bit area.
- NO means that the phase is not loaded at all.

```
STATUS DISPLAY       SDL  AND  SVA                  DATE: 2014-10-07
                                                    TIME: 07:15
--------------------------------------------------------------------
SDL       TOTAL ENTRIES :    908   (100%)
          USED  ENTRIES :    513   ( 56%)
          FREE  ENTRIES :    395   ( 44%)

SVA(24)   TOTAL SPACE   :   1292K  (100%)
          USED  SPACE   :    977K  ( 76%)
           - PFIXED AREA:     87K  (  7%)  START AT: 001FC108
          FREE  SPACE   :    315K  ( 24%)

SVA(31)   TOTAL SPACE   :  10664K  (100%)
          USED  SPACE   :   7846K  ( 74%)
           - PFIXED AREA:   1050K  ( 10%)  START AT: 0A3635A0
          FREE  SPACE   :   2818K  ( 26%)
--------------------------------------------------------------------

DIRECTORY DISPLAY    SDL SORTED BY PHASE NAME       DATE: 2014-10-07
                                                    TIME: 07:15
--------------------------------------------------------------------
 M E M B E R    ORIGIN SVA/MOVE  LOADED  PHASE  ADDRESS  ENTRY POINT
NAME     TYPE   SYSLIB   MODE   INTO SVA SIZE   IN SVA   IN SVA
--------------------------------------------------------------------
$IJBLBR  PHASE    YES     24       24    70088  000D9678 000D9678
```

Figure 40. Output Format of an SDL Display for a single Phase (O=NORMAL)

```
STATUS DISPLAY       SDL  AND  SVA                  DATE: 2014-10-07
                                                    TIME: 07:17
--------------------------------------------------------------------
SDL       TOTAL ENTRIES :    908   (100%)
          USED  ENTRIES :    513   ( 56%)
          FREE  ENTRIES :    395   ( 44%)

SVA(24)   TOTAL SPACE   :   1292K  (100%)
          USED  SPACE   :    977K  ( 76%)
           - PFIXED AREA:     87K  (  7%)  START AT: 001FC108
          FREE  SPACE   :    315K  ( 24%)

SVA(31)   TOTAL SPACE   :  10664K  (100%)
          USED  SPACE   :   7846K  ( 74%)
           - PFIXED AREA:   1050K  ( 10%)  START AT: 0A3635A0
          FREE  SPACE   :   2818K  ( 26%)
--------------------------------------------------------------------

DIRECTORY DISPLAY    SDL SORTED BY PHASE NAME       DATE: 2014-10-07
                                                    TIME: 07:17
--------------------------------------------------------------------
 M E M B E R              M E M B E R              M E M B E R
NAME     TYPE            NAME     TYPE            NAME     TYPE
-----------------        -----------------        -----------------
$$BACLOS PHASE           $$BODADE PHASE           $$BOPLBL PHASE
$$BATTNA PHASE           $$BODADS PHASE           $$BOPNR2 PHASE
$$BATTNK PHASE           $$BOESTV PHASE           $$BOPNR3 PHASE
$$BATTNR PHASE           $$BOKUL1 PHASE           $$BOSDC1 PHASE
$$BCLOSE PHASE           $$BOMLTA PHASE           $$BOSMMW PHASE
$$BCLOS2 PHASE           $$BOMSVA PHASE           $$BOSMXT PHASE
$$BCLOS5 PHASE           $$BOMSV2 PHASE           $$BOTLTA PHASE
$$BCLRPS PHASE           $$BOPEN  PHASE           $$BOTUSR PHASE
$$BCVSAM PHASE           $$BOPENR PHASE           $$BOUR01 PHASE
$$BCVS02 PHASE           $$BOPEN1 PHASE           $$BOVSAM PHASE
$$BCVS03 PHASE           $$BOPEN2 PHASE           $$BOVS01 PHASE
$$BOCP01 PHASE           $$BOPEN3 PHASE           $$BTCLOS PHASE
$$BOCP03 PHASE
```

Figure 41. Output Format of an SDL Display (O=SHORT)

```
 STATUS  DISPLAY      SUBLIBRARY=PRD2.CONFIG         DATE: 2014-10-07
                                                    TIME: 07:20
--------------------------------------------------------------------
CREATION DATE: 2014-04-22   14:40
MEMBERS      :       77
SPACE REUSAGE: AUTOMATIC
USED    SPACE:     1120 LIBRARY BLOCKS    (  1% OF LIBRARY SPACE)
DELAYED SPACE:        0 LIBRARY BLOCKS    (  0% OF LIBRARY SPACE)

--------------------------------------------------------------------

DIRECTORY DISPLAY    SUBLIBRARY=PRD2.CONFIG         DATE: 2014-10-07
                                                    TIME: 07:20
--------------------------------------------------------------------
 M E M B E R      CREATION   LAST     BYTES    LIBR
NAME     TYPE     DATE      UPDATE   RECORDS   BLKS  LOCKID
--------------------------------------------------------------------
TCPAPPL  B        14-04-22 14-10-07    272 R      13 MYLOCK
```

Figure 42. Output Format of Locked Members

An example of identifying locked members with the **SEARCH** command is provided in Figure 47 on page 126.

Lock a Member

In an interactive programming environment, it is often necessary to lock a member for a long period of time. For example, if a user at a workstation wants to edit a member, this member has to be locked for any write or update access until the user sends the member back to the host library.

The LOCK command allows you to lock single members with the *lockid* specified in the command unless one of the following applies:

- The member is already locked.
- The user has no UPDATE access right for the member.
- The member is MSHP controlled and MSHP bypass is not active.
- The job control option IGNLOCK is active.

Usually, the same user who locks a member will also unlock it. There are two exceptions, however. MSHP will update a member even if it is locked and the updated member remains unlocked. In a system with security active, any user with the UPDATE access right to a locked member can unlock it.

The locking facility is also available with the application program interface of the Librarian.

Locking Rules

A locked member remains locked until:

- An UNLOCK command for this member, its library, or sublibrary is given.
- A DEFINE sublibrary command with RESETLOCK=YES is given for the sublibrary containing the specified member.
- A MOVE, COPY, or RESTORE command with TLOCK=RESET updated the specified member.
- A MOVE, COPY or RESTORE command with TLOCK=COPY changed the *lockid* of the specified member.
- The member is replaced or deleted while the job control option IGNLOCK is active.
- The member is unlocked by one of the following:

  1. Jobs created by dialogs such as the following (and by other IBM service dialogs):

     – Fast Service Upgrade (FSU)

     – PTF Handling

     – Install Programs - V2 Format

  2. System utilities such as the following:

  - Maintain System History Program (MSHP)
  - DTRSETP utility program
  - DTRISTRT utility program
- The member is updated by MSHP (the member is MSHP controlled and MSHP bypass is not active).

The UNLOCK command unlocks a member unless:

- The user has no UPDATE access right for the specified library, sublibrary, or member.
- The member is locked with a *lockid*, which does not correspond to the one specified in the UNLOCK command.
- The job control option IGNLOCK is active.

For a detailed description of the TLOCK and RESETLOCK operands and the job control option IGNLOCK, refer to z/VSE System Control Statements.

For a non-generic member specification, the command returns an appropriate message and return code, if the unlock failed and leaves the member unchanged.

Job Control IGNLOCK Option

A library member that has unintendedly been locked may cause programs or job streams to damage the system if they cannot be executed completely. To allow modification of such jobs and ensure their normal execution despite of members that are possibly locked, the job control options IGNLOCK and NOIGNLOCK are available.

IGNLOCK and NOIGNLOCK have the following characteristics:

- // OPTION IGNLOCK causes locks to be ignored, therefore, use this option with care. A library member is then treated as if it had not been locked. The option stays in effect until end-of-job or until an // OPTION NOIGNLOCK is given. No STDOPT equivalent exists for this option.
- // OPTION NOIGNLOCK, which is always the default, causes // OPTION IGNLOCK to be reset.

Librarian Handling of IGNLOCK

If the // OPTION IGNLOCK is set, any Librarian function in this job will delete, rename or update members even if they are locked, or will delete libraries/sublibraries or rename sublibraries even if they contain locked members. The renamed or updated member will be unlocked. A corresponding message will be issued in all cases.

Any LOCK or UNLOCK command and any LIBRM LOCK/UNLOCK macro will be ignored and a corresponding message will be issued.

In detail, the Librarian commands react as follows if // OPTION IGNLOCK is specified:

- CATALOG command

  REPLACE=YES causes the specified member to be replaced even if it is locked; the replaced member will be unlocked.
- COPY command

  The TLOCK option, if specified, is ignored. If REPLACE=YES is in effect, a target member will be replaced and unlocked even if it is locked. An existing sublibrary will be replaced also if it contains locked members. The sublibrary will not contain any locked members after processing.
- DEFINE command

  REPLACE=YES causes an existing sublibrary to be deleted even if it contains locked members.
- DELETE command

  The specified library or sublibrary will be deleted even if it contains locked members; members will also be deleted if they are locked.
- LOCK/UNLOCK command

The command is ignored; no member will be locked/unlocked. Return code is 0.

- MOVE command

  The TLOCK option, if specified, is ignored. If REPLACE=YES is in effect, a target member will be replaced and unlocked even if it is locked. An existing sublibrary will be replaced even if it contains locked members. The sublibrary will contain no locked members after processing. The source sublibrary will be deleted even if it contains locked members; members will also be deleted if they are locked.

- RENAME command

  A member will be renamed even if it is locked; the renamed member will be unlocked. A sublibrary will be renamed also if it contains locked members. The members remain locked in the renamed library. (The old status of the sublibrary can be recreated by renaming the sublibrary back to its old name).

- RESTORE command

  The TLOCK option, if specified, is ignored. If REPLACE=YES is in effect, a target member will be replaced and unlocked even if it is locked. An existing sublibrary will be replaced also if it contains locked members. The sublibrary will not contain any locked members after processing.

- UPDATE command

  The specified member will be updated even if it is locked. The member will be unlocked after updating.

- Macro LIBRM OPEN

  For TYPEFLE=INOUT or TYPEFLE=(OUTPUT,REPLACE), the specified member will be opened even if it is locked.

- Macro LIBRM DELETE

  The specified member will be deleted even if it is locked.

- Macro LIBRM RENAME

  The specified member will be renamed even if it is locked.

- Macro LIBRM CLOSE

  For COMMIT=YES, the specified member will be closed and unlocked even if it is locked.

- Macro LIBRM LOCK/UNLOCK

  The macro is ignored; no member will be locked/unlocked. The return code is 0, feedback code 8.

LOCK Example

```
    // JOB LOCK
    // EXEC LIBR
    ACCESS SUBL=OURLIB.YSUB4
(1) LOCK YPROG.A LOCKID=LWW3
    /*
    /&
```

(1) Locks member YPROGA.A using LWW3 as lock ID.

UNLOCK Examples

```
    // JOB UNLOCK
    // EXEC LIBR
    ACCESS SUBL=OURLIB.YSUB4
(1) UNLOCK YPROG.A LOCKID=LWW3
(2) UNLOCK SUBL=OURLIB.YSUB5 LOCKID=*
(3) UNLOCK SUBL=OURLIB.YSUB6 LOCKID=A*
    /*
    /&
```

(1) Unlocks member YPROGA.A in YSUB4 if the lock ID is LWW3.
(2) Unlocks all members in YSUB5.
(3) Unlocks all members in YSUB6 whose lock ID starts with A.

Punch and Re-Catalog a Member

The PUNCH command allows you to punch one or more members from the sublibrary specified in a preceding ACCESS command. The output device is SYSPCH. For example:

```
// JOB PUNCHCAT
ASSGN SYSPCH,TAPE
// EXEC LIBR
ACCESS SUBL=YOURLIB.YSUB1
PUNCH OAZPRG1.OBJ
/*
/&
```

With this job, a punched version of the member OAZPRG1.OBJ in sublibrary YOURLIB.YSUB1 is created on the tape unit assigned to SYSPCH.

The punched version includes the end-of-file indication of the original member and is preceded by a CATALOG statement for a subsequent catalog run. The CATALOG statement includes the EOD and DATA parameter if applicable.

A CATALOG statement is punched for object modules, source books, procedures, and user-defined member types. A PHASE statement is punched for phases to allow easy cataloging with a Linkage Editor run.

You only have to provide the sublibrary name (via the ACCESS command) in order to re-catalog the punched object module in a different sublibrary as shown by the following skeleton job:

```
// JOB CAT
// ASSGN SYSIPT,...            (to SYSPCH of punch job)
// EXEC LIBR,PARM='ACCESS SUBLIB=...'
(Librarian reads input from SYSIPT)
/*
/&
```

Refer also to “Catalog a Member” on page 97.

In the following sequence, the same job steps as shown above, punching and cataloging a member, are performed. But instead of SYSRDR, SYSLOG is used as the input device for the control statements. No JOB statement and no // in front of the job control statements are necessary when running the Librarian from SYSLOG.

```
(1)  ASSGN SYSPCH,280
(2)  EXEC LIBR
(3)  ACCESS SUBL=YOURLIB.YSUB1
(4)  PUNCH OAZPRG1.OBJ
(5)  END
(6)  MTC REW,280
(7)  ASSGN SYSIPT,280
(8)  EXEC LIBR
(9)  ACCESS SUBL=OURLIB.OSUB2
(10) INPUT SYSIPT
```

(1)

Tape unit 280 is assigned to SYSPCH.

(2)

The Librarian program is called.

(3)

The sublibrary to be accessed (YOURLIB.YSUB1) is defined.

(4)

The member OAZPRG1.OBJ stored in YOURLIB.YSUB1 is punched and the output is written to SYSPCH.

(5)

The Librarian run is ended.

(6)

The SYSPCH tape is rewound to the start of the punched output.

(7)
Tape unit 280 is re-assigned as input device for the catalog step.

(8)
The Librarian program is called again.

(9)
OURLIB.OSUB2 is accessed as target sublibrary for the following CATALOG statement.

(10)
With the INPUT Librarian command you change the input device from SYSLOG to SYSIPT. From now on the Librarian reads all input from tape unit 280. Since the CATALOG command and the EOD indication are part of the punched output, all information for cataloging is available.

Release Space for a Library or Sublibrary

When you delete a member, the space occupied is usually released and can be used again. However, this is not true for libraries and sublibraries that are either shared between the partitions of one VSE System (via LIBDEF statements, for example) or that are shared across CPUs.

Note: A library with several extents distributed over several DASDs is considered to be shared across CPUs if the first extent of the library resides on a shared DASD.

For such libraries or sublibraries the space is only released when it is known that they are in a non-shared status. This ensures that a read operation can continue although the member is being deleted at the same time.

With the RELEASE command you can force the release of space for shared libraries and sublibraries. For example:

```
// JOB RELEASE SPACE
EXEC LIBR
RELEASE SPACE SUBLIB=SHARLIB.SHSUB1
/*
/&
```

This job frees all space for sublibrary SHARLIB.SHSUB1 that is recorded internally for release, although the sublibrary may still be in a shared status.

Specifying LIB=SHARLIB in the RELEASE command frees the space of all sublibraries of SHARLIB recorded for release.

It is also possible to define a shared sublibrary so that the space freed by deletion of members becomes available for reuse at once. This is done by specifying REUSE=IMMEDIATE in the DEFINE command for the sublibrary.

Note: This command must be used carefully. It may occur that a library member stored in a shared sublibrary is being retrieved and deleted at the same time. Since the release of space is normally delayed until a non-shared status exists, the member can still be read and no problem arises. However, if in such a situation the release of space is forced, jobs reading the deleted members terminate abnormally.

Rename a Sublibrary or a Member

With the RENAME command you can change the name of one or more sublibraries, or the name and type of one or more members. If the new name already exists, the sublibrary or member is not renamed.

Note: For details about the locking function in connection with the RENAME command, refer also to “Librarian Handling of IGNLOCK” on page 115.

Renaming Sublibraries

A sublibrary can only be renamed if it is not being accessed at same time by another VSE partition. Also, a LIBDEF statement including a sublibrary name that is to be changed must be dropped (LIBDROP statement), or changed accordingly, before the rename job is run.

Note that sublibraries with locked members cannot be renamed. Refer also to "Locking Rules" on page 114.

The following job stream changes the names of two sublibraries:

```
// JOB RENAME SUBLIBS
// EXEC LIBR
RENAME SUBLIB=YOURLIB.YSUB1:YOURLIB.YSUB11 -
           YOURLIB.YSUB2:=.YSUB12
/*
/&
```

YOURLIB.YSUB1 becomes YOURLIB.YSUB11 and YOURLIB.YSUB2 becomes YOURLIB.YSUB12. The library name must always be the same for the first and the second operand. The library name may be replaced by the character '=' in the second operand.

Renaming Members

Note that locked members cannot be renamed. Refer also to "Locking Rules" on page 114.

The sublibrary in which a member resides must be defined with an ACCESS command. The following job stream changes the names of two members:

```
// JOB RENAME1
// EXEC LIBR
ACCESS SUBL=YOURLIB.YSUB3
RENAME CHECKRB.PHASE:VERFYRB.=, -
           CHECKEX1.PROC:VERFYEX1.=
/*
/&
```

CHECKRB.PHASE becomes VERFYRB.PHASE and CHECKEX1.PROC becomes VERFYEX1.PROC. Since the member type does not change you can use the character '=' in the second operand. This is also true for the member name if it remains the same and only the member type is changed.

You can specify the member name and type in generic format as well. But both the first and the second operand must then be specified in the generic format. For example:

```
// JOB RENAME2
// EXEC LIBR
ACCESS SUBL=YOURLIB.YSUB3
RENAME A*.PROC:AB*.= FF*.X*:GGA*.Y*
/*
/&
```

With this job all procedures whose names begin with 'A' are identified and 'A' is replaced by 'AB'.

In addition, all members whose names begin with 'FF' and whose types start with 'X' are identified and 'FF' is replaced by 'GGA' and 'X' is replaced by 'Y' in the member type.

If this renaming would result in a name that is longer than eight characters, the rename function is not carried out, and the Librarian sets a return code of 8.

Renaming for Assigning User-Defined Member Types

It may be useful, for example, to maintain several versions of a cataloged procedure. This can be done by assigning the same name but different member types. One version is assigned the member type PROC, to the other versions user-defined member types (PROC1,PROC2,PROC3..., for example). The version of type PROC can be considered as activated and when the procedure is called this version is chosen. You can easily deactivate this procedure by changing its type, for example to PROC4, and activate another version of these cataloged procedures by assigning the member type PROC to it, using the RENAME command. The job stream which calls this procedure need not be changed.

Restore a SYSRES File, Library, Sublibrary, or a Member

By using the restore function (command RESTORE) of the Librarian you can restore backup tapes created by the backup function (command BACKUP) of the Librarian. Single members of backed-up sublibraries, or sublibraries of backed-up libraries can be restored selectively. During restore, an internal reorganization is performed (scattered free space is removed), which usually results in faster read access after restore.

Note:

1. Refer also to "Backup a SYSRES File, Library, Sublibrary, or Member" on page 91 since both commands (BACKUP and RESTORE) have close dependencies.
2. For details about the locking function in connection with the RESTORE command, refer also to "Locking Rules" on page 114 and "Librarian Handling of IGNLOCK" on page 115.

The following can be changed between creating a backup tape and restoring it:

- The number of extents for a library.
- The disk device type.
- The type of library (system into private, for example).

A library that resided in non-VSAM-managed space may be restored into VSAM-managed space and vice versa.

The RESTORE command accepts **unlabeled tapes** and tapes with **standard labels**.

The following may be part of a backup tape and restored:

- SYSRES files

  This may be either IJSYSRS, the IPLed SYSRES file, or SYSRES files created additionally. The names IJSYSR1 through IJSYSR9 are available for that purpose. A SYSRES file consists of the system sublibrary SYSLIB and may in addition contain one or more private sublibraries as well as free space. A SYSRES file can be restored stand-alone or online.

- Libraries and sublibraries

  These can only be restored online.

- Library members

  Selected library members can be restored online. This can be considered as a copy function - the restored members are added to the sublibrary specified, and replace members of the same name and type, if REPLACE=YES is specified.

Restoring a SYSRES file, library, or sublibrary includes its creation. This means that an existing version is deleted and replaced by the version being restored, if REPLACE=YES is specified in the RESTORE command; or that the restore fails, if REPLACE=NO is specified or the REPLACE operand is omitted.

The following restrictions apply:

- An existing library is not restored if it is being accessed by another partition at the same time.
- An existing library is not restored if one of its sublibraries is specified in a LIBDEF statement. This statement has to be dropped (LIBDROP statement) or changed accordingly.
- A library or sublibrary is not restored if existing library members are locked. The same is true of a single library member that is locked. In such cases, use the RESTORE operands TLOCK=RESET or TLOCK=COPY.
- Since the active system sublibrary IJSYSRS.SYSLIB cannot be replaced by a restore run, a sublibrary of the name SYSLIB that is to be restored into the system library IJSYSRS must be assigned a different name.
- A SYSRES file with the name IJSYSRS must be assigned a different name if it is to be restored online. IJSYSRS is reserved for the SYSRES file used for IPL.

- A SYSRES file can be restored as a private library by assigning a name different from IJSYSR1 through IJSYSR9, but not vice versa.

The tape created with the backup function contains one or more backup files. Stand-alone programs or the system history file, which may be on the backup tape, are recognized automatically and skipped. A system history file must be restored with an MSHP INSTALL or RESTORE job, which can be used to restore libraries at the same time.

Retrieving Information from the Backup Tape

Before performing an actual restore run you may need to know the precise contents of the backup tape, and you will certainly need to know the space requirements for restoring the libraries and sublibraries. To gather this information you can use the SCAN operand of the RESTORE command. With the SCAN operand specified no restore is performed, but the complete content of an unlabeled backup tape is scanned and the required information is printed on SYSLST. For a labeled backup tape, only the content of the backup file identified by the label is scanned.

The replacing of members requires temporary disk space over and above the space needed for the members themselves. SCAN provides the following functions:

- The names of all libraries, sublibraries, (and their space requirements), and the names of all members are retrieved and listed. For libraries the space requirements for all supported disk devices are given; for sublibraries in libraries on the backup tape, the number of library blocks they need is given.
- For sublibraries backed-up separately, the disk space requirements for all supported disk devices are given.
- When specific libraries, sublibraries, or members are to be restored, the Librarian searches for them and prints their names with an indication whether they are on the backup tape or not. For libraries and sublibraries the space requirements are also given.
- By using a generic member specification all members of the backup tape matching that generic specification are listed.

The following job causes the next backup file on the unlabeled backup tape mounted on physical unit 281 to be scanned. The names of all libraries and sublibraries as well as the space requirements for restore (on all DASD types supported) are printed on SYSLST. If the backup file contains only members, the names of all members are printed on SYSLST.

```
// JOB SCAN
// EXEC LIBR
RESTORE *,SCAN=YES,TAPE=281
/*
/&
```

The command:

```
RESTORE * ID=* SCAN=YES T=281
```

causes all following backup files on the unlabeled backup tape to be scanned. Note that blanks and commas are allowed between the operands of Librarian commands. For a labeled tape the command would look as follows:

```
RESTORE * SCAN=YES T=281 TAPELABEL=filename
```

Restoring Online SYSRES Files, Libraries, and Sublibraries

With the following job stream two libraries (one is a SYSRES file) and one sublibrary are restored. They are all part of a single, unlabeled backup tape mounted on the physical unit 281, and the specified ID= operands identify the correct backup file:

```
// JOB RESTORE
// DLBL YOURLIB,...
// EXTENT ,volser,...
// DLBL MYLIB,...
// EXTENT ,volser,...
// DLBL IJSYSR1,...
// EXTENT ,volser,...
// EXEC LIBR
RESTORE LIB=IJSYSR1 TAPE=281 ID=BA1
RESTORE LIB=YOURLIB TAPE=281 ID=BA3
RESTORE SUBLIB=MYLIB.MSUB1 TAPE=281 ID=BA4
/*
/&
```

Note: A backed-up SYSRES file must be restored to a disk volume separate from the currently IPLed system. The SYSRES file IJSYSR1, library YOURLIB, and sublibrary MYLIB.MSUB1 are restored from tape to the DASD locations defined by the DLBL and EXTENT statements. The RESTORE commands must reflect the sequence of the backup files (BA1,BA3, and BA4) on the backup tape.

By specifying:

```
RESTORE * ID=* TAPE=281
```

all libraries and/or sublibraries of the unlabeled backup tape are restored. In the example above, the old names are also the new names for the SYSRES file, library and sublibrary restored. You may run such a restore job stream in case of a destroyed library or for internal reorganization to improve performance. If you want to use the restore function for a copy operation, you have to specify two names: the old name under which a library, for example, is stored on the backup tape and the new name under which it is to be restored. The restore commands might then look as follows:

```
RESTORE LIB=IJSYSR1:IJSYSR3,TAPE=281,ID=BA1
RESTORE LIB=YOURLIB:OURLIB,TAPE=281,ID=BA3
RESTORE SUBLIB=MYLIB.MSUB1:COMLIB.CSUB5,TAPE=281,ID=BA4
```

SYSRES file IJSYSR1 becomes SYSRES file IJSYSR3, library YOURLIB becomes library OURLIB, and sublibrary MYLIB.MSUB1 becomes sublibrary COMLIB.CSUB5.

For details about tape positioning refer to "Tape Positioning" on page 95.

Restoring Online Library Members

Library members can be selectively restored from the backup tape into a target sublibrary. The system sublibrary SYSLIB may also be specified as a target sublibrary. Refer to the following example:

```
    // JOB RESMEM
    // EXEC LIBR
(1) ACCESS SUBLIB=COMLIB.CSUB3
(1) RESTORE YOURLIB.YSUB1.CRMUPD6.PHASE, -
          YOURLIB.YSUB1.CRMUPD7.PHASE, -
(2)       YOURLIB.YSUB1.JOBACC4.PHASE:IJSYSRS.SYSLIB,TAPE=281
    /*
    /&
```

(1)
Phases CRMUPD6 and CRMUPD7 of YOURLIB.YSUB1 are restored into the target sublibrary CSUB3, defined with an ACCESS command.

(2)
The target sublibrary can also be defined with the RESTORE command as shown. Phase JOBACC4 of YOURLIB.YSUB1 is restored into the system sublibrary IJSYSRS.SYSLIB.

Existing members of the same name and type are deleted and replaced by the member restored, if REPLACE=YES is specified. **Example with Labeled Tape**:

The following example is for an online restore with a labeled tape:

```
// JOB RESTORE
// TLBL TL1, ...
// MTC REW,181
// EXEC LIBR
   REST SUBL=MYLIB.MSUB1 TAPE=181 TAPELABEL=TL1
/*
/&
```

Duplicate Names in RESTORE Command

In one RESTORE command you may specify a list of library, sublibrary or member names to be restored. If, for some reason, one name occurs more than once in the list, one of two things may happen:

- If the name exists only once on the backup tape, it is restored when the name is first read in the RESTORE command. When the name is read again in the same RESTORE statement, the Librarian issues the message:

  ```
  LIBRARY (or SUBLIBRARY or MEMBER) NOT FOUND
  ```

- If the name occurs more than once (for example in another backup file of an unlabeled tape containing the same library), the second occurrence on tape overwrites the first, if REPLACE=YES is specified. If REPLACE=NO is in effect, the Librarian issues a message and sets a return code of 8.

Restoring a SYSRES File Stand-Alone

The stand-alone restore function allows a single SYSRES file to be restored. This file may be stored on a distribution tape, for example, or on any tape created using the backup function with **RESTORE=STANDALONE** specified. If the backup tape contains more than one SYSRES file, one has to be selected. If private libraries, sublibraries, or members are also part of the backup tape, they must be restored online.

A backup tape for a stand-alone restore need not include a SYSRES file; the SYSRES file can be on another tape.

The stand-alone restore program prompts you for a **TLBL** statement. If you have an unlabeled backup tape, simply respond by pressing ENTER. For a labeled tape enter the tape label in the format

```
// TLBL UIN,'x ... x'
```

where the file name must be UIN.

Figure 43 on page 124 shows a console communication example of a restore run (not initial installation) with an unlabeled tape. In the example, input entered from the console is indicated by arrows. The following preliminary steps are required:

1. Mount the tape with the stand-alone programs.
2. IPL the tape.
3. Press ENTER.

```
   i 490
   BG 0000 SA01I *********** STAND ALONE PROGRAMS LOADED ***********
   BG 0000 SA02D IF YOU WANT A LISTING, SPECIFY PCUU OF PRINTER, ELSE (ENTER)
   BG-0000
==>0
   BG 0000 SA08D DATE IS 10/10/2014. ACCEPT DATE (ENTER) OR SPECIFY DATE MM/DD/
YYYY
   BG-0000
==>0
   BG 0000 SA70D IF YOU WANT TO USE SCSI  DEVICES SPECIFY YES, ELSE NO
   BG-0000
==>0 no
   BG 0000 SA09I SELECT ONE OF THE FOLLOWING PROGRAMS, OR TYPE END
   BG 0000 SA10D FASTCOPY, RESTORE, ICKDSF, DITTO, REIPL
   BG-0000
==>0 restore
   BG 0000 SA11D SPECIFY ADDRESS OF INPUT DEVICE PCUU
   BG-0000
==>0 490
   BG 0000 SA03D DEVICE TYPE IS TPA   . ACCEPT (ENTER) OR SPECIFY ALTERNATE TYPE
   BG-0000
==>0
   BG 0000 SA15D FOR TAPE LABEL CHECKING SPECIFY // TLBL, ELSE (ENTER)
   BG-0000
==>0
```

```
   BG 0000 SA16D SPECIFY ADDRESS OF SYSRES DISK PCUU
   BG-0000
==>0 200
   BG 0000 SA03D DEVICE TYPE IS ECKD  . ACCEPT (ENTER) OR SPECIFY ALTERNATE TYPE
   BG-0000
==>0
   BG 0000 L302A ENTER YES TO RESTORE SYSRES FILE IJSYSRS OR NO TO SKIP TO NEXT
SYSRES
   BG-0000
==>0 yes
   BG 0000 L315I ORIGINAL FILE ID= VSE.SYSRES.LIBRARY
   BG 0000 L316A ENTER YES TO KEEP OR NO TO RESPECIFY THE SYSRES FILE ID
   BG-0000
==>0 yes
   BG 0000 L309I ORIGINAL ALLOCATION=      899 TRACKS =      59 CYLINDERS 14
TRACKS
   BG 0000 L310A ENTER YES TO KEEP OR NO TO RESPECIFY THE ALLOCATION
   BG-0000
==>0 no
   BG 0000 L312I MINIMUM ALLOCATION=      771 TRACKS =      51 CYLINDERS  6
TRACKS
   BG 0000 L304I ENTER THE DESIRED ALLOCATION AS NUMBER OF TRACKS OR
CYLINDERS.TRACKS

   BG 0000 L313A ALLOC=
   BG-0000
==>0 959
   BG 0000 L329A ENTER YES TO RESTORE ALL SUBLIBRARIES OR NO FOR SELECTIVE
RESTORE
   BG-0000
==>0 yes
   BG 0000 L338I SUMMARY OF RESTORE PARAMETERS:
   BG 0000 L318I FILE NAME = IJSYSRS
   BG 0000 L319I FILE ID = VSE.SYSRES.LIBRARY
   BG 0000 L321I ALLOCATION=      959 TRACKS
   BG 0000 L344I START= CYLINDER 0 TRACK 1 - END= CYLINDER      63 TRACK 14
   BG 0000 L327I RESTORE ALL SUBLIBRARIES
   BG 0000 L322A ENTER YES IF THE SPECIFICATION IS CORRECT OR NO TO RESPECIFY
   BG-0000
==>0 yes
   BG 0000 L300I FORMATTING OF LIBRARY IJSYSRS IN PROGRESS
   BG 0000 L306I RESTORE OF LIBRARY IJSYSRS IN PROGRESS
   BG 0000 L325I RESTORE OF SUBLIBRARY IJSYSRS.SYSLIB IN PROGRESS
   BG 0000 L326I RESTORE COMPLETE FOR LIBRARY IJSYSRS
   BG 0000 SA09I SELECT ONE OF THE FOLLOWING PROGRAMS, OR TYPE END
   BG 0000 SA10D FASTCOPY, RESTORE, ICKDSF, DITTO, REIPL
   BG-0000
==>0 end
   BG 0000 SA17W ***** END OF STAND ALONE PROCESSING *****
```

Figure 43. Example of a Stand-Alone Restore (Unlabeled Input Tape)

Restoring with the Librarian Time-Stamp Control

The Librarian program stores (and displays or prints) time stamps for events as follows:

- Creation of a library or sublibrary – the date and time when a library or sublibrary is created or replaced.
- Creation of a library member – the date and time when a library member is newly stored into a sublibrary.

  Note: In a directory listing (the output of the LISTDIR command), the words `CREATION DATE` are used instead of `PUT INTO SUBLIBRARY` as done formerly.
- Last-update – the date and time when an existing library member is replaced or modified.

It may be desirable to preserve the original time stamp beyond and in spite of intervening backups and restores. The restore DATE option allows you to specify whether or not the existing time stamp is to be preserved.

The option can be specified as DATE=OLD or DATE=NEW in the RESTORE command. DATE=OLD indicates that the date and time stored on the backup tape is to be used.

DATE=OLD cannot be specified for a stand-alone restore run.

The DATE option is ignored for a restore run if SCAN=YES is specified on the RESTORE statement.

For a detailed description of the DATE option refer to z/VSE System Control Statements.

Search for Members

The **SEARCH** command searches for specified members in libraries or sublibraries. For every member found, the name and the name of the sublibrary is printed. Additionally, a list of all libraries in which a member was found is provided at the end. If a lock ID is specified, only the members locked by the specified lock ID are displayed.

```
    // JOB SEARCH
    // EXEC LIBR
(1) SEARCH A.A* LIB=TESTLIB,JSLIB O=NORMAL
(2) SEARCH *.A LBR=CONNECT O=FULL
(3) SEARCH TRERP.PHASE LIBDEF=PHASE P=F3 O=FULL
(4) SEARCH *.* LIB=TESTLIB LOCK=* O=FULL
    /*
    /&
```

(1)
The Librarian searches libraries TESTLIB and JSLIB for members whose member name is A and their member type starts with A. Refer to Figure 44 on page 125.

(2)
The Librarian searches in the library chains created with the **CONNECT** command for all members whose member type is A. Refer to Figure 45 on page 126.

(3)
The Librarian searches for phase TRERP in the libraries of the active LIBDEF chain for phases for partition F3. Refer to Figure 46 on page 126.

(4)
The Librarian searches for all locked members in library TESTLIB. Refer to Figure 47 on page 126.

The OUTPUT option (short form: **O**) has two parameters. **O=FULL** provides a SUMMARY OF AFFECTED LIBRARIES at the end of a display. **O=NORMAL** does not.

```
 M E M B E R                                    CREATION    LAST
NAME     TYPE      LIBRARY SUBLIB      CHAIN      DATE     UPDATE
----------------------------------------------------------------------
A        A12
                   TESTLIB S1                   2014-12-14 2015-01-18
                   TESTLIB S2                   2014-12-14     -  -
                   JSLIB   PROD                 2014-12-14     -  -
```

Figure 44. Output Format of Searched Members in Libraries (OUTPUT=NORMAL)

```
 M E M B E R                                     CREATION    LAST
NAME     TYPE      LIBRARY SUBLIB      CHAIN       DATE     UPDATE
---------------------------------------------------------------------
MWW12    A
                  TESTLIB S1        CONN-FROM   2014-12-14 2015-01-12
DUMMY3   A
                  TESTLIB S1        CONN-FROM   2014-12-16 2015-01-12
TESTSL1  A
                  TESTLIB S1        CONN-FROM   2014-12-16     -  -
                  VSAM    SYSLIB    CONN-TO     2014-12-16     -  -
---------------------------------------------------------------------
SUMMARY OF AFFECTED LIBRARIES                         DATE 2015-01-24
                                                       TIME 12:41
---------------------------------------------------------------------
               CREATION     # OF
  LIBRARY         DATE      SUBLIBS     DEVICE       VOLID
---------------------------------------------------------------------
TESTLIB        2014-12-14        7        3380        JSCB30
VSAM           2014-12-14        4        3380        JSCB30
```

Figure 45. Output Format of Searched Members in CONNECT Libraries (OUTPUT=FULL)

```
 M E M B E R                                     CREATION    LAST
NAME     TYPE     LIBRARY SUBLIB      CHAIN        DATE     UPDATE
----------------------------------------------------------------------
TRERP    PHASE
                  VSE1B3  AA26    PHASE :SRCH    2014-08-02   -  -
----------------------------------------------------------------------
SUMMARY OF AFFECTED LIBRARIES                          DATE 2015-01-24
                                                        TIME 12:41
----------------------------------------------------------------------
               CREATION     # OF
  LIBRARY         DATE      SUBLIBS    DEVICE         VOLID
----------------------------------------------------------------------
VSE1B3         2014-12-14      1       3380          JSCB50
```

Figure 46. Output Format of Search for a Single Phase (OUTPUT=FULL)

```
SEARCH DISPLAY
RESULT OF SEARCH                                      DATE 2015-01-24
                                                      TIME 12:41
----------------------------------------------------------------------
 M E M B E R
NAME     TYPE       LIBRARY SUBLIB      CHAIN        LOCKID
----------------------------------------------------------------------
LOCKII4  NOW
                    TESTLIB LOCKLIB                  IT$NOW
INLPLOCK OBJ
                    TESTLIB S1                       WIN99
LOCKAB   PHASE
                    TESTLIB S1                       WIN99
TESTMOD1 PROC
                    TESTLIB S1                       WIN99
----------------------------------------------------------------------
 SUMMARY OF AFFECTED LIBRARIES                         DATE 2015-01-24
                                                       TIME 08:58
----------------------------------------------------------------------
               CREATION     # OF
  LIBRARY         DATE      SUBLIBS     DEVICE       VOLID
----------------------------------------------------------------------
TESTLIB      2014-12-14        7        3380        JSCB30
```

Figure 47. Output Format of Search for Locked Members (OUTPUT=FULL)

Note: If a library or sublibrary was specified in the **SEARCH** command the chain column is left blank. Otherwise, the chain column shows one of the following:

ACCESS
sublibrary was specified in **ACCESS** command

CONN-FROM
FROM-sublibrary in **CONNECT** command

CONN-TO
TO-sublibrary in **CONNECT** command

PHASE :SRCH
sublibrary from **LIBDEF PHASE,SEARCH**

PHASE :CTLG
sublibrary from **LIBDEF PHASE,CATALOG**

SOURCE:SRCH
sublibrary from **LIBDEF SOURCE,SEARCH**

OBJECT:SRCH
sublibrary from **LIBDEF OBJ,SEARCH**

DUMP :CTLG
sublibrary from **LIBDEF DUMP,CATALOG**

Test a Library or Sublibrary

This command is mainly intended as a debugging aid for IBM personnel. You may use it to find out which library, sublibrary, or member causes an error if a system message indicates a library defect. To test a library, for example, you may specify

```
TEST LIB=YOURLIB
```

The information retrieved by the TEST command is displayed on SYSLOG, when the command was entered from SYSLOG, or on SYSLST, when the command was submitted via SYSRDR. z/VSE System Control Statements provides further details about the command.

Unlock a Member

Refer to the description of the LOCK command under “Lock a Member” on page 114.

Update a Member

The UPDATE command allows modification of library members of any type except PHASE and DUMP (phases can be updated by using MSHP). Updating is done by adding, deleting, or replacing lines of the member to be modified but only if the member is not locked. The following subcommands are provided:

)ADD -- To add lines
)DEL -- To delete lines
)REP -- To replace lines
)END -- To indicate end of the update statement sequence.

The)ADD statement must be followed by the statements to be added and the)REP statement must be followed by the statements that are to replace existing ones.

Note: For details about the locking function in connection with the UPDATE command, refer also to “Librarian Handling of IGNLOCK” on page 115.

A time stamp indicates when a member was updated last in addition to the time stamp indicating when the member was first cataloged. The time stamp information can be displayed by using the LISTDIR command.

Assume that you want to update source book YBOOK1 of the member type L by adding, deleting, and replacing lines. The source book is stored in sublibrary YOURLIB.YSUB1. The job required may look as follows:

```
    // JOB UPDATE
    // EXEC LIBR
(1) ACCESS SUBL=YOURLIB.YSUB1
(2) UPDATE YBOOK1.L SAVE=YBOOK2.L
(3) )REP 1260
    OPTNFND CLC 0(1,REGG),BLANKS
(4) )REP 1290,1340
    OPTNCHK CLC 0(5,REGG),=C'PUNCH'
            BE  PUNCH
            CLC 0(5,REGG),=C'DSPCH'
```

```
(5) )ADD 1410
    PUNCH   MVI SWITCH,C'X'
            B   CHKOPND
(6) )DEL 3280,3290
(7) )END
    /*
    /&
```

(1)
The ACCESS command defines the sublibrary to be accessed.

(2)
The UPDATE command defines first the member to be modified. In addition, you may optionally use the parameters SAVE,SEQUENCE, and COLUMN.

SAVE=YBOOK2.L causes the Librarian to save the current version of YBOOK1 under the name YBOOK2.

Since SEQUENCE is not explicitly specified, the default SEQUENCE=10 is used. After modification, the Librarian re-sequences YBOOK1 (starting with number 10) using an increment of 10 between each line number.

SEQUENCE=FS would cause no re-sequencing but a check that the line number sequence is still valid after modification.

SEQUENCE=NO means that no check is performed. For correct modifications the updates have to be supplied in ascending order.

COLUMN specifies the start and end columns that contain the sequence number. The defaults are:

```
columns 77:80 if SEQUENCE=n or SEQUENCE=NO
columns 73:78 if SEQUENCE=FS
```

The member in this example will therefore be numbered in columns 77 to 80.

(3)
The contents of line number 1260 is replaced by the line following)REP 1260.

(4)
The lines numbered 1290 through 1340 are replaced by the three lines following)REP 1290,1340.

(5)
The lines following)ADD 1410 are inserted after line number 1410 in the member.

(6)
The lines numbered 3280 through 3290 are deleted.

(7)
Indicates the end of the update statements.

To number an unnumbered member, use the UPDATE command with the SEQ=n operand (and, if required, the COL operand), followed by an)END subcommand.

Library Access for Application Programs

The application program interface (API) of the Librarian provides access to the objects of VSE libraries for application programs. These objects are:

Library
The data file (library) allocation

Sublibrary
The logical library description

Member
The collection of user data

Chain
The concatenation of sublibraries

Access to these objects is provided through the macros LIBRDCB and LIBRM.

Note: The following information about accessing member data, retrieving status information, and about return code conventions is intended as introduction to the code examples provided under “Example of a STATE Member Request” on page 131 and “Example of OPEN/GET/CLOSE Requests” on page 135. For a detailed description of the LIBRDCB and LIBRM macros and their functions refer to z/VSE System Macros Reference and z/VSE System Macros User's Guide.

The macro LIBRDCB defines the central control block for the library interface. In LIBRDCB you specify, for example, the names of the libraries or sublibraries you want to access and the characteristics and processing requirements of the library members which you want to work on.

Note: For details about the locking function in connection with the LIBRM macro, refer also to “Locking Rules” on page 114 and “Librarian Handling of IGNLOCK” on page 115.

The various options of the LIBRM macro perform Librarian functions, such as:

LIBDEF
Defines a chain of sublibraries.

LIBDROP
Drops a chain of sublibraries with a chain ID.

STATE
Checks whether a particular member, library, sublibrary, or chain exists.

DELETE
Deletes a member from a sublibrary.

RENAME
Renames a member.

OPEN
Opens a member for reading/writing of records.

GET
Retrieves one or more records of a member in the user's work area.

PUT
Writes one or more records from the user's work area into the member specified.

NOTE
Notes the current position within a member.

POINT
Points to the position within a member noted earlier.

CLOSE
Closes a member. No further GET/PUT or NOTE/POINT access is possible.

LOCK
Locks a member with a specified lock ID.

UNLOCK
Unlocks a member with a specified lock ID.

Accessing Member Data

The API provides a record I/O (input/output) interface for application programs.

Record I/O enables application programs to access member data, either as records or substrings (bytes). It is designed to process single records or bytes with GET or PUT requests.

The amount of data to be read or written can be controlled by the user application. For string-type members the number of bytes, for record-type members the number of records can be specified.

Normally, the access is logically sequential but the starting position for GET or PUT can be altered with a macro option. For PUT this is only possible for string type members.

Record I/O must be started with an OPEN request. With this OPEN, three types of processing options can be selected:

OPEN for INPUT
Provides a read-only access to an existing member. Only GET,NOTE and POINT requests are accepted.

OPEN for OUTPUT
Provides a write-only access to a new member. The replacement of an existing member with the same name is controlled by the REPLACE/NOREPLACE option. Only PUT requests are accepted.

OPEN for INOUT
Provides read/write access to a member. Any PUT requests to the member are written to a new copy of that member which replaces the original member at CLOSE time. GET,NOTE and POINT requests are always working with the original (old) member data.

With the CLOSE macro, record I/O is terminated. The access to a member is stopped and all resources owned (such as GETVIS space and locks) are released. Newly created members become accessible.

Retrieving Status Information

With the STATE function, application programs can retrieve information about the status of Librarian objects.

STATE member-name,....
Checks whether the specified member exists in a certain sublibrary or a chain of sublibraries. If it exists, attributes like number of records (bytes), record format or modification time stamp are returned.

STATE sublib-name,....
Checks whether the specified sublibrary exists in a certain library. If it exists, attributes like number of members, used space and delayed space are returned.

STATE lib-name,....
Checks whether the specified library exists. Here, the service returns attributes like number of sublibraries, used and free space, and the physical file allocation.

STATE chain,.......
Checks whether a sublibrary chain exists. If it exits, the sublibrary names are returned to the caller.

Return Code Conventions

Each API macro passes information back to the invoking program to inform about the completion of the requested function. This return information consists of three parts:

- A severity code (Register 15), which gives a global statement about the success or failure of the requested service. This code is used uniformly throughout the library access service macros.
- A reason code (Register 0), which informs the caller in detail about any failing condition or exception.
- A Librarian message Lxxx describing the failure.

In addition, the API provides exits to handle the different levels of errors. An exit can be specified for the following conditions:

- Object not found
- End of member data
- Unexpected error

The STATE Member Request example shown in Figure 48 on page 131 is available as skeleton LIBRSTAT in VSE/ICCF library 59.

Example of a STATE Member Request

Figure 48. Code Example for a Librarian STATE Member Request

```
* * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * *
*                                                                     *
*         EXAMPLE FOR THE LIBRARIAN LIBRM SERVICES                    *
*                                                                     *
*                                                                     *
*                                                                     *
*  Ask for all procedures in a specific sublibrary and process        *
*  the returned member attributes.                                    *
*                                                                     *
*         LIBRM STATE,ENITITY=MEMBER                                  *
*                                                                     *
* * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * *
         SPACE
APIMON00 START X'78'
         BALR  R11,0                   ESTABLISH ADDRESSABILITY
         USING *,R11,R12
         LA    R12,4095(R11)
         LA    R12,1(R12)              SECOND BASE
         SPACE
         LA    R13,APIMONS1            SAVE AREA FOR THIS TASK
         SPACE
* --------------------------------------------------------------------*
*        STATEMENTS FOR SET UP FIELDS NEEDED FOR LIBRM                *
*        E.G. LIBRARY NAME, MEMBER NAME, MEMBER TYPE ....             *
* --------------------------------------------------------------------*
         SPACE
         MVI    APIMEMBN,C'*'                   MEMBER NAME GENERIC
         MVC    APIMEMTY(L'APINAMPR),APINAMPR   MEMBER TYPE PROC
*        .....  ..............                  ..................
SPACE* --------------------------------------------------------------------*
*        GET LDCB MAP                                                 *
* --------------------------------------------------------------------*
         SPACE
         LIBRDCB FUNC=MAP
         SPACE
* --------------------------------------------------------------------*
*        ALLOCATE GETVIS FOR LDCB USED IN THIS TASK                   *
* --------------------------------------------------------------------*
         SPACE
         LIBRM SHOWCB,CB=LDCB,CBLEN=LEN1  LENGTH OF LDCB
         L     R0,LEN1
*
         SPACE

         SPACE

         BNZ    APIERR10                NOT ZERO, WRITE MESSAGE AND EXIT
         ST     R0,APISTADL             SAVE GETVIS AREA LENGTH
         ST     R1,APISTADA             SAVE LDCB ADDRESS
         LR     R3,R1                   USED FOR ADDRESSING
* --------------------------------------------------------------------*
*        GENERATE LDCB                                                *
* --------------------------------------------------------------------*
         SPACE
         LIBRDCB FUNC=GEN,                                            *
               AREA=(3),                                              *
               CONT=YES,                                              *
               ERRAD=APIERR00
* --------------------------------------------------------------------*
*        INVOKE STATE MEMBER SERVICE                                  *
* --------------------------------------------------------------------*
         SPACE
         BAL   R10,APISTM00           CHECK MEMBER AVAILABILITY
* --------------------------------------------------------------------*
*        FREE LDCB SPACE                                              *
* --------------------------------------------------------------------*
         SPACE
APISTA95 EQU   *
         L     R0,APISTADL            RELOAD GETVIS AREA LENGTH
         L     R1,APISTADA            RELOAD GETVIS AREA ADDRESS
         SPACE
         FREEVIS ADDRESS=(1),LENGTH=(0)
         SPACE
         LTR   R15,R15                FREEVIS RC ZERO
         BNZ   APIERR20               NO MESSAGE EXIT
         SPACE
```

```
* ---------------------------------------------------------------------*
*        EOJ AND DUMP EXITS                                            *
* ---------------------------------------------------------------------*
         SPACE
APIMONEX EQU   *
         EOJ
         SPACE
APIMONDP EQU   *
         JDUMP
SPACE* * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * *
*                                                                      *
*         EXAMPLE      LIBRM STATE MEMBER                              *
*                                  MEMBER NAME = GENERIC               *
*                                  MEMBER TYPE = PROC                  *
*                                                                      *
* * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * *
         SPACE
APISTM00 EQU   *
         LIBRM SHOWCB,CB=MBST,,CBLEN=LEN1
         L     R4,LEN1                 SET LENGTH OF ONE ENTRY
         ST    R4,APISAVIL             SAVE DIRINF LENGTH
APISTM10 EQU   *
         L     R3,APISTADA             RELOAD LDCB ADDRESS
         L     R4,APISAVIL             RELOAD DIRINF LENGTH
         SPACE
         LIBRM STATE,                                                  *
               ENTITY=MEMBER,                                          *
               LDCB=(3),                                               *
               LIB=APILIB,                                             *
               SUBLIB=APISUBLB,                                        *
               MEMBER=APIMEMBN,                                        *
               TYPE=APIMEMTY,                                          *
               DIRINF=APIDIRNF,                                        *
               DIRINFL=(4),                                            *
               DIRNO=APIDIRNO,                                         *
               CONT=YES,                                               *
               NOTFND=APISTM30,                                        *
               EROPT=RET,                                              *
               ERRAD=APIERR00
         SPACE
         ST    R15,APISAVSC            SAVE SEVERITY CODE
         ST    R0,APISAVRC             SAVE REASON CODE
         L     R6,APISTMMT             MESSAGE TABLE FOR STATE MEMBER
         BAL   R7,APICRC00             CHECK RETURN CODE
         SPACE
* ---------------------------------------------------------------------*
*        ANALYZE RETURNED INFORMATION                                  *
* ---------------------------------------------------------------------*
         SPACE
         LA    R2,APIDIRNF
         USING INLCMBST,R2       ADDRESSABILITY
APISTM20 EQU   *
         SPACE
*        ..... .........
*        ..... .........   P R O C E S S Returned Entry
*        ..... .........
*        ..... .........
*        ..... .........
*        ..... .........
         CLI   APISAVSC+3,X'04'        SEVERITY CODE EQUAL 4
         BNER  R10                     NO, RETURN TO CALLER
         B     APISTM10                YES, GET CONTINUATION
         SPACE
         DROP  R2
         SPACE
APISTM30 EQU   *
         L     R6,APISTMMT             MESSAGE TABLE FOR STATE MEMBER
         BAL   R7,APICRC00             CHECK RETURN CODE
*        ..... .........   SETUP FOR NEXT CALL OR EXIT
         B     APIMONEX
         SPACE
* ---------------------------------------------------------------------*
*        CHECK RETURN CODES FROM SERVICES                              *
*        INPUT : REGISTER 7  LINK REGISTER                             *
*                REGISTER 6  MESSAGE TABLE ADDRESS                     *
*                REGISTER 15 SEVERITY CODE FROM SERVICE                *
*                REGISTER 0  REASON CODE FROM SERVICE                  *
* ---------------------------------------------------------------------*
         SPACE
APICRC00 EQU   *                       CHECK RETURN CODES
         SLR   R4,R4
         LR    R5,R4
         IC    R5,2(R6)                GET MAX SEVERITY CODE
         CR    R15,R5                  SEVERITY CODE WITHIN LIMIT
         BH    APICRC30                NO
         IC    R4,3(R6)                GET MAX REASON CODE
         CR    R0,R4                   REASON CODE WITHIN LIMIT
         BH    APICRC30                NO
         LR    R2,R6                   ADDRESS TABLE ENTRY
         LA    R2,8(R2)                POINT TO FIRST MESSAGE ENTRY
         AR    R2,R15                  CORRECT DISPLACEMENT FOR SC
         MVC   APICRFL1,0(R2)          GET FLAG BYTE
         NI    APICRFL1,X'0F'          CLEAR BIT 0-3
         SLR   R3,R3                   CLEAR WORK REGISTER
         IC    R3,APICRFL1             LOAD POSSIBLE CORRECTION
         AR    R2,R3                   CORRECT TABLE DISPLACEMENT
         AR    R2,R0                   CORRECT DISPLACEMENT FOR RC
         TM    0(R2),RCINV             INVALID SEVERITY/REASON CODE
         BO    APICRC30                YES
         TM    0(R2),RCCON             CAN LIVE WITH RC
         BZ    APICRC25                YES
         LA    R7,APIMONEX             NO, EXIT AFTER MESSAGE
APICRC25 EQU   *
         L     R1,0(R2)                MESSAGE TEXT ADDRESS
         BAL   R8,APIERR94             WRITE MESSAGE
         BR    R7                      RETURN
         SPACE
APICRC30 EQU   *
         LA    R1,APIMSG08             GET MESSAGE ADDRESS
         CVD   R15,APIRCPAK            TRANSLATE SEVERITY CODE
         UNPK  21(2,R1),APIRCPAK+6(2)  UNPACK
         OI    22(R1),C'0'             CORRECT SIGN BYTE
         CVD   R0,APIRCPAK             TRANSLATE REASON CODE
         UNPK  27(2,R1),APIRCPAK+6(2)  UNPACK
         OI    28(R1),C'0'             CORRECT SIGN BYTE
         L     R3,0(R6)                GET STRING ADDRESS FROM TABLE
         MVC   4(11,R1),0(R3)          SERVICE STRING TO MESSAGE
         B     APIERR92                WRITE MESSAGE AND EXIT
         SPACE
* ---------------------------------------------------------------------*
*        ERROR HANDLING FOR RETURN CODE GT 12                          *
* ---------------------------------------------------------------------*
         SPACE
APIERR00 EQU   *
         ST    R15,APISAVSC            SAVE SEVERITY CODE
         LR    R15,R0                  GET REASON CODE
         LA    R1,APIMSG03             MESSAGE TEXT ADDRESS
         CLI   APISAVSC+3,RC16         IS THIS RC 16
         BE    APIERR90                YES
         LA    R1,APIMSG04             MESSAGE TEXT ADDRESS
         CLI   APISAVSC+3,RC20         IS THIS RC 20
         BE    APIERR90                YES
         LA    R1,APIMSG05             MESSAGE TEXT ADDRESS
         CLI   APISAVSC+3,RC32         IS THIS RC 32
         BE    APIERR92                YES
         LA    R1,APIMSG06             MESSAGE TEXT ADDRESS
         L     R15,APISAVSC            RELOAD RETURN CODE
         B     APIERR90                UNEXPECTED RETURN CODE
         SPACE
* ---------------------------------------------------------------------*
*        GETVIS AND OTHER ERROR EXITS                                  *
* ---------------------------------------------------------------------*
         SPACE
APIERR10 EQU   *                       NOT ENOUGH GETVIS FOR EXECUTION
         LA    R1,APIMSG01             MESSAGE TEXT ADDRESS
         B     APIERR90
APIERR20 EQU   *                       FREEVIS RETURN CODE NOT ZERO
         LA    R1,APIMSG02             MESSAGE TEXT ADDRESS
         B     APIERR92
APIERR30 EQU   *                       NUMBER OF DIRECTORY ENTRIES ZERO
         LA    R1,APIMSG07             MESSAGE TEXT ADDRESS
         B     APIERR92
         SPACE
* ---------------------------------------------------------------------*
*        WRITE MESSAGE ON SYSLOG                                       *
* ---------------------------------------------------------------------*
         SPACE
APIERR90 EQU   *
         CVD   R15,APIRCPAK            TRANSLATE RETURN CODE
         UNPK  35(3,R1),APIRCPAK+5(3)  UNPACK
         OI    37(R1),C'0'             CORRECT SIGN BYTE
APIERR92 EQU   *
         LA    R8,APIMONDP             SET CONTINUATION
APIERR94 EQU   *
```

```
         STCM  R1,7,APIERCCW+1          TO ERROR CCW
         LA    R1,APIERCCB              LOAD CORRESPONDING CCB
         SPACE
         EXCP  (1)                      WRITE MESSAGE
         WAIT  (1)                      WAIT FOR I/O COMPLETE
         BR    R8                       CONTINUE AS INDICATED
         SPACE
* ---------------------------------------------------------------------*
*        REGISTER EQUATES                                              *
* ---------------------------------------------------------------------*
         SPACE
R0       EQU   0
R1       EQU   1
R2       EQU   2
R3       EQU   3
R4       EQU   4
R5       EQU   5
R6       EQU   6
R7       EQU   7
R8       EQU   8
R9       EQU   9
R10      EQU   10
R11      EQU   11
R12      EQU   12
R13      EQU   13
R14      EQU   14
R15      EQU   15
         SPACE
* ---------------------------------------------------------------------*
*        SAVE AREA DECLARATIONS                                        *
* ---------------------------------------------------------------------*
         SPACE
         DS    0F
APIMONS1 DC    XL72'00'                 MONITOR TASK SAVE AREA
         SPACE
APISAVSC DC    F'0'                      SEVERITY CODE FROM REG. 15
APISAVRC DC    F'0'                      REASON CODE FROM REG. 0
RC16     EQU   X'10'                     SEVERITY CODE 16
RC20     EQU   X'14'                     SEVERITY CODE 20
RC32     EQU   X'20'                     SEVERITY CODE 32
APISTADA DC    F'0'                      GETVIS LDCB ADDRESS
APISTADL DC    F'0'                      GETVIS LDCB LENGTH
APISAVIL DC    F'0'                      SAVE DIRINF LENGTH
LEN1     DC    F'0'                      TEMP FIELD
         SPACE
* ---------------------------------------------------------------------*
*        WORKFIELDS                                                    *
* ---------------------------------------------------------------------*
         SPACE
         DS    0D
APIRCPAK DC    D'00'                     CVD AREA
         SPACE
* ---------------------------------------------------------------------*
*        PARAMETER FIELD DEFINITIONS FOR STATE MEMBER                  *
* ---------------------------------------------------------------------*
         SPACE
APILIB   DC    CL7'LIB0001'            REQUESTED LIBRARY
APISUBLB DC    CL8'SUBLIB01'           REQUESTED SUBLIBRARY
         SPACE
APIMEMBN DC    XL8'00'                 REQUESTED MEMBER NAME
APIMEMTY DC    XL8'00'                 REQUESTED MEMBER TYPE
         SPACE
APIDIRNF DC    128F'0'                 DIRECTORY INFORMATION
APIDIRNL EQU   *-APIDIRNF              LENGTH OF DIRINF AREA
APIDIRNO DC    F'0'                    NUMBER OF RETURNED DIRENTRIES
         SPACE
APINAMPR DC    C'PROC'                 USED FOR MEMBER TYPE
         SPACE
* ---------------------------------------------------------------------*
*        MESSAGE TABLES                                                *
* ---------------------------------------------------------------------*
         SPACE
         DS    0F                        ALIGNEMENT
APISTMMT EQU   *                         STATE MEMBER MESSAGE TABLE
         DC    XL4'00000C08'             MAX SEVERITY/REASON CODE
         DC    A(APISTMST)               STATE MEMBER CHARACTER STRING
         DC    AL1(X'00'),AL3(APISTMM1) SC 00 RC 00
         DC    AL1(X'44'),AL3(APISTMM2) SC 00 RC 04
         DC    AL1(X'08'),AL3(APISTMM3) SC 04 RC 00
         DC    AL1(X'48'),AL3(APISTMM4) SC 04 RC 04
         DC    AL1(X'40'),AL3(APISTMM5) SC 08 RC 00
         DC    AL1(X'40'),AL3(APISTMM6) SC 12 RC 00
         DC    AL1(X'40'),AL3(APISTMM7) SC 12 RC 04
         DC    AL1(X'40'),AL3(APISTMM8) SC 12 RC 08
         SPACE
APISTMST DC    CL11'STATE MEMB '        CHARACTER STRING FOR MESSAGE
APICRFL1 DC    X'00'                    WORKFIELD FOR MESSAGE FLAG
RCINV    EQU   X'80'                    INVALID SEVERITY/REASON CODE
RCCON    EQU   X'40'                    CAN'T CONTINUE WITH THIS RC
         SPACE
* --------------------------------------------------------------------*
*        ERROR MESSAGE DEFINITIONS                                    *
* --------------------------------------------------------------------*
         SPACE
         DS    0D
APIERCCB CCB   SYSLOG,APIERCCW
APIERCCW CCW   X'09',APIMSG01,X'20',L'APIMSG01
         SPACE
APIMSG01 DC    C'API MON :     GETVIS FAILED   RC = XXX               '
APIMSG02 DC    C'API MON :    FREEVIS FAILED   RC = XXX               '
APIMSG03 DC    C'API  EXT.SYS ERROR  RC = 16   FB = XXX               '
APIMSG04 DC    C'API  INT.SYS ERROR  RC = 20   FB = XXX               '
APIMSG05 DC    C'API  ACCESS CONTROL RC = 32   (PREECEDING MSG L163I) '
APIMSG06 DC    C'API  UNDEFINED  RETURN CODE   RC = XXX               '
APIMSG07 DC    C'API STATE MEMBER : NO DIRECTORY INFORMATION RETURNED '
APIMSG08 DC    C'API             : SC=XX RC=XX  UNEXPECTED OR INVALID '
         SPACE
APISTMM1 DC    C'API STATE MEMB SC=00 RC=00 INFORM.STORED IN DIRINF   '
APISTMM2 DC    C'API STATE MEMB SC=00 RC=04 DIRINF NOT SPECIFIED      '
APISTMM3 DC    C'API STATE MEMB SC=04 RC=00 DIRINF TOO SMALL CONT=YES '
APISTMM4 DC    C'API STATE MEMB SC=04 RC=04 DIRINF TOO SMALL CONT=NO  '
APISTMM5 DC    C'API STATE MEMB SC=08 RC=00 MEMBER NOT FOUND          '
APISTMM6 DC    C'API STATE MEMB SC=12 RC=00 SUBLIBRARY DOES NOT EXIST '
APISTMM7 DC    C'API STATE MEMB SC=12 RC=04 LIBRARY DOES NOT EXIST    '
APISTMM8 DC    C'API STATE MEMB SC=12 RC=08 CHAIN DOES NOT EXIST      '
         SPACE
* --------------------------------------------------------------------*
*        LIBR MEMBER STATUS INFORMATION BLOCK (DSECT)                 *
* --------------------------------------------------------------------*
         SPACE
         INLCMBST
         END
```

Example of OPEN/GET/CLOSE Requests

Figure 49. Code Example for Librarian OPEN/GET/CLOSE Requests

```
* * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * *
*         LIBRM OPEN/CLOSE/GET EXAMPLE                                  *
* * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * *
         SPACE
APIMON00 START X'78'
*        ....                          ESTABLISH ADDRESSABILITY
*        .... Same as in STATE Member Request
*        ....
* --------------------------------------------------------------------*
*        INVOKE GET MEMBER SERVICE                                    *
* --------------------------------------------------------------------*
         SPACE
         BAL   R10,APIRDM00           READ A MEMBER
         SPACE
* --------------------------------------------------------------------*
*        FREE LDCB SPACE                                              *
* --------------------------------------------------------------------*
         SPACE
         L     R0,APISTADL            RELOAD GETVIS AREA LENGTH
         L     R1,APISTADA            RELOAD GETVIS AREA ADDRESS
         SPACE
         FREEVIS ADDRESS=(1),LENGTH=(0)
         SPACE
         LTR   R15,R15                FREEVIS RC ZERO
         BNZ   APIERR20               NO MESSAGE EXIT
         SPACE
* --------------------------------------------------------------------*
*        EOJ AND DUMP EXITS                                           *
* --------------------------------------------------------------------*
         SPACE
APIMONEX EQU   *
         EOJ
         SPACE
```

```
APIMONDP EQU   *
         JDUMP
         SPACE
* * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * *
*         EXAMPLE      LIBRM GET MEMBER                                *
*                            GET AT LEAST 3 RECORDS                    *
* * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * *
         SPACE
APIRDM00 EQU   *
         SPACE
* ---------------------------------------------------------------------*
*        MODIFY LDCB FOR GET MEMBER REQUEST                            *
* ---------------------------------------------------------------------*
         SPACE
         LIBRDCB FUNC=MOD,                                             *
               AREA=(3),                                               *
               LIB=GETLIB,                                             *
               SUBLIB=GETSUBLB,                                        *
               MEMBER=GETMEMBN,                                        *
               TYPE=GETMEMTY,                                          *
               DIRINF=GETDIRNF,                                        *
               DIRINFL=GETDIRNL,                                       *
               BUFFER=GETBUFER,                                        *
               BUFSIZE=GETBUFLN,                                       *
               RECNO=0,                                                *
               UNITS=0,                                                *
               EROPT=RET,                                              *
               ERRAD=APIERR00
         SPACE
* ---------------------------------------------------------------------*
*        LIBR OPEN TYPEFLE=INPUT                                       *
* ---------------------------------------------------------------------*
         SPACE
APIOPN00 EQU   *
         LIBRM OPEN,                                                   *
               TYPEFLE=INPUT,                                          *
               NOTFND=APIOPN20,                                        *
               LDCB=(3)
         SPACE
         L     R6,APIOPNMT             MESSAGE TABLE FOR OPEN
         BAL   R7,APICRC00             CHECK RETURN CODE
         SPACE
         LA    R2,GETDIRNF       POINT TO MEMBER DIRECTORY INFO
         USING INLCMBST,R2       SET ADDRESSABILITY
         SPACE
*               MEMBER INFORMATION CAN BE RETRIEVED FROM THE
*               INFORMATION STORED IN DIRECTORY INFORMATION FROM
*               OPEN
*        MVC   ........,MBSTNORL       SAVE NO OF RECORDS
*        MVC   ........,MBSTRLEN       SAVE LOGICAL RECORD LENGTH
*        MVC   ........,MBSTRCFM       SAVE RECORD FORMAT
         DROP  R2                      RELEASE INLCMBST
         SPACE
         B     APIGET00                GET FULL OR PART OF MEMBER
APIOPN20 EQU   *
         L     R6,APIOPNMT             MESSAGE TABLE FOR OPEN
         BAL   R7,APICRC00             CHECK RETURN CODE
*        ..... .........   SETUP FOR NEXT CALL OR EXIT
         B     APIMONEX
         SPACE
* ---------------------------------------------------------------------*
*        LIBR GET                                                      *
* ---------------------------------------------------------------------*
         SPACE
APIGET00 EQU   *
         LA    R2,1                    SET UNITS TO 1 BYTE/RECORD
         SLR   R4,R4                   SET STARTING BYTE/RECORD
         SPACE
         LIBRM GET,                                                    *
               LDCB=(3),                                               *
               RECNO=(4),                                              *
               UNITS=(2),                                              *
               MOVELEN=GETMOVEL
         SPACE
         L     R6,APIGETMT             MESSAGE TABLE FOR GET
         BAL   R7,APICRC00             CHECK RETURN CODE
         SPACE
APIGET10 EQU   *
*        ..... .........               COMPARE RECORD CONTENTS
*        ..... .........               OR SAVE CONTENTS FOR
*        ..... .........               LATER USE
         SPACE
```

```
* ---------------------------------------------------------------------*
*        LIBR CLOSE                                                    *
* ---------------------------------------------------------------------*
         SPACE
APICLO00 EQU   *
         LIBRM CLOSE,                                                  *
               LDCB=(3),                                               *
               COMMIT=YES
         SPACE
         L     R6,APICLOMT             MESSAGE TABLE FOR CLOSE
         BAL   R7,APICRC00             CHECK RETURN CODE
         BR    R10                     RETURN
         SPACE
* ---------------------------------------------------------------------*
*        CHECK RETURN CODES FROM SERVICES                              *
*        INPUT : REGISTER 7  LINK REGISTER                             *
*                REGISTER 6  MESSAGE TABLE ADDRESS                     *
*                REGISTER 15 SEVERITY CODE FROM SERVICE                *
*                REGISTER 0  REASON CODE FROM SERVICE                  *
* ---------------------------------------------------------------------*
         SPACE
APICRC00 EQU   *                       CHECK RETURN CODES
*
*    as in "Example of a STATE Member Request" on page 131
* ---------------------------------------------------------------------*
*        ERROR HANDLING FOR RETURN CODE GT 12                          *
* ---------------------------------------------------------------------*
         SPACE
APIERR00 EQU   *
*
*    as in "Example of a STATE Member Request" on page 131
* ---------------------------------------------------------------------*
*        GETVIS AND OTHER ERROR EXITS                                  *
* ---------------------------------------------------------------------*
         SPACE
*
*    as in "Example of a STATE Member Request" on page 131
* ---------------------------------------------------------------------*
*        WRITE MESSAGE ON SYSLOG                                       *
* ---------------------------------------------------------------------*
         SPACE
*
*    as in "Example of a STATE Member Request" on page 131
* ---------------------------------------------------------------------*
*        REGISTER EQUATES                                              *
* ---------------------------------------------------------------------*
         SPACE
*
*    as in "Example of a STATE Member Request" on page 131
*
         SPACE
* ---------------------------------------------------------------------*
*        SAVE AREA DECLARATIONS                                        *
* ---------------------------------------------------------------------*
         SPACE
*    as in "Example of a STATE Member Request" on page 131
         SPACE
* ---------------------------------------------------------------------*
*        WORKFIELDS                                                    *
* ---------------------------------------------------------------------*
         SPACE
*    as in "Example of a STATE Member Request" on page 131
         SPACE
* ---------------------------------------------------------------------*
*        PARAMETER FIELD DEFINITIONS FOR STATE MEMBER                  *
* ---------------------------------------------------------------------*
         SPACE
*    as in "Example of a STATE Member Request" on page 131
         SPACE
* ---------------------------------------------------------------------*
*        PARAMETER FIELD DEFINITIONS FOR GET MEMBER                    *
* ---------------------------------------------------------------------*
         SPACE
GETLIB   DC    CL7'LIB0002'           REQUESTED LIBRARY
GETSUBLB DC    CL8'SUBLIB02'          REQUESTED SUBLIBRARY
         SPACE
GETMEMBN DC    CL8'MEMBER01'          REQUESTED MEMBER NAME
GETMEMTY DC    CL8'PROC    '          REQUESTED MEMBER TYPE
         SPACE
GETDIRNF DC    256X'00'               DIRECTORY INFORMATION
GETDIRNL EQU   *-GETDIRNF             LENGTH OF DIRINF AREA
GETMOVEL DS    F                      RECORDS/BYTES TRANSFERED
```

```
GETBUFER DC    10XL80'00'              BUFFER TO CONTAIN DATA
GETBUFLN EQU   *-GETBUFER              BUFFER LENGTH
         SPACE
* --------------------------------------------------------------------*
*        MESSAGE TABLES                                               *
* --------------------------------------------------------------------*
         SPACE
*    similar to “Example of a STATE Member Request” on page 131
         SPACE
* --------------------------------------------------------------------*
*        ERROR MESSAGE DEFINITIONS                                    *
* --------------------------------------------------------------------*
*
*    similar to “Example of a STATE Member Request” on page 131
* --------------------------------------------------------------------*
*        LIBR MEMBER STATUS INFORMATION BLOCK (DSECT)                 *
* --------------------------------------------------------------------*
         INLCMBST
     END
```

Processing Macros with the ESERV Program

Assembler programs that were assembled with the DOS/VSE Assembler could use two types of source books (macros): A and E. Since the DOS/VSE Assembler has been replaced by the High Level Assembler, E-Deck processing is different. E-Deck refers to a macro (source book) of type E. The High Level Assembler cannot create E-Decks but can process existing E-Decks through an exit provided by z/VSE. The exit is described in detail under “Using the High Level Assembler Library Exit for Processing E-Decks” on page 139.

All macro definitions of type E have been preprocessed by the DOS/VSE Assembler; they are said to be edited. Edited macros cannot be updated directly, they must first be converted to A-Books.

You can use the ESERV program to convert an edited macro (E) back to source format (A): this is called de-editing. If the output is to be cataloged directly as source book of type A, you can specify the GENCATALS control statement. This causes a catalog statement for the Librarian (LIBR) program to be generated before each macro and a /* card after the last macro. If the GENCATALS control statement is not specified after the // EXEC ESERV statement, GENCATALS is assumed.

You can de-edit and update a macro in a single job stream by including the necessary update control statements. The following job stream example converts a macro of type E back into a macro of type A (de-editing), updates it, and catalogs the macro as source book of type A into the sublibrary specified. A listing of the updated macro on SYSLST is also created.

```
* $$ JOB JNM=EXAMPL,CLASS=C,DISP=D
* $$ PUN DISP=I
// JOB EXAMP1
// OPTION DECK
// EXEC ASMA90,SIZE=ASMA90
   PUNCH '// JOB EXAMP2'
   PUNCH '// EXEC LIBR'
   PUNCH 'ACCESS S=lib.sublib'
   END
/*
// EXEC ESERV
   DSPCH E.macroname
   .
   .
   update control statements
   .
   .
/*
/&
* $$ EOJ
```

Note:

1. The statement

```
* $$ PUN DISP=I
```

creates automatically a new job with the statements following it and places the created job into the VSE/POWER reader queue for processing.

2. With no

   ```
   update control statements
   ```

   included, the job stream just converts the specified macro of type E into type A and catalogs it into the sublibrary specified.

z/VSE System Control Statements describes the ESERV control statements available. For ESERV information on de-editing and updating macro definitions, you can also consult the *Guide to the DOS/VSE Assembler*

High Level Assembler Considerations

Starting with VSE/ESA 2.1, the DOS/VSE Assembler is no longer part of VSE. It has been replaced by the High Level Assembler.

Starting with z/VSE 4.1 HLASM 1.5 is shipped. This version of HLASM does not use work files anymore, instead virtual/partition storage is used. This requires assemblies to run in partitions of at least 6 MB (BG). For a Supervisor assembly you need 60 MB.

The statement

```
// EXEC ASMA90....
```

calls the High Level Assembler. A complete statement is shown below under "Using the Exit" on page 139. Refer also to z/VSE Planning for additional details about calling the High Level Assembler.

Since the High Level Assembler itself cannot process E-Decks, z/VSE provides an exit (EDECKXIT) for this purpose. If your programs use E-Decks (created by the DOS/VSE Assembler), you have to activate this exit when calling the High Level Assembler as discussed in detail below.

Using the High Level Assembler Library Exit for Processing E-Decks

The High Level Assembler cannot process E-Decks (as the DOS/VSE Assembler could). For that reason, z/VSE provides a library exit for the High Level Assembler to enable the processing of E-Decks. Once activated, the High Level Assembler branches to it whenever it has to process a macro call. The exit analyses the macro and in case of an E-Deck calls the ESERV program. ESERV changes, line by line, the E-Deck definitions back into source code which the High Level Assembler can then process.

Tailoring the Exit

The library exit is a general interface to the High Level Assembler for macro processing. The specialized exit for E-Deck processing is shipped as phase EDECKXIT in PRD1.BASE. If you want to modify the exit or add new functions, you can use skeleton **SKEDECKX** which is stored in VSE/ICCF library 59. The skeleton, as shipped, only establishes the interface to the ESERV program and provides support for E-Deck conversion. The skeleton includes detailed comments on how to use it and explains the setup and the working of the exit.

Using the Exit

To activate the exit, you must specify the name of the exit, EDECKXIT, in the EXEC statement calling the High Level Assembler:

```
// EXEC ASMA90,SIZE=(ASMA90,64K),PARM='EXIT(LIBEXIT(EDECKXIT)),SIZE(MAXC
               -200K,ABOVE)'
```

The 64K are used for loading the ESERV program. The exit establishes the interface to ESERV via phase IPKVX, which is further discussed under "Function Description of Phase IPKVX" on page 142.

E-Deck Processing:

As long as macros in E-Deck format are used by programs, the High Level Assembler must be activated with the library exit (EDECKXIT) capable of translating E-Decks back into source format.

The EDECKXIT has an invocation parameter: ORDER E/A, which is the default, or ORDER A/E.

- E/A means, that EDECKXIT searches all sublibraries of the whole LIBDEF chain of type SOURCE for the respective macro in E-Deck format. If found, the E-Deck is translated back into source format and passed as input to the High Level Assembler. If no E-Deck is found, the A-Book format of the macro is searched for.
- A/E means, that EDECKXIT searches for macro type A first and then (if not found) for type E.

Among others, the following error situations may occur:

1. EDECKXIT needs GETVIS storage for its own processing and for that reason a minimum default area of 200K is reserved. If that size is not sufficient, EDECKXIT issues the message:

   ERROR IN LIBR STATE

   In this case, increase the value of the assembler variable &GVLEN in skeleton SKEDECKX (set to 200*1024 by default) and regenerate EDECKXIT.
2. The partition is too small for an assembly run, because:

   EDECKXIT needs more GETVIS storage (see above), or the
   High Level Assembler itself needs more GETVIS storage.

   In this case, increase the partition size.

For a complete list of error messages, refer to "Error Messages Issued by EDECKXIT" on page 141.

Modifying User Macros:

The following applies when existing E-Decks need to be modified.

Macros to be modified must be in A-Book format. E-Decks to be modified must therefore first be converted into A-Books (as shown in the job stream example under "Processing Macros with the ESERV Program" on page 138). For assembly runs using EDECKXIT (because older macros are still in E-Deck format), the following must be considered when changing a macro.

- If EDECKXIT is invoked with ORDER E/A:
  1. If the macro to be changed exists only in E-Deck format, proceed as follows:
     a. Convert the E-Deck to an A-Book via the ESERV program.
     b. Delete the E-Deck, otherwise EDECKXIT will continue using it.
     c. Modify the A-Book.
  2. If the macro to be changed exists only in A-Book format:

     Modify the A-Book.
  3. If the macro to be changed exists in both formats, proceed as follows:
     a. Ensure that the E-Deck and the A-Book are logically equivalent (if not, delete the A-Book and proceed as described under 1 above).
     b. Delete the E-Deck.
     c. Modify the A-Book.
- If EDECKXIT is invoked with ORDER A/E:
  1. If the macro to be changed exists only in E-Deck format, proceed as follows:
     a. Convert the E-Deck to an A-Book via the ESERV program.
     b. Deleting the E-Deck is not absolutely required (but advised) since A/E will always select the A-Book version of the macro.
     c. Modify the A-Book.
  2. If the macro to be changed exists only in A-Book format:

Modify the A-Book.

3. If the macro to be changed exists in both formats:

   Modify the A-Book and consider deleting the E-Deck.

Error Messages Issued by EDECKXIT

EDECKXIT issues error messages in case of problems. The message text appears as part of the High Level Assembler message ASMA940U. The following list provides explanations for possible EDECKXIT error messages.

CALLED FOR WRONG EXIT TYPE: xx
: **Explanation**: EDECKXIT can only be specified as a "library exit" to the High Level Assembler. EDECKXIT was specified as the name of an exit which is not a library exit. The value given for TYPE identifies the wrong type of exit called for.

GETVIS FAILURE, RC=xx
: **Explanation**: This indicates a storage problem related to the partition GETVIS area. The value issued for RC is the return code from the GETVIS macro (in hexadecimal). If RC indicates "not enough storage", increase the partition size.

INPUT NOT "ORDER=EA" NOR "ORDER=AE"
: **Explanation**: In the call for the High Level Assembler, an invalid specification for the ORDER parameter was given.

CDLOAD OF IPKVX FAILED, RC=xx
: **Explanation**: The problem is related to the loading of phase IPKVX into the partition GETVIS area. The value issued for RC is the return code from the CDLOAD macro (in hexadecimal). Possible reasons for the error are "phase not found" or "partition GETVIS area too small".

ERROR IN LIBR STATE, RC=xx
: **Explanation**: This indicates a problem related to the Librarian API (application program interface). The value issued for RC is the return code (2 bytes) and the reason code (2 bytes) from the Librarian in hexadecimal format. A value of X'00100064' indicates a shortage of partition GETVIS storage. In this case, increase the value of the assembler variable &GVLEN in skeleton SKEDECKX (set to 50*1024 by default) and regenerate EDECKXIT.

macro NOT A VALID EDECK
: **Explanation**: The named "macro" is not a valid EDECK.

The following error situations are unlikely to occur. Contact IBM for support if you get such an error and you cannot identify its cause.

ESERV INIT FAILED, RC=xx
: **Explanation**: ESERV initialization failed. The value issued for RC originates from the INIT function of IPKVX (in hexadecimal).

ESERV LOAD FAILED, RC=xx
: **Explanation**: The loading of ESERV into the program area of the partition failed. The value issued for RC is the return code from the LOAD macro (in hexadecimal).

ESERV FIND FAILED, RC=xx
: **Explanation**: The EDECK to be processed cannot be read. The value issued for RC originates from the FIND function of IPKVX (in hexadecimal).

ESERV READ FAILED, RC=xx
: **Explanation**: The EDECK cannot be processed. The value issued for RC originates from the READ function of IPKVX (in hexadecimal).

ERROR IN STACK HANDLING
: **Explanation**: This indicates a problem related to the register save stack of this module. Probably the value set for the variable &DEPTH of EDECKXIT is too small.

FREEVIS FAILURE, RC=xx
: **Explanation**: This indicates a storage problem related to the partition GETVIS area. The value issued for RC is the return code from the FREEVIS macro (in hexadecimal).

CALLED FOR BAD REQUEST TYPE: xx

Explanation: The High Level Assembler called the exit with a request type not known to EDECKXIT. The value given for TYPE (in hexadecimal) identifies the wrong request type called for.

Function Description of Phase IPKVX

The following description provides background information useful if exit tailoring is required. Phase IPKVX has the following characteristics:

It is not reentrant.
It has an RMODE and AMODE of 24.

The phase provides the functions INIT, FIND, READ, and TERM which are activated with the statement

```
BAL  14,OFFSET(15)
```

where the contents of register 15 is the entrypoint of IPKVX (as returned by CDLOAD, for example).

INIT Function (Offset=0)

The INIT function loads ESERV into the program area of the calling partition and starts ESERV to perform initialization.

ESERV uses up the remaining program area of the calling partition. This means, that after an INIT up to the next TERM the caller must **not** load any further code into the partition's program area otherwise unpredictable results may occur.

ESERV requires two workfiles, IJSYS01 and IJSYS02. If the related logical units (SYS001, SYS002) are not assigned, the function terminates unsuccessfully.

Required Input

Register 13 must specify whether processing is to be initialized for E-Decks or F-Decks.

R13=0: E-Deck processing
R13=1: F-Deck processing

Other values are rejected.

Note: F-Decks are macros needed for NCP (Network Control Program) generation.

Return Codes for Function INIT (Register 15)

RC=4

Invalid input in Register 13. Initialization was not performed.

RC=8

ESERV could not be loaded. Register 13 contains the return code of the LOAD request.

RC=12

ESERV was loaded but could not initialize successfully. There may be several reasons, a console message is issued. Examples are:

The work files are not assigned correctly.
The program area of the partition is too small.

FIND Function (Offset=4)

The FIND function tells ESERV which macro is to be processed next. ESERV tries to locate the E-Deck format of the macro in the active LIBDEF source chain, and if successful converts the macro into source format.

Required Input

Register 13 must point to an 8-byte area containing one macro name.

Return Codes for Function FIND (Register 15)

RC=4
ESERV could not find the EDECK, or the found EDECK is not valid, or the specified macro name is not allowed. If further diagnostics are needed, ESERV must be run and its output must be analyzed.

RC=8
INIT was not run successfully before.

RC=12
ESERV experienced a run-time problem. There may be several reasons; a console message is issued in addition.

READ Function (Offset=8)

The READ function returns the next sequential line of source code of the macro being converted.

Required Input

Register 13 must point to an 80-byte area which is to receive one line of source code.

Return Codes for Function READ (Register 15)

RC=4
No more data; the last line has already been returned in a previous READ operation.

RC=8
FIND was not run successfully before.

RC=12
ESERV experienced a run-time problem. There may be several reasons; a console message is issued in addition.

TERM Function (Offset=12)

The TERM function tells ESERV to terminate.

There is no input required.

Return Codes for Function TERM (Register 15)

RC=12
ESERV experienced a run-time problem. There may be several reasons; a console message is issued in addition.

Chapter 5. Linking Programs

The basic functions of the Linkage Editor and the linking of programs for a 24-bit environment are described here. If you use enhanced functions such as the z/VSE 31-bit addressing support, see z/VSE Extended Addressability, which describes the Linkage Editor support for programs that use such functions.

Before a program can run, it must be processed by the Linkage Editor. This section briefly discusses the role of the Linkage Editor and describes how you can communicate with it through control statements.

The Linkage Editor prepares a program for execution by link editing the output of a language translator into one or more executable phases.

A program can be link edited into one or more phases and either

- stored temporarily and executed immediately (one phase only), or
- cataloged permanently and executed repeatedly.

If the phase is stored temporarily and executed immediately, the linkage editor is required again the next time the phase is to be run. If a phase is cataloged permanently, the Linkage Editor is no longer required for that phase, because the supervisor can load it directly from the sublibrary in response to an EXEC job control statement, or a FETCH or LOAD macro.

Phases are either stored temporarily or cataloged permanently, depending on the option specified in the OPTION job control statement:

```
    // OPTION LINK
```

causes the Linkage Editor to store the generated (single) phase temporarily in the VIO area for immediate execution in the same job. This phase is no longer available when the linkage editor starts processing the next phase.

```
    // OPTION CATAL
```

causes the Linkage Editor to catalog the generated phase permanently in the sublibrary specified in the current LIBDEF PHASE,CATALOG... statement for the partition in which the Linkage Editor job runs. This phase can be executed any time after the link edit run.

The Linkage Editor may run in any partition, and the phases produced by the Linkage Editor are executable in any partition. The linkage editor can be active in more than one partition concurrently without endangering the integrity of your sublibraries. This holds true even if each Linkage Editor run updates (that is, catalogs into) the same sublibrary. The Linkage Editor passes the following return codes to job control:

```
     0 = successful
     2 = warning message issued but phases are cataloged
     4 = warning or error message issued but phases are cataloged
     8 = single phase not replaced
    16 = severe error(s), phases are not cataloged
```

Note: If the linked phase contains only unresolved address constants for external symbols identified by the WXTRN assembler instruction, the Linkage Editor returns (starting with VSE/ESA 1.3.0) return code 2 (instead of return code 4 as in previous releases).

Structure of a Program

To understand the functions of the Linkage Editor, you must understand the structure of a program during the various stages of its development. The following sections discuss source books, object modules, and phases as summarized below.

Stages of Program Development

1. A set of source statements is to be processed by a language translator.

   The source statements might first be cataloged as a book into a sublibrary.
2. The output of the language translator, an object module, can be processed by the Linkage Editor.

   The language translator output might first be cataloged as a module into a sublibrary.
3. The output of the linkage editor, a phase, is either stored temporarily into the VIO (virtual input/output) area or is cataloged permanently into a sublibrary.

To catalog a source book or an object module, the Librarian must be used. A phase, however, is cataloged by the Linkage Editor directly.

Source Books

After planning an application, a programmer writes a set of source statements in a programming language. This set of source statements is processed by a language translator. The language translator assembles source statements written in assembler language, or it compiles source modules written in a high-level language (for example, COBOL, PL/I, or RPG II). The language translator transforms the source into an object module, which is in machine language.

You can either submit your source, consisting of one or more control sections (CSECTs), directly to the language translator for processing, or you can catalog it as a source book into a sublibrary for later processing by the language translator.

Using Macros

In a VSE environment two member types of macros can exist:

A
for macros in source format;

E
for edited macros which have been created by the DOS/VSE Assembler.

Since the DOS/VSE Assembler has been replaced by the High Level Assembler, the processing of edited macros is different. Refer to "Processing Macros with the ESERV Program" on page 138 for details.

Object Modules

Language translators process source code and produce output in the form of object modules. These modules need to be processed by the Linkage Editor to become executable phases. During the link-editing of a module, other modules may have to be included. If so, the Linkage Editor searches the sublibraries specified for the modules in question. In this way, sections of code that are used by a number of different programs need to be written, translated, and cataloged in object format only once.

An object module, the output of a language translator, consists of dictionary and text records as shown in Figure 50 on page 146.

For more information about these records refer to z/VSE System Control Statements.

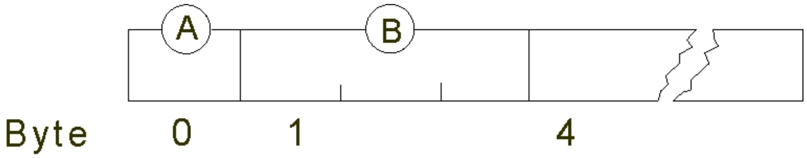


Figure 50. Record Types of an Object Module

- **A** contains X'02' and identifies the record as one of an object module.
- **B** indicates the record type and can be one of the following:

- C'ESD' - External symbol dictionary. It contains symbols defined in this module and referred to by one or more other modules and symbols referred to in this module but defined in another module.
- C'TXT' - Text. Contains actual code plus control information needed by the Linkage Editor.
- C'RLD' - Relocation list dictionary. Identifies those portions of the text which must be modified when the program is relocated for execution.
- C'END' - End of module. Indicates the end of a module. The record may contain an address where the execution is to begin (transfer address) or the length of the control section or both.

Occasionally, there might be a need to change the information contained in a TXT record. You can prepare a REP (user replace) statement and submit it with your object module for processing by the Linkage Editor. A REP statement must be submitted between the TXT record it modifies and the END record.

Cataloging Multiple Object Modules into One Member

In a Linkage Editor run, you can use the INCLUDE statement to catalog a phase composed of multiple object modules as discussed later in this chapter. You can also catalog several object modules as a single library member which is then called a multiple-object member. For details refer to “Cataloging Multiple Object Modules” on page 100.

Retrieving a Multiple-Object Member

The complete multiple-object file in an OBJ member is included by the Linkage Editor when the name of the member is specified in the INCLUDE statement. It is not possible to include only one object module of a multiple-object file.

Including Parts of Modules

If you want to catalog a phase consisting of selected control sections (CSECTs) of an object module, use the INCLUDE statement with its "namelist" parameter. In a multiple-object file, you can select control sections of the first object module only.

Phases

The Linkage Editor produces a phase from the object module(s) you identify in Linkage Editor control statements. A phase is the functional unit that the system loader can load for processing into a partition in response to a single EXEC job control statement (or a FETCH or a LOAD macro instruction in an assembler language program).

In the PHASE control statement you instruct the Linkage Editor to produce a phase of either relocatable, self-relocating, or non-relocatable type. If a phase built by the Linkage Editor includes areas defined by DS (define storage), those areas are set to zeros.

Relocatable Phases

A phase is relocatable if it can be loaded for execution in any partition. The Linkage Editor produces a relocatable phase, unless you specify an absolute origin (load) address. It is recommended that you always specify a relative origin address. A relative load address is represented by a symbol with or without a displacement.

If a relocatable phase is also designed as a re-enterable phase, it is eligible to be loaded into the shared virtual area (SVA). Phases resident in the SVA can be shared concurrently by programs running in either real or virtual mode.

Self-Relocating Phases

Before a loader with the relocating capability became available, some programmers coded self-relocating programs in order to gain the advantages of relocatability. If you have to perform maintenance on such a program, you must write this program in assembler language according to the rules described in z/VSE

System Macros User's Guide. In the PHASE control statement you indicate an origin address of +0. The program must relocate all its addresses at execution time to correspond with the addresses available in the partition in which the program is loaded.

Non-Relocatable Phases

A non-relocatable phase is link edited to be loaded at a specific location (absolute address) associated with a partition. If you request execution of a non-relocatable phase in a given partition, the starting and ending addresses of the phase must be included within that partition. Otherwise, the job is canceled. If you want to execute a non-relocatable phase in more than one partition, you must catalog a separate copy of the phase for each partition.

Year Representation

The linkage editor uses a year representation of 4 digits (retrieved from field JOBDATE in partition COMREG). Programs scanning the printout of the linkage editor might have to be adapted because of the new date format.

There is one exception to this rule. The linkage editor does not use 4 digits to display the year if the // DATE job control statement has been used, specifying only 2 digits for the year. In this case, the header lines also show a 2-digit year (from field JOBDATE in COMREG). The following changes apply:

1. One header line is printed on each new page on SYSLST when reading input from SYSLNK and printing input statements on SYSLST. This header line contains the date with a 4-digit year, instead of 2 digits followed by 2 blanks. The rest of the header line and all the following lines remain unchanged.
2. One header line is printed on each new page on SYSLST when printing the linkage editor map, displaying phase names, CSECT names, linked module names, and so on. In this header line, the date is contained in the first 8 bytes (digits and slashes), followed by just one blank before the next header text (PHASE) begins. If 4 digits are displayed for the year, the date is displayed using 10 bytes instead of 8 bytes. The next header text (PHASE) following the date is moved 2 bytes to the right.

Basic Applications of the Linkage Editor

The three basic applications of the Linkage Editor are:

- Link-editing and cataloging phases into a sublibrary;
- Link editing and executing;
- Assembling (or compiling), link editing, and executing.

Following is an overview of these applications. For detailed job stream examples refer to the section "Examples of Linkage Editor Applications".

Cataloging Phases into a Sublibrary

When you have a program, which you expect to use frequently, you should catalog it permanently as a PHASE-type member in a sublibrary. You can do this as shown in the skeleton job below:

```
(1)  // JOB CATALOG
(2)  // LIBDEF PHASE,CATALOG=library.sublibrary
(3)  // OPTION CATAL
(4)     ACTION ...
(5)     PHASE  ...
(6)     INCLUDE
        ...
        ... object
        ... module(s)
        ...
     /*
(7)     ENTRY
(8)  // EXEC LNKEDT
     /&
```

Figure 51. Job Stream to Catalog a Program Permanently into a Sublibrary

Except for the object module(s) and the /* statement, which are read from SYSIPT, all other statements are read from SYSRDR.

(1)
Job statement.

(2)
The phase produced by the Linkage Editor will be cataloged into the sublibrary specified here.

(3)
// OPTION CATAL causes the Linkage Editor to catalog the generated phase permanently in the applicable sublibrary.

Job control writes the input from the SYSRDR device into the disk extent that is assigned to SYSLNK.

(4)
ACTION is used to specify Linkage Editor options.

(5)
When a phase is to be cataloged permanently, the PHASE statement is mandatory.

(6)
An INCLUDE without an operand indicates that job control is to read one or more object modules from SYSIPT.

(7)
An ENTRY statement signals to job control the end of input for the Linkage Editor. This statement is optional.

(8)
The statement causes the Linkage Editor to be loaded into the particular partition.

A phase cataloged as SVA eligible (operand in the linkage editor PHASE statement) is loaded into the SVA, if it is cataloged in the system sublibrary IJSYSRS.SYSLIB, and:

- A phase of that name is already stored in the SVA, or
- A phase of that name has been requested to be loaded into the SVA (via the SET SDL command).

Also, if the phase has an entry in the SDL, that entry is updated.

Link-Edit and Execute

You do not always need to catalog a permanent copy of your program into a sublibrary in order to execute the program. For instance, if you have modified parts of your program and want to test these modifications, you can request the linkage editor to store the program temporarily in either a test sublibrary or in the VIO, the virtual I/O area (one phase only).

Using a Test Sublibrary

The following steps are required:

1. Create a test sublibrary with DEFINE SUBLIB=lib.sublib and REUSE=IMMEDIATE.
2. Link-edit your programs with // OPTION CATAL as often as needed (library space is reused when phases are replaced).
3. Execute the link-edited programs with LIBDEF PHASE,SEARCH=lib.sublib.
4. After completion of the test, copy the phases from the test sublibrary into the production sublibrary. This requires one link-edit run less than with the OPTION LINK approach below.
5. If required, delete the test sublibrary with DELETE SUBLIB=lib.sublib.

Using the Virtual I/O Area

Figure 52 on page 150 illustrates how the job in Figure 51 on page 148 is to be changed for such a run.

```
     // JOB TEMP
(1)  // OPTION LINK
        ACTION
        INCLUDE
        ...
        ... object
        ... module(s)
        ...
     /*
        ENTRY
     // EXEC LNKEDT
(2)  // EXEC
     /&
```

Except for the object module(s) and the /* statement which are read from SYSIPT, all other statements are read from SYSRDR.

Figure 52. Job to Link-Edit and Store a Program Temporarily for Immediate Execution

Note: No LIBDEF statement is needed, because the phase generated is not stored in a sublibrary but in the VIO, the virtual I/O area (one phase only).

(1)
// OPTION LINK indicates to the Linkage Editor that the generated phase is to be stored temporarily in the virtual I/O area for execution immediately after this job is completed. The // OPTION CATAL statement can be used instead of the // OPTION LINK statement. The program would then be cataloged permanently in a sublibrary and also executed after job completion. When // OPTION CATAL is specified, a PHASE statement is required.

(2)
// EXEC (without a program name operand) causes the program to be loaded for execution.

Assemble (or Compile), Link-Edit, and Execute

You can also combine the job steps described above with a job step for assembling or compiling your source program. This is especially useful when you are developing a program. Figure 53 on page 150 shows how your job should be set up.

```
     // JOB TEST
(1)  // OPTION LINK
(2)  // EXEC ASMA90....,GO
        ...
        ... source
        ... code
        ...
     /*
     /&
```

Figure 53. Job to Assemble, Link-Edit, and Execute a Program Stored Temporarily

Note: The statement

```
// EXEC ASMA90....
```

calls the High Level Assembler. Refer to “High Level Assembler Considerations” on page 139 for further details.

Except for the source code and the /* statement which are read from SYSIPT, all other statements are read from SYSRDR.

(1)
// OPTION LINK causes the assembler to write the object module created on SYSLNK.

(2)
GO in the EXEC statement causes the Linkage Editor to use the object module stored on SYSLNK as input for a link-edit run. Moreover, after link-editing the phase is stored temporarily in the virtual I/O area and is executed once. Without the GO parameter an additional // EXEC LNKEDT and // EXEC

statement would be required for achieving the same result. The GO parameter has the same effect when working with other VSE language translators.

For multiple assemblies (compilations), an OPTION LINK statement must precede the first EXEC statement for an assembly or compilation. This is true also when Linkage Editor control statements like INCLUDE or PHASE are used. If no LINK option is set, the GO parameter will be in effect only for the EXEC statement it appears on, and the ACTION default will be set to NOMAP (Linkage Editor control statements are described below, in "Preparing Input for the Linkage Editor", later in this section).

When you make use of the GO parameter, your executable program has to run in virtual mode, and the partition GETVIS area available to this program will be of the IBM set default size unless you have changed that value using the SIZE command.

If errors occur in one job step causing an abnormal termination, the remaining job steps are ignored. Certain Linkage Editor errors do not cause job step termination. If you do not want to execute the program when these errors occur, you may specify ACTION CANCEL after the // OPTION LINK.

Processing Requirements for the Linkage Editor

The subsequent paragraphs discuss processing requirements as follows:

- Symbolic units required
- Sublibrary definitions required

Symbolic Units Required

The Linkage Editor requires the following symbolic units:

SYSIPT
Object module input (see Note)

SYSLST
Programmer messages and listings (if SYSLST is not assigned, no map is printed and programmer messages appear on SYSLOG)

SYSLOG
Operator messages

SYSRDR
Control statement input (see Note)

SYSLNK
Input to the Linkage Editor; must not be assigned to a tape unit

SYS001
Work file. Used only if a large number of RLD items (approximately 400) is to be processed.

Note: Both SYSRDR and SYSIPT may contain input for the Linkage Editor. This input is written to SYSLNK by job control.

Sublibrary Definitions

With the LIBDEF job control statement you define the sublibrary (or chain of sublibraries) to be accessed by the Linkage Editor. The Linkage Editor has to access a sublibrary

- To retrieve object modules whenever an INCLUDE statement or the AUTOLINK function (this function is explained under “Using the AUTOLINK Function” on page 155) requires it, and
- To store and catalog a phase that has been link-edited.

When you start your job with an assemble or compile step you may have to specify for the language translator a sublibrary (or chain of sublibraries) to allow the retrieval of source books to be included in the object module.

For a detailed description of how to set up LIBDEF specifications refer to Chapter 4, “Using VSE Libraries,” on page 79.

Preparing Input for the Linkage Editor

A Linkage Editor job consists of job control statements and Linkage Editor input.

The Linkage Editor control statements direct the execution of the Linkage Editor. The statements are: ACTION, ENTRY, INCLUDE, and PHASE. The following sections describe how to prepare these control statements in context with a discussion of Linkage Editor input.

Naming a Phase

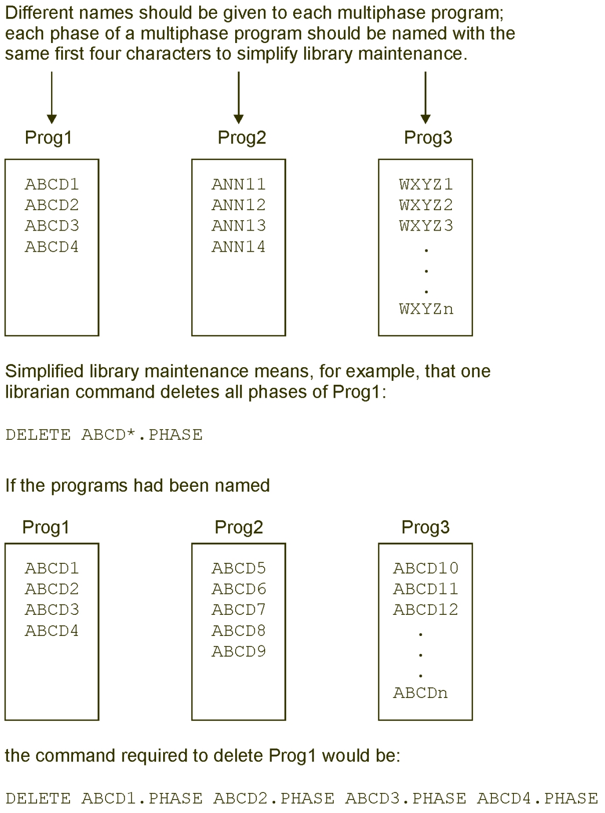
Each phase the Linkage Editor is to produce has to have a name, which you specify in the PHASE statement. When a phase is cataloged in a sublibrary, the phase name identifies that phase for subsequent retrieval. In other words, this name must be used as the operand in the EXEC job control statement or in a FETCH or a LOAD macroinstruction.

When you catalog a phase with the same name as a phase already residing in the particular sublibrary, the earlier entry with the same phase name is deleted.

For job control, all phases whose names start with the same first four characters are classified as a multiphase program. When a phase of a multiphase program is fetched, the available address space must be large enough to contain the largest of these phases even if that phase is not part of the program which is being executed. To bypass this mechanism, specify SIZE=phasename in the EXEC statement. This directs the job control program to acquire only as much space as the particular phase needs.

In choosing a name for any multiphase program, make sure that the first four characters are the same for all phases of that program but different from those of other programs. Such names simplify the deleting, displaying, punching, and copying of the entire program. Figure 54 on page 152 summarizes this recommendation.

Phase names are to be formed only from characters 0-9, A-Z, #, $, and @. Otherwise, the phase statement is invalid. The names "S", "ALL", and "ROOT" are invalid phase names. For phases which are to be cataloged, keep to the librarian program's naming conventions.



Different names should be given to each multiphase program; each phase of a multiphase program should be named with the same first four characters to simplify library maintenance.

Simplified library maintenance means, for example, that one librarian command deletes all phases of Prog1:

```
DELETE ABCD*.PHASE
```

If the programs had been named

Prog1
Prog2
Prog3
ABCD1 ABCD2 ABCD3 ABCD4
ABCD5 ABCD6 ABCD7 ABCD8 ABCD9
ABCD10 ABCD11 ABCD12 . . . ABCDn

the command required to delete Prog1 would be:

```
DELETE ABCD1.PHASE ABCD2.PHASE ABCD3.PHASE ABCD4.PHASE
```

Figure 54. Naming Multiphase Programs

Defining a Load Address for a Phase

For link-editing, you specify in the PHASE statement where your program is to be loaded for execution. You have several choices.

A phase can be link-edited to be loaded into and executed from:

- A partition's address area
- The shared virtual area (SVA)
- An absolute address.

A phase can be link edited as a relocatable phase, a self-relocating phase, or a non-relocatable phase.

The load address you specify in the PHASE statement determines the relocatability status of the link-edited phase:

- For a phase to be relocatable, that is, executable in any partition, specify a symbolic address with or without a displacement.
- For a phase which you wrote to be self-relocating, specify +0.
- For a phase to be non-relocatable, specify an absolute address.

For full details on possible load address (also called origin address) specifications, refer to z/VSE System Control Statements.

Link-Editing for Execution in Any Partition

If the Linkage Editor determines that a phase is to be relocatable, it flags the directory entry for that phase and inserts the relocation information behind the text of the phase in the sublibrary.

When a relocatable phase is link edited, it is assigned a load address relative to the beginning of the partition's address area in which the Linkage Editor was executed. When executing the phase from the same partition, relocation is not required. (This assumes that storage allocations were not changed between link-editing and executing the phase.)

Executing the phase from a different partition requires relocation by the operating system.

Link-Editing for Inclusion in the SVA

If a relocatable phase is also re-enterable, it can be included in the SVA. The Linkage Editor cannot check whether a phase is re-enterable; however, a protection check can occur when executing a phase from the SVA that modifies itself and therefore is not re-enterable.

Phases resident in the SVA can be shared concurrently by more than one partition. It is advantageous to include frequently-used phases in the SVA because these are then resident when requested for execution (they are not reloaded from a sublibrary).

To indicate that a phase should reside in the SVA, you must specify the SVA operand in the PHASE statement. You can also use the PHASE statement to PFIX a phase in the SVA to prevent paging. Refer to z/VSE System Control Statements for a detailed description of the PHASE statement.

Immediately after a phase is cataloged as SVA eligible into the system sublibrary IJSYSRS.SYSLIB, it is loaded into the SVA if this phase is either already in the SVA or (via the SET SDL command) has been requested to be loaded into the SVA. This is described in more detail under "User Options for the SVA" in Chapter 2, “Starting the System,” on page 15.

Link-Editing for Execution at an Absolute Address

If you specify an absolute address in the PHASE statement, your program can be loaded only at this address at the time of program execution. The address you specify and the phase's end address must be within the boundaries of the area allocated to the partition where you request that phase to be executed.

If you wish to force a phase to be executed in real mode, you may link edit that phase with the absolute address of a given partition's real address space.

Link-Editing a Self-Relocating Phase

You should identify a self-relocating phase by a PHASE statement with an origin point of +0:

PHASE PROGA,+0

The Linkage Editor assumes that the program is loaded at location zero, and computes all addresses accordingly. Job control recognizes such a phase and adjusts the origin address. It then gives control to the updated entry address of that phase.

Linkage Editor Input - Source and Sequence

The Linkage Editor can only read input from SYSLNK or from sublibraries. Job control, therefore, has to take care that all linkage editor input, available on SYSRDR as part of the job and on SYSIPT, is transferred to SYSLNK. All Linkage Editor control statements found on SYSRDR are transferred to SYSLNK. An INCLUDE statement without parameters is interpreted by job control to switch from SYSRDR to SYSIPT. Data records are read from SYSIPT without checking or interpretation and transferred to SYSLNK until end-of-data (/*) is reached. Then job control resumes reading from SYSRDR. The data from SYSIPT can be any linkage editor input, that is, control statements or object modules or both in any sequence.

When job control finds a // EXEC LNKEDT statement, the Linkage Editor is loaded and gets control. It starts reading and interpreting records from SYSLNK. INCLUDE statements with a module name interrupt processing from SYSLNK. The Linkage Editor then searches for the specified member of type OBJ, and if found, continues processing by reading records from this member until a record type of an object END statement is found as last member record. The member can contain an object module (single or multiple) or further linkage editor control statements or both. Up to five levels of INCLUDE "nesting" are possible. After detecting the END statement as last member record, processing is resumed at the previous member level (or at SYSLNK). The finding of an ENTRY statement at any level causes the Linkage Editor to complete processing.

The sequential mode of processing is also changed in case of several "INCLUDE ,(namelist)" statements preceding the object module. Each statement is processed completely before the next one is handled. To do that, all control statements between the INCLUDE just being processed and the beginning of the object module are skipped and the selection of the control sections for the phase is done. Then the Linkage Editor returns to the first statement skipped to handle it in the same way. If all INCLUDE statements have been processed, the Linkage Editor continues processing after the object module END statement.

Linkage Editor Storage Requirements

The storage requirements for a link edit run depend on the number of PHASE and INCLUDE statements and the number of unique ESD items processed during a link edit run.

A unique ESD item is an occurrence in the control dictionary. All symbols that appear in the MAP are unique occurrences. A symbol that occurs several times in the input stream is normally incorporated into a unique ESD item. However, if the same symbol occurs in different phases (for example, control sections), each resolved occurrence of the symbol within a different phase is a unique ESD item.

The following table shows approximate values for phases and unique ESD items that can be handled by the Linkage Editor running in a given partition size.

Table 4. Partition Size and Related Number of Phases and ESD Items

Partition Size	**128 KB**	**256 KB**	**512 KB**	**1 MB**	**2 MB**
Phases	5	10	20	50	50
Unique ESD Items and Includes	200	1400	3900	12800	32000

With partition sizes below 512KB the I/O buffer size is reduced which increases the number of I/O operations which in turn decreases performance. The maximum number of unique ESD items that the Linkage Editor can handle is approximately 32000.

Using the AUTOLINK Function

For each phase the automatic library look-up function (referred to as AUTOLINK) collects any unresolved external references and attempts to resolve them. An external reference is an ER item in the control dictionary that has not been matched with an entry point. AUTOLINK searches the current phase search chain until a cataloged object module with the same name as the external reference is found. When found, the module is included in the phase (autolinked). This retrieved module must have an entry point matching the external reference in order to resolve its address.

The following examples show how the AUTOLINK feature works.

Assume that a sublibrary contains the following:

```
Module Name     Entry Names     External References
   A            A, B, C
   D                                   A
   E                                   B
   F                                   A, C
```

Following are some examples:

If you specify INCLUDE D in your Linkage Editor input stream A is autolinked (included with module D) because the external reference A is also a module name in the sublibrary accessed.

If you specify INCLUDE E, module A is not autolinked because the external reference B does not relate to a module name. Additionally, you must also specify INCLUDE A, so that the external reference B can be resolved. No autolink is required.

If you specify INCLUDE D and INCLUDE E, then module A is autolinked by module D, and the external reference B in module E can then be resolved.

If you specify INCLUDE F, then module A is autolinked as a result of the reference to A, and the reference to C is also resolved.

Suppressing the AUTOLINK Feature

You can suppress the AUTOLINK feature as follows:

- By specifying NOAUTO in a PHASE statement, AUTOLINK is canceled for that phase only.
- By specifying NOAUTO in the ACTION statement, AUTOLINK is canceled for the remaining execution of this Linkage Editor step, starting with the current phase.
- By writing a weak external reference (WXTRN), you can cancel AUTOLINK for one symbol.

You can do this in assembler language by specifying for example:

```
  DC    A(LABEL)
  WXTRN     LABEL

    or

  DC    V(LABEL)
  WXTRN     LABEL
```

For more information, refer to the assembler language publications.

NOAUTO can be used to force an object module into a specific phase within an overlay structure. For example, the four phases of the program shown below have a V-type address constant called PETE, but in the overlay structure you want the coding for PETE included only in the third phase:

```
  PHASE PROGA,*,NOAUTO
  PHASE PROGB,*,NOAUTO
  PHASE PROGC,*
  PHASE PROGD,*,NOAUTO
```

causes PETE to be included in PROGC only.

Specifying Linkage Editor Helps

You can specify that the Linkage Editor helps you avoid certain problems in your programs or isolate problems if they occur. The actions discussed below are MAP and CANCEL, which may be specified as operands of the ACTION statement.

Obtaining a Storage Map

You can obtain a Linkage Editor storage map and a listing of Linkage Editor error diagnostics, which assist you in determining the reasons for particular errors in your program. If SYSLST is assigned, ACTION MAP is the default. You can specify ACTION NOMAP if you are not interested in this service of the Linkage Editor.

Refer to z/VSE Diagnosis Tools for a description of the Linkage Editor map.

Terminating an Erroneous Job

The Linkage Editor may encounter errors that cause the link job to be terminated with return code 16. Other errors may occur that allow processing to continue, unless you specify CANCEL in an ACTION statement. In this case, the linkage editor job is canceled.

Designing an Overlay Program

The nature of virtual storage normally makes it unnecessary to write programs in an overlay structure, because partitions can be allocated enough virtual storage to accommodate very large programs. Designers of complex application programs might want to use the overlay programming technique nevertheless.

Note: If you plan to write overlay programs, consider first the use of 31-bit addressing and related macros such as CDLOAD to avoid overlay programs altogether. Refer to z/VSE Extended Addressability and z/VSE System Macros Reference for details about this support.

Overlay programs consist of control sections organized in an overlay tree structure. An example of an overlay tree structure is shown in Figure 55 on page 157. This structure does not imply the order of execution, although the root phase is normally the first to receive control.

The manner in which control should be passed between control sections is discussed later under "Using FETCH and LOAD Macros".

Relating Control Sections to Phases

After organizing the control sections of your program into an overlay tree structure, you must prepare a corresponding set of Linkage Editor control statements. Next, link-edit your complete overlay program in a single job step, and conversely, do not include in this job step any phases that are not related to the overlay. Otherwise, the Linkage Editor might be unable to resolve external references correctly.

Figure 56 on page 157 is an example of the job stream that ensures the overlay tree structure shown in Figure 55 on page 157.

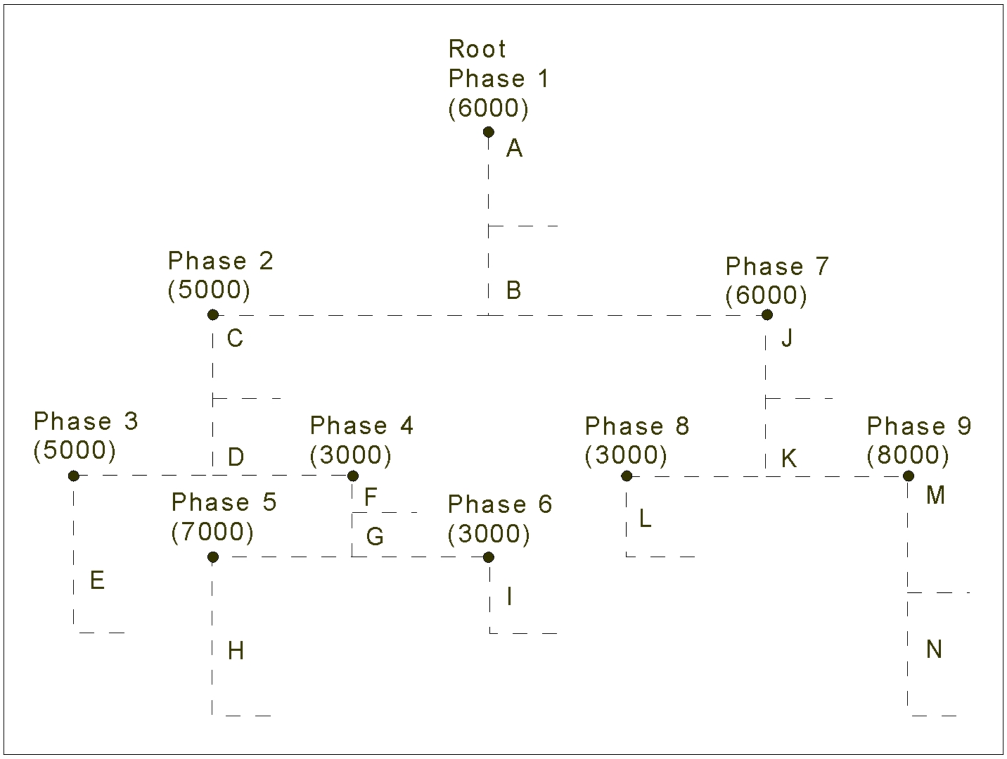


Figure 55. Overlay Tree Structure

The letters A through N represent control sections, which are organized to form nine phases in one program. The root phase resides in storage during the entire execution of the program. The remaining phases can overlay each other during execution.

You must guarantee a partition size that is equal to the longest combination of phases that can possibly reside in storage together, namely, phases 1, 2, 4, and 5, which total 42,000 bytes. If the program had not been organized in an overlay structure, it would have required an address space of 92,000 bytes.

```
// JOB OVERLAY
// OPTION CATAL
   INCLUDE
   PHASE   PHASE1,ROOT            PHASE1 stays in storage during
   INCLUDE ,(CSECTA,CSECTB)       execution of the entire program.
   PHASE   PHASE2,*               PHASE2 is to be loaded
   INCLUDE ,(CSECTC,CSECTD)       immediately behind PHASE1.
   PHASE   PHASE3,*               Since PHASE3 needs PHASE2, PHASE3
   INCLUDE ,(CSECTE)              is not allowed to overlay PHASE2.
   PHASE   PHASE4,PHASE3          PHASE4 will occupy the same
   INCLUDE ,(CSECTF,CSECTG)       storage locations as PHASE3.
   PHASE   PHASE5,*               PHASE5 will be loaded
   INCLUDE ,(CSECTH)              immediately behind PHASE4.
   PHASE   PHASE6,PHASE5          PHASE6 will be loaded at the
   INCLUDE ,(CSECTI)              same address as PHASE5.
   PHASE   PHASE7,PHASE2          PHASE7 will be loaded at the
   INCLUDE ,(CSECTJ,CSECTK)       end of the root phase.
   PHASE   PHASE8,*               PHASE8 will be loaded at the
   INCLUDE ,(CSECTL)              end of PHASE7.
   PHASE   PHASE9,PHASE8          PHASE9 will overlay
   INCLUDE ,(CSECTM,CSECTN)       PHASE8.
       (Object modules containig CSECTs A through N)
/*
// EXEC LNKEDT
/&
```

Figure 56. Link-Editing an Overlay Program

Using LOAD and FETCH Macros

During execution, an overlay program uses LOAD or FETCH macros to request that a subsequent phase be brought into the partition.

Use a LOAD macro in a phase that is to remain in control after the requested phase is brought into the partition.

Use a FETCH macro if you want the requested phase to gain control immediately after it is brought into the partition. If a phase loaded by the FETCH macro is relocatable, it will be relocated if necessary. You cannot issue a FETCH macro for a self-relocating phase.

Refer to z/VSE System Macros Reference for a detailed description of these macros.

Pseudo-Register Support

In PL/I, programmers can use pseudo-registers to define storage that will not be reserved in the load module but can be allocated dynamically during execution. The external dummy sections generated by the High Level Assembler correspond to the pseudo-registers of PL/I.

Implementation Details

The High Level Assembler implementation of pseudo-registers provides two instructions (DXD and CXD) and a special address constant (Q-type). An external dummy section is defined, if a

DXD instruction (specifying length), or a
DSECT instruction (specifying length and structure of the section)

is coded together with a Q-type address constant referring to the label of the DXD or DSECT instruction.

External dummy sections (pseudo-registers) defined in this way, are accumulated by the Linkage Editor to a single, consecutive storage block. The Linkage Editor provides the total cumulative length of this block at the fullword position of the CXD instruction. The displacement of each single, external dummy section within the storage block is placed at the locations of the related Q-type constants. Refer also to the corresponding assembler and compiler documentation.

Note that external dummy sections are accumulated for the complete Linkage Editor job. This is also the case if more than phase is generated.

The Linkage Editor receives the necessary information in ESD entries provided for DXD and DSECT statements and in RLD entries provided for CXD statements and Q-type address constants:

- The ESD entries are of type X'06' and contain name, length, and an alignment requirement of the external dummy section.
- The RLD entries for Q-type constants have type B'10', the RLD entries for CXD fields have type B'11' and a zero R-pointer.

If two or more external dummy sections for different source modules have the same name, the linkage editor uses the most restrictive alignment and the largest section to compute the total length. A doubleword alignment is more "restrictive" than fullword alignment; a fullword alignment is more "restrictive" than halfword alignment; and so on.

Further Characteristics

- External dummy sections can coexist with other ESD types of the same name (such as control sections, entry definitions, or external references).
- Multiple CXD instructions are allowed. The pseudo-register cumulative length value is stored in all CXD fields.
- If a displacement is too large to fit into the Q-constant (which can be defined as 1 to 4 byte field) or the cumulative length is beyond 31-bit addressability, an error message is issued.

Note:

1. The design of this z/VSE Linkage Editor function (including the layout of ESD and RLD entries) follows closely the MVS Linkage Editor.
2. The interface between the object code and the z/VSE Linkage Editor (ESD or RLD type) is identical to that between the High Level Assembler and the MVS Linkage Editor.

Coding Example

The following example is for an external dummy section in High Level Assembler code.

```
CSECT1    START
            .
            .
          GETVIS  ,LENGTH=CUML  Start address of the pool
                                of external dummy sections
                                is returned in register 1
          L     8,DISP1
          L     9,DISP2
            .
            .
          L     2,OFFSET1(8,1)  Access to offset within
                                1. external dummy section
          ST    2,OFFSET2(9,1)  Access to offset within
                                2. external dummy section
            .
            .
EXTDS1    DXD   19C             1. ext. dummy sect. (length 19 bytes)
EXTDS2    DXD   20F             2. ext. dummy sect. (length 20 words)
DISP1     DC    Q(EXTDS1)       LE loads X'0' into DISP1
DISP2     DC    Q(EXTDS2)       LE loads X'14' into DISP2 (FW alignment)
CUML      CXD                   LE loads X'64' into CUML (cumulative length)
          END
```

Support of Named Common Control Sections

Common control sections (common areas) are control sections used to reserve "common storage" for shared use between phases. This common storage is reserved in the partition in front of the phase with the lowest start address of the linked phases; usually at the beginning of the partition. Common sections are defined in the source module by the assembler instruction COM (or directly by the compilers). If the COM instruction has a label, the common area is referred to as named, otherwise it is blank or unnamed. In addition to blank common areas, named common areas are supported as well.

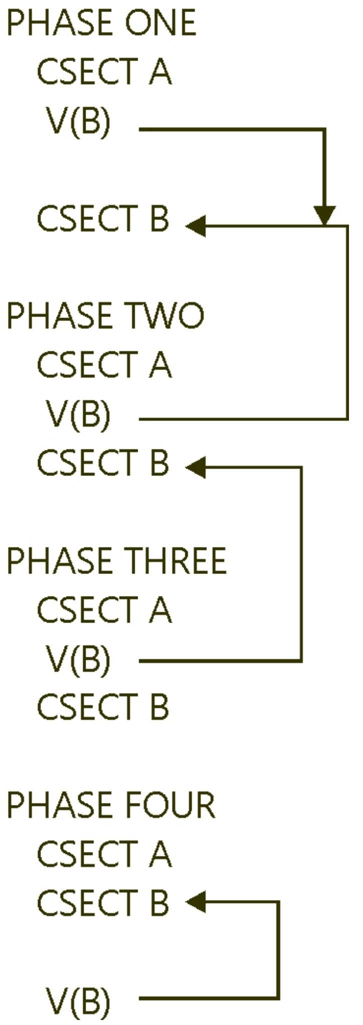
A control section which has the same name as a common area must have at least the same length.

The z/VSE Linkage Editor accepts ESD records as produced by the High Level Assembler for common sections.

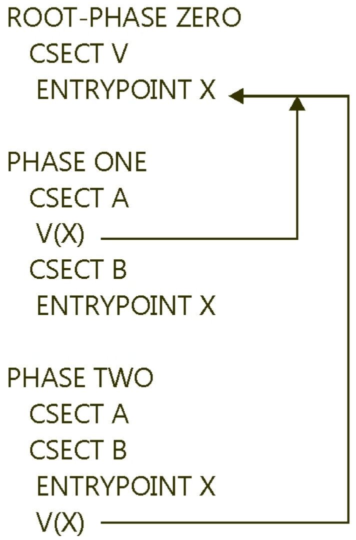
How External References are Resolved

The Linkage Editor allows the inclusion of the same control section (CSECT) within each of several phases of a multiphase link edit. If a CSECT appears in a ROOT phase, it does not appear in any other phase (this does not apply to CSECTs that begin with the letters IBM). A duplicate CSECT within the same phase is ignored.

The following examples show how external references are resolved, depending on whether or not a ROOT phase exists. The first example shows how external references are resolved without a ROOT phase:

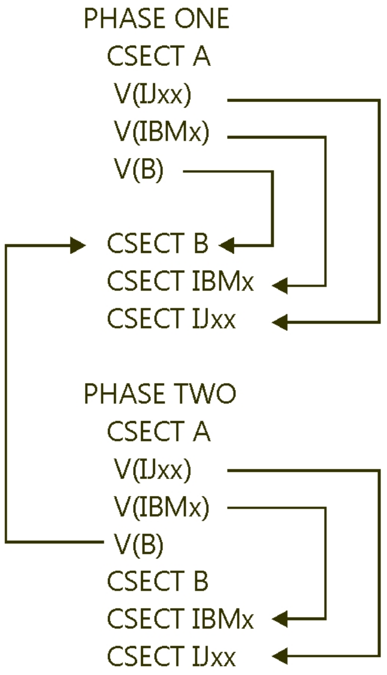


The second example shows the resolution of external references with a ROOT phase:



Privileged external references (names beginning with the letters IJ or IBM) are always resolved within the current phase or the ROOT phase. If this is not possible, the resolution will be attempted at the end of the phase via the AUTOLINK function (if NOAUTO is specified, the IJ or IBM prefix is not privileged). The other previously defined phases are not examined for possible resolution. If an external reference does not match the name of a module in the sublibraries to be searched, it will be an unresolved external reference.

The following example shows the resolution of privileged external references:



Examples of Linkage Editor Applications

The Linkage Editor examples on the following pages illustrate the use of and relation between Linkage Editor and job control statements.

Catalog a Phase into a Sublibrary

```
     // JOB CATALPHA
(1)  // LIBDEF PHASE,CATALOG=YOURLIB.YSUB1,TEMP
(1)  // LIBDEF OBJ,SEARCH=(YOURLIB.YSUB1,MYLIB.MSUB1),TEMP
(2)  // ASSGN SYSLNK,190
(3)  // OPTION CATAL
(4)     PHASE PROGB,*,SVA
(5)     INCLUDE
      Object deck
     /*
      INCLUDE SUBRX
      INCLUDE SUBRY
      INCLUDE
      Object deck
     /*
(6)  // EXEC LNKEDT
     /&
```

This example illustrates the cataloging of a single phase composed of multiple object modules. These modules are located in the input stream and in a sublibrary.

(1)

These are temporary library definitions that override existing permanent definitions. A LIBDEF PHASE,CATALOG=... statement (temporary or permanent) is always required since no default sublibrary exists for cataloging phases. The LIBDEF OBJ,SEARCH=... statement defines the sublibraries to be searched for object modules.

Label information for the libraries containing the required sublibraries must have been stored in the label information area or DLBL and EXTENT statements and librarian DEFINE commands must precede the LIBDEF statements.

(2)

The statement is required, unless SYSLNK is permanently assigned. If the statement is included, it must precede the OPTION statement (3).

(3)

The OPTION CATAL statement indicates that any phase generated in the subsequent link-edit run is to be cataloged permanently.

(4)

Only one PHASE is produced. It is cataloged into the sublibrary defined and can be retrieved by the name PROGB. The specified origin *, indicates that this phase begins at the starting address of the partition plus the length of the partition save area, and the COMMON pool (if any). The SVA operand indicates that the phase is SVA-eligible.

If the phase is cataloged into the system sublibrary, the following applies: If the phase PROGB is already loaded in the SVA or has been requested (via the SET SDL command) to be loaded into the SVA, PROGB is loaded into the shared virtual area immediately after it is cataloged into the system sublibrary. If it is a private sublibrary, as in this example, a phase is not automatically loaded into the SVA but the SET SDL command must be used.

Note: The COMMON pool is used, for example, by FORTRAN programs to store data shared by multiple programs.

(5)

Four object modules make up this phase. The first and last are not cataloged. Job control reads them from SYSIPT and writes them onto SYSLNK. Each of these object modules must be followed by /*. SUBRX and SUBRY are cataloged in the sublibraries defined in the LIBDEF OBJ,SEARCH statement.

(6)

The EXEC LNKEDT statement causes the Linkage Editor program to be loaded. SYSLNK, the input to the Linkage Editor, now contains:

```
PHASE PROGB,*,SVA
First uncataloged deck
INCLUDE SUBRX
INCLUDE SUBRY
Second uncataloged deck
ENTRY
```

When link-editing is completed, the Linkage Editor catalogs the phase.

The example can be modified to illustrate a link-and-execute operation as shown in the following paragraphs.

Link-Edit and Execute Example

```
     // JOB LINKEXEC
(1)  // OPTION LINK
(2)       PHASE PROGA,*
(3)       INCLUDE
        object deck
     /*
(4)  // EXEC LNKEDT
(5)       Any job control statement required for execution
          such as ASSGN or label statements
(6)  // EXEC
          input data as required
     /*
     /&
```

This example illustrates the link-editing and execution of a single phase that is constructed from a single object module contained in punched cards. No assignments are necessary because the system units and sublibraries required for link-editing are assumed to be permanently assigned.

(1)

The statement indicates that a link-edit operation without cataloging is to be performed. Using the data on SYSLNK as input, the Linkage Editor generates executable code and stores this code temporarily in the virtual I/O area for immediate execution.

(2)

The phase that is built by the Linkage Editor starts at the beginning of the partition plus the length of the save area plus the length of the area assigned to the COMMON pool (if any).

(3)

Since the INCLUDE statement has no operands, job control reads the records from SYSIPT and writes them on SYSLNK until SYSIPT has an end-of-data (/*) record. The data on SYSIPT is expected to be an object module in card image format.

(4)

The statement causes the Linkage Editor program to be loaded.

Using the data on SYSLNK as input, the Linkage Editor generates executable code and stores this code temporarily in the virtual I/O area for immediate execution. No ACTION options are specified. Therefore, when resolving external references, if any, the Linkage Editor uses its AUTOLINK function. Error diagnostics and a storage map are written on SYSLST.

(5)

Because the program is not cataloged, it must be executed immediately. Any pertinent job control statements are entered at this point.

(6)

An EXEC statement with no program name operand indicates that the phase to be executed was just link-edited. Therefore, no search of a sublibrary for a phase is required. The program is brought from the VIO area into partition storage and control is transferred to it. The example can be modified to illustrate a compile (assemble)-and-execute operation as shown in the following paragraphs.

Compile and Execute Example

```
     // JOB COMPEXEC
(1)  // OPTION LINK
(2)       PHASE PROGA,S
(3)  // EXEC FCOBOL
        COBOL source statements
     /*
(4)     INCLUDE SUBRX
```

```
(4)     INCLUDE
        object module
     /*
(5)     ENTRY BEGIN1
     // EXEC LNKEDT
        Any job control statements required for PROGA
        execution
     // EXEC
        Any input data required for PROGA execution
     /*
     /&
```

This example illustrates the compiling (assembling), link-editing, and execution of a single phase constructed of multiple object modules. All three possible sources of object module input to the Linkage Editor are used: SYSIPT, a sublibrary, and the output from a language translator. It is assumed that only sequential disk files or unlabeled tape files are processed. All necessary assignments are considered as permanent.

(1)

The statement indicates that a link-edit operation without cataloging is to be performed.

(2)

A specification of S as the origin causes the generated phase to start at the first doubleword in the partition following the partition save area, and the area assigned to the COMMON pool (if any). This gives the same effect as the specification of an asterisk (*).

(3)

The appropriate language translator is called (in this case, DOS/VS COBOL). The normal rules for compiling are as follows; the source deck must be on the unit assigned to SYSIPT and the /* defines the end of the source data. The output of the language translator is written on SYSLNK.

Note: Some compilers generate under certain conditions additional dummy PHASE cards. If so, the output cannot be processed with OPTION LINK (which allows only the processing of single phases).

(4)

The statement is written to SYSLNK.

INCLUDE without an operand causes job control to read from SYSIPT up to the next /* statement and to write the records to SYSLNK.

(5)

The ENTRY statement is written on SYSLNK as the last linkage editor control statement. The symbol BEGIN1 must be either the name of a CSECT or a label definition that occurs in an ENTRY source statement defined in the first and only phase. The address of BEGIN1 becomes the transfer address for the program. The ENTRY statement is used to provide a specific entry point rather than to use the point specified in the program.

The remaining statements follow the same pattern as discussed in the Link-Edit and Execute example.

If certain types of errors are detected during compilation of a source program, the LINK option is suppressed. Under these circumstances the EXEC LNKEDT and EXEC statements are ignored and the message 'STATEMENT OUT OF SEQUENCE' results. This should be kept in mind if a series of programs is to be compiled and cataloged as a single job. Failure of one job step would cause failure of all succeeding steps.

This problem can be handled, however, by using conditional job control. The processing of such a job stream can then be controlled by making the processing of subsequent job steps dependent on the return code passed by the Linkage Editor in a previous job step. For details refer to the section "Using Conditional Job Control" in Chapter 3, “Controlling Jobs,” on page 33.

Chapter 6. Using VSE Facilities and Options

This section discusses ways and means for monitoring certain activities of the system. This involves the coding of program exit routines and of user programs to be used as IPL and job control exit routines and the coding of a job accounting interface routine. In addition, this chapter discusses the checkpointing facility, DASD switching and designing programs for virtual mode execution.

User-Written Exit Routines

Program Exit Routines

If required, the supervisor can permit user routines to gain control when any of the following types of events occurs:

- Interval Timer Interrupt (IT)
- Program Check Interrupt (PC)
- Abnormal Termination (AB)
- Operator Communication Interrupt (OC)
- Page Fault Handling Overlap

Both the supervisor and the problem program that contains the user routine must have the proper code to establish an interface.

A problem program that wants to utilize the options must contain code to set up the interface. For the events IT, PC, AB, and OC the STXIT macro is to be used. For page fault handling overlap, the SETPFA macro is available (further discussed under “Page Fault Handling Overlap Exit” on page 166).

Figure 57 on page 165 is a summary of the supervisor determined conditions for which an exit routine may be coded and the operand to be coded in the STXIT macro. The STXIT macro and its operands are discussed in detail in z/VSE System Macros Reference.

Operand of the STXIT Macro
Condition

AB
Abnormal termination of the problem program.

IT
Interval timer external interrupt.

OC
Operator communications interrupt.

PC
Program check interrupt.

Figure 57. Summary of Program Exit Conditions (STXIT Macro)

Short descriptions of the support for each of the types of program exit routines follow, indicating the associated problem program macros. For information on how multitasking affects this support and what happens if multiple events coincide, refer to z/VSE System Macros Reference. Some high-level languages offer similar facilities, for details of which see the appropriate programmer's guide.

Interval Timer Exit

Suppose you want to cancel a job at a certain time if it has not already completed. Code the STXIT to set up the interface of your exit routine with the supervisor; use the SETIME macro to set a time interval. When that interval elapses, an interval timer interrupt occurs and control is given to your user routine. The user routine need not be entered immediately. For instance, if the user routine is in the background partition, and a foreground partition is active, the user routine will not be entered until the background partition becomes active.

To find out the time remaining in an interval, a program can issue the TTIMER macro instruction. The supervisor then loads this value in general register 0. This macro can also be used to cancel the remaining time in the interval.

Program Check Exit

Programs can establish linkage from the supervisor to a user program-check exit routine by coding an STXIT macro. If a program check occurs within the program, the supervisor gives control to the user routine instead of discontinuing the program. The user routine can analyze the program check and choose to ignore, to correct, or to accept it.

If the check is ignored or if the exit routine can correct the error condition, the routine can request via the EXIT PC macro that processing of the main line program continues.

If the problem cannot be resolved, the program check is accepted as valid. The user routine can then terminate further processing of the program by issuing a CANCEL, DUMP, JDUMP, or EOJ macro.

The ability to include a user routine to process program checks can be especially advantageous when using LIOCS. In that case, I/O housekeeping such as closing files and freeing tracks can be performed before termination of the job or task.

Abnormal Termination Exit

Programs can establish linkage from the supervisor to an abnormal termination exit routine by issuing an STXIT AB macro.

The macro allows a user routine to get control from the supervisor before an abnormal end-of-job condition discontinues the processing of the program. The user routine normally ends with one of the termination macros (CANCEL, DUMP, JDUMP or EOJ) to terminate the problem program and to return control to the supervisor, rather than by initiating the continuation of the problem program.

Operator Communications Exit

Programs can provide a routine for handling external interrupts from the operator. This support is useful in a number of applications, for example:

- To let the operator indicate that a required action has been taken.
- To allow the operator to communicate with CICS to start and stop activities on certain communication lines or terminals, or to invoke diagnostic procedures.

The external interrupt that links to an OC user exit routine is caused by entering the MSG command. Refer to z/VSE System Control Statements for further details.

Page Fault Handling Overlap Exit

A user routine can continue processing during the time a page fault is being handled by the system, provided this page fault occurs in the same task and not in a supervisor routine invoked by this task. This support is of interest only for programs executed in virtual mode and making use of user-developed subtasking rather than IBM-supplied multitasking.

Such programs may issue the SETPFA macro instruction to establish linkage from the page management routines in the supervisor to a user routine, called the page fault appendage routine. Linkage can be established for only one task per partition. The usage of the SETPFA macro is described in z/VSE System Macros Reference.

Writing an IPL Exit Routine

The IPL Exit allows you to do some processing at the end of IPL and prior to execution of the job control program. You might want to check the IPL options included, for example, whether the support for job accounting or access control is activated.

Note: IPL exit routines are restricted to a 24-bit environment.

Before you start coding your exit routine, take account of any system requirements that should be met at the time the routine is to be executed. The exit routine and any routines that are called by your routine must be present in the system sublibrary IJSYSRS.SYSLIB.

Observe the following conventions for the exit routine:

- Register 15 contains the entry point of the routine.
- Register 14 contains the return address to job control.
- The format of the PHASE statement must be:

```
    PHASE $SYSOPEN,*
```

After IPL, the job control program executes the exit routine as an overlay phase; an area of 4 K has been reserved for the exit routine. While the routine is being executed, the job control program is unable to read any job control statements.

In your exit routine, you can issue SVCs and perform I/O operations to SYSLOG and/or SYSRES. To do so, you can only use the EXCP macro. Any use of LIOCS or of a DTFPH would obstruct proper execution of the job control program.

Phase $SYSOPEN is executed with a storage protect key of zero. If the phase is abnormally terminated, the job control program is loaded for execution.

Figure 58 on page 168 illustrates a user-written routine that is executed once each time the IPL procedure is performed.

```
* THIS PROGRAM CHECKS WHETHER THE INSTALLATION INCLUDES
* JOB ACCOUNTING SUPPORT. IPL WITHOUT ACTIVATING JOB
* ACCOUNTING IS CONSIDERED AS NOT ALLOWED.
* A MESSAGE INFORMS THE OPERATOR WHY HE/SHE HAS TO
* REPEAT IPL. THEN A HARD WAIT IS FORCED.
*
        START    0
        USING    *,R15
BEGIN   ST       R14,RETURN                SAVE RETURN ADDRESS
        COMRG    REG=R2
        TM       56(R2),X'80'              JOB ACCOUNTING SUPPORTED?
        BZR      R14                       YES, RETURN TO JOB CONTROL
        LA       R1,LOGCCB                 NO, WRITE MESSAGE TO
        EXCP     (1)                       OPERATOR
        WAIT     (1)
        L        R11,HWCODE                LOAD HARD WAIT CODE
        ST       R11,0                     STORE IT IN LOW CORE
        OI       SVCNPSW+1,X'02'           SET ON WAIT BIT
        SVC      7                         FORCE HARD WAIT
SVCNPSW EQU      96                        LOCATION OF SVC NEW PSW
LOGCCB  CCB      SYSLOG,LOGCCW,X'0400'     CCB WITH POST AT DEVICE END
LOGCCW  CCW      X'09',LOGMSG,X'20',L'LOGMSG
LOGMSG  DC       C'JOB ACCOUNTING SUPPORT NOT ACTIVATED, RE-IPL'
RETURN  DC       F'0'
HWCODE  DC       C'NOJA'
R0      EQU      0
R1      EQU      1
R2      EQU      2
R11     EQU      11
R12     EQU      12
R13     EQU      13
R14     EQU      14
R15     EQU      15
        END      BEGIN
```

Figure 58. IPL Exit Routine Example

Writing a Job Control Exit Routine

After a job control statement (or command) has been read, and before any symbolic parameters have been substituted, control can be passed to one or more user exit routines. Such a routine can examine and alter the statement (or command) before symbolic parameters are substituted, and before it is processed by job control.

As shipped, z/VSE contains dummy phase $JOBEXIT in the system library which is automatically loaded into the SVA at IPL. If you do not modify $JOBEXIT, it has no effect on your system. If you replace it by your own user-exit routine, it is activated for each control statement (command) after that statement (command) has been read by job control.

In your routine you are free to modify the operands of the job control statement and to add comments. You must not, however, modify the operation field of the statement. For example, // EXEC IBM can be modified to // EXEC USER; the operation field (EXEC) cannot be modified. In your exit routine neither perform any I/O operations nor issue any SVCs nor request the system to cancel the job step.

Link-edit your routine to the system library using a PHASE and a MODE control statement as follows:

```
  PHASE $JOBEXIT,S,NOAUTO,SVA
  MODE  AMODE(24),RMODE(24)
```

Your routine must be coded re-enterable; it must be SVA eligible, and it must reside in the SVA. The PHASE statement must include the SVA parameter. This ensures that when the phase is cataloged it will also be loaded into the SVA replacing the dummy phase provided by IBM.

Note: JCL user exit routines must be loaded into the SVA (24-Bit).

Phase $JOBEXIT is executed with a storage protection key of zero. The code is shared between partitions.

When your routine receives control, registers contain control information as shown on the following page.

Register Number
    Contents of Register

0
System identification characters 'SDOS'.

1
Address of partition communication region.

2
Address of system communication region.

3
Address of current statement's vector table entry.

4
Address of buffer that contains the currently processed job control statement.

5
Number of continuation lines if read from SYSDR, otherwise 0.

6
Anchor field. At the very first call (after IPL) JCL will load X'00000000' into register 6 before passing control to the exit routine. For all subsequent calls register 6 will contain the value that was returned from the last preceding call. This will allow an exit routine, for example, to acquire GETVIS storage and get its address saved from call to call. In case of multiple job control exit routines, a SET SDL for one single $JOBEX0n will cause all anchor fields to be re-initialized to X'00000000'.

13
Skip mode indicator:

R13=X'00000000'
Job Control will process the statement.

R13=X'000000FF'
Job control will ignore the statement (that is, job control is in skip mode, the statement does not come from SYSLOG, and the statement is not JOB, /&, /+, or /.).

14
Return address to job control.

15
Entry address of $JOBEXIT.

Prior to returning control to job control, your routine must store a return code value into register 15:

a zero value
requests job control to continue processing the current statement.

a value of X'D5D3D6C7'
requests job control to ignore the current statement. The statement will *not* be logged, neither on SYSLST nor on SYSLOG.

a value of X'C3D3D6C7'
requests job control to ignore the current statement. The statement will be logged *conditionally*, that is depending whether LOG and/or // OPTION LOG are currently in effect or not.

any other non-zero value
requests job control to ignore the current statement. The statement will be logged *unconditionally*, both on SYSLST and on SYSLOG.

The vector table whose layout is given below shows which job control statement is being processed by job control. You must not modify its contents. Use it for comparison only. Continuation lines are located in storage immediately behind the statement pointed to by register 4, and are each 80 bytes long.

In the buffer, you may modify any part of the statement, except for the operation field. After having set the return code, your routine should pass control back to job control.

Layout of the vector table:

Bytes 0 through 6:
Operation field (name of job control statement)

Bytes 7 through 13:
Internal control information

Do not attempt to modify the table or modify the operation field in the buffer.

Note: Make sure your exit routine is free of errors that could cause abnormal termination in a production environment.

The Job Control Exit Routine skeleton shown in Figure 59 on page 170 is available as member JOBEXIT in VSE/ICCF library 59.

Figure 59. Job Control Exit Routine Example

```
* $$ JOB JNM=IESJEXT,CLASS=0,DISP=D,NTFY=YES
* $$ LST CLASS=Q,DISP=H
// JOB IESJEXT ASSEMBLE
// LIBDEF *,CATALOG=IJSYSRS.SYSLIB
// OPTION CATAL
// EXEC ASMA90,SIZE=(ASMA90,64K),PARM='EXIT(LIBEXIT(EDECKXIT)),SIZE(MAXC
               -200K,ABOVE)'
 TITLE 'JCLE   $JOBEXIT - DUMMY PHASE - JCL-EXIT TO USER-ROUTINE'
***********************************************************************
*  USER EXIT FROM JOB-CONTROL AFTER A STATEMENT IS READ               *
*                                                                     *
*        RESIDENCE                                                    *
*               $JOBEXIT PHASE IS LOADED INTO THE SVA DURING IPL.     *
*               IF THIS JOB IS EXECUTED, THE PHASE $JOBEXIT IS        *
*               REPLACED IN THE SVA. TO ACTIVATE THE NEW EXIT         *
*               ROUTINE, A SET SDL COMMAND WITH $JOBEXIT,SVA HAS      *
*               TO BE ISSUED. IF MULTIPLE EXITS ARE USED, ALL EXIT    *
*               ROUTINES HAVE TO BE LOADED AND ACTIVATED.             *
*                                                                     *
*        ACTIVATE/DEACTIVATE STATUS                                   *
*               THE USER EXIT ROUTINES MAY BE ACTIVATED OR            *
*               DEACTIVATED USING THE JCLEXIT COMMAND ISSUED FROM     *
*               THE BACKGROUND PARTITION. THIS COMMAND ALLOWS ALSO    *
*               TO DISPLAY THE STATUS IF ENTERED WITHOUT OPERAND.     *
*               THE SET SDL COMMAND FOR ANY JOB EXIT ROUTINE WILL     *
*               ALSO ACTIVATE THE ROUTINE.                            *
*                                                                     *
*        FUNCTION                                                     *
*               FOR A DETAIL INFORMATION REFER TO THE 'GUIDE TO       *
*               SYSTEM FUNCTIONS' MANUAL.                             *
*                                                                     *
*               THE OPERANDS OF THE JCL STATEMENT CAN BE MODIFIED     *
*               IN THE EXIT ROUTINES TO THE USER'S CONVENIENCE.       *
*               SOME COMMENTS CAN BE ADDED IN THE LENGTH OF THE AREA  *
*               'BUFFER', WHERE THE JCL STATEMENT RESIDES. THIS AREA  *
*               HAS A LENGTH OF 121 BYTES.                            *
*               FROM THE START OF THE AREA 'BUFFER' THE LENGTH OF     *
*               71 BYTES IS PRINTED ONTO SYSLOG AND THE LENGTH OF     *
*               121 BYTES IS PRINTED ONTO SYSLST.                     *
*               EXCEPTION:                                            *
*               AT EOJ BYTE 11-71 ARE AVAILABLE FOR USER PURPOSE ONLY.*
*               THE USER CAN ISSUE A RETURN CODE:                     *
*                - REG. 15 = ZERO    : STATEMENT WILL BE PROCESSED    *
*                - REG. 15 = NON-ZERO: STATEMENT IS TREATED AS COMMENT*
*                                                                     *
*        REGISTER USAGE                                               *
*                                                                     *
*              R0  PRELOADED WITH ID.: 'SDOS'    FUTURE USE           *
*              R1  PRELOADED WITH ADDR OF PART. COMREG                *
*              R2  PRELOADED WITH ADDR OF SYSCOM                      *
*              R3  PRELOADED WITH ADDR OF JCL VECTOR TABLE            *
*              R4  PRELOADED WITH ADDR OF AREA 'BUFFER', WHERE        *
*                            THE JCL STATEMENT RESIDES                *
*              R5  NUMBER OF CONTINUATION LINES WHEN READ             *
*                            FROM SYSRDR                              *
*              R6  ANCHOR FIELD: ANY VALUE CAN BE PASSED BY           *
*                            THE USER EXIT TO JCL. IT WILL BE         *
*                            SAVED BY JCL AND BE PASSED TO            *
*                            THE USER EXIT ROUTINE AT ITS             *
*                            NEXT INVOCATION. ITS INITIAL             *
*                            VALUE WILL BE ZERO.                      *
*                            CARE SHOULD BE TAKEN SINCE               *
*                            THE USER EXIT ROUTINE CAN BE             *
*                            INVOKED ASYNCHRONOUSLY BY ALL            *
*                            PARTITIONS. THEREFORE IT IS              *
*                            RECOMMENDED TO MODIFY THE ANCHOR         *
*                            ONLY AT THE FIRST INVOCATION OF          *
*                            A EXIT ROUTINE AND LEAVE IT              *
*                            UNCHANGED AT ALL FURTHER CALLS.          *
*                            WHENEVER A 'SET SDL' COMMAND FOR ANY     *
*                            JCL EXIT ROUTINE IS ISSUED A COMPLETELY  *
*                            NEW JOB EXIT ENVIRONMENT IS BUILT UP AND *
*                            THE ANCHOR FIELDS FOR ALL EXIT ROUTINES  *
*                            ARE RESET TO ZERO.                       *
*              R14 LINK-RETURN TO JOB-CONTROL ROOT PHASE              *
*              R15 AT ENTRY: BASE-ADDRESS OF THIS PHASE               *
*                  AT EXIT:  TO BE LOADED WITH RETURN CODE            *
*                                                                     *
***********************************************************************
         EJECT
         PUNCH ' PHASE $JOBEXIT,S,SVA '
IJBJEXIT START 0
* HERE YOU CAN INSERT YOUR CODE IF YOU HAVE ONLY ONE EXIT ROUTINE
JOBEXIT  XR    15,15              ZERO VALUE MEANS NORMAL PROCESSING
         BR    14                 RETURN TO CALLER
***********************************************************************
*                                                                     *
*  INSTALLATION OF MULTIPLE JCL EXIT ROUTINES:                        *
*                                                                     *
*   A)  CATALOG YOUR JCL EXITS INTO IJSYSRS.SYSLIB AND ADD THEM TO    *
*       THE SVA LOAD LIST $SVA0000.                                   *
*                                                                     *
*   B)  IPL YOUR SYSTEM TO LOAD THE EXIT ROUTINES INTO THE SVA.       *
*                                                                     *
*   C)  RUN THIS JOB.                                                 *
*                                                                     *
*    EXAMPLE OF TWO EXIT ROUTINES:                                    *
*                                                                     *
*    IF YOU WANT JCL TO INVOKE TWO JCL EXIT ROUTINES                  *
*    YOU HAVE TO REMOVE THE PREVIOUS TWO LINES OF CODE AND            *
*    TO MODIFY THE FOLLOWING COMMENT LINES BY                         *
*      - REMOVING THE ASTERISK IN COLUMN 1                            *
*      - CHANGING THE LAST DIGIT OF THE USER EXIT ROUTINE             *
*        NAMES. VALID CHARACTERS ARE NUMBERS FROM 0 THROUGH 9.        *
*      - CHANGING THE IDENTIFIER OF THE ROUTINES TO A NAME OF YOUR    *
*        CHOICE.                                                      *
*    FOR EVERY ADDITIONAL USER EXIT TWO LINES MUST BE ADDED:          *
*      - ONE FOR THE USER EXIT ROUTINE NAME - $JOBEX0N                *
*      - ONE FOR THE IDENTIFIER OF THE USER EXIT ROUTINE              *
*                                                                     *
*    DO NOT FORGET TO REMOVE THE ASTERISK BEFORE THE IDENTIFIER       *
*    OF THE USER EXIT LIST AND DON'T CHANGE ANY LENGTH.               *
*                                                                     *
*    DO NOT FORGET TO REMOVE THE ASTERISK BEFORE THE END OF TABLE     *
*    INDICATOR.                                                       *
*                                                                     *
*                                                                     *
*                                                                     *
*                                                                     *
*                                                                     *
*                                                                     *
*                                                                     *
*                                                                     *
*                                                                     *
*                                                                     *
*                                                                     *
***********************************************************************
*        DC    CL8'JCLLUSEX'      IDENTIFIER OF USER EXIT LIST
*        DC    CL8'$JOBEX00'      USER EXIT ROUTINE NAME
*        DC    CL8'IDENTIF0'      IDENTIFIER OF THE ROUTINE
*        DC    CL8'$JOBEX01'      USER EXIT ROUTINE NAME
*        DC    CL8'IDENTIF1'      IDENTIFIER OF THE ROUTINE
*        DC    X'FFFFFFFF'        END OF TABLE
         END   IJBJEXIT
/*
// EXEC LNKEDT,PARM='MSHP'
/*
/&
* $$ EOJ
```

Multiple Job Control Exit Routines

You can also specify a list of up to **10** JCL exit routines in the phase $JOBEXIT. This means that up to 10 JCL exit routines can be invoked for each JCL statement (command).

Phase $JOBEXIT is cataloged in the system library IJSYSRS.SYSLIB. JCL checks the first eight characters of phase $JOBEXIT and interprets them as follows:

- If they are equal to **JCLLUSEX**, JCL assumes that you cataloged a list of JCL exit routines in phase $JOBEXIT. JCL then checks whether all of these JCL exit routines are stored in the Shared Virtual Area (SVA). JCL invokes each exit routine defined provided it has been activated (as described on the following pages). Refer to Figure 60 on page 172.
- If the first eight characters are not equal to JCLLUSEX, JCL calls phase $JOBEXIT as a single exit routine.

Creating a List of JCL Exit Routines

The IBM-provided source program JOBEXIT helps you create a list of JCL exit routines. Sample JOBEXIT is available as a member in VSE/ICCF library 59 for modification. You have to edit this sample, compile it, and catalog it into your system library IJSYSRS.SYSLIB. Figure 60 on page 172 shows a selected portion of sample JOBEXIT as shipped with z/VSE:

```
         PUNCH ' PHASE $JOBEXIT,S,SVA '
         START 0
         XR    15,15              ZERO VALUE MEANS NORMAL PROCESSING
         BR    14                 RETURN TO CALLER
*        DC    CL8'JCLLUSEX'      IDENTIFIER OF EXIT LIST
*        DC    CL8'$JOBEX00'      EXIT ROUTINE NAME
*        DC    CL8'IDENTIF0'      IDENTIFIER OF THE ROUTINE
*        DC    CL8'$JOBEX01'      EXIT ROUTINE NAME
*        DC    CL8'IDENTIF1'      IDENTIFIER OF THE ROUTINE
*        DC    X'FFFFFFFF'        END OF TABLE
         END
```

Figure 60. Part of Content of JOBEXIT Sample

Installing a List of JCL Exit Routines

If you want to install several JCL exit routines, you must proceed in the following sequence:

1. Catalog your JCL exit routines into system library IJSYSRS.SYSLIB and add them to the SVA load list $SVA0000. $SVA0000, initially shipped as an empty phase, is the one SVA load book, that is reserved for private use. Refer to "Automatic SVA Loading During System Startup" on page 29 for further details.

   Figure 61 on page 172 shows a sample job to catalog and add phases to an SVA load list. In this example, three JCL exit routines ($JOBEX01, $JOBEX05 and $JOBEX07) are added to $SVA0000 by means of the SVALLIST macro.

   ```
   // JOB BUILD IPL LOAD LIST
   // OPTION CATAL
   // LIBDEF PHASE,CATALOG=IJSYSRS.SYSLIB,PERM
   // EXEC ASMA90....
      TITLE '$SVA0000 - IPL LOAD LIST FOR JCL EXIT ROUTINES'
      SVALLIST $SVA0000,($JOBEX01),($JOBEX05),                            C
               ($JOBEX07)
      END
   /*
   // EXEC LNKEDT,PARM='MSHP'
   /&
   ```

   Figure 61. Creating an SVA Load List for JCL Exit Routines

2. IPL your z/VSE system. Your JCL exit routines are loaded into the SVA during IPL, while the load book $SVA0000 is being processed automatically.
3. Modify sample JOBEXIT as shown in Figure 62 on page 173. After making the required changes, assemble, catalog and load phase $JOBEXIT into the SVA.

```
         PUNCH ' PHASE $JOBEXIT,S,SVA '
         START 0
*        XR    15,15              ZERO VALUE MEANS NORMAL PROCESSING
*        BR    14                 RETURN TO CALLER
         DC    CL8'JCLLUSEX'      IDENTIFIER OF EXIT LIST
         DC    CL8'$JOBEX05'      ROUTINE NUMBER 5
         DC    CL8'ACCOUNT'       IDENTIFIER, SELECTED BY THE USER
         DC    CL8'$JOBEX01'      ROUTINE NUMBER 1
         DC    CL8'TUNING'        IDENTIFIER, SELECTED BY THE USER
         DC    CL8'$JOBEX07'      ROUTINE NUMBER 7
         DC    CL8'MY APPL '      IDENTIFIER, SELECTED BY THE USER
         DC    X'FFFFFFFF'        END OF TABLE
         END
```

Figure 62. JOBEXIT Sample with Several JCL Exit Routines

Note:

1. If you start with step 3 and omit steps 1 and 2, each job control (JCL) statement (even SET SDL) may cause a problem, since JCL tries to load the JCL exit routines, which cannot be found in the SVA.
2. The statement

   ```
        // EXEC ASMA90....
   ```

   calls the High Level Assembler. Refer to “High Level Assembler Considerations” on page 139 for further details.

Naming Convention for JCL Exit Routines

The names of the JCL exit routines must have the format **$JOBEX0***n*, where *n* is a decimal digit (from 0 - 9).

The value used for *n* influences the sequence of invocation. The JCL exit routine with the smallest digit is invoked first. The JCL exit routine with the highest digit is invoked last. Thus in Figure 62 on page 173, $JOBEX01 is invoked first even though it is the second JCL exit routine defined in sample JOBEXIT.

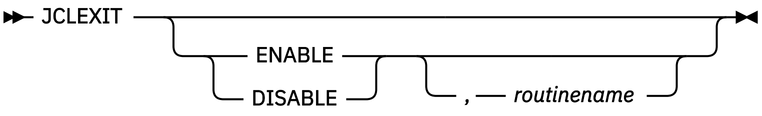
Whenever you make changes to a JCL exit routine, you must catalog it and then load it into the SVA via the **SET SDL** command. **SET SDL** can only be issued in the BG partition. To use this command permanently, modify procedure $0JCL (by using skeleton SKJCL0). Procedure $0JCL and skeleton SKJCL0 are described in z/VSE Administration.

Activating and Deactivating JCL Exit Routines

The JCL command, JCLEXIT, supports multiple JCL exit routines. With this command, you can activate or deactivate:

- A single JCL exit routine.
- All routines listed in $JOBEXIT.

Format



Following are examples for activating and deactivating individual JCL exit routines specified in $JOBEXIT:

```
JCLEXIT DISABLE,$JOBEX01
JCLEXIT ENABLE,$JOBEX05
```

You can also activate or deactivate $JOBEXIT as a whole. Depending on what you have specified in $JOBEXIT this means activating or deactivating a single JCL exit routine or a list of JCL exit routines (as shown in Figure 62 on page 173):

```
JCLEXIT DISABLE,$JOBEXIT
JCLEXIT ENABLE,$JOBEXIT
```

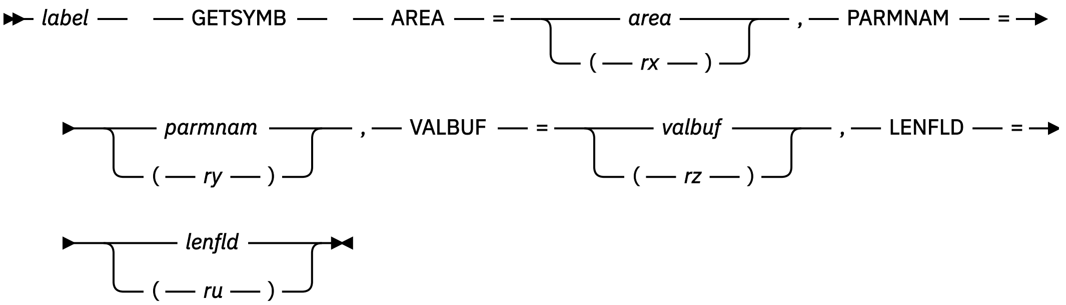
Note: If no operand is specified in the JCLEXIT command, you get a report on SYSLOG about the status (enabled or disabled) of all JCL exit routines.

Without operands JCLEXIT can be issued in any dynamic or static partition. With the operands ENABLE and DISABLE it can only be issued in the BG partition.

Resolving Symbolic Parameters in JCL Commands

Macro GETSYMB resolves symbolic parameters in the JCL commands of JCL exit routines. To resolve symbolic parameters, you first have to scan the JCL statement and isolate the symbolic parameters. You then invoke the macro GETSYMB to get the value of a symbolic parameter. GETSYMB has four required operands. They can be specified either by a symbolic address or a pointer in a Register.

Format



Parameters

As shown, you need to specify the addresses for the following:

AREA=
A work area of 100 bytes which is used as control block for saving macro call-related information.

PARMNAM=
A 7-byte field, containing the symbolic parameter name. A parameter name shorter than 7 bytes must start with the first position from the left. Unused bytes must be blank.

VALBUF=
A buffer of 50 bytes which will receive the character string that was defined in a previous SETPARM statement for the symbolic parameter name. Since this value can be up to 50 characters, the length of the buffer must be 50 bytes.

LENFLD=
A 2-byte field. The system moves the length of the value in VALBUF into this 2-byte field.

Registers 0, 1, 13, 14, and 15 are destroyed by the GETSYMB macro. Register 15 always contains the return code: return code 0 means that the request was successful and the symbol was found. Return code 10 means that the symbol was not found.

Register Conventions

Register conventions apply for every JCL exit routine. They are shown in "Writing a Job Control Exit Routine" on page 168.

Note: JCL exit routines are invoked asynchronously by all partitions. A change of the anchor field may not be meaningful if more than one partition is active. The anchor field is maintained only per exit routine, not per partition.

Handling of a Changed JCL Statement

If a JCL exit routine changes one or more operands of a JCL statement and further exit routines are to be activated, then the changed statement is passed to the remaining exit routines. If an exit routine sets a statement into "IGNORE" status (on return, Register 15 is not equal to zero), none of the subsequent exit routines will get control for this statement.

Thus it is important for you to consider the:

- Priority of the JCL exit routines in sample JOBEXIT (see “Naming Convention for JCL Exit Routines” on page 173).
- Effect of each exit routine on the statements of your job stream.

Writing a Job Accounting Interface Routine

Job accounting interface support is available for all partitions in the system. At the end of each job step or job, accounting information is accumulated in a table for that partition and can be processed by a user-written routine. This routine can extract data for such purposes as charging system usage and supervising system operation, or for planning new applications or changing the system configuration.

The routine must be relocatable, and should be SVA eligible (see Note below) for performance reasons. With the distribution volume, IBM provides a dummy phase $JOBACCT as part of the system sublibrary IJSYSRS.SYSLIB. If you decide to use the job accounting facility, you must catalog your routine into the system sublibrary. At IPL, $JOBACCT is automatically loaded into the SVA if it is SVA eligible. If not, $JOBACCT is loaded into the corresponding partition at the end of each job step. A message is issued during IPL if $JOBACCT could not be loaded into the SVA. To catalog your routine as SVA eligible, the PHASE statement must include the SVA parameter; this causes the phase, after it has been cataloged, to be loaded into the SVA, replacing the dummy phase provided by IBM.

Since the processing of this kind of information is an overhead element, the user routine should be efficient and avoid unnecessary reduction or reformatting of data. For details about the VSE/POWER job accounting support, refer to VSE/POWER Application Programming.

Note: Normally, an SVA eligible routine is programmed to be read-only and re-enterable. The job accounting interface routine is an exception. $JOBACCT runs with a PSW protection key of 0 which means it does not have to be read-only and may modify itself or may modify tables contained within itself (but not by I/O operations; I/O operations for $JOBACCT must be performed in the related partition GETVIS area). Also, it is called by job control, and the job control program serializes $JOBACCT execution. In other words, concurrent execution for more than one partition cannot happen and, therefore, the routine need not be re-enterable.

z/VSE provides skeleton SKJOBACC in VSE/ICCF library 59.

Job Accounting Information

When the support for basic job accounting is activated, a job accounting table comprising fourteen fields is included for each partition in the system. At the end of each job step and job, information is stored in fields 1 to 14 of the Job Accounting Table (see Table 5 on page 176).

SIO accounting (refer to fields 15 and 16 of the job accounting table) is performed for each partition for the devices specified during IPL. The maximum is 255 and has no relation to the number of devices specified for the total VSE system. If more devices are accessed than the number specified, SIOs on the excess devices will not be counted.

Note: The job accounting table resides below 16 MB (RMODE 24).

Note that the difference between Start and Stop times will not necessarily equal the sum of CPU, All Bound, and Overhead times. All Bound and Overhead times will vary, depending on the number of active partitions and the type of partition activity. CPU time is accurate for each partition, but it may not be reproducible. That is, the same job being executed under different system conditions (varying number of active partitions, logical transient area available, etc.) may show differences in CPU time.

Table 5. Job Accounting Table

Fld	Disp	Len	Contents
1	0-7	8	Job name. 8-byte character string taken from JOB statement.
2	8-23	16	User information.16 characters of information taken from the JOB statement.
3	24-25	2	Partition ID: BG, FB, FA, F9, and so on.
4	26	1	Cancel Code. Refer to z/VSE Messages and Codes Volume 1.
5	27	1	Type of Record. S=job step; L=last step of job.
6	28-35	8	Date when job step ended, depending on the JCL STDOPT DATE option.
7	36-39	4	Previous Job Step Stop Time. 0hhmmssF, where h=hours, m=minutes, s=seconds, F is a sign (in packed decimal format).
8	40-43	4	Job Step Stop Time (in same format as start time).
9	44-47	4	Job Step Duration in 300ths of a second.
10	48-55	8	Phase name, 8-byte character string taken from the EXEC card.
11	56-59	4	(4 K) multiplied by (number of pages referenced or PFIXed for real execution) in the current job step.
12	60-63	4	CPU Time. 4 binary bytes given in 300ths of a second. Time is calculated from exit of the user-written routine called during job control to the next entry of the routine. Time used by the user-written output routine is charged to overhead of the next record.
13	64-67	4	Overhead Time. 4 binary bytes given in 300ths of a second. Includes time taken by functions that cannot be charged readily to one partition (such as attention routine and error recovery). System overhead time is distributed to the partitions in proportion to the used CPU time.
14	68-71	4	All Bound Time. 4 binary bytes in 300ths of a second. Includes the time the system is in the wait state divided by the number of partitions running.
15	72-		SIO Tables. Variable number of bytes. Six bytes are reserved for each device accessed by the Job Step. First two bytes are X'0cuu', next four are hex count of SIOs for the Job Step. Stacker Select commands for MICR devices are not counted. Error recovery SIOs are not charged to the Job Accounting table. Devices are added to the table as they are used.
16		1	Contents: X'20'. Indicates end of SIO tables.

Programming Considerations

If physical IOCS is used for printing, you must 'space after' to prevent overwriting of job control statements.

For efficiency, an overlay structure should be avoided and the length of the program should preferably not exceed one library block.

If the job accounting program is canceled as the result of an error condition, the current information cannot be retrieved, the job accounting information for the current job step is unreliable. However, provision is made that the job accounting information for any subsequent job steps will be correct, provided the cancelation was not caused by an error in the $JOBACCT routine itself. If there was an error in the $JOBACCT routine, it must be corrected first.

In order to avoid unintentional cancelation of the job accounting program by operator action, the operator should issue the MAP command and check the job name for the running partition. If the job name is 'JOBACCT', the job accounting routine is active; the CANCEL command should not be issued until the original job name is displayed after another MAP command.

Register Usage

Important data for the user's job accounting routine are passed in the following general registers:

11	Length of the job accounting table
12	Base address for $JOBACCT
13	Address of the user save area
14	Return address to job control
15	Address of the job accounting table

If $JOBACCT uses LIOCS, the contents of general registers 14 and 15 must be saved (also registers 0 and 1, if used) because LIOCS uses these registers.

Save Area for the User's Routine

The address of a save area that can be used by the job accounting routine is passed in general register 13.

Tailoring the Program

The requirements of the program can be simply to record the accounting information as part of the SYSLST output for each job step or job, or it can be to gather information to be used for charging system usage.

If data is to be written out on a disk or tape, the save area can be used for communicating between job steps. Such information as the disk address for the next record or an indication that tape labels have been successfully processed, or even the DTF used to control the output, may be stored in the save area.

Checkpointing Facility

Note: The checkpointing facility can be used with static but not with dynamic partitions. Also, it does not support 31-bit addressing and data spaces. The macro CHKPT is canceled when issued from a partition that crosses the 16 MB line and data spaces that a program can access are not recorded during CHKPT requests. Virtual tapes (VTAPEs) cannot be used with the checkpointing facility.

The progress of a program that performs considerable processing in one job step should be protected against destruction in case the program is canceled. z/VSE provides support for taking up to 9999 checkpoint records in a job. Through this facility, information can be preserved at regular intervals and in sufficient quantity to allow restarting a program at an intermediate point. The CHKPT macro (or the corresponding high-level language statement) causes the checkpoint record to be stored on a magnetic tape or disk. For details about the CHKPT macro, refer to z/VSE System Macros Reference.

The RSTRT job control statement restarts the program from the last or any specified checkpoint that is taken before cancellation.

When a checkpointed program is to be restarted after a new IPL, the partition must start at the same location as when the program was checkpointed. Also, its end address must not be lower than at that time, unless a lower end address was specified in the CHKPT macro instruction. Unless you reestablish all linkages to SVA phases yourself, the contents and location of the modules in the SVA when restarting must also be the same as when the program was checkpointed. The SDL must be identical if the restarted program uses a local directory list. For example, one that was generated by the assembler language macro GENL.

In a program using checkpoints, avoid having linkage into the SVA at the point in the program where the CHKPT macro call is issued.

If any pages of a virtual mode program were fixed when the checkpoint record was taken, the real address area allocation for the partition must also start at the same or a lower location and its end address must be at least as high as at checkpoint time. The pages that were fixed are refixed by the supervisor when the program is restarted.

Restarting a Program from a Checkpoint

To restart a program from a checkpoint the RSTRT job control statement is used. The sequence of job control statements that must be submitted to restart a program is as follows:

1. A JOB statement specifying the job name used when the checkpoints were taken.
2. ASSGN statements, if necessary, to establish the I/O assignments for the program that is to be restarted.
3. A RSTRT statement specifying
   a. the symbolic name of the tape or disk device on which the checkpoint records are stored.
   b. the sequence number of the checkpoint record to be used for restart.
   c. for checkpoint records on disk the filename (DTF name) of the checkpoint file.
4. An end-of-job (/&) statement.

Figure 63 on page 178 shows the sequence of job control statements needed to restart a checkpointed program that ended abnormally due to, for example, a power failure. Following are the characteristics of the checkpointed program that must be considered for restart:

- The job name specified in the JOB statement was CHECKP; the same name must be used for restart.
- The checkpoint records were written on magnetic tape; therefore, no filename need be specified in the RSTRT statement.
- The symbolic device name SYS006 is used for the checkpoint file.
- The sequence number of the last checkpoint record written was 0013; this or any previous checkpoint record can be used for restart (the sequence numbers are printed by VSE/Advanced Functions on the SYSLOG device).

In reconstructing the job stream note that the // RSTRT statement physically and functionally replaces the // EXEC statement originally used.

Another important consideration is the repositioning of files on magnetic tape or disk. Assembler language users can consult z/VSE System Macros Reference, which discusses the topic in context with using the CHKPT macro. High-level language users should consider printing a file processing status record for each checkpoint that is taken during the execution of a program. This record should indicate the name of the files read or written on magnetic tape or disk when the checkpoint is taken.

```
// JOB CHECKP
// ASSGN SYS006,380          CHKPT TAPE
// ASSGN ...
// ASSGN ...
// RSTRT SYS006,0013
/&
```

Figure 63. Example of a RESTART Job

Note: If you are restarting a program that uses multi-extent disk files, always use a checkpoint that was taken on the last opened extent of the file. If you do not, the job is canceled with message

```
4n40D
```

.

Using Timer Services

The following timer services are available:

- Time-of-day clock
- Interval timer

The time-of-day clock is a hardware feature. The interval timer is a software feature which makes use of the hardware features CPU timer and clock comparator. The use of timer services is briefly discussed below. Timer services are automatically provided.

Time-of-Day Clock

The time-of-day (TOD) clock provides a consistent measure of elapsed time suitable for time-of-day indication.

The TOD clock support also enables programs to issue the GETIME macro instruction, which causes the exact time-of-day to be stored in general register 1. A description of the use of the GETIME macro instruction is given in z/VSE System Macros User's Guide.

The time-of-day and the date are automatically included with each // JOB and /& job control statement that is printed on SYSLST or SYSLOG.

During the IPL procedure, if IPL is performed from SYSLOG, a message is printed on the operator console to inform the operator of the status of the date, clock, and zone. If necessary, the operator can correct this information in the SET command.

Interval Timer

The interval timer can be used by programs (main tasks or subtasks or both) that need to schedule certain processing based on discrete time intervals. If a problem program is written with the appropriate macros and routines, the interval timer causes an external interrupt when the time limit established by the program has elapsed.

Several VSE macros relate to interval timer support. For information about using these macros, refer to z/VSE System Macros User's Guide.

DASD Sharing with Multiple VSE Systems

If your installation consists of more than one VSE system, you may consider sharing disk devices (called DASD sharing) among them. Rather than assigning a fixed number of devices to the different systems, you can combine the total number of available devices into a disk pool which is shared by all VSE systems. DASD sharing between two or more VSE systems has several advantages:

- Library maintenance is easier, if only one set of libraries needs to be maintained.
- If you run several CICS subsystems, file maintenance of CICS files becomes easier.
- Several VSE systems can share the VSE/POWER files thus distributing the batch work load.
- Direct access storage space can be saved, as only one copy of the data is required instead of multiple copies.

As long as the different VSE systems access the shared devices for reading only, the integrity of your data is preserved. If, however, data on the shared disk devices are accessed in write mode by more than one system at the same time, data integrity is no longer ensured, unless special precautions are taken. The Track Hold and DASD File Protect functions do not apply here because none of the sharing systems is aware of what the other is doing.

z/VSE provides programming support which allows to access a DASD device from different VSE systems in read and write mode. This programming support is based on the channel switching and/or the string switching feature.

Note: If a DASD sharing environment includes at least two CPUs, procedure $COMVAR must identify these CPUs. You can update this procedure with skeleton SKCOMVAR described in z/VSE Administration. Skeleton SKCOMVAR resides in ICCF library 59.

Reserving Devices for Exclusive Use

Channel command words (DEVICE RESERVE / DEVICE RELEASE) allow one I/O interface to reserve a disk drive for exclusive use. Any other I/O interface that attempts to access such a reserved disk drive will receive a 'device busy' indication.

Reserving disk devices has several disadvantages:

- An entire disk pack has to be reserved even if only a single record is to be updated. This may lead to a severe performance degradation.
- If one CPU tries to access a volume which is already reserved by another CPU, no specific indication is given that the volume is not available.
- When an application program terminates abnormally, the system does not automatically release reserved disk drives; the other VSE system(s) may have to wait indefinitely if they try to access data on the reserved disk drives.

z/VSE provides a method that avoids those risks. The sharing of data on disk is controlled on the resource level, not on the device level. This method, called "resource locking", is described in the remainder of this section.

Resource Locking

A program running under z/VSE is capable of protecting data by reserving ('locking') and releasing ('unlocking') a named resource. This resource may, for example, be a table in storage, a phase name, a disk volume identifier, or a library name.

If a job is canceled, any resources which it has locked are unlocked automatically.

Locking and unlocking occurs

- within a partition: the resource is shared between tasks belonging to the partition,
- within one computing system: the resource is shared between partitions, or
- within a multiple-CPU installation: the resource (a catalog or a file, for example) is shared between VSE systems.

Locking within one computing system is called 'internal locking', locking across systems is called 'external locking' or 'cross-system locking'. All functions provided for internal locking are available for external locking as well.

Compared with the method of reserving of entire volumes, locking by named resource offers the following advantages:

- protection can be limited to a portion of an entire volume (a file, for example);
- data can be shared, comparable to shareoptions 1 and 2 of VSE/VSAM, that is, locking is not necessarily exclusive;
- if a lock request cannot be satisfied because the corresponding resource is already under exclusive control by another task (by another VSE system perhaps), the requestor can be immediately notified.

If you are planning to switch from a one-system to a multiple-system setup and you have used the VSE/VSAM access method in the past, you do not have to change your source programs in order to utilize DASD sharing across systems. Resource protection across systems is accomplished by the VSE/VSAM open routine. For SAM files in VSE/VSAM-managed space, the open routine performs the cross-system locking, too.

If a VSE/VSAM file defined with shareoption 1 or 2 is opened for update by one program, then no other user (in another partition of the same VSE system or in another system) can open the file for update at the same time. Concurrent updating of a VSE/VSAM file defined with shareoption 4,4 is allowed for programs running in one system or in different systems. While a file is opened for update by one program, a second program running in another partition may open the file for update.

For libraries in VSE/VSAM-managed space, use shareoption 3. Refer also to “Defining a Library, Sublibrary, or a SYSRES File” on page 82.

Files of other types should be locked explicitly in order to have the file protected against concurrent update by other tasks.

IBM-supplied programs such as the linkage editor or the librarian do this locking whenever they are about to update a library. If you want to do your own resource locking, you must use the assembler language macros

- DTL, GENDTL, and/or MODDTL to define the named resource
- LOCK and UNLOCK to perform the actual locking control.

Via the resource definition macros, a resource lock control block is generated. Among other things, it defines

- the name of the resource
- the level of locking: exclusive or shared with other tasks
- the scope of locking: within one system or across systems
- the time of automatic unlocking: at the end of the job step or at end-of-job.

Note that the locking mechanism functions only if each task that shares a particular resource subjects itself to the lock control and uses one and only one name for the resource.

The following macro statement

```
EXAMPLE DTL NAME=SHAREFL,CONTROL=S,LOCKOPT=2,SCOPE=EXT
```

defines a lock control block for the resource SHAREFL. The SCOPE parameter indicates that the resource should be shared across systems. The combination of CONTROL=S and LOCKOPT=2 means: for a lock request to be granted, other tasks with a definition of CONTROL=S may have concurrent access, but not more than one task with a definition of CONTROL=E.

The LOCK macro requests access to a named resource. The requestor may specify which action the system is to take if the lock request cannot be granted. For the above DTL, the statement

```
LOCK EXAMPLE,FAIL=WAIT
```

requests access to the resource with the name SHAREFL. If the resource is locked such that no concurrent access is allowed, the requesting task should be set into the wait state until the access can be granted.

The use of the lock control macros is described in detail in z/VSE System Macros Reference and z/VSE System Macros User's Guide

Lock Communication File

Resource protection across systems requires a special system file which reflects the system-wide locking status to all the sharing systems at any time. A resource which is locked across systems will be entered by the operating system into this lock communication file (or 'lock file' for short). The disk device where this file resides must be defined to all the sharing systems by the DLF command at IPL.

There must be an agreement between the sharing systems which ensures that all systems use the same lock communication file. All systems which take part in the DASD sharing must define the disk drive where this file is located as shareable.

How to Initialize a Shared VSE Environment

To define a disk device as shareable across systems, you must include the SHR parameter in the IPL ADD command. For example:

```
    ADD 140,3390,SHR
```

All disk devices of the shared disk pool should be defined (in all sharing systems) as shareable. At least the disk drive where the lock file resides has to be defined as shareable.

If you have to add disk devices whose volume labels are non-unique within your VSE system, then problems may occur when disk devices are addressed by VOLID. For example:

```
    DLF   VOLID=SYSWK1, ...
```

If SYSWK1 exists twice in your environment, it is not predictable on which of the two volumes the lock communication file will be allocated. To make VOLID addressing unique, use the DVCDN (device down) operand when adding non-unique volumes you do not want to have accessed by VOLID addressing. For example:

```
    ADD   133,3390,SHR,DVCDN
```

You must not specify DVCDN when adding devices that will be addressed by VOLID.

The lock communication file is created as a special system file with the dedicated file name 'DOS.LOCK.FILE' via the IPL command DLF (Define Lock File). The DLF command has to be issued immediately after the ADD and DEL commands. When the DLF command is missing in the IPL procedure and at least one device ADDed as shareable, the operator is prompted for entering the DLF command. Two versions of the DLF command are available:

- a long version used to create a new lock file or to open an existing one, depending on the TYPE operand, and
- a short version to refer to an already existing lock file.

Refer to z/VSE System Control Statements for a detailed description of the DLF command, its operands, and its syntax.

You should try to place the lock file on a disk drive that is not subject to heavy I/O traffic; for example, keep it separate from files such as SYSRES, the page data set, or VSE/POWER files.

The operand DSF defines the lock file as secured or not secured. The DASD sharing support depends heavily on the availability and integrity of the lock communication file. This file should therefore be defined as a secured file.

The default size of the lock file is one cylinder on a CKD device or 80 blocks on an FBA device.

The TYPE operand specifies whether or not a new lock file is to be created. With TYPE=F (for Format), a new lock file is created every time. With TYPE=N, an existing lock file is opened, if the extent information in the DLF command matches that of the existing lock file. If the extent information does not match, the system prompts the operator to decide whether a new lock file is to be created.

The short forms of the DLF command

```
    DLF UNIT=cuu
    DLF VOLID=volser
    DLF UNIT=cuu,VOLID=volser
```

are used by the other CPUs which join the sharing environment to reference the already existing lock file. The short form may be used also by the first IPLing CPU if you want to resume with the lock file as it existed at the end of a preceding production period. On the other hand, submitting the long form for an already existing lock file is not harmful if TYPE=N is specified.

Note: During the execution of the DLF command, no other sharing system can access the lock file. Therefore, lock and unlock requests cannot be serviced. A performance degradation may be encountered on the already active systems while another (new) system is in the process of IPL.

DASD Sharing under VM

DASD sharing is also possible under VM. Disks which are defined with the multiple write feature (MWV) can be used by different VM users as shared disks (minidisks or full disk packs).

Resource sharing across systems functions properly only if each sharing (virtual) CPU is discernible by a unique CPU identification. Therefore, for any virtual machine, a different CPU identification must be defined. Before performing IPL for a virtual machine, the VM user has to define a unique CPU identification via the OPTION CPUIDxxxx in the directory or the CP command SET CPUID xxxxxx. Without this command, severe lock file errors will occur.

Special Considerations for Shared Libraries

Libraries may reside on shared disks and may be accessed by more than one CPU. IBM's Librarian and Linkage Editor programs utilize the LOCK/UNLOCK management thereby protecting libraries against concurrent write access.

The following precautions should be kept in mind:

1. If an SVA-resident phase is updated in a shared library, the update is not reflected in the SVA or SDL of the other sharing system. You have two options:
   a. Continue to work on the other system with the old copy of the particular phase.
   b. Run a SET SDL on the other sharing systems, with the appropriate phase name. This would refresh the contents of the SDL or SVA.
2. Multiple VSE/ICCF systems may not share one VSE/ICCF library, but should rather have their own dedicated VSE/ICCF library, each.
3. To ensure read integrity when sharing libraries across CPUs under VM, the SHR parameter of the IPL ADD statement must be used.

Recorder, Hardcopy, and History Files in a DASD Sharing Environment

Three system files are usually referenced by the logical unit name SYSREC:

- The recorder file (file name IJSYSRC)
- The hardcopy file (file name IJSYSCN)
- The history file (file name IJSYSHF)

The IPL DEF Command "assigns" SYSREC to a physical device. The recorder and the hardcopy file must be part of SYSREC. The history file need not reside on SYSREC. However, it is a good practice to define the history file also as part of SYSREC. For the placement of these files within a DASD Sharing environment, the following rules should be observed.

To ensure that library maintenance under control of the MSHP program is recorded in only one history file, the system standard label area of each sharing system has to contain identical DLBL/EXTENT information for the history file. The definition (DEF SYSREC=cuu) or assignment (ASSGN SYSnnn, cuu) must be for the same physical device on which the common history file resides. This enables you to do library maintenance on the shared SYSRES file and on any of the shared or non-shared private libraries without loosing track of the change status of your libraries.

For the recorder and the hardcopy file, each sharing system has to keep its own extent on the pack where SYSREC is defined. The DLBL statement must contain, for each sharing system, a unique file identifier of IJSYSRC and IJSYSCN; non-overlapping extents on the SYSREC pack must be defined in the EXTENT statement.

An Example of a Two-System Installation

The following example shows how two VSE systems are set up to share a string of IBM disk devices. Both systems run on separate processors, named Processor I and Processor II. Figure 64 on page 184 presents the configuration of disk devices.

Processor I

Number of Devices	IBM Device Type
8	3390
4	3390 (shared)

Processor II

Number of Devices	IBM Device Type
4	3390
4	3390 (shared)

Figure 64. Example of a DASD Sharing Configuration

The following files are shareable by the two systems:

- The SYSRES file
- The history file
- VSE/POWER files
- Private libraries
- VSE/VSAM catalog and files
- Other data files

Each supervisor used must be cataloged with a unique name.

Similarly, two VSE/POWER phases are generated, each with a unique name; the VSE/POWER macro must specify the SYSID and SHARED parameters. You can operate with only one VSE/POWER phase if SYSID is changed dynamically at autostart time.

If during VSE/POWER bring-up no FORMAT statement is included in the AUTOSTART file, the operator will be prompted as to whether VSE/POWER files are to be formatted or not. If the operator replies D,A or the AUTOSTART file contains a FORMAT=D,A statement, VSE/POWER asks the operator whether another system is already IPLed and whether the shared files can be formatted.

Note: VSE/POWER files should be formatted only once.

```
 *  IPL Procedure for Processor I:

    01F,$$A$SUP1,NOLOG,VSIZE=250M,VIO=512K,VPOOL=256K
    ADD 148:155,3390
(1) ADD 230:233,3390,SHR            VIA CHANNEL 2
       .
       .
    unit record devices, terminals, etc.
       .
       .
(2) DLF UNIT=231
(3) DEF SYSREC=230,SYSCAT=231
    DPD UNIT=233,CYL=450,DSF=N
    SYS ....
    SVA SDL=700,GETVIS=(768K,6M),PSIZE=(320K,6M)

 *  IPL Procedure for Processor II:

    01F,$$A$SUP1,NOLOG,VSIZE=250M,VIO=512K,VPOOL=256K
    ADD 340:343,3390
(1) ADD 330:333,3390,SHR            VIA CHANNEL 3
       .
       .
    unit record devices, terminals, etc.
       .
       .
(2) DLF UNIT=331
(3) DEF SYSREC=330,SYSCAT=331
    DPD UNIT=333,BLK=80000,DSF=N
    SYS ....
    SVA SDL=700,GETVIS=(768K,6M),PSIZE=(320K,6M)
```

Figure 65. Example of IPL Procedures for a DASD Sharing Environment

Note that the values given in Figure 65 on page 185 are examples which do not necessarily reflect a running environment. The figure shows two sets of IPL commands (for two DASD sharing systems). Notice that both ADD commands for the shared disks, statements (1) in Figure 65 on page 185, refer to the same packs although they specify different device addresses. Each CPU accesses the shared disks via different channels: 2 and 3.

The short form of the DLF command is shown here, statements (2). If the system which performs the first IPL refers to a non-existent lock file, it prompts the operator to submit the long form of the DLF command. On Processor I, for example, the long form would include the CYL and DSF specifications. If one or more ADD statements have the operand SHR, you **must** enter a DLF command. Use the long form if the lock file does not exist yet, or the short form if the lock file has already been created.

The SYSREC specifications, statements (3), refer to the same pack by different device addresses.

Complete ASI JCL procedures are not shown here. These procedures would contain DLBL/EXTENT statements for the shared resources listed below:

- To be cataloged in the system standard label area (OPTION STDLABEL):

  IJSYSRS
  SYSRES file

  IJSYSHF
  history file

- To be cataloged in the system standard label area (OPTION STDLABEL) or in the partition standard label area (OPTION PARSTD):

  IJAFILE
  VSE/POWER account file

  IJQFILE
  VSE/POWER queue file

  IJDFILE
  VSE/POWER data file

  IJSYSCT
  VSE/VSAM catalog

VSMSPCE
VSE/VSAM data space

xxxxxxxx
shared private libraries

Be aware that in addition labels for the following non-shared resources must be uniquely defined for each system:

IJSYSCN
hardcopy file

IJSYSRC
recorder file

xxxxxxxx
dedicated files and libraries

Error Recovery after System Breakdown

When one of the sharing systems breaks down, for example, due to a hardware error, the other system(s) may enter the wait state.

Two error situations are possible:

1. The hardware malfunction occurred while the system was executing a LOCK or UNLOCK request. The system has reserved the disk drive containing the lock file by a DEVICE RESERVE channel program. Thus the other systems are unable to execute LOCK or UNLOCK requests. The operator should press "system reset" on the failing CPU; the device reserves will be reset.
2. Prior to the system breakdown, the failing VSE system has locked some vital resources (for example, a VSE/VSAM catalog). The sharing VSE systems trying to lock these resources will enter the wait state. They will remain in the wait state until the failing system has been re-IPLed.

   If an IPL on the failing system is not possible at once, use the attention command UNLOCK SYSTEM=sys-id to unlock all resources locked by the failing CPU. This command can be entered at any other CPU sharing the same lock file. You should be extremely careful with the use of the attention command UNLOCK. Enter this command only when you are absolutely sure that the failing system has stopped and a new IPL is not possible. The attention command UNLOCK when used to break the lock of a running system will cause severe errors.

Designing Programs for Virtual Mode Execution

This section describes programming techniques that may improve the efficiency of programs that execute in virtual mode. Consider these techniques for new programs to be written and old programs to be revised. The section also contains information on the use of certain macros that are provided especially for virtual storage. Programming conventions for the shared virtual area are also discussed.

Programming Hints for Reducing Page Faults

It may be worthwhile to spend some extra programming effort for tuning virtual-mode programs that are used frequently or that require long periods of processing time so that they will cause fewer page faults during execution. Page faults generally occur when the size of the virtual-mode program exceeds the number of page frames available to it during execution. Efforts to reduce the number of page faults occurring in a program generally involve techniques for reducing the size of the "working set" of the program. The term "working set" is one that recurs often in discussions of virtual storage systems.

The working set of a program comprises those program pages which contain the most frequently used sequences of instructions for a given period of time. The working set of a program is not a fixed number of pages or instructions of that program; this set changes as the execution of the program proceeds. For example, a program doing an internal sort and writing a formatted table based on the results of this sort would have two completely different basic working sets; one for the sort function and one for the write functions.

Although the following section does not tell you how to determine the size of the working set, it does provide techniques for reducing its size.

General Hints for Reducing the Working Set

There are three general rules to keep in mind when working toward a reduction of a program's working set. The first is locality of reference; that is, instructions and data used together should be in storage near each other. Second is minimum processor storage. In other words, the amount of processor storage necessary for a program to do something should be kept as low as possible. Third is validity of reference; that is, references should be made only to data which will actually be used.

The chief means of achieving locality of reference is to make execution sequential whenever possible by avoiding excessive branching.

A program that executes sequentially normally requires a partition larger than the same program when it does not execute sequentially. For example, the functions of a section of code repeat themselves several times throughout the logic of your program. You are tempted to write this code once and branch to it whenever necessary, but branching violates the principle of locality of reference. Branching may cause more page faults than would coding the routine in line each time it is used. Also, it is easier for someone else to follow the logic of a program which is written to execute sequentially.

Locality of reference can be achieved only to a limited extent by programs written in a high-level language. Elements in arrays in FORTRAN or PL/I can be referred to in the order in which they appear in storage. In FORTRAN, for example, arrays are ordered by columns. The elements of the array DIMENSION (2,2,2) are arranged as follows in contiguous virtual storage locations:

```
    (1,1,1) (2,1,1)
    (1,2,1) (2,2,1)
    (1,1,2) (2,1,2)
    (1,2,2) (2,2,2)
```

For array structures of other compilers, refer to the appropriate programming language reference manuals.

A routine which processes all the elements of such an array should refer to them in this order. If only certain elements of an array are processed, the elements should be arranged in the order in which they are to be processed. If arranging an array in a certain manner causes it to be processed advantageously one time, but disadvantageously another time, you should consider writing two arrays, even at the cost of additional virtual storage.

In an assembler language program, you should keep frequently used data and constants near each other in storage, and near the instructions which use them. This contrasts with the traditional practice of having one area at the end of the program reserved for all the data areas and constants. Also, seldom used data should be separated from the frequently used data and placed with the routines which use it.

Avoid, if possible, using chains which must be searched each time a data item is required. If chains are unavoidable they should be kept in a compact area of storage. This may result in some wasted (virtual) storage but will be better than searches of large areas of storage.

Another good practice to help reduce paging is to initialize variables just before they are to be used. For example, in PL/I instead of the following:

```
  DCL A FIXED INIT (10);
  .
  .
  DO B=1 TO 100;
  A=A+B;
  END;
```

use:

```
  DCL A FIXED;
  .
  .
  A=10;
  DO B=1 TO 100;
```

```
   A=A+B;
   END;
```

In the first example, PL/I initializes the automatic variable at the beginning of execution. The second example does not require the page containing A to be in processor storage until just before A is used.

An important help in reducing the amount of processor storage needed for execution is to keep coding used for errors or other unusual occurrences in a separate routine. If, for example, the main routine contains code for conditions that occur only 5% of the time, by moving this error code to a separate section of your program, you can reduce the amount of needed processor storage for 95% of the processing.

Frequently-used subroutines should be loaded near each other. Because of their frequent use, these routines tend to be in processor storage almost continuously. If they are scattered over several pages, each of these pages will need to be in processor storage most of the time, thus increasing the size of the working set. By loading these routines near each other, you reduce the number of pages required in processor storage at any one time.

Subroutines should be designed to do as much processing as possible whenever they are called. It is better to duplicate some code from the calling routine in the called routine in order to avoid switching back and forth between routines. One technique for accomplishing this is to have the calling program pass several parameters to the subroutine and make one call, rather than passing one parameter at a time and make several calls.

You should try to keep code that can be modified and code that cannot be modified in separate sections of a large program. This will reduce page traffic by reducing the number of pages that are changed. Also, try to prevent I/O buffers from crossing page boundaries unnecessarily. Check the assembler listing and the linkage editor map to determine where page boundaries occur in your programs.

Using Virtual Storage Macros

The macros designed for use by virtual-mode programs, which are discussed below, perform the following services:

- Fix pages in processor storage (PFIX macro) and later free the same pages for normal paging (PFREE macro).
- Indicate the mode of execution of a program (RUNMODE macro).
- Influence the paging mechanism in order to reduce the number of page faults, to minimize the page I/O activity, and to control the page traffic within a specific partition.

In order to use these macros you must use assembler language or, if your program is written in a high-level language, you must write an assembler subroutine to make use of them. Refer to z/VSE System Macros Reference, and for a detailed description of these macros.

Fixing Pages in Processor Storage

Parts of virtual-mode programs must be in processor storage only at certain times. These parts include not only the instructions and data being processed at any one moment, but also data areas for use by channel programs. Instructions and data are always in processor storage when being used. Because of the nature of I/O operations, the data areas for these operations could be paged out during the I/O operation if something were not done to keep them in processor storage during the entire operation. The operating system therefore fixes I/O areas in processor storage during the I/O operation.

There are other parts of a program, however, which cannot tolerate paging, and these parts are not necessarily kept in storage by the operating system. For example, programs that control time-dependent I/O operations cannot tolerate paging. If a page fault were to occur during the execution of one of these programs, the results would be unpredictable. A page fault in one of these programs can be avoided by fixing the affected pages in processor storage, using the PFIX macro.

Note:

1. You should define the amount of storage available for PFIX request with the JCL command SETPFIX. Refer also to "Defining Real Storage" on page 11 for additional details.
2. You can use the PFIX macro for programs running in a static or in a dynamic partition.

It is also possible to PFIX pages in the SVA. Refer to the description of the PHASE statement in z/VSE System Control Statements.

The pages that you fix by the PFIX macro are fixed in the processor storage that is allocated to the partition in which the PFIX request is issued.

The PFIX macro fixes the pages in processor storage, regardless of whether the pages are stored in contiguous page frames or not. The supervisor keeps a count of the number of times a page has been fixed without being freed.

The PFREE macro does not directly free a page for paging out, but each time it is issued, the counter of fixes is reduced by one. As soon as the counter for a page reaches zero, the page can be paged out. At the end of a job step, all pages that are fixed during the job step are freed.

Use the PFREE macro as soon as possible to make a maximum number of page frames available to all programs running in virtual mode.

Figure 66 on page 190 is a skeleton example using the PFIX and PFREE macros. After the execution of a PFIX macro, a return code is given in register 15. The meanings of the return codes are:

0
: The pages were fixed successfully.

4
: You requested more page frames than the number of PFIXable page frames available to the partition.

8
: Insufficient number of free page frames were available at the time.

12
: You specified invalid addresses in your macros, or the begin address was higher than the end address, or a negative length was found.

16
: A PFIX request was given with RLOC=BELOW, but at least one page of the requested area is already PFIXed in a frame above 16 MB.

20
: Inconsistent function or option code in register 15.

Note in the example how the return code can be used to establish a branch to parts of the program that handle these specific conditions.

```
            .
            .
FIXRT    PFIX   ARTN,ARTNEND+2     FIX ARTN IN STORAGE
         B      *+4(15)            BRANCH BY RETURN CODE
         B      HERE         CONTINUE IF OK
         B      NOPAGES      GO TO CANCEL IF PART TOO SMALL
         B      WAIT         GO TO WAIT UNTIL PAGES FREED
         B      CANCL        GO TO CANCEL IF ADDR INVALID
            .
            .
HERE     BAL    14,ARTN      GO TO ARTN
         PFREE  ARTN,ARTNEND+2     FREE ROUTINE AFTER EXECU-
            .                      TION
            .
ARTN      (time-dependent processing which cannot be
           paged out during execution)

ARTNEND  BR     R14      RETURN
            .
            .
NOPAGES  LA     R1,OPCCB
         EXCP   (1)      WRITE MESSAGE TO OPERATOR
         WAIT   (1)      WAIT FOR COMPLETION
CANCL    CANCEL ALL
            .
            .
END      EOJ
OPCCB    CCB    SYSLOG,OPCCW
OPCCW    CCW    X'09',MSG,X'20',61
MSG      DC     CL32'AM CANCELING PLEASE ENLARGE REAL'
         DC     CL29'ADDR AREA AND RESTART THE JOB'
            .
            .
```

Figure 66. PFIX and PFREE Example

Indicating the Execution Mode of a Program

You may have a program that must do different processing depending upon its execution mode. It may be impractical to have two separate programs cataloged in a library (one program for real mode and another program for virtual mode). The RUNMODE macro can be issued during the execution of the program to inquire which mode of execution is being used. A return code is issued to the program in register 1.

Influencing the Paging Mechanism

Releasing Pages

With the RELPAG macro, you inform the page management routines that the contents of one or more pages is no longer required and need not be saved on the page data set. Thus, page frames occupied by these released pages can be claimed for use by other pages, and page I/O activity is reduced.

Forcing Page-out

The FCEPGOUT macro is used to inform the page management routines that one or more pages will not be needed until a later stage of processing. The pages are given the highest page-out priority, with the result that other pages, which may be needed immediately, are kept in storage. Except when the RELPAG macro is in operation, the contents of any pages written out are saved.

Page-in in Advance

The PAGEIN macro allows you to request that one or more pages be read into processor storage in advance, in order to avoid page faults when the specified pages are needed in processor storage. If the specified pages are already in processor storage when the macro is issued, they are given the lowest priority for page-out.

Coding for the Shared Virtual Area

Note: The description and the example shown in Figure 67 on page 192 are primarily intended for a 24-bit environment. For the linkage conventions valid for a 31-bit environment, refer to z/VSE Extended Addressability.

Besides accommodating the system directory list (SDL) and phases that are needed by the system, the shared virtual area (SVA) may contain user-written phases that can be used concurrently by more than one program. SVA phases must be re-enterable and relocatable; code that modifies itself will cause a protection check when executed from the SVA. This section presents some advice on coding phases to use SVA facilities and suggests some standards for base-register usage.

The basic assumptions for coding an SVA phase are:

- The re-enterable code must not modify any storage within its own storage area. Therefore, the code must not contain DTFs, CCBs, or other control blocks that are modified during execution.
- The phase can modify registers only if it saves and restores them for each user.
- A user-specified work area (within the calling partition) must be provided for storing registers and for any storage modifications.

Suggested register conventions:

- Use register 12 as the base register in both the main routine and the re-enterable code.
- Use register 13 as base for the working storage area. It is the responsibility of the main routine to provide addressability to the work area by loading register 13; the re-enterable routine must not modify register 13. The easiest way to address the working storage area in the re-enterable code is by a DSECT that defines the fields of the work area and a USING dsectname,13. In this way symbolic addressing can be used.
- Use CALL, SAVE, and RETURN macros. As register 13 is the base register, SAVE (14,12) and RETURN (14,12) result. Use register notation for CALL, for example, CALL (15) .... Before issuing the CALL, load register 15 with the transfer address. Register 14 will always contain the return address. The standard is thus established of register 15 for calling and register 14 for returning.
- Switches, and other areas that may be modified, can be placed in the working storage area using base register 13.

Figure 67 on page 192 illustrates the suggested conventions: MAINRTN is the main routine, SUBRTN is the SVA phase.

```
MAINRTN   CSECT
          BALR     12,0
          USING    *,12
          LA       13,SAVE
          LOAD     SUBRTN,WORKAREA       CANCELS IF SUBRTN NOT IN LIB
 *                                       LOADS SUBRTN INTO WORKAREA
 *                                       IF SUBRTN IS NOT IN SVA
          LR       15,1
          CALL     (15),(SWITCH,TECB,FIELDA,FIELDB,WORKAREA)
          .
          .
          EJO
 SAVE     DS       9D
 WORKAREA DS       200D                  SUBRTN IS LOADED HERE
 *                                       IF NOT IN SVA
 SWITCH   DC       XL1'00'
 TECB     DS       CL4
 FIELDA   DS       CL15
 FIELDB   DS       CL11
          END

 SUBRTN   CSECT                          MUST BE SEPARATE ASSEMBLY
          SAVE     (14,12)
          BALR     12,0
          USING    *,12
          USING    WORKAREA,6
          LM       2,6,0(1)
          MVC      0(15,4),DATA1
          MVC      0(11,5),DATA2
          CLI      0(2),X'FF'
          BE       EXIT
          SETIME   3,(3)                 SETIME ALTERS THE TECB
          WAIT     (3)
          .
          .
 EXIT     XI       0(2),X'FF'
          RETURN   (14,12)
 DATA1    DC       CL15'THIS IS FIELDA'
 DATA2    DC       CL11'THIS IS FIELDB'
          LTORG
 WORKAREA DSECT
 FIELDC   DS       3D
 FIELDD   DS       3D
          END
```

Figure 67. Example of Conventions for SVA Coding

Appendix A. Understanding Syntax Diagrams

This section describes how to read the syntax diagrams.

To read a syntax diagram follow the path of the line. Read from left to right and top to bottom.

- The ▶▶─── symbol indicates the beginning of a syntax diagram.
- The ───▶ symbol, at the end of a line, indicates that the syntax diagram continues on the next line.
- The ▶─── symbol, at the beginning of a line, indicates that a syntax diagram continues from the previous line.
- The ───▶◀ symbol indicates the end of a syntax diagram.

Syntax items (for example, a keyword or variable) can be:

- Directly on the line (required)
- Above the line (default)
- Below the line (optional)

Uppercase Letters

Uppercase letters denote the shortest possible abbreviation. If an item appears entirely in uppercase letters, it cannot be abbreviated.

You can type the item in uppercase letters, lowercase letters, or any combination. For example:

▶▶─ KEYWOrd ─▶◀

In this example, you can enter KEYWO, KEYWOR, or KEYWORD in any combination of uppercase and lowercase letters.

Symbols

You must code these symbols exactly as they appear in the syntax diagram.

*****

Asterisk

:

Colon

,

Comma

=

Equal sign

-

Hyphen

//

Double slash

()

Parenthesis

.

Period

+

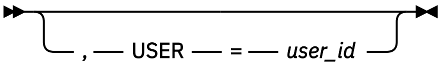
Add

For example:

```
* $$ LST
```

Variables

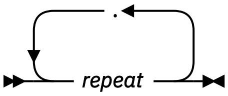
Highlighted lowercase letters denote variable information that you must substitute with specific information. For example:



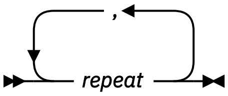
Here you must code USER= as shown and supply an ID for user_id. You can enter USER in lowercase, but you should not change it otherwise.

Repetition

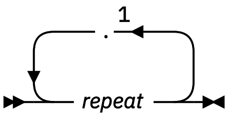
An arrow returning to the left means that the item can be repeated.



A character within the arrow means you must separate repeated items with that character.



A footnote (1) by the arrow references a limit that tells how many times the item can be repeated.

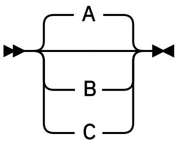


Notes:

[1] Specify *repeat* up to five times.

Defaults

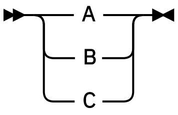
Defaults are above the line. The system uses the default unless you override it. You can override the default by coding an option from the stack below the line. For example:



In this example, A is the default. You can override A by choosing B or C.

Required Choices

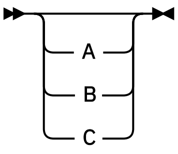
When two or more items are in a stack and one of them is on the line, you must specify one item. For example:



Here you must enter either A or B or C.

Optional Choice

When an item is below the line, the item is optional. You can only choose one item. For example:



Here you can enter either A or B or C, or omit the field.

Required Blank Space

A required blank space is indicated as such in the notation. For example:

```
* $$ EOJ
```

This indicates that at least one blank is required before and after the characters $$.

Frequent Abbreviations

1. *cuu* is the VSE address. It can be any value between X'000' and X'FFF'. It is the address by which the device was defined during I/O configuration.

   During IPL the **ADD** and **DEL** commands also accept physical device address up to X'FFFF'(*pcuu*). These physical device addresses are mapped to corresponding VSE addresses and can be queried with the QUERY IO command.
2. *volser* represents the six-character identifier (the volume serial number) of a tape or disk volume. If you specify fewer than six characters, the value that is passed to the system is padded to the left with zeros, unless you enclose the specification in quotation marks. In this case, the value is padded to the right with blanks. For example,

   ```
   The specification        is passed to the system as

        VOL1                     00VOL1
        'VOL1'                   VOL1__
   ```

   Bear in mind that these two specifications do not match when compared by label checking routines. The IPL program always pads to the right with blanks.

Alphanumeric characters are defined to include the following: A - Z, 0 - 9, @, $, and #.

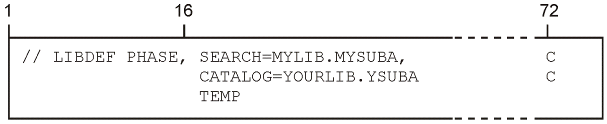
In case of any difference between the conventions that are given in this publication for control program functions and those appearing in IBM-supplied VSE component publications, observe the deviations given in the component publication.

Continuation of Commands and Statements

When job control statements are entered through SYSRDR, job control will accept continuation cards or lines only for the **ASSGN, DLBL, EXEC, IF, KEKL, LIBDEF, LIBDROP, LIBLIST, LIBSERV, PROC, PRTY, SETPARM, SETPRT, TLBL,** and **VTAPE** statements. In these statements, up to nine continuation lines are accepted. The operands of the line to be continued can be:

- Entered up to and including column 71, or
- Interrupted after the comma or equals sign separating two operands. Any columns between the interruption and column 72 must contain blanks.

A character string enclosed in single quotes ' ' is regarded as a single operand. Do not interrupt it before column 71, even if it contains commas or equal signs. Position 72 of the line to be continued must always contain a nonblank continuation character (usually a C). The continuation line must start in column 16. For example:



```
// LIBDEF PHASE, SEARCH=MYLIB.MYSUBA,                C
               CATALOG=YOURLIB.YSUBA                 C
               TEMP
```

If entered through SYSLOG, all job control statements and commands (except those which have no separating commas) and all attention routine commands can be continued on subsequent lines. The existence of a continuation line is indicated by a minus sign immediately following the last delimiting comma on the current line. The command or statement is then continued at the start of the next line. Example:

```
ALLOC R,F1=128K,-
F2=228K,F3=128K,-
F4=128K
```

Continuation lines can also be entered on SYSLOG in the same way as on SYSRDR.

Notices

This information was developed for products and services offered in the US. This material might be available from IBM in other languages. However, you may be required to own a copy of the product or product version in that language in order to access it.

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 grant you any license to these patents. You can send license inquiries, in writing, to:

IBM Director of Licensing
IBM Corporation
North Castle Drive, MD-NC119
Armonk, NY 10504-1785
US

For license inquiries regarding double-byte character set (DBCS) information, contact the IBM Intellectual Property Department in your country or send inquiries, in writing, to:

Intellectual Property Licensing
Legal and Intellectual Property Law
IBM Japan Ltd.
19-21, Nihonbashi-Hakozakicho, Chuo-ku
Tokyo 103-8510, Japan

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 jurisdictions 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 websites are provided for convenience only and do not in any manner serve as an endorsement of those websites. The materials at those websites are not part of the materials for this IBM product and use of those websites is at your own risk.

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

Licensees of this program who wish to have information about it for the purpose of enabling: (i) the exchange of information between independently created programs and other programs (including this one) and (ii) the mutual use of the information which has been exchanged, should contact:

IBM Director of Licensing
IBM Corporation
North Castle Drive, MD-NC119
Armonk, NY 10504-1785
US

Such information may be available, subject to appropriate terms and conditions, including in some cases, payment of a fee.

The licensed program described in this document and all licensed material available for it are provided by IBM under terms of the IBM Customer Agreement, IBM International Program License Agreement or any equivalent agreement between us.

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.

Statements regarding IBM's future direction or intent are subject to change or withdrawal without notice, and represent goals and objectives only.

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 actual people or business enterprises is entirely coincidental.

COPYRIGHT LICENSE:

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

Programming Interface Information

This publication documents intended Programming Interfaces that allow the customer to write programs to obtain services of z/VSE.

Trademarks

IBM, the IBM logo, and ibm.com are trademarks or registered trademarks of International Business Machines Corp., registered in many jurisdictions worldwide. Other product and service names might be trademarks of IBM or other companies. A current list of IBM trademarks is available on the web at "Copyright and trademark information" at www.ibm.com/legal/copytrade.shtml.

Adobe, the Adobe logo, PostScript, and the PostScript logo are either registered trademarks or trademarks of Adobe Systems Incorporated in the United States, and/or other countries.

IPv6/VSE is a registered trademark of Barnard Software, Inc.

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

Linux is a registered trademark of Linus Torvalds in the United States, other countries, or both.

Microsoft, Windows, Windows NT, 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.

Terms and Conditions for Product Documentation

Permissions for the use of these publications are granted subject to the following terms and conditions.

Applicability

These terms and conditions are in addition to any terms of use for the IBM website.

Personal use

You may reproduce these publications for your personal, noncommercial use provided that all proprietary notices are preserved. You may not distribute, display or make derivative work of these publications, or any portion thereof, without the express consent of IBM.

Commercial use

You may reproduce, distribute and display these publications solely within your enterprise provided that all proprietary notices are preserved. You may not make derivative works of these publications, or reproduce, distribute or display these publications or any portion thereof outside your enterprise, without the express consent of IBM.

Rights

Except as expressly granted in this permission, no other permissions, licenses or rights are granted, either express or implied, to the publications or any information, data, software or other intellectual property contained therein. IBM reserves the right to withdraw the permissions granted herein whenever, in its discretion, the use of the publications is detrimental to its interest or, as determined by IBM, the above instructions are not being properly followed. You may not download, export or re-export this information except in full compliance with all applicable laws and regulations, including all United States export laws and regulations.

IBM MAKES NO GUARANTEE ABOUT THE CONTENT OF THESE PUBLICATIONS. THE PUBLICATIONS ARE PROVIDED "AS-IS" AND WITHOUT WARRANTY OF ANY KIND, EITHER EXPRESSED OR IMPLIED, INCLUDING BUT NOT LIMITED TO IMPLIED WARRANTIES OF MERCHANTABILITY, NON-INFRINGEMENT, AND FITNESS FOR A PARTICULAR PURPOSE.

Accessibility

Accessibility features help a user who has a physical disability, such as restricted mobility or limited vision, to use software products successfully. The major accessibility features in z/VSE enable users to:

- Use assistive technologies such as screen readers and screen magnifier software
- Operate specific or equivalent features using only the keyboard
- Customize display attributes such as color, contrast, and font size

Using Assistive Technologies

Assistive technology products, such as screen readers, function with the user interfaces found in z/VSE. Consult the assistive technology documentation for specific information when using such products to access z/VSE interfaces.

Documentation Format

The publications for this product are in Adobe Portable Document Format (PDF) and should be compliant with accessibility standards. If you experience difficulties when you use the PDF files and want to request a web-based format for a publication, you can either write an email to s390id@de.ibm.com, or use the Reader Comment Form in the back of this publication or direct your mail to the following address:

```
IBM Deutschland Research & Development GmbH
Department 3282
Schoenaicher Strasse 220
D-71032 Boeblingen
Federal Republic of Germany
```

In the request, be sure to include the publication number and title.

When you send information to IBM, you grant IBM a nonexclusive right to use or distribute the information in any way it believes appropriate without incurring any obligation to you.

Glossary

This glossary includes terms and definitions for IBM z/VSE.

The following cross-references are used in this glossary:

1. See refers the reader from a term to a preferred synonym, or from an acronym or abbreviation to the defined full form.
2. See also refers the reader to a related or contrasting term.

A

Access Control Logging and Reporting

An IBM licensed program to log all attempts of access to protected data and to print selected formatted reports on such attempts.

access control table (DTSECTAB)

A table that is used by the system to verify a user's right to access a certain resource.

access list

A table in which each entry specifies an address space or data space that a program can reference.

access method

A program, that is, a set of commands (macros) to define files or addresses and to move data to and from them; for example VSE/VSAM or VTAM.

account file

A disk file that is maintained by VSE/POWER containing accounting information that is generated by VSE/POWER and the programs running under VSE/POWER.

addressing mode (AMODE)

A program attribute that refers to the address length that a program is prepared to handle on entry. Addresses can be either 24 bits, 31 bits, or 64 bits in length. In 24 bit addressing mode, the processor treats all virtual addresses as 24-bit values; in 31 bit addressing mode, the processor treats all virtual addresses as 31-bit values and in 64-bit addressing mode, the processor treats all virtual addresses as 64-bit values. Programs with an addressing mode of ANY can receive control in either 24 bit or 31 bit addressing mode. 64 bit addressing mode cannot be used as program attribute.

administration console

In z/VSE, one or more consoles that receive all system messages, except for those that are directed to one particular console. Contrast this with the user console, which receives only those messages that are directed to it, for example messages that are issued from a job that was submitted with the request to echo its messages to that console. The operator of an administration console can reply to all outstanding messages and enter all system commands.

alternate block

On an FBA disk, a block that is designated to contain data in place of a defective block.

alternate index

In systems with VSE/VSAM, the index entries of a given base cluster that is organized by an alternate key, that is, a key other than the prime key of the base cluster. For example, a personnel file preliminary ordered by names can be indexed also by department number.

alternate library

An interactively accessible library that can be accessed from a terminal when the user of that terminal issues a connect or switch library request.

alternate track

A library, which becomes accessible from a terminal when the user of that terminal issues a connect or switch (library) request.

AMODE

Addressing mode.

APA

All points addressable.

APAR

Authorized Program Analysis Report.

appendage routine

A piece of code that is physically located in a program or subsystem, but logically and extension of a supervisor routine.

application profile

A control block in which the system stores the characteristics of one or more application programs.

application program

A program that is written for or by a user that applies directly to the user's work, such as a program that does inventory control or payroll. See also batch program and online application program.

AR/GPR

Access register and general-purpose register pair.

ASC mode

Address space control mode.

ASI (automated system initialization) procedure

A set of control statements, which specifies values for an automatic system initialization.

attention routine (AR)

A routine of the system that receives control when the operator presses the Attention key. The routine sets up the console for the input of a command, reads the command, and initiates the system service that is requested by the command.

automated system initialization (ASI)

A function that allows control information for system startup to be cataloged for automatic retrieval during system startup.

autostart

A facility that starts VSE/POWER with little or no operator involvement.

auxiliary storage

Addressable storage that is not part of the processor, for example storage on a disk unit. Synonymous with external storage.

B

B-transient

A phase with a name beginning with $$B and running in the Logical Transient Area (LTA). Such a phase is activated by special supervisor calls.

bar

2 GigyByte (GB) line

basic telecommunications access method (BTAM)

An access method that permits read and write communication with remote devices. BTAM is not supported on z/VSE.

BIG-DASD

A subtype of Large DASD that has a capacity of more than 64 K tracks and uses up to 10017 cylinders of the disk.

block

Usually, a block consists of several records of a file that are transmitted as a unit. But if records are very large, a block can also be part of a record only. On an FBA disk, a block is a string of 512 bytes of data. See also a control block.

block group

In VSE/POWER, the basic organizational unit for fixed-block architecture (FBA) devices. Each block group consists of a number of 'units of transfer' or blocks.

C

CA splitting

Is the host part of the VSE JavaBeans, and is started using the job STARTVCS, which is placed in the reader queue during installation of z/VSE. Runs by default in dynamic class R. In VSE/VSAM, to double a control area dynamically and distribute its CIs evenly when the specified minimum of free space get used up by more data.

carriage control character

The fist character of an output record (line) that is to be printed; it determines how many lines should be skipped before the next line is printed.

catalog

A directory of files and libraries, with reference to their locations. A catalog may contain other information such as the types of devices in which the files are stored, passwords, blocking factors. To store a library member such as a phase, module, or book in a sublibrary. See also VSE/VSAM catalog.

cell pool

An area of virtual storage that is obtained by an application program and managed by the callable cell pool services. A cell pool is located in an address space or a data space and contains an anchor, at least one extent, and any number of cells of the same size.

central location

The place at which a computer system's control device, normally the systems console in the computer room, is installed.

chained sublibraries

A facility that allows sublibraries to be chained by specifying the sequence in which they must be searched for a certain library member.

chaining

A logical connection of sublibraries to be searched by the system for members of the same type (phases or object modules, for example).

channel command word (CCW)

A doubleword at the location in main storage that is specified by the channel address word. One or more CCWs make up the channel program that directs data channel operations.

channel program

One or more channel command words that control a sequence of data channel operations. Execution of this sequence is initiated by a start subchannel instruction.

channel scheduler

The part of the supervisor that controls all input/output operations.

channel subsystem

A feature of z/Architecture that provides extensive additional channel (I/O) capabilities to IBM Z.

channel to channel attachment (CTCA)

A function that allows data to be exchanged

1. Under the control of VSE/POWER between two virtual VSE machines running under VM or
2. Under the control of VTAM between two processors.

character-coded request

A request that is encoded and transmitted as a character string. Contrast with *field-formatted request*.

checkpoint

1. A point at which information about the status of a job and the system can be recorded so that the job step can be restarted later.
2. To record such information.

CICS (Customer Information Control System)

An IBM program that controls online communication between terminal users and a database. Transactions that are entered at remote terminals are processed concurrently by user-written application programs. The program includes facilities for building, using, and servicing databases.

CICS ECI

The CICS External Call Interface (ECI) is one possible requester type of the *CICS business logic interface* that is provided by the CICS Transaction Server for z/VSE. It is part of the CICS client and allows workstation programs to CICS function on the z/VSE host.

CICS EXCI

The EXternal CICS Interface (EXCI) is one possible requester type of the *CICS business logic interface* that is provided by the CICS Transaction Server for z/VSE. It allows any BSE batch application to call CICS functions.

CICS system definition data set (CSD)

A VSAM KSDS cluster that contains a resource definition record for every record defined to CICS using resource definition online (RDO).

CICS Transaction Server for z/VSE

A z/VSE base program that controls online communication between terminal users and a database. This is the successor system to CICS/VSE.

CICS TS

CICS Transaction Server

CICS/VSE

Customer Information Control System/VSE. No longer shipped on the Extended Base Tape and no longer supported, cannot run on z/VSE 5.1 or later.

class

In VSE/POWER, a group of jobs that either come from the same input device or go to the same output device.

CMS

Conversational monitor system running on z/VM.

common library

A library that can be interactively accessed by any user of the (sub)system that owns the library.

communication adapter

A circuit card with associated software that enables a processor, controller, or other device to be connected to a network.

communication region

An area of the supervisor that is set aside for transfer of information within and between programs.

component

1. Hardware or software that is part of a computer system.
2. A functional part of a product, which is identified by a component identifier.
3. In z/VSE, a component program such as VSE/POWER or VTAM.
4. In VSE/VSAM, a named, cataloged group of stored records, such as the data component or index component of a key-sequenced file or alternate index.

component identifier

A 12-byte alphanumeric string, uniquely defining a component to MSHP.

conditional job control

The capability of the job control program to process or to skip one or more statements that are based on a condition that is tested by the program.

connect

To authorize library access on the lowest level. A modifier such as "read" or "write" is required for the specified use of a sublibrary.

connection pooling

Introduced with an z/VSE 5.1 update to manage (reuse) connections of the z/VSE database connector in CICS TS.

connector

In the context of z/VSE, a connector provides the middleware to connect two platforms: Web Client and z/VSE host, middle-tier and z/VSE host, or Web Client and middle-tier.

connector (e-business connector)

A piece of software that is provided to connect to heterogeneous environments. Most connectors communicate to non-z/VSE Java-capable platforms.

container

Is part of the JVM of application servers such as the IBM WebSphere Application Server, and facilitates the implementation of servlets, EJBs, and JSPs, by providing resource and transaction management resources. For example, an EJB developer must not code against the JVM of the application server, but instead against the interface that is provided by the container. The main role of a container is to act as an intermediary between EJBs and clients, Is the host part of the VSE JavaBeans, and is started using the job STARTVCS, which is placed in the reader queue during the installation of z/VSE. Runs by default in dynamic class R. and also to manage multiple EJB instances. After EJBs have been written, they must be stored in a container residing on an application server. The container then manages all threading and client-interactions with the EJBs, and co-ordinate connection- and instance pooling.

control interval (CI)

A fixed-length area of disk storage where VSE/VSAM stores records and distributes free space. It is the unit of information that VSE/VSAM transfers to or from disk storage. For FBA it must be an integral multiple to be defined at cluster definition, of the block size.

control program

A program to schedule and supervise the running of programs in a system.

conversational monitor system (CMS)

A virtual machine operating system that provides general interactive time sharing, problem solving, and program development capabilities and operates under the control of z/VM.

count-key-data (CKD) device

A disk device that store data in the record format: count field, key field, data field. The count field contains, among others, the address of the record in the format: cylinder, head (track), record number, and the length of the data field. The key field, if present, contains the record's key or search argument. CKD disk space is allocated by tracks and cylinders. Contrast with *FBA disk device. See also extended count-key-data device.*

cross-partition communication control

A facility that enables VSE subsystems and user programs to communicate with each other; for example, with VSE/POWER.

cryptographic token

Usually referred to simply as a *token*, this is a device, which provides an interface for performing cryptographic functions like generating digital signatures or encrypting data.

cryptography

1. A method for protecting information by transforming it (encrypting it) into an unreadable format, called ciphertext. Only users who possess a secret key can decipher (or decrypt) the message into plaintext.
2. The transformation of data to conceal its information content and to prevent its unauthorized use or undetected modification .

D

data block group

The smallest unit of space that can be allocated to a VSE/POWER job on the data file. This allocation is independent of any device characteristics.

data conversion descriptor file (DCDF)

With a DCDF, you can convert individual fields within a record during data transfer between a PC and its host. The DCDF defines the record fields of a particular file for both, the PC and the host environment.

data import

The process of reformatting data that was used under one operating system such that it can subsequently be used under a different operating system.

Data Interfile Transfer, Testing, and Operations (DITTO) utility

An IBM program that provides file-to-file services for card I/O, tape, and disk devices. The latest version is called DITTO/ESA for VSE.

Data Language/I (DL/I)

A database access language that is used with CICS.

data link

In SNA, the combination of the link connection and the link stations joining network noes, for example, a z/Architecture channel and its associated protocols. A link is both logical and physical.

data security

The protection of data against unauthorized disclosure, transfer, modification, or destruction, whether accidental or intentional .

data set header record

In VSE/POWER abbreviated as DSHR, alias NDH or DSH. An NJE control record either preceding output data or, in the middle of input data, indicating a change in the data format.

data space

A range of up to 2 gigabytes of contiguous virtual storage addresses that a program can directly manipulate through z/Architecture instructions. Unlike an address space, a data space can hold only user data; it does not contain shared areas, or programs. Instructions do not execute in a data space. Contrast with address space.

data terminal equipment (DTE)

In SNA, the part of a data station that serves a data source, data sink, or both.

database connector

Is a function introduced with z/VSE 5.1.1, which consists of a client and server part. The client provides an API (CBCLI) to be used by applications on z/VSE, the server on any Java capable platform connects a JDBC driver that is provided by the database. Both client and server communicate via TCP/IP.

Database 2 (Db2)

An IBM rational database management system.

Db2-based connector

Is a feature introduced with VSE/ESA 2.5, which includes a customized Db2 version, together with VSAM and DL/I functionality, to provide access to Db2, VSAM, and DL/I data, using Db2 Stored Procedures.

Db2 Runtime only Client edition

The Client Edition for z/VSE comes with some enhanced features and improved performance to integrate z/VSE and Linux on z Systems.

Db2 Stored Procedure

In the context of z/VSE, a Db2 Stored Procedure is a Language Environment (LE) program that accesses Db2 data. However, from VSE/ESA 2.5 onwards you can also access VSAM and DL/I data using a Db2 Stored Procedure. In this way, it is possible to exchange data between VSAM and Db2.

DBLK

Data block.

DCDF

Data conversion descriptor file.

deblocking

The process of making each record of a block available for processing.

dedicated (disk) device

A device that cannot be shared among users.

device address

1. The identification of an input/output device by its device number.
2. In data communication, the identification of any device to which data can be sent or from which data can be received.

device driving system (DDS)

A software system external to VSE/POWER, such as a CICS spooler or PSF, that writes spooled output to a destination device.

Device Support Facilities (DSF)

An IBM supplied system control program for performing operations on disk volumes so that they can be accessed by IBM and user programs. Examples of these operations are initializing a disk volume and assigning an alternative track.

device type code

The four- or five-digit code that is used for defining an I/O device to a computer system. See also ICKDSF

dialog

In an interactive system, a series of related inquiries and responses similar to a conversation between two people. For z/VSE, a set of panels that can be used to complete a specific task; for example, defining a file.

dialog manager

The program component of z/VSE that provides for ease of communication between user and system.

digital signature

In computer security, encrypted data, which is appended to or part of a message, that enables a recipient to prove the identity of the sender.

Digital Signature Algorithm (DSA)

The Digital Signature Algorithm is the US government-defined standard for digital signatures. The DSA digital signature is a pair of large numbers, computed using a set of rules (that is, the DSA) and a set of parameters such that the identity of the signatory and integrity of the data can be verified. The DSA provides the capability to generate and verify signatures.

directory

In z/VSE the index for the program libraries.

direct access

Accessing data on a storage device using their address and not their sequence. This is the typical access on disk devices as opposed to magnetic tapes. Contrast with *sequential access*.

disk operating system residence volume (DOSRES)

The disk volume on which the system sublibrary IJSYSRS.SYSLIB is located including the programs and procedures that are required for system startup.

disk sharing

An option that lets independent computer systems uses common data on shared disk devices.

disposition

A means of indicating to VSE/POWER how a job input or output entry is to be handled: according to its local disposition in the RDR/LST/PUN queue or its transmission disposition when residing in the XMT queue. A job might, for example, be deleted or kept after processing.

distribution tape

A magnetic tape that contains, for example, a preconfigured operating system like z/VSE. This tape is shipped to the customer for program installation.

DITTO/ESA for VSE

Data Interfile Transfer, Testing, and Operations utility. An IBM program that provides file-to-file services for disk, tape, and card devices.

DSF

Device Support Facilities.

DSH (R)

Data set header record.

dummy device

A device address with no real I/O device behind it. Input and output for that device address are spooled on disk.

duplex

Pertaining to communication in which data can be sent and received at the same time.

DU-AL (dispatchable unit - access list)

The access list that is associated with a z/VSE main task or subtask. A program uses the DU-AL associated with its task and the PASN-AL associated with its partition. See also “PASN-AL (primary address space number - access list)” on page 224.

dynamic class table

Defines the characteristics of dynamic partitions.

dynamic partition

A partition that is created and activated on an 'as needed' basis that does not use fixed static allocations. After processing, the occupied space is released. Dynamic partitions are grouped by class, and jobs are scheduled by class. Contrast with *static partition*.

dynamic space reclamation

A librarian function that provides for space that is freed by the deletion of a library member to become reusable automatically.

E

ECI

See “CICS ECI” on page 207.

emulation

The use of programming techniques and special machine features that permit a computer system to execute programs that are written for another system or for the use of I/O devices different from those that are available.

emulation program (EP)

An IBM control program that allows a channel-attached 3705 or 3725 communication controller to emulate the functions of an IBM 2701 Data Adapter Unit, or an IBM 2703 Transmission Control.

end user

1. A person who makes use of an application program.
2. In SNA, the ultimate source or destination of user data flowing through an SNA network. Might be an application program or a terminal operator.

Enterprise Java Bean

An EJB is a distributed bean. "Distributed" means, that one part of an EJB runs inside the JVM of a web application server, while the other part runs inside the JVM of a web browser. An EJB either represents one data row in a database (entity bean), or a connection to a remote database (session bean). Normally, both types of an EJB work together. This allows to represent and access data in a standardized way in heterogeneous environments with relational and non-relational data. See also *JavaBean*.

entry-sequenced file

A VSE/VSAM file whose records are loaded without respect to their contents and whose relative byte addresses cannot change. Records are retrieved and stored by addressed access, and new records are added to the end of the file.

Environmental Record Editing and Printing (EREP) program

A z/VSE base program that makes the data that is contained in the system record file available for further analysis.

EPI

See *CICS EPI.*

ESCON Channel (Enterprise Systems Connection Channel)

A serial channel, using fiber optic cabling, that provides a high-speed connection between host and control units for I/O devices. It complies with the ESA/390 and IBM Z I/O Interface until z114. The zEC12 processors do not support ESCON channels.

exit routine

1. Either of two types of routines: installation exit routines or user exit routines. Synonymous with exit program.
2. See *user exit routine*.

extended addressability

The ability of a program to use 31 bit or 64 bit virtual storage in its address space or outside the address space.

extended recovery facility (XRF)

In z/VSE, a feature of CICS that provides for enhanced availability of CICS by offering one CICS system as a backup of another.

External Security Manager (ESM)

A priced vendor product that can provide extended functionality and flexibility that is compared to that of the Basic Security Manager (BSM), which is part of z/VSE.

F

FASTCOPY

See "VSE/Fast Copy" on page 235.

fast copy data set program (VSE/Fast Copy)

See "VSE/Fast Copy" on page 235.

fast service upgrade (FSU)

A service function of z/VSE for the installation of a refresh release without regenerating control information such as library control tables.

FAT-DASD

A subtype of Large DASD, it supports a device with more than 4369 cylinders (64 K tracks) up to 64 K cylinders.

FCOPY

See *VSE/Fast Copy*.

fence

A separation of one or more components or elements from the remainder of a processor complex. The separation is by logical boundaries. It allows simultaneous user operations and maintenance procedures.

fetch

1. To locate and load a quantity of data from storage.

2. To bring a program phase into virtual storage from a sublibrary and pass control to this phase.
3. The name of the macro instruction (FETCH) used to accomplish 2. See also *loader.*

Fibre Channel Protocol (FCP)

A combination of hardware and software conforming to the Fibre Channel standards and allowing system and peripheral connections via FICON and FICON Express feature cards on IBM zSeries processors. In z/VSE, zSeries FCP is employed to access industry-standard SCSI disk devices.

fragmentation (of storage)

Inability to allocate unused sections (fragments) of storage in the real or virtual address range of virtual storage.

FSU

Fast service upgrade.

FULIST (FUnction LIST)

A type of selection panel that displays a set of files and/or functions for the choice of the user.

G

generation

See *macro generation.*

generation feature

An IBM licensed program order option that is used to tailer the object code of a program to user requirements.

GETVIS space

Storage space within partition or the shared virtual area, available for dynamic allocation to programs.

guest system

A data processing system that runs under control of another (host) system. On the mainframe z/VSE can run as a guest of z/VM.

H

hard wait

The condition of a processor when all operations are suspended. System recovery from a hard wait is impossible without performing a new system startup.

hash function

A hash function is a transformation that takes a variable-size input and returns a fixed-size string, which is called the hash value. In cryptography, the hash functions should have some additional properties:

- The hash function should be easy to compute.
- The hash function is one way; that is, it is impossible to calculate the 'inverse' function.

- The hash function is collision-free; that is, it is impossible that different input leads to the same hash value.

hash value

The fixed-sized string resulting after applying a *hash function* to a text.

High-Level Assembler for VSE

A programming language providing enhanced assembler programming support. It is a base program of z/VSE.

home interface

Provides the methods to instantiate a new EJB object, introspect an EJB, and remove an EJB instantiation., as for the remote interface is needed because the deployment tool generates the implementation class. Every Session bean's home interface must supply at least one *create()* method.

host mode

In this operating mode, a PC can access a VSE host. For programmable workstation (PWS) functions, the Move Utilities of VSE can be used.

host system

The controlling or highest level system in a data communication configuration.

host transfer file (HTF)

Used by the Workstation File Transfer Support of z/VSE as an intermediate storage area for files that are sent to and from IBM personal computers.

HTTP Session

In the context of z/VSE, identifies the web-browser client that calls a servlet (in other words, identifies the connection between the client and the middle-tier platform).

I

ICCF

See *VSE/ICCF.*

ICKDSF (Device Support Facilities)

A z/VSE base program that supports the installation, use, and maintenance of IBM disk devices.

include function

Retrieves a library member for inclusion in program input.

index

1. A table that is used to locate records in an indexed sequential data set or on indexed file.
2. In, an ordered collection of pairs, each consisting of a key and a pointer, used by to sequence and locate the records of a key-sequenced data set or file; it is organized in levels of index records. See also *alternate index.*

input/output control system (IOCS)

A group of IBM supplied routines that handle the transfer of data between main storage and auxiliary storage devices.

integrated communication adapter (ICA)

The part of a processor where multiple lines can be connected.

integrated console

In z/VSE, the service processor console available on IBM Z that operates as the z/VSE system console. The integrated console is typically used during IPL and for recovery purposes when no other console is available.

Interactive Computing and Control Facility (ICCF)

An IBM licensed program that serves as interface, on a time-slice basis, to authorized users of terminals that are linked to the system's processor.

interactive partition

An area of virtual storage for the purpose of processing a job that was submitted interactively via VSE/ICCF.

Interactive User Communication Vehicle (IUCV)

Programming support available in a VSE supervisor for operation under z/VM. The support allows users to communicate with other users or with CP in the same way they would with a non-preferred guest.

intermediate storage

Any storage device that is used to hold data temporarily before it is processed.

IOCS

Input/output control system.

IPL

Initial program load.

irrecoverable error

An error for which recovery is impossible without the use of recovery techniques external to the computer program or run.

IUCV

Interactive User Communication Vehicle.

J

JAR

Is a platform-independent file format that aggregates many files into one. Multiple applets and their requisite components (.class files, images, and sounds) can be bundled in a JAR file, and then downloaded to a web browser using a single HTTP transaction (much improving the download speed). The JAR format also supports compression, which reduces the files size (and further improves the download speed). The compression algorithm that is used is fully compatible with the ZIP algorithm. The owner of an applet can also digitally sign individual entries in a JAR file to authenticate their origin.

Java application

A Java program that runs inside the JVM of your web browser. The program's code resides on a local hard disk or on the LAN. Java applications might be large programs using graphical interfaces. Java applications have unlimited access to all your local resources.

Java bytecode

Bytecode is created when a file containing Java source language statements is compiled. The compiled Java code or "bytecode" is similar to any program module or file that is ready to be executed (run on a computer so that instructions are performed one at a time). However, the instructions in the bytecode are really instructions to the *Java Virtual Machine*. Instead of being interpreted one instruction at a time, bytecode is instead recompiled for each operating-system platform using a just-in-time (JIT) compiler. Usually, this enables the Java program to run faster. Bytecode is contained in binary files that have the suffix**.CLASS**

Java servlet

See *servlet.*

JHR

Job header record.

job accounting interface

A function that accumulates accounting information for each job step, to be used for charging the users of the system, for planning new applications, and for supervising system operation more efficiently.

job accounting table

An area in the supervisor where accounting information is accumulated for the user.

job catalog

A catalog made available for a job by means of the file name IJSYSUC in the respective DLBL statement.

job entry control language (JECL)

A control language that allows the programmer to specify how VSE/POWER should handle a job.

job step

In 1 of a group of related programs complete with the JCL statements necessary for a particular run. Every job step is identified in the job stream by an EXEC statement under one JOB statement for the whole job.

job trailer record (JTR)

As VSE/POWER parameter JTR, alias NJT. An NJE control record terminating a job entry in the input or output queue and providing accounting information.

K

key

In VSE/VSAM, one or several characters that are taken from a certain field (key field) in data records for identification and sequence of index entries or of the records themselves.

key sequence

The collating sequence either of records themselves or of their keys in the index or both. The key sequence is alphanumeric.

key-sequenced file

A VSE/VSAM file whose records are loaded in key sequence and controlled by an index. Records are retrieved and stored by keyed access or by addressed access, and new records are inserted in the file in key sequence.

KSDS

Key-sequenced data sets. See *key-sequenced file.*

L

label

1. An identification record for a tape, disk, or diskette volume or for a file on such a volume.
2. In assembly language programming, a named instruction that is generally used for branching.

label information area

An area on a disk to store label information that is read from job control statements or commands. Synonymous with *label area*.

Language Environment for z/VSE

An IBM software product that is the implementation of Language Environment on the VSE platform.

language translator

A general term for any assembler, compiler, or other routine that accepts statements in one language and produces equivalent statements in another language.

Large DASD

A DASD device that

1. Has a capacity exceeding 64 K tracks and
2. Does not have VSAM space created prior to VSE/ESA 2.6 that is owned by a catalog.

LE/VSE

Short form of Language Environment for z/VSE.

librarian

The set of programs that maintains, services, and organizes the system and private libraries.

library block

A block of data that is stored in a sublibrary.

library directory

The index that enables the system to locate a certain sublibrary of the accessed library.

library member

The smallest unit of a data that can be stored in and retrieved from a sublibrary.

line commands

In VSE/ICCF, special commands to change the declaration of individual lines on your screen. You can copy, move, or delete a line declaration, for example.

linkage editor

A program that is used to create a phase (executable code) from one or more independently translated object modules, from one or more existing phases, or from both. In creating the phase, the linkage editor resolves cross-references among the modules and phases available as input. The program can catalog the newly built phases.

linkage stack

An area of protected storage that the system gives to a program to save status information for a branch and stack or a stacking program call.

link station

In SNA, the combination of hardware and software that allows a node to attach to and provide control for a link.

loader

A routine, commonly a computer program, that reads data or a program into processor storage. See also *relocating loader*.

local shared resources (LSR)

A VSE/VSAM option that is activated by three extra macros to share control blocks among files.

lock file

In a shared disk environment under VSE, a system file on disk that is used by the sharing systems to control their access to shared data.

logical partition

In LPAR mode, a subset of the server unit hardware that is defined to support the operation of a system control program.

logical record

A user record, normally pertaining to a single subject and processed by data management as a unit. Contrast with *physical* record, which may be larger or smaller.

logical unit (LU)

1. A name that is used in programming to represent an I/O device address. *physical unit (PU), system services control point (SSCP), primary logical unit (PLU), and secondary logical unit (SLU).*
2. In SNA, a port through which a user accesses the SNA network,
   a. To communicate with another user and
   b. To access the functions of the SSCP. An LU can support at least two sessions. One with an SSCP and one with another LU and might be capable of supporting many sessions with other LUs.

logical unit name

In programming, a name that is used to represent the address of an input/output unit.

logical unit 6.2

A SNA/SDLC protocol for communication between programs in a distributed processing environment. LU 6.2 is characterized by

1. A peer relationship between session partners,
2. Efficient utilization of a session for multiple transactions,
3. Comprehensive end-to-end error processing, and
4. A generic Application Programming Interface (API) consisting of structured verbs that are mapped into a product implementation.

logons interpret interpret routine

In VTAM, an installation exit routine, which is associated with an interpret table entry, that translates logon information. It also verifies the logon.

LPAR mode

Logically partitioned mode. The CP mode that is available on the Configuration (CONFIG) frame when the PR/SM feature is installed. LPAR mode allows the operator to allocate the hardware resources of the processor unit among several logical partitions.

M

macro definition

A set of statements and instructions that defines the name of, format of, and conditions for generating a sequence of assembler statements and machine instructions from a single source statement.

macro expansion

See *macro generation*

macro generation

An assembler operation by which a macro instruction gets replaced in the program by the statements of its definition. It takes place before assembly. Synonymous with *macro expansion*.

macro (instruction)

1. In assembler programming, a user-invented assembler statement that causes the assembler to process a set of statements that are defined previously in the macro definition.

2. A sequence of VSE/ICCF commands that are defined to cause a sequence of certain actions to be performed in response to one request.

maintain system history program (MSHP)

A program that is used for automating and controlling various installation, tailoring, and service activities for a VSE system.

main task

The main program within a partition in a multiprogramming environment.

master console

In z/VSE, one or more consoles that receive all system messages, except for those that are directed to one particular console. Contrast this with the *user* console, which receives only those messages that are specifically directed to it, for example messages that are issued from a job that was submitted with the request to echo its messages to that console. The operator of a master console can reply to all outstanding messages and enter all system commands.

maximum (max) CA

A unit of allocation equivalent to the maximum control area size on a count-key-data or fixed-block device. On a CKD device, the max CA is equal to one cylinder.

memory object

Chunk of virtual storage that is allocated above the bar (2 GB) to be created with the IARV64 macro.

message

In VSE, a communication that is sent from a program to the operator or user. It can appear on a console, a display terminal or on a printout.

MSHP

See maintain system history program.

multitasking

Concurrent running of one main task and one or several subtasks in the same partition.

MVS

Multiple Virtual Storage. Implies MVS/390, MVS/XA, MVS/ESA, and the MVS element of the z/OS (OS/390) operating system.

N

NetView

A z/VSE optional program that is used to monitor a network, manage it, and diagnose its problems.

network address

In SNA, an address, consisting of subarea and element fields, that identifies a link, link station, or NAU. Subarea nodes use network addresses; peripheral nodes use local addresses. The boundary function in the subarea node to which a peripheral node is attached transforms local addresses to network addresses and vice versa. See also *network name*.

network addressable unit (NAU)

In SNA, a logical unit, a physical unit, or a system services control point. It is the origin or the destination of information that is transmitted by the path control network. Each NAU has a network address that represents it to the path control network. See also *network name, network address*.

Network Control Program (NCP)

An IBM licensed program that provides communication controller support for single-domain, multiple-domain, and interconnected network capability. Its full name is ACF/NCP.

network definition table (NDT)

In VSE/POWER networking, the table where every node in the network is listed.

network name

1. In SNA, the symbolic identifier by which users refer to a NAU, link, or link station. See also *network address*.
2. In a multiple-domain network, the name of the APPL statement defining a VTAM application program. This is its network name, which must be unique across domains.

node

1. In SNA, an end point of a link or junction common to several links in a network. Nodes can be distributed to host processors, communication controllers, cluster controllers, or terminals. Nodes can vary in routing and other functional capabilities.
2. In VTAM, a point in a network that is defined by a symbolic name. Synonymous with *network node*. See *major node and minor node*.

node type

In SNA, a designation of a node according to the protocols it supports and the network addressable units (NAUs) it can contain.

O

object module (program)

A program unit that is the output of an assembler or compiler and is input to a linkage editor.

online application program

An interactive program that is used at display stations. When active, it waits for data. Once input arrives, it processes it and send a response to the display station or to another device.

operator command

A statement to a control program, issued via a console or terminal. It causes the control program to provide requested information, alter normal operations, initiate new operations, or end existing operations.

optional licensed program

An IBM licensed program that a user can install on VSE by way of available installation-assist support.

output parameter text block (OPTB)

in VSE/POWER's spool-access support, information that is contained in an output queue record if a * $$ LST or * $$ PUN statement includes any user-defined keywords that have been defined for autostart.

P

page data set (PDS)

One or more extents of disk storage in which pages are stored when they are not needed in processor storage.

page fixing

Marking a page so that it is held in processor storage until explicitly released. Until then, it cannot be paged out.

page I/O

Page-in and page-out operations.

page pool

The set of page frames available for paging virtual-mode programs.

panel

The complete set of information that is shown in a single display on terminal screen. Scrolling back and forth through panels like turning manual pages. See also *selection panel*.

partition balancing

A z/VSE facility that allows the user to specify that two or more or all partitions of the system should receive about the same amount of time on the processor.

PASN-AL (primary address space number - access list)

The access list that is associated with a partition. A program uses the PASN-AL associated with its partition and the DU-AL associated with its task (work unit). See also *DU-AL*.

Each partition has its own unique PASN-AL. All programs running in this partition can access data spaces through the PASN-AL. Thus a program can create a data space, add an entry for it in the PASN-AL, and obtain the ALET that indexes the entry. By passing the ALET to other programs in the partition, the program can share the data space with other programs running in the same partition.

PDS

Page data sets.

phase

The smallest complete unit of executable code that can be loaded into virtual storage.

physical record

The amount of data that is transferred to or from auxiliary storage. Synonymous with *block*.

PNET

Programming support available with VSE/POWER; it provides for the transmission of selected jobs, operator commands, messages, and program output between the nodes of a network.

POWER

See *VSE/POWER*.

pregenerated operating system

An operating system such as z/VSE that is shipped by IBM mainly in object code. IBM defines such key characteristics as the size of the main control program, the organization, and size of libraries, and required system areas on disk. The customer does not have to generate an operating system.

preventive service

The installation of one or more PTFs on a VSE system to avoid the occurrence of anticipated problems.

primary address space

In z/VSE, the address space where a partition is executed. A program in primary mode fetches data from the primary address space.

primary library

A VSE library owned and directly accessible by a certain terminal user.

printer/keyboard mode

Refers to 1050 or 3215 console mode (device dependent).

Print Services Facility (PSF)/VSE

An access method that provides support for the advanced function printers.

private area

The virtual space between the shared area (24 bit) and shared area (31 bit), where (private) partitions are allocated. Its maximum size can be defined during IPL. See also *shared area*.

private memory object

Memory object (chunk of virtual storage) that is allocated above the 2 GB line (bar) only accessible by the partition that created it.

private partition

Any of the system's partitions that are not defined as shared. See also *shared partition*.

production library

1. In a pre-generated operating system (or product), the program library that contains the object code for this system (or product).
2. A library that contains data that is needed for normal processing. Contrast with *test library*.

programmer logical unit

A logical unit available primarily for user-written programs. See also *logical unit name*.

program temporary fix (PTF)

A solution or by-pass of one or more problems that are documented in APARs. PTFs are distributed to IBM customers for preventive service to a current release of a program.

PSF/VSE

Print Services Facility/VSE.

PTF

See *Program temporary fix*.

Q

Queue Control Area (QCA)

In VSE/POWER, an area of the data file, which might contain:

- Extended checkpoint information
- Control information for a shared environment.

queue file

A direct-access file that is maintained by VSE/POWER that holds control information for the spooling of job input and job output.

R

random processing

The treatment of data without respect to its location on disk storage, and in an arbitrary sequence that is governed by the input against which it is to be processed.

real address area

In z/VSE, processor storage to be accessed with dynamic address translation (DAT) off

real address space

The address space whose addresses map one-to-one to the addresses in processor storage.

real mode

In VSE, a processing mode in which a program might not be paged. Contrast with *virtual mode*.

recovery management support (RMS)

System routines that gather information about hardware failures and that initiate a retry of an operation that failed because of processor, I/O device, or channel errors.

refresh release

An upgraded VSE system with the latest level of maintenance for a release.

relative-record file

A VSE/VSAM file whose records are loaded into fixed-length slots and accessed by the relative-record numbers of these slots.

release upgrade

Use of the FSU functions to install a new release of z/VSE.

relocatable module

A library member of the type object. It consists of one or more control sections cataloged as one member.

relocating loader

A function that modifies addresses of a phase, if necessary, and loads the phase for running into the partition that is selected by the user.

remote interface

In the context of z/VSE, the remote interface allows a client to make method calls to an EJB although the EJB is on a remote z/VSE host. The container uses the remote interface to create client-side stubs and server-side proxy objects to handle incoming method calls from a client to an EJB.

remote procedure call (RPC)

1. A facility that a client uses to request the execution of a procedure call from a server. This facility includes a library of procedures and an external data representation.
2. A client request to service provider in another node.

residency mode (RMODE)

A program attribute that refers to the location where a program is expected to reside in virtual storage. RMODE 24 indicates that the program must reside in the 24-bit addressable area (below 16 megabytes), RMODE ANY indicates that the program can reside anywhere in 31-bit addressable storage (above or below 16 megabytes).

REXX/VSE

A general-purpose programming language, which is particularly suitable for command procedures, rapid batch program development, prototyping, and personal utilities.

RMS

Recovery management support.

RPG II

A commercially oriented programming language that is specifically designed for writing application programs that are intended for business data processing.

S

SAM ESDS file

A SAM file that is managed in VSE/VSAM space, so it can be accessed by both SAM and VSE/VSAM macros.

SCP

System control programming.

SDL

System directory list.

search chain

The order in which chained sublibraries are searched for the retrieval of a certain library member of a specified type.

second-level directory

A table in the SVA containing the highest phase names that are found on the directory tracks of the system sublibrary.

Secure Sockets Layer (SSL)

A security protocol that allows the client to authenticate the server and all data and requests to be encrypted. SSL was developed by Netscape Communications Corp. and RSA Data Security, Inc..

segmentation

In VSE/POWER, a facility that breaks list or punch output of a program into segments so that printing or punching can start before this program has finished generating such output.

selection panel

A displayed list of items from which a user can make a selection. Synonymous with *menu*.

sense

Determine, on request or automatically, the status or the characteristics of a certain I/O or communication device.

sequential access method (SAM)

A data access method that writes to and reads from an I/O device record after record (or block after block). On request, the support performs device control operations such as line spacing or page ejects on a printer or skip some tape marks on a tape drive.

service node

Within the VSE unattended node support, a processor that is used to install and test a master VSE system, which is copied for distribution to the unattended nodes. Also, program fixes are first applied at the service node and then sent to the unattended nodes.

service program

A computer program that performs function in support of the system. See with *utility program*.

service refresh

A form of service containing the current version of all software. Also referred to as a *system refresh*.

service unit

One or more PTFs on disk or tape (cartridge).

shared area

In z/VSE, shared areas (24 bit) contain the Supervisor areas and SVA (24 bit) and shared areas (31 bit) the SVA (31 bit). Shared areas (24 bit) are at the beginning of the address space (below 16 MB), shared area (31 bit) at the end (below 2 GB).

shared disk option

An option that lets independent computer systems use common data on shared disk devices.

shared memory objects

Chunks of virtual storage allocated above the 2 GB line (bar), that can be shared among partitions.

shared partition

In z/VSE, a partition that is allocated for a program (VSE/POWER, for example) that provides services and communicates with programs in other partitions of the system's virtual address spaces. In most cases shared partitions are no longer required.

shared spooling

A function that permits the VSE/POWER account file, data file, and queue file to be shared among several computer systems with VSE/POWER.

shared virtual area (SVA)

In z/VSE, a high address area that contains a list system directory list (SDL) of frequently used phases, resident programs that are shared between partitions, and an area for system support.

SIT (System Initialization Table)

A table in CICS that contains data used the system initialization process. In particular, the SIT can identify (by suffix characters) the version of CICS system control programs and CICS tables that you have specified and that are to be loaded.

skeleton

A set of control statements, instructions, or both, that requires user-specific information to be inserted before it can be submitted for processing.

socksified

See *socks-enabled*.

Socks-enabled

Pertaining to TCP/IP software, or to a specific TCP/IP application, that understands the *socks protocol*. "Socksified" is a slang term for socks-enabled.

socks protocol

A protocol that enables an application in a secure network to communicate through a firewall via a *socks server*.

socks server

A circuit-level gateway that provides a secure one-way connection through a firewall to server applications in a nonsecure network.

source member

A library member containing source statements in any of the programming languages that are supported by VSE.

split

To double a specific unit of storage space (CI or CA) dynamically when the specified minimum of free space gets used up by new records.

spooling

The use of disk storage as buffer storage to reduce processing delays when transferring data between peripheral equipment and the processor of a computer. In z/VSE, this is done under the control of VSE/POWER.

Spool Access Protection

An optional feature of VSE/POWER that restricts individual spool file entry access to user IDs that have been authenticated by having performed a security logon.

spool file

1. A file that contains output data that is saved for later processing.
2. One of three VSE/POWER files on disk: queue file, data file, and account file.

SSL

See Secure Sockets Layer.

stacked tape

An IBM supplied product-shipment tape containing the code of several licensed programs.

standard label

A fixed-format record that identifies a volume of data such as a tape reel or a file that is part of a volume of data.

stand-alone program

A program that runs independently of (not controlled by) the VSE system.

startup

The process of performing IPL of the operating system and of getting all subsystems and applications programs ready for operation.

start option

In VTAM, a user-specified or IBM specified option that determines conditions for the time a VTAM system is operating. Start options can be predefined or specified when VTAM is started.

static partition

A partition, which is defined at IPL time and occupying a defined amount of virtual storage that remains constant. See also *dynamic partition*.

storage director

An independent component of a storage control unit; it performs all of the functions of a storage control unit and thus provides one access path to the disk devices that are attached to it. A storage control unit has two storage directors.

storage fragmentation

Inability to allocate unused sections (fragments) of storage in the real or virtual address range of virtual storage.

suballocated file

A VSE/VSAM file that occupies a portion of an already defined data space. The data space might contain other files. See also *unique file*.

sublibrary

In VSE, a subdivision of a library. Members can only be accessed in a sublibrary.

sublibrary directory

An index for the system to locate a member in the accessed sublibrary.

submit

A VSE/POWER function that passes a job to the system for processing.

SVA

See shared virtual area.

Synchronous DataLink Control (SDLC)

A discipline for managing synchronous, code-transparent, serial-by-bit information transfer over a link connection. Transmission exchanges might be duplex or half-duplex over switched or non-switched links. The configuration of the link connection might be point-to-point, multipoint, or loop.

SYSRES

See system residence volume.

system control programming (SCP)

IBM supplied, non-licensed program fundamental to the operation of a system or to its service or both.

system directory list (SDL)

A list containing directory entries of frequently used phases and of all phases resident in the SVA. The list resides in the SVA.

system file

In z/VSE, a file that is used by the operating system, for example, the hardcopy file, the recorder file, the page data set.

System Initialization Table (SIT)

A table in CICS that contains data that is used by the system initialization process. In particular, the SIT can identify (by suffix characters) the version of CICS system control programs and CICS tables that you have specified and that are to be loaded.

system recorder file

The file that is used to record hardware reliability data. Synonymous with *recorder file*.

system refresh

See *service refresh*.

system refresh release

See *refresh release*.

system residence file (SYSRES)

The z/VSE system sublibrary IJSYSRS.SYSLIB that contains the operating system. It is stored on the system residence volume DORSES.

system residence volume (SYSRES)

The disk volume on which the system sublibrary is stored and from which the hardware retrieves the initial program load routine for system startup.

system sublibrary

The sublibrary that contains the operating system. It is stored on the system residence volume (SYSRES).

T

task management

The functions of a control program that control the use, by tasks, of the processor and other resources (except for input/output devices).

time event scheduling support

In VSE/POWER, the time event scheduling support offers the possibility to schedule jobs for processing in a partition at a predefined time once repetitively. The time event scheduling operands of the * $$ JOB statement are used to specify the wanted scheduling time.

TLS

See Transport Layer Security.

track group

In VSE/POWER, the basic organizational unit of a file for CKD devices.

track hold

A function that protects a track that is being updated by one program from being accessed by another program.

transaction

1. In a batch or remote batch entry, a job or job step. 2. In CICS TS, one or more application programs that can be used by a display station operator. A given transaction can be used concurrently from one or more display stations. The execution of a transaction for a certain operator is also referred to as a task.
2. A given task can relate only to one operator.

transient area

An area within the control program that is used to provide high-priority system services on demand.

Transport Layer Security

The newest SSL cryptographic protocol. It provides additional strength to privacy and data integrity.

Turbo Dispatcher

A facility of z/VSE that allows to use multiprocessor systems (also called CEC: Central Electronic Complexes). Each CPU within such a CEC has accesses to be shared virtual areas of z/VSE: supervisor, shared areas (24 bit), and shared areas (31 bit). The CPUs have equal rights, which means that any CPU might receive interrupts and work units are not dedicated to any specific CPU.

U

UCB

Universal character set buffer.

universal character set buffer (UCB)

A buffer to hold UCS information.

UCS

Universal character set.

user console

In z/VSE, a console that receives only those system messages that are specifically directed to it. These are, for example, messages that are issued from a job that was submitted with the request to echo its messages to that console. Contrast with *master console*.

user exit

A programming service that is provided by an IBM software product that can be requested during the execution of an application program for the service of transferring control back to the application program upon the later occurrence of a user-specified event.

V

variable-length relative-record data set (VRDS)

A relative-record data set with variable-length records. See also *relative-record data set*.

variable-length relative-record file

A VSE/VSAM relative-record file with variable-length records. See also *relative-record file*.

VIO

See virtual I/O area.

virtual address

An address that refers to a location in virtual storage. It is translated by the system to a processor storage address when the information stored at the virtual address is to be used.

virtual addressability extension (VAE)

A storage management support that allows to use multiple virtual address spaces.

virtual address space

A subdivision of the virtual address area (virtual storage) available to the user for the allocation of private, nonshared partitions.

virtual disk

A range of up to 2 gigabytes of contiguous virtual storage addresses that a program can use as workspace. Although the virtual disk exists in storage, it appears as a real FBA disk device to the user program. All I/O operations that are directed to a virtual disk are intercepted and the data to be written to, or read from, the disk is moved to or from a data space.

Like a data space, a virtual disk can hold only user data; it does not contain shared areas, system data, or programs. Unlike an address space or a data space, data is not directly addressable on a virtual disk. To manipulate data on a virtual disk, the program must perform I/O operations.

Starting with z/VSE 5.2, a virtual disk may be defined in a shared memory object.

virtual I/O area (VIO)

An extension of the page data set; used by the system as intermediate storage, primarily for control data.

virtual mode

The operating mode of a program, where the virtual storage of the program can be paged, if not enough processor (real) storage is available to back the virtual storage.

virtual partition

In VSE, a division of the dynamic area of virtual storage.

virtual storage

Addressable space image for the user from which instructions and data are mapped into processor storage locations.

virtual tape

In z/VSE, a virtual tape is a file (or data set) containing a tape image. You can read from or write to a virtual tape in the same way as if it were a physical tape. A virtual tape can be:

- A VSE/VSAM ESDS file on the z/VSE local system.
- A remote file on the server side; for example, a Linux, UNIX, or Windows file. To access such a remote virtual tape, a TCP/IP connection is required between z/VSE and the remote system.

volume ID

The volume serial number, which is a number in a volume label that is assigned when a volume is prepared for use by the system.

VRDS

Variable-length relative-record data sets. See *variable-length relative record file*.

VSAM

See *VSE/VSAM*.

VSE (Virtual Storage Extended)

A system that consists of a basic operating system and any IBM supplied and user-written programs that are required to meet the data processing needs of a user. VSE and hardware it controls form a complete computing system. Its current version is called z/VSE.

VSE/Advanced Functions

A program that provides basic system control and includes the supervisor and system programs such as the Librarian and the Linkage Editor.

VSE Connector Server

Is the host part of the VSE JavaBeans, and is started using the job STARTVCS, which is placed in the reader queue during installation of z/VSE. Runs by default in dynamic class R.

VSE/DITTO (VSE/Data Interfile Transfer, Testing, and Operations Utility)

An IBM licensed program that provides file-to-file services for disk, tape, and card devices.

VSE/ESA (Virtual Storage Extended/Enterprise Systems Architecture)

The predecessor system of z/VSE.

VSE/Fast Copy

A utility program for fast copy data operations from disk to disk and dump/restore operations via an intermediate dump file on magnetic tape or disk.

VSE/FCOPY (VSE/Fast Copy Data Set program)

An IBM licensed program for fast copy data operations from disk to disk and dump/restore operations via an intermediate dump file on magnetic tape or disk. There is also a stand-alone version: the FASTCOPY utility.

VSE/ICCF (VSE/Interactive Computing and Control Facility)

An IBM licensed program that serves as interface, on a time-slice basis, to authorized users of terminals that are linked to the system's processor.

VSE/ICCF library

A file that is composed of smaller files (libraries) including system and user data, which can be accessed under the control of VSE/ICCF.

VSE JavaBeans

Are JavaBeans that allow access to all VSE-based file systems (VSE/VSAM, Librarian, and VSE/ICCF), submit jobs, and access the z/VSE operator console. The class library is contained in the *VSEConnector.jar* archive. See also *JavaBeans*.

VSE library

A collection of programs in various forms and storage dumps stored on disk. The form of a program is indicated by its member type such as source code, object module, phase, or procedure. A VSE library consists of at least one sublibrary, which can contain any type of member.

VSE/POWER

An IBM licensed program that is primarily used to spool input and output. The program's networking functions enable a VSE system to exchange files with or run jobs on another remote processor.

VSE/VSAM (VSE/Virtual Storage Access Method)

An IBM access method for direct or sequential processing of fixed and variable length records on disk devices.

VSE/VSAM catalog

A file containing extensive file and volume information that VSE/VSAM requires to locate files, to allocate and deallocate storage space, to verify the authorization of a program or an operator to gain access to a file, and to accumulate use statistics for files.

VSE/VSAM managed space

A user-defined space on disk that is placed under the control of VSE/VSAM.

W

wait for run subqueue

In VSE/POWER, a subqueue of the reader queue with dispatchable jobs ordered in execution start time sequence.

wait state

The condition of a processor when all operations are suspended. System recovery from a hard wait is impossible without performing a new system startup. See *hard wait*.

Workstation File Transfer Support

Enables the exchange of data between IBM Personal Computers (PCs) linked to a z/VSE host system where the data is kept in intermediate storage. PC users can retrieve that data and work with it independently of z/VSE.

work file

A file that is used for temporary storage of data being processed.

Numerics

24-bit addressing

Provides addressability for address spaces up to 16 megabytes.

31-bit addressing

Provides addressability for address spaces up to 2 gigabytes.

64-bit addressing

Provides addressability for address spaces up to 2 gigabytes and above.

Index

Special Characters

/& statement 38
$$A$CDL0 22
$ABEND 65
$ASIPROC master procedure 19
$CANCEL 65
$COMVAR procedure 179
$EOJ 63
$SVAASMA 29
$SVACICS 29
$SVAREXX 29
$SVAVTAM 29
$SYSOPEN phase 32

Numerics

31-bit addressing 145, 156
3800 printing subsystem 50
64-bit address space 13

A

ABEND condition 65
access method 43
accessibility 201
accessing files 39
address space layout
    real address space 4
    virtual address space 4
ALLOC command 14
alphanumeric characters, definition of 195
application program interface (API), Librarian 79
application program, access to libraries 128
assembler 139
assembling programs 50
ASSGN command 33, 40
ASSGN statement 39
attention routine commands
    continuation of 195
automated system initialization (ASI)
    IPL procedure 15
    JCL procedure 17
    master procedure $ASIPROC 19
    overriding 19
    overview 15
    partition bring-up 17
    TYPE command 20
    TYPE operand 19

B

background partitions 17
BACKUP command 91
BG partition 172
block size 43
books 145
buffer load 49

C

calling procedures 67
CANCEL condition 65
carriage control character 59
CATAL operand (OPTION statement) 145
CATALOG a member command 97
CATALOG command 100
cataloged procedures 66, 82
cataloging multiple object modules 100, 146
cataloging phases 148
CDLOAD macro 6, 156
CHANGE command 102
checkpointing facility 177
CHKPT macro 177
CISIZE 43
CLASSTD option 18, 47
CLASSTD, label information 45
CLOSE (LIBRM macro option) 128
CLOSE command 58
command notation explained 193
command symbols 193
communication device list (CDL) 22
communication device, for IPL 21
COMPARE command 102
compile, link and go 150
compiling programs 50
conditional job control 59
continuation character 195
continuation lines 195
continuation of commands and statements 195
control interval (FBA) 43
controlling jobs 33
conventions, command 193
COPY command 103
creation date 45
cuu, definition of 195

D

DASD sharing 179
data area 43
DATA operand (CATALOG) 67
data secured file 43
data spaces 12
DEF SCSI command 15
default value 69
DEFINE command 79, 82
DELETE (LIBRM macro option) 128
DELETE command 105
device address, definition of 195
device assignment 33
device assignment, permanent 40
device assignment, shared 41

device assignment, temporary 40
device class 33
device sensing (IPL) 19
device type 33
direct access files 43
disability 201
disk extent 39
disk labels 43
display storage layout dialog 9
DLBL statement 34, 43
DOSRES 80
dump library 82
DUMP macro 59
dump options 53
dumps 82
duplicate phase names 152
dynamic partition 4
dynamic space GETVIS area 9

E

E-Deck library exit, High Level Assembler 139
end-of-job processing 38
end-of-job statement 38
EOJ macro 59
ESERV program 138
example
    conventions for SVA coding 191
    for changing the PASIZE 28
    for changing the VSIZE 28
    IPL exit routine 167
    IPL procedures for a DASD sharing environment 183
    JCL exit routine 168
    OPEN/GET/CLOSE request, Librarian API 135
    partition allocations 6
    PFIX and PFREE macro example 188
    STATE member request, Librarian API 131
EXEC PROC statement 67, 69
EXEC statement 37
executing programs 50
execution mode 54
exit
    abnormal termination 166
    E-Deck processing, High Level Assembler 139
    interval timer 166
    IPL, system startup 167
    job accounting 175
    job control, system startup 168
    operator communications 166
    page fault handling overlap 166
    program check 166
exit routine 32
expiration date 43
extent 39
EXTENT statement 34, 43
extent, size of 43

F

FETCH macro 145, 158
file labels 43
file labels, processing of 34
file name 34, 43, 45
file organization 43
file sequence number 45
file-ID 43, 45
files, accessing 39
files, relating them to programs 33
fixing pages 14
foreground partitions 17
forms control buffer (FCB) 49
FREEVIS macro 54

G

GET (LIBRM macro option) 128
GETSYMB macro 174
GETVIS area 55
GETVIS areas 9
GETVIS macro 54
global condition 63
GO operand (EXEC statement) 150
GOTO statement 63

H

High Level Assembler 139
High Level Assembler, library exit for processing E-Decks 139

I

I/O assignments 33
I/O devices, assigning 39
I/O spooling 57
IBM 3800 printing subsystem 50
ID statement 37
IF statement 62
IGNLOCK job control option 115
IJSYSIN 58
IJSYSLS 58
IJSYSPH 58
IJSYSRS.SYSLIB 80
index area 43
initial program load (IPL)
    overview 15
inline data 66, 82
INPUT command 106
integrated console 23
interrupting IPL processing 23
interrupting job stream 38
interval timer 179
IOCS 43
IPL communication device 21
IPL communication device list 22
IPL exit routine 167
IPL load parameter 23
IPL procedure, contents 15
IPL processing, interrupt 23

J

JCL exit routine 168
JCL procedure, contents 17
JCLEXIT command 173
job accounting interface routine 175

job control
- cataloged procedures 66
- commands, entering 33
- conditional 37, 59
- defining a job 37
- device assignments 39
- end-of-job statement (/&) 38
- for program control 54
- JOB statement 37
- job streams 38
- label information 43
- language 37
- loading 33
- logging 53
- nested procedures 73
- processing a program 50
- program 33
- symbolic parameters 69

job control statements and commands
- continuation of 195

job definition 37
job mix 38
JOB statement 37
job step 37
job stream 38
job-to-job transition 38
job, defining a 37

L

label areas 45
label checking 34
label information
- adding 47
- deleting 47
- search order 47

label information area 36
label options 36
label statement 63
labels 34
language translator 146
LIBDEF (LIBRM macro option) 128
LIBDROP (LIBRM macro option) 128
Librarian API, application program interface 79
librarian commands
- BACKUP 91
- CATALOG 97
- CHANGE 102
- COMPARE 102
- conditional command execution 89
- COPY 103
- DEFINE 82
- DELETE 105
- generic notation 90
- INPUT 106
- interactive execution 90
- LIST 107
- LISTDIR 107
- LOCK 114
- merge 104
- MOVE 103
- PUNCH 117
- RELEASE 118
- RENAME 118

librarian commands *(continued)*
- RESTORE 120
- SEARCH 125
- TEST 127
- UNLOCK 127
- UPDATE 127

Librarian commands 91
Librarian program 88
Librarian return codes 88
Librarian time stamp 124
libraries
- catalog members 85
- define library access (LIBDEF) 85
- defining 79
- display library access (LIBLIST) 88
- overview 79
- private 80
- private, non-VSAM managed space 82
- private, VSAM-managed space 82
- reset library access (LIBDROP) 87
- system 80
- time stamp control 124
- using 79

library access for application programs
- example of a STATE member request 131
- example of an OPEN/GET/CLOSE request 135
- LIBRDCB macro 128
- LIBRM macro 128

library exit, High Level Assembler 139
library members 80
LIBRDCB macro 128
LIBRM macro 128
link and go 149
LINK operand (OPTION statement) 145
linkage editor
- AUTOLINK function 155
- cataloging phases 148
- compile, link-edit, and execute 150
- FETCH macro 158
- how external references are resolved 159
- link-edit and execute 149
- LOAD macro 158
- overlay programs 156
- preparing the input 152
- processing requirements 151
- samples 161
- specifying helps 156
- storage requirements 154

Linkage Editor 81, 145
Linkage Editor error handling 150
Linkage Editor, assignments for 151
Linkage Editor, LIBDEFs for 151
linking programs
- object modules 146
- phases 147, 148
- source books 146

LIOCS 43
LIST command 107
list of JCL exit routines 172
list options 53
list-directed IPL 23
LISTDIR command 107

load list 29
LOAD macro 145, 158
load parameter facility 24
loading phases 145
loading phases into the SVA 28
LOCK a member command 114
lock file 181
locking library members 115
logical IOCS 43
logical units 33, 39
LSERV 47

M

macros
    CDLOAD 156
    CHKPT 177
    DUMP 59
    editing 138
    EOJ 59
    FETCH 145, 158
    FREEVIS 54
    GETSYMB 174
    GETVIS 54
    LOAD 145, 158
    PFIX 14
    PFREE 14
    source format 146
    SVALLIST 30
magnetic tape control 48
MAP command 9
member types, library members 80
members
    cataloging 81, 82
    DUMP type 82
    MSHP controlled 88
    OBJ type 81, 146
    PHASE type 81
    PHASE-type 52
    PROC type 82
    PROC, cataloging 67
    renaming 82
    SOURCE type 81
    SOURCE types 146
    user type 82
merging sublibraries 104
modules 145
MOVE command 103
MSHP controlled members 88
MTC command 48
multi-step procedures 68
multiphase program 152
multiple JCL exit routines 171
multiple object modules 146

N

nested procedures 73
nesting levels 73
notation, syntax 193
notations, command 193
NOTE (LIBRM macro option) 128

O

object module 50
object modules 81, 145, 146
ON conditions, default 62
ON statement 62
OPEN (LIBRM macro option) 128
OPTION CATAL 145
OPTION CLASSTD command 45
OPTION LINK 145
OPTION PARSTD command 45
OPTION statement 36, 53
OPTION STDLABEL command 45
OPTION USRLABEL command 45
overflow area 43
overlay programs 156

P

padding of volume serial number 195
page 1
page data set 1, 3
page fixing 14
page frame 1
page management 2
page out 3
page pool 1
paging 2
PARM parameter 54
PARSTD, label information 45
partition allocation 6
partition GETVIS area 9
partition-related procedures 67
partitions
    allocation 6
    GETVIS area 55
    layout 6
    processor storage 54
    real allocation 14
    real GETVIS area 54
    real storage allocation 13
    standard label subarea 45
    starting (ASI) 17
    temporary label subarea 45
PASIZE 4
passing parameters 54, 71
PAUSE command 38
PAUSE statement 38
permanent assignment 40
PFIX limits 11
PFIX macro 14
PFREE macro 14
phase 50
PHASE statement (Linkage Editor) 147, 152
phases
    duplicate 152
    loading in SVA 148
    naming 152
    non-relocatable 147, 148
    re-enterable 147, 148
    relocatable 147, 148
    self-relocating 147, 148
    SVA-eligible 147, 148
    testing 149

phases, cataloging 52
phases, loading of 81, 145
phases, storing of 145
physical unit 33
POINT (LIBRM macro option) 128
print buffers 49
printer, controlling 49
priority 38
private libraries 80
private sublibraries 80
PROC statement 69
procedure nesting 73
procedures 82
procedures, calling 67
procedures, partition-related 67
processor storage allocation 14
program switches 54
programmer logical units 39
programs
    assembling 50
    cataloged 52
    compiling 50
    designing for virtual mode execution 186
    executing 50, 52
    linking 50
programs, structure of 145
pseudo register support 158
PUNCH command 117
PUT (LIBRM macro option) 128

R

real execution, need for 54
real GETVIS area, defining 54
real mode, executing programs 54
real mode, program execution 13
real storage, defining 11
real storage, layout 11
record format, system files 59
register conventions, return code 59
RELEASE command 118
RENAME (LIBRM macro option) 128
RENAME command 82, 118
renaming members 82
replacing SVA phases 31
REQTEXT
    cataloging 146
    enhanced object module support 146
    enhanced support 146
    in register 15 174
    including parts 146
RESTORE command 120
retention period 45
return code setting by job control 60
return codes 59
return codes from Librarian 88
rewind 48
rotational position sensing 55
RSIZE 4

S

SDL (system directory list) 28
SDL entries 31
SEARCH for member command 125
selective restore 95
sequential files 43
SET SDL command 31
SET ZONEBDY command 15
SET ZONEDEF command 15
SETDF command 50
SETPARM statement 63, 69, 71
SETPFIX command 11
SETPRT command 50
shared device assignment 41
shared virtual area (SVA), loading phases into 28
shared virtual area, size of 31
SIZE command 55
SIZE operand (EXEC) 54, 55
SIZE parameter/command 6
SIZE=phasename (EXEC) 152
SKCOMVAR skeleton 179
SKEDECKX skeleton, High Level Assembler 139
skipping statements 63
source books 81, 145, 146
source program 50
source statements 145
SOURCE type members 81
spooling 57
stacker selection character 59
starting the system 15
startup 15
STATE (LIBRM macro option) 128
statement notation explained 193
STDLABEL, label information 45
STDOPT statement 53
STOP processing (IPL) 27
stoplist (IPL) 19
storage layout
    address space layout 4
    dynamic address space and GETVIS areas 10
    real address space 4
    real storage 11
    static address space and GETVIS areas 10
    virtual address spaces 4
storage management
    address space layout 4
    defining real storage 11
    dynamic address space and GETVIS areas 10
    page management 2
    partition allocation 6
    real storage layout 11
    static address space and GETVIS areas 10
    virtual storage concept 1
storing phases 145
STXIT macro 165
sublibraries
    accessing 90
    chaining 80
    concatenation 80
    members 80
    private 80
    system 80
support of named common areas 159
SVA command, IPL 31
SVA load list 29
SVA, load single phase 29

SVA, loading phases 148
SVA, loading phases into 28
SVA, PFIX a phase 153
SVALLIST macro 30
symbolic parameters
  assigning values to 69, 71
  concatenating 71
  default value 69
  defining 69, 71
  nullifying 69
  passing 71
  rules 69
  setting 63
  substitution 63
  values 63
syntax notation explained 193
syntax of commands 193
syntax symbols 193
SYSBUFLD program 49
SYSDUMP library 82
SYSIN 58
SYSIN, assigning 40
SYSIPT 57
SYSIPT data 66
SYSLNK 151
SYSLOG 33
SYSLOG, defining 21
SYSLST 57
SYSLST on tape 58
SYSOUT 58
SYSOUT, assigning 40
SYSPCH 57
SYSPCH on tape 58
SYSRDR 33, 57
SYSRES file, restore stand-alone 123
system console device list 22
system directory list (SDL) 28
system GETVIS area 9, 55
system library 80
system logical units 39
system standard subarea 45
system startup
  ASI IPL procedure 15
  ASI JCL procedure 17
  communication device for IPL 21
  communication device list (CDL) 22
  interrupt IPL processing 23
  IPL exit routine 167
  JCL exit routine 168
  master procedure $ASIPROC 19
  modify IPL parameters 26
  multiple JCL exit routines 171
  shared virtual area (SVA), loading phases into 28
  STOP processing (IPL) 27
  SYSLOG 21
  system directory list (SDL) 28
  user-defined processing 32
system sublibrary 80

T

tape control 48
tape labels 45
tape positioning 95
tape, system input 58
tapemarks, writing 48
telecommunication systems 22
temporary assignment 40
TEST command 127
testing phases 149
testing return codes 59
time stamp control, libraries 124
time-dependent programs 13
time-of-day clock 179
timer services 179
TLBL statement 34, 45
TYPE command (IPL) 20

U

universal character-set buffer (UCB) 49
UNLOCK a member command 127
UPDATE command 127
updating macros 146
UPSI byte 54
USRLABEL, label information 45

V

version number for files 45
VIO area 149
virtual address space 13
virtual I/O area 149
virtual machine 20
virtual mode, executing programs 54
virtual mode, program execution 13
virtual storage
  address translation 3
  loading programs in 2
  map 9
  overview 1
  paging 2
  size of 1
virtual storage map 9
volume 34
volume sequence number 45
volume serial number
  padding 195
VSE/ESA
  processor (real) storage 1
  virtual storage concept 1
VSE/POWER 57

Y

year 148
year representation (Linkage Editor) 148

Product Number: 5686-VS6

SC34-2705-01