z/VM
7.4

Saved Segments Planning and Administration

SC24-6322-74

Note:

Before you use this information and the product it supports, read the information in “Notices” on page 93.

This edition applies to version 7, release 4 of IBM® z/VM® (product number 5741-A09) and to all subsequent releases and modifications until otherwise indicated in new editions.

Last updated: 2024-09-18

Contents

Figures

Tables

About This Document

This document provides information about planning and administering saved segments on an IBM® z/VM® system. It describes tasks associated with planning for CP saved segments, CMS saved segments, and building saved segments using VMSES/E.

Some of this information is based on the experiences of IBM customers. The recommendations are meant to help installations run their z/VM systems more efficiently.

Intended Audience

This information is intended for anyone responsible for planning, installing, and updating a z/VM system.

You should have a general understanding of data processing and teleprocessing techniques, and you should have thought about:

- What z/VM functions your site requires
- Which guest operating systems you will be running
- How many users you are going to have and the type of environment under which they will be running their applications

Where to Find More Information

For more information about z/VM functions, see the documents listed in the “Bibliography” on page 97.

Links to Other Documents and Websites

The PDF version of this document contains links to other documents and websites. A link from this document to another document works only when both documents are in the same directory or database, and a link to a website works only if you have access to the Internet. A document link is to a specific edition. If a new edition of a linked document has been published since the publication of this document, the linked document might not be the latest edition.

How to provide feedback to IBM

We welcome any feedback that you have, including comments on the clarity, accuracy, or completeness of the information. See How to send feedback to IBM for additional information.

Summary of Changes for z/VM: Saved Segments Planning and Administration

This information includes terminology, maintenance, and editorial changes. Technical changes or additions to the text and illustrations for the current edition are indicated by a vertical line (**|**) to the left of the change.

SC24-6322-74, z/VM 7.4 (September 2024)

This edition supports the general availability of z/VM 7.4. Note that the publication number suffix (-74) indicates the z/VM release to which this edition applies.

SC24-6322-73, z/VM 7.3 (December 2023)

This edition includes terminology, maintenance, and editorial changes.

SC24-6322-73, z/VM 7.3 (September 2022)

This edition supports the general availability of z/VM 7.3. Note that the publication number suffix (-73) indicates the z/VM release to which this edition applies.

SC24-6322-01, z/VM 7.2 (September 2020)

This edition supports the general availability of z/VM 7.2.

Updates reflect the removal of KANJI language files from base z/VM components. The only currently supported languages are American English and uppercase English.

Chapter 1. Planning and Defining CP Saved Segments

This chapter tells you what a saved segment is and how to:

- Plan for saved segments
- Create saved segments and where to save them
- Load licensed programs into saved segments
- Use a saved segment

Page ranges shown for program products are approximate. Use the actual number of pages required by the release, modification, and service level you wish to run.

Note: The VMSES/E component provides functions for managing your saved segments. Saved segments are defined using the VMFSGMAP EXEC and built using the PUT2PROD EXEC, which calls the VMFBLD EXEC. See Chapter 3, “Using VMSES/E to Define, Build, and Manage Saved Segments,” on page 57.

- To use the VMSES/E functions, you must understand the principles of defining saved segments in CP, which are discussed in the following sections.
- VMSES/E does not support saved segments that contain pages above 2047 MB.

Saved Segment Overview

A *segment* (also called an *architected segment*) is a 1 MB portion of real storage defined by the hardware architecture.

A *saved segment* is a range of pages of virtual storage that you can define to hold data or reentrant code (programs).

Why Use Saved Segments?

Defining frequently used data and code (such as licensed programs) as saved segments provides several advantages:

- Because several users can access the same physical storage, real storage use is minimized.
- Using saved segments decreases the I/O rate and DASD paging space requirements, thereby improving virtual machine performance.
- Saved segments attached to a virtual machine can reside above its defined virtual storage. This allows the virtual machine to use its defined storage for other purposes.

Saved segments allow code or data in an area of virtual storage to be saved and assigned a name. A saved segment can then be dynamically attached to, and detached from, a virtual machine.

Programs residing within the page ranges of a saved segment that are reenterable can be shared by concurrently operating virtual machines. This allows you to place code that is required only some of the time in a saved segment and load it into a virtual machine when needed.

Note that a saved segment differs from a named saved system (NSS) in that a DIAGNOSE code X'64' loads it rather than an IPL.

Figure 1 on page 2 shows how different virtual machines can access the same saved segment in z/VM.

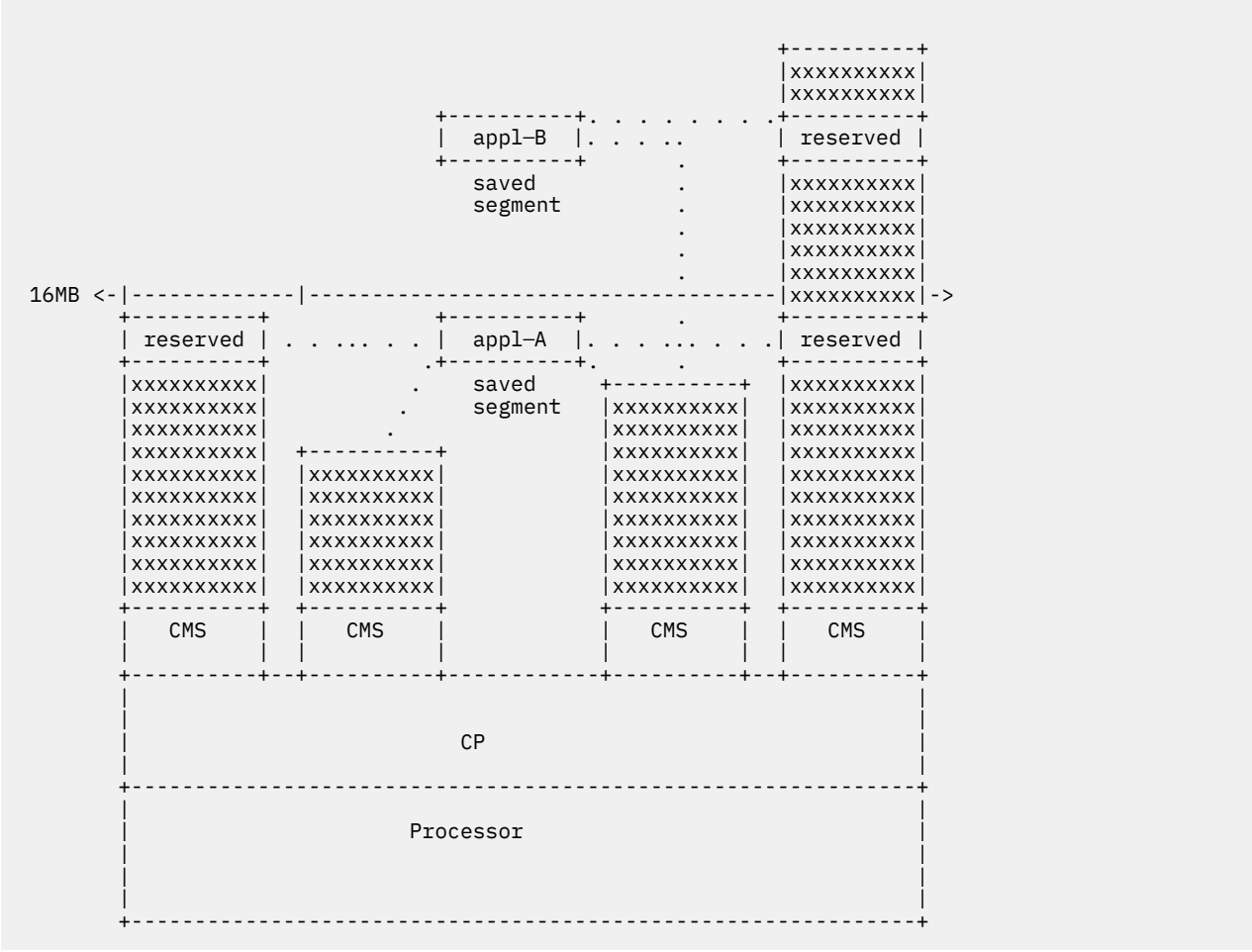


Figure 1. Sharing Saved Segments

Application B operates from a saved segment residing above the 16 MB line so it executes in 31-bit addressing mode. Also, note that the virtual machine with greater than 16 MB of storage has both application A and B attached, causing two areas of its address space to be reserved.

Using Saved Segments—An Overview

The following summarizes what you need to do to access code or data from within a saved segment:

1. Create the code or data that you want to define as a saved segment.
2. Define the saved segment:
   - If you are using VMSES/E to manage your saved segments, see "Changing, Adding, and Deleting Saved Segment Definitions" on page 62.
   - If you are not using VMSES/E:
     a. Use the CP DEFSEG command. The DEFSEG command creates a *skeleton* (class S) system data file (SDF) for the saved segment you specify. The saved segment cannot be accessed until you enter a corresponding SAVESEG command.
     b. Load the code or data to be saved into the location indicated by the ranges you specify on the DEFSEG command.
     c. Use the CP SAVESEG command to save the saved segment. The SAVESEG command writes the contents of the saved segment to spool space on DASD and changes the skeleton file to an active (class A or R) file, which can then be accessed by a virtual machine.

   For more details on creating saved segments, see "Planning Considerations" on page 7.
3. Load the saved segment into a virtual machine using one of the following methods:

- Use DIAGNOSE code X'64'.

  If you load a segment with DIAGNOSE code X'64', you must purge it with DIAGNOSE code X'64'.

  DIAGNOSE X'64' provides 64-bit subcodes for manipulating DCSSs that contain page addresses above 2047 MB.

  For more information on using DIAGNOSE code X'64', see *z/VM: CP Programming Services*.
- Use the CMS SEGMENT LOAD command or macro.

  If a saved segment is to reside within a virtual machine's address space, you should consider using the SEGMENT RESERVE command to reserve space before you enter the SEGMENT LOAD command.

  The SEGMENT command and macro provide CMS interfaces to DIAGNOSE code X'64'. For more information on using the SEGMENT command, see *z/VM: CMS Commands and Utilities Reference*. For more information on using the SEGMENT macro, see *z/VM: CMS Macros and Functions Reference*.

  Note: CMS does not support saved segments that include pages above 2047 MB. To load a saved segment that contains pages above 2047 MB, you must use DIAGNOSE code X'64'.

An application programmer generally:

- Creates the code or data that resides in a saved segment
- Provides code in the form of an installation exec that loads the data to be saved into the page ranges indicated on the DEFSEG command
- Provides code in the form of an exec or a CMS module that invokes either SEGMENT LOAD or DIAGNOSE code X'64'.

A system programmer generally defines saved segments from a class E virtual machine. DEFSEG and SAVESEG are class E CP commands; therefore, to define and save a saved segment, you need class E command privileges.

Types of Saved Segments

There are three types of saved segments: *discontiguous saved segments*, *segment spaces*, and *member saved segments*.

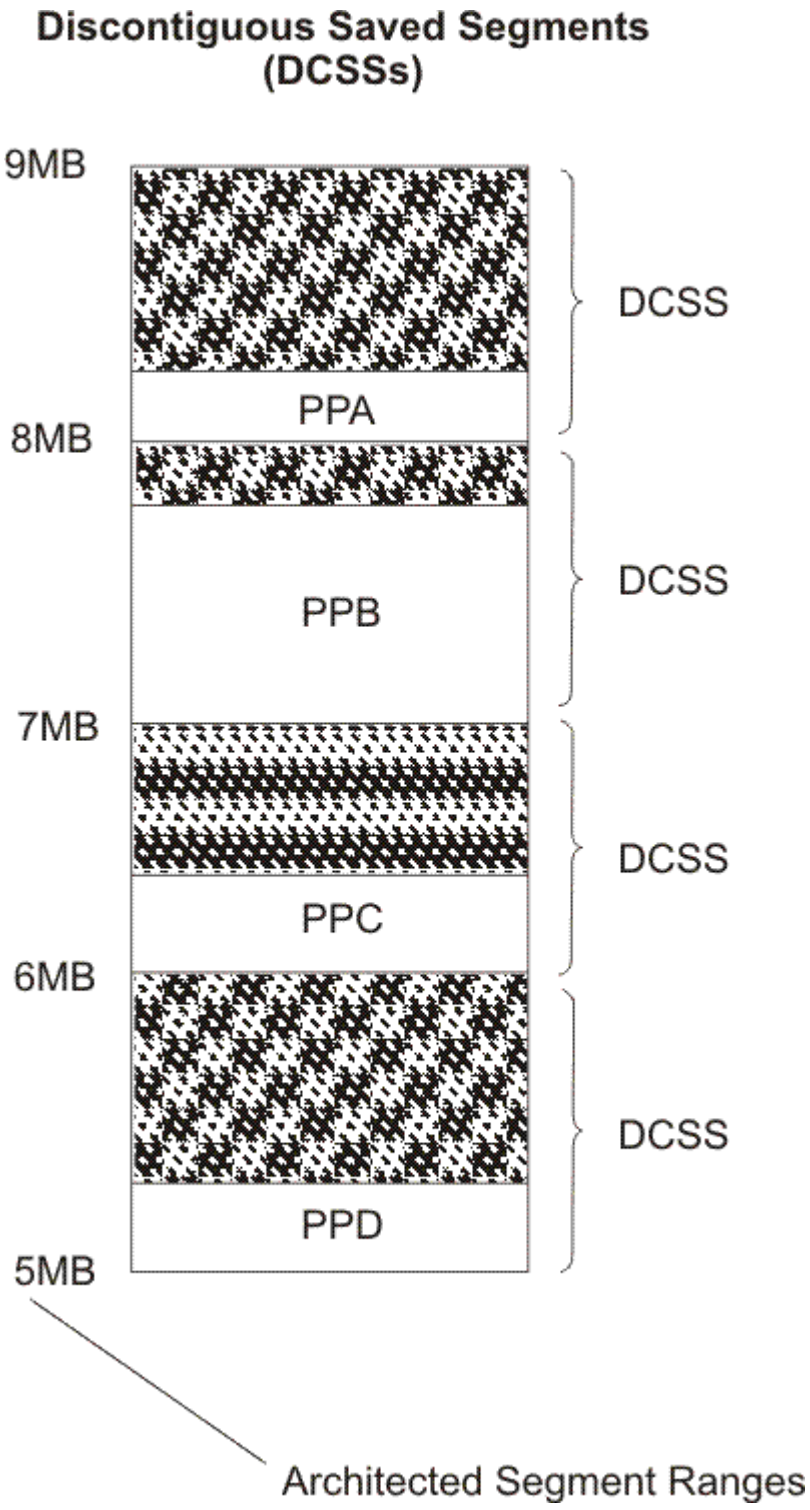


Figure 2. Saved Segments

Discontiguous saved segment (DCSS)

A DCSS occupies one or more architected segments. It is accessed by name and can be embedded above the virtual machine's defined storage size. Although individual address ranges are specified on page boundaries anywhere within an architected segment, a DCSS begins and ends on a MB boundary. Figure 2 on page 4 shows four DCSSs defined in the 5 MB to 9 MB range of architected segments. Each DCSS contains an application (represented by PPA, PPB, PPC, and PPD). By *application*, we mean a licensed program or other shared code or data.

Segment space
A segment space is a special type of DCSS that is composed of up to 64 member saved segments referred to by a single name. A segment space occupies one or more architected segments. Although individual address ranges are specified on page boundaries anywhere within an architected segment, a segment space begins and ends on a MB boundary. A user with access to a segment space has access to all its members.

Member saved segment
A member saved segment is a special type of DCSS that belongs to up to 64 segment spaces. A member saved segment begins and ends on a page boundary and is accessed either by its own name or by a segment space name. When a virtual machine loads any member of a segment space, the virtual machine has access to all members of the space. However, the virtual machine should load the other members before trying to use them. Figure 3 on page 6 shows a segment space defined in the 5 MB to 8 MB range of architected segments. This segment space contains several member saved segments, which are used to hold applications.

Notes:

1. The terms *discontiguous saved segment* and *DCSS* generally refer to saved segments that are not segment spaces or member saved segments.
2. A segment space or member saved segment can include pages up to 2047 MB. A DCSS (that is, a saved segment that is not a segment space or a member saved segment) can include pages over 2047 MB.
3. A *physical saved segment* is a DCSS or member saved segment in which CMS logical saved segments may be defined. For more information, see Chapter 2, “Planning and Defining CMS Logical Saved Segments,” on page 39. (CMS logical saved segments cannot be defined in a DCSS that contains pages above 2047 MB.)

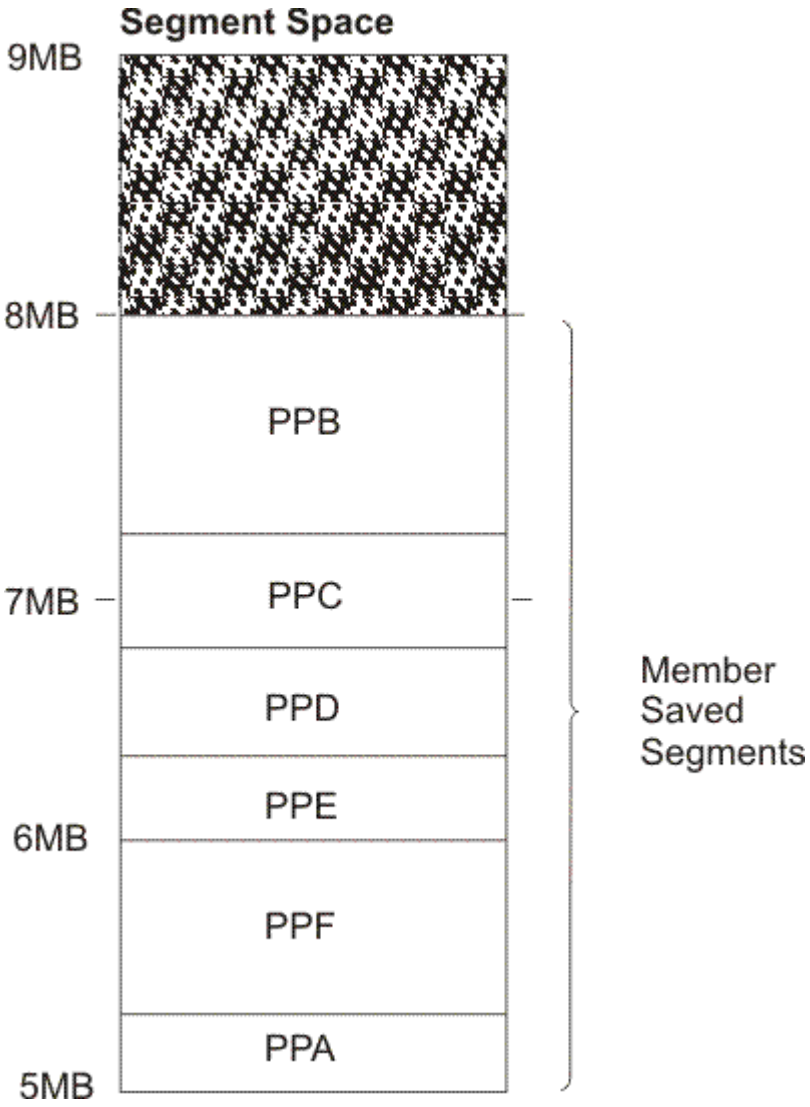


Figure 3. Member Saved Segments

Defining Saved Segments

Use the CP DEFSEG command to define a saved segment and thereby set aside storage for applications. By omitting the SPACE operand on the DEFSEG command, you define a DCSS. A DCSS is at least 1 MB in size. The address range for an application within a DCSS begins and ends on a page boundary, but the DCSS itself begins and ends on a MB boundary. *Only one application can reside in each DCSS*.

By including the SPACE operand on the DEFSEG command, you define a member of a segment space. Like a DCSS, the address range for an application within a segment space (that is, in a member saved segment) begins and ends on page boundaries, but the segment space itself begins and ends on a MB boundary. A segment space is at least 1 MB in size.

A segment space differs from a DCSS in the following ways:

- Segment spaces allow different, nonoverlapping applications to occupy the same architected segment.
- A segment space is composed of member saved segments (also called members). A member is a licensed program or application, or a component thereof, that you run in a segment space. A member begins and ends on a page boundary, and is able to span a MB boundary. A member can belong to more than one segment space.
- A segment space is created dynamically when you define member saved segments.

A segment space allows you to pack licensed programs into the same architected segment. Segment packing reclaims the address ranges that are unused within DCSSs and makes more licensed programs available to virtual machines. However, there is some CP overhead involved when segments are packed. If the programs are to be used by ESA and XA virtual machines, which can address over 2047 MB of virtual storage (XC virtual machines can address up to 2047 MB), you might prefer to avoid the overhead and complexity of segment packing by installing the programs in DCSSs.

When packing DCSSs (members) into a segment space, you should check the CP Directory entry MAINT to ensure that any NAMESAVE control statements for DCSS names are changed to the segment space name. For more information on the NAMESAVE control statement, see *z/VM: CP Planning and Administration*.

Shared and Exclusive Segments

You can specify that a program or application be placed in a shared segment, an exclusive segment, or a segment having both shared and exclusive areas. However, a saved segment having both shared and exclusive areas cannot have both areas within the same 1 MB architected segment. *Each 1 MB architected segment must be defined entirely as shared or entirely as exclusive*. For an example of defining a saved segment that has both shared and exclusive areas, see "Defining a Saved Segment with Both Shared and Exclusive Page Ranges" on page 81.

When you define a program in a shared saved segment, a virtual machine accessing it receives a shared copy of the program. When you define a program in an exclusive saved segment, a virtual machine accessing it receives its own copy of the program.

Planning Considerations

In planning for saved segments, it is important that you consider the following. These planning tips apply to both saved segments and the applications you install in saved segments. By an *application*, we mean a licensed program or other shared code or data.

1. Know your applications and their requirements. Take the following into account:
   - Make sure you are aware of the prerequisites and corequisites of the applications you will be installing. One program may require the use of others. You should make a list of all the applications your installation uses and any dependencies they have on other products. This information can be found in the application's installation manual or in the *Memo to Users* that is shipped on the installation tape.
   - Know which applications are *not* required to run together. You may be able to overlay these products by having them run in separate saved segments defined in the same address range. For more information, see "Overlaying Your Applications" on page 25.
   - Know how many pages of storage each saved segment requires.
   - Know what architecture the application exploits (or tolerates):
     - **z/Architecture® exploitation**

       The application uses 64-bit addressing and can run above the 2 GB line. See note "1.b" on page 8.
     - **ESA/390 exploitation**

       The application uses 31-bit addressing and can run above or below the 16 MB line. See note "1.a" on page 8.
     - **ESA/390 toleration**

       The application can run in a virtual machine without taking advantage of 31-bit addressing. This type of application runs under the 16 MB line, but it can call programs that reside above the 16 MB line. See note "1.a" on page 8.
     - **System/370**

       The application was written to run in a 370-mode (System/370 architecture) virtual machine, which implies that it runs under the 16 MB line. See note "1.c" on page 8.

Notes:

a. ESA and XA virtual machines are functionally equivalent, and process according to ESA/390 (31-bit) architecture. The XA mode designation is supported for compatibility, because some CMS applications may require CMS to be running in an XA virtual machine.

b. A guest operating system in an ESA virtual machine might have the capability to switch the virtual machine from ESA/390 mode to z/Architecture mode. A z/Architecture mode virtual machine supports 64-bit addressing and addressable storage beyond 2 GB.

c. 370-mode virtual machines are no longer supported. However, most CMS applications once restricted to running in a 370 virtual machine can now run in an XA or XC mode virtual machine if you issue the CP SET 370ACCOM ON command or the CMS SET CMS370AC ON command. In addition, modules generated with the 370 option of the GENMOD command can be executed in an XA or XC virtual machine by issuing the CMS SET GEN370 OFF command. See the *z/VM: CP Commands and Utilities Reference* for information on the CP SET 370ACCOM command. See the *z/VM: CP Programming Services* for more information on how to run your 370-only CMS applications in an XA or XC virtual machine. See *z/VM: CMS Commands and Utilities Reference* for information on the CMS SET GEN370 command.

- Consider the type of storage this program requires: exclusive-read or exclusive-write storage cannot be placed in the same segment as shared-read or shared-write storage. When a program requires 3 pages of exclusive-write storage and 8 pages of shared-read storage, the program will require parts of 2 segments.
- Determine whether the program has storage location dependencies.

2. Know your users and their product requirements:
   - You may not be able to supply every application that your users require. If that is the case, determine what programs are the most essential.
   - Products that are used concurrently need to be available at the same time and should not overlay each other.
   - Know if any national languages are required for a product.
   - Decide on an average virtual machine size for your users. This will help you when you install saved segments. For example, suppose a typical user at your installation needs a 4 MB virtual machine. Based on this, you should install saved segments from the 4 MB line on up. (Be aware, however, of where CMS and other system-related saved segments are loaded. This is discussed under "CMS Considerations" on page 11.)
   - Specific users may have unique product requirements. For example, a user might have FORTRAN programs that interface with GDDM®. In this case, all of these programs have to be available to this user simultaneously, so they should not be defined in saved segments that overlay each other.

By gathering the previous information, you develop a set of rules and guidelines that your installation needs to follow. After you establish these guidelines, planning for saved segments becomes a matter of moving your applications around until they fit together without breaking any of the guidelines.

To avoid defining more than one saved segment in the same address range, consider the size of a virtual machine that will access a saved segment.

Planning for Saved Segments Based on Virtual Machine Size

Assuming a saved segment is active, whether the virtual machine can load the information in the saved segment depends on:

- Where the saved segment is located
- The size of the virtual machine

CMS uses the uppermost segments of the virtual machine's free storage. Because of this, if a saved segment resides just below the 8 MB line, a 2 MB or 4 MB virtual machine can use it; an 8 MB virtual machine cannot. A 9 MB or greater virtual machine can use it if the saved segment is loaded with the

SEGMENT command or macro. If the saved segment is just below the 5 MB line, an 8 MB virtual machine can use it but a 5 MB virtual machine can not.

You should plan for saving segments on the basis of the most frequently used virtual machine size at your installation (such as the default size in the user's directory entry). If most users in your system run with 4 MB of virtual machine storage, placing all saved segments above the 4 MB line prevents collisions.

Users whose virtual machine size conflicts with saved segments should not have a problem unless they try to use DIAGNOSE code X'64' to access the saved segment. (The SEGMENT LOAD command, however, should not present any problems.) Users that need to use DIAGNOSE code X'64' can make their virtual storage size either larger or smaller and re-IPL. This may, however, cause a conflict with an entirely different saved segment.

The following sections describe how you can prevent collisions between CMS and saved segments.

Saved Segments in a CMS Machine Whose Size Is Greater Than 21 MB

For virtual machines that are 21 MB or larger, CMS uses an area that extends downward from the end of virtual machine storage. The size of this area depends on the size of the virtual machine. For virtual machines larger than 21 MB, the MB below this area represents the first available location (moving from the top down) for defining segment space.

The formula for determining the amount of storage used by CMS is:

```
CMS storage = Virtual Machine Size * 1 1/2 + 1
```

Note: Fractional values are rounded upward to the next whole number.

`CMS storage`
is measured in pages.

`Virtual Machine Size`
is measured in MB.

For example, for a 999 MB virtual machine, CMS requires 1500 pages. For a 64 MB virtual machine, CMS uses 97 pages.

To determine the address where you can safely define saved segments, subtract the storage required by CMS (rounded up to the nearest MB) from the size of the virtual machine. The MB below this value then represents the first available location for defining saved segments below the end of the virtual machine. The following table shows these calculations. It starts with a 22 MB virtual machine, because 16-20 MB virtual machines do not have room for a 1 MB saved segment above the 21 MB line.

Virtual Machine Size	CMS Storage	Safe Address for Defining Saved Segments (MB)
21-170 MB	33-256 pages	Virtual Machine Size - 1 MB
171-340 MB	257-512 pages	Virtual Machine Size - 2 MB
341-511 MB	513-768 pages	Virtual Machine Size - 3 MB
512-682 MB	769-1024 pages	Virtual Machine Size - 4 MB
683-852 MB	1025-1280 pages	Virtual Machine Size - 5 MB
853-1023 MB	1281-1536 pages	Virtual Machine Size - 6 MB
1024-1194 MB	1537-1792 pages	Virtual Machine Size - 7 MB
1195-1364 MB	1793-2048 pages	Virtual Machine Size - 8 MB
1365-1535 MB	2049-2304 pages	Virtual Machine Size - 9 MB
1536-1706 MB	2305-2560 pages	Virtual Machine Size - 10 MB
1707-1876 MB	2561-2816 pages	Virtual Machine Size - 11 MB

Virtual Machine Size	CMS Storage	Safe Address for Defining Saved Segments (MB)
1877-2047 MB	2817-3072 pages	Virtual Machine Size - 12 MB

Figure 4 on page 10 illustrates the location of storage available for saved segments for a virtual machine that is larger than 21 MB.

Figure 4. Storage Configuration for a CMS Virtual Machine Whose Size Is Greater than 21 MB

The figure shows CMS extending downward from the end of virtual machine storage, followed by an area for saved segments extending down to the 20 MB line.

Saved Segments in CMS Virtual Machines Whose Size Is Less than 21 MB

If a virtual machine is less than 20 MB, CMS uses an area that extends downward from the end of the virtual machine's storage or the starting address of the CMS nucleus whichever is smaller. For example, suppose CMS is a saved system starting at the 15 MB line. If it is IPLed in an 8 MB virtual machine, CMS uses pages extending down from the 8 MB line. If the same saved system is IPLed in a 2 MB virtual machine, CMS uses pages extending down from the 2 MB line. In a 16 MB virtual machine, CMS uses pages extending down from the 15 MB line, because the starting address of the CMS nucleus (15 MB) is less than the virtual machine size (16 MB).

Figure 5 on page 11 illustrates the location of storage available for segment spaces for a virtual machine that is less than or equal to 20 MB.

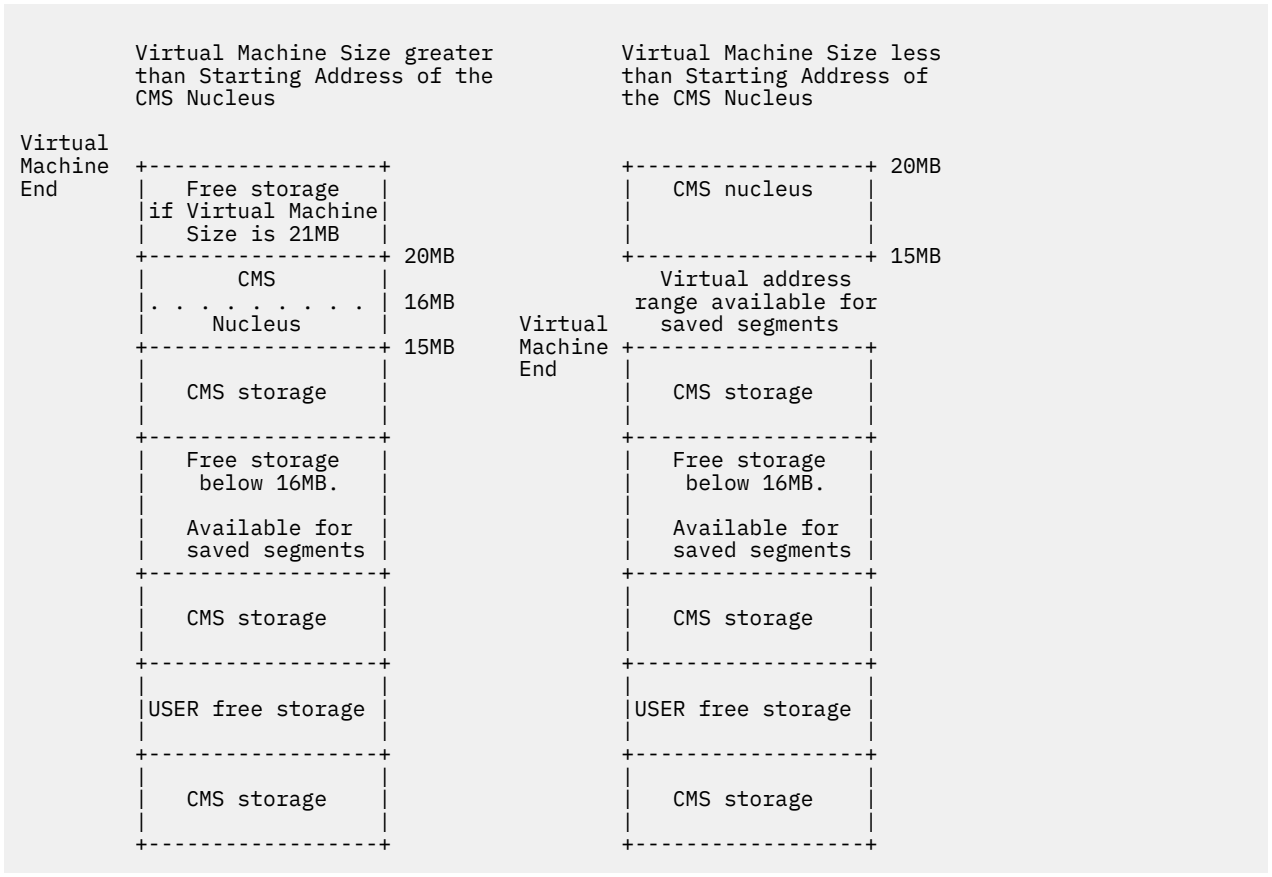


Figure 5. Storage Configuration for a CMS Virtual Machine Whose Size Is Less than 21 MB

The left half of the figure shows the storage configuration when the virtual machine size is greater than the starting address of the CMS nucleus; the right half shows the storage configuration when the virtual machine size is less than the starting address of the CMS nucleus.

CMS Considerations

For programs that operate under CMS, 2047 segments are available, addresses X'0' through X'7FEFF000'.

CMS and the default saved segments (CMSBAM, CMSDOS, DOSINST, and INSTSEG) occupy the same segments. Unless otherwise defined, CMS occupies segments 0 (EW), F, 10, 11, 12, and 13 (all SR). CMS uses portions of other segments, whose location depends on the size of the user's virtual machine and the location of the CMS nucleus. For a further explanation, see “Saved Segments in a CMS Machine Whose Size Is Greater Than 21 MB” on page 9 and “Saved Segments in CMS Virtual Machines Whose Size Is Less than 21 MB” on page 10.

Check the installation instructions for each product that you will install in your z/VM system; note any restrictions about the location at which to load the program.

System Performance Considerations

Defining saved segments at high storage addresses that will be shared by many users might affect real storage availability. For each virtual machine, CP creates dynamic address translation (DAT) tables to reference the virtual machine storage. DAT tables include page tables, segment tables, and higher level (region) tables.

CP keeps the page tables in page management blocks (PGMBKs). Each 8 KB PGMBK references 1 MB of virtual machine storage. For shared page ranges within a saved segment that is loaded shared, the associated segment table entries will point to the same page tables. However, for a saved segment that is loaded nonshared, or for exclusive page ranges within a saved segment, unique page tables are

created for each user. PGMBKs might be pageable; as such, their impact on real storage depends on how frequently the MBs of storage they reference are used.

Segment tables and region tables are allocated from host real storage and are not pageable:

- To reference the page tables for a primary address space or data space up to 2 GB, 1 - 4 contiguous frames are allocated for the segment table, one frame for each 512 MB of storage.
- For a primary address space larger than 2 GB, multiple segment tables are created, plus one or more region tables to reference the segment tables. Each region table occupies 1 - 4 contiguous frames. If needed, multiple levels of region tables are created.

Because CP dynamically expands the size of a virtual machine to incorporate a saved segment loaded at an address outside the virtual machine, the DAT tables for the virtual machine also expand. (However, addresses between the top of the virtual machine and the bottom of the saved segment are not addressable by the virtual machine.) To conserve real storage, you should try to define your saved segments at addresses closer to the sizes of the virtual machines that will use them.

Creating Saved Segments

This section describes how to set up saved segments into which you can later install applications.

To create a saved segment, you must:

1. Enter the CP DEFSEG command (DEFSEG and SAVESEG are class E CP commands). The DEFSEG command creates a *skeleton* (class S) system data file for the saved segment you specify. The saved segment cannot be accessed until you enter a corresponding SAVESEG command.
2. Load the application into the area of storage you set aside with the DEFSEG command.

   Note: CMS uses storage in the uppermost MB of the virtual machine (for a virtual machine size that is less than or equal to 16 MB) or the MB just below the 16 MB line (for a virtual machine size that is greater than 16 MB). As a result, you may not be able to load the application into this MB to create the saved segment because the storage is already in use.
3. Enter the SAVESEG command. The SAVESEG command changes a skeleton file to an active (class A or R) file.

For examples of defining and saving the different types of saved segments, see Appendix A, “Defining CP Saved Segments—Examples,” on page 81.

Using the DEFSEG Command

For the syntax and a detailed description of the DEFSEG command, see *z/VM: CP Commands and Utilities Reference*.

When the DEFSEG command is entered, system data files (SDFs) are created containing information related to the DEFSEG command input. Understanding the information contained in these SDFs will help you manage saved segments.

The scenarios in “Results of Entering the DEFSEG Command” on page 12 show which files are created (or affected) after various DEFSEG commands are entered. The abbreviations used (such as PPW, PPX, and GRP1) are the names of saved segments.

Results of Entering the DEFSEG Command

1. *Assume that no saved segment files currently exist.*
2. Enter the following command:

   ```
   defseg ppw 700-7ff sr
   ```

   A class S (skeleton) SDF with the name PPW is created and a unique spool ID number is assigned to the file.

- Because the SPACE operand was *not* specified on the DEFSEG command, the SDF defines a DCSS (that is, not a member saved segment or a segment space).
- The page range (700-7FF) and type (SR) information is saved. For multiple range specifications, the ranges are sorted from lowest to highest.

A class S file now exists for PPW.

3. Enter the following command:

```
defseg ppx 800-820 sr space grp1
```

   a. A class S SDF with the name PPX is created and a unique spool ID number is assigned to the file.
      - Because the SPACE operand was specified on the DEFSEG command, the SDF defines a member saved segment.
      - The page range (800-820) and type (SR) information is saved.
      - A count is maintained indicating how many segment spaces are associated with this member. In this case the value is 1, because GRP1 is the only segment space associated with the member PPX.
      - CP notes that GRP1 is a segment space containing PPX.

   b. A class S SDF with the name GRP1 is created and a unique spool ID number is assigned to the file.
      - Because the SPACE operand was specified on the DEFSEG command, the SDF defines a segment space.
      - The lowest and highest page range values specified for any member defined for this segment space are maintained. In this way, the overall range of a segment space is determined by its member definitions. (In the above example, the lowest value is 800 and the highest is 820.) These values are rounded down and up respectively to MB boundaries to determine the true range of pages that a segment space affects when any of its members is attached to a virtual machine. The rounded values are the ones returned by the FINDSPACE function of DIAGNOSE code X'64'.
      - A count is maintained indicating how many members are associated with this segment space. In this case, the value is 1.
      - CP notes that PPX is a member of this segment space.
      - The page range (800 through 820) and type (SR) information is saved. The lowest (800) and highest (820) page range values specified for the member are maintained.
      - The status of this entry is *not saved*, meaning no corresponding SAVESEG has been issued.

   Class S files now exist for PPX, GRP1, and PPW.

4. Enter the following command:

```
defseg ppy 821-830 sr space grp1
```

   The same processing as outlined under the DEFSEG command for PPX occurs for PPY. Note, however, the following changes to the SDF for the GRP1 segment space:

   - Because a class S SDF already exists for GRP1, this file does not have to be created.
   - The page ranges of PPY are checked to make sure that they do not overlap the ranges of any other member in GRP1. In this case, the page ranges of PPY are checked with those of PPX.
   - The count indicating how many members have been defined for this segment space is incremented by 1. In this case, the value is now 2.
   - The lowest and highest page range values specified for any member defined for this segment space are maintained. In this case, the lowest value (800) is maintained, and the highest value is updated from 820 to 830.
   - CP notes that PPY is a member of this segment space. The same information as indicated under the PPX member entry is captured for the PPY member entry.

Class S files now exist for PPY, PPX, GRP1, and PPW.

5. Enter the following command:

```
defseg ppu 800-820 sr space grp2
```

The same processing as outlined under the DEFSEG command for PPX and GRP1 occurs for PPU and GRP2.

Class S files now exist for PPU, GRP2, PPY, PPX, GRP1, and PPW.

6. Enter the following command:

```
defseg ppy same space grp2
```

a. Prior to updating the file for PPY, an existing class S file, a check is made to see if a class S SDF file exists for GRP2 (it does). The PPY file is then updated.

- The count indicating how many segment spaces have been defined for this member is incremented by 1. In this case, the value is now 2.
- CP notes that GRP2 is a segment space containing the member PPY. The class S file for PPY now has entries for GRP1 and GRP2.

b. The existing class S SDF for GRP2 is updated as follows:

- The page ranges of PPY are checked to make sure they do not overlap the ranges of any other member in GRP2. In this case, the page ranges of PPY are checked with those of PPU.
- The count indicating how many members have been defined for the segment space GRP2 is incremented by 1. The value is now 2.
- The lowest and highest page range values specified for any member defined for this segment space are maintained, and thus the overall range of GRP2 is defined. In this case, the lowest value (800) is maintained, and the highest value is updated from 820 to 830.
- CP notes that PPY is a member of this segment space. The same information as indicated under the PPX member entry is captured for the PPY member entry.

Class S files now exist for PPU, GRP2, PPY, PPX, GRP1, and PPW.

Restrictions for Using the SAMERANGE Operand

The following rules apply to the SAMERANGE operand (also called the SAME operand) of the DEFSEG command:

- You can specify the SAME operand only if you also specify the SPACE operand.
- With type EW, EN, SW, and SN saved segments, you *cannot* use the SAME operand to cause a member to belong to two different segment spaces. You can use SAME to update a segment space, replace a segment space, or add a member to the same segment space. The SAME operand is useful when you need to reinstall a read-only (SR) member of a space containing other members with writeable pages.
- You should enter a DEFSEG command with the SAME operand on the first definition of a segment space only if the member exists as a skeleton file.
- A DEFSEG command with a SAME operand needs no corresponding SAVESEG command unless the member is not yet saved. (If a definition of a member in a segment space does not include the SAME operand, a SAVESEG is required.)
- A member of a segment space cannot overlay any of the ranges specified for an existing member within the same segment space.
- There is no effect when you specify a DEFSEG command with the SAME operand for an existing member that has the same segment space name. A change to the state descriptor of the member's SDF occurs in the directory section only if you define the member in a new or additional segment space.

Using the SAVESEG Command

For the syntax and a detailed description of the SAVESEG command, see *z/VM: CP Commands and Utilities Reference*.

Note: The SAVESEG command writes all page ranges, except those defined as EN or SN, to the associated system data file. The amount of time it takes for the command to complete is directly proportional to the amount of data to be written. A subsequent DEFSEG or SAVESEG command issued by another user will be delayed behind any SAVESEG in progress.

Using SAVESEG with Your Installation Procedures

In general, customers use an installation procedure (normally an exec) to initialize the page ranges given on previously specified DEFSEG commands. After this is done, the SAVESEG command can be entered to capture the contents of the pages in the spool file that was created by the DEFSEG command and thereby save the segment. Such installation procedures vary depending on the type of code that makes up the application you plan to install.

For the virtual machine issuing the SAVESEG to get addressability to the saved segment, one of the following must be true:

- The virtual machine size must include the page ranges of the saved segment. For example, for a saved segment defined in the B00 through BFF address range, the virtual machine must be at least 12 MB.
- The virtual machine must define or use an existing saved segment containing the required pages. In other words, one of the following must be done for the saved segment:
  - It must be defined with writeable pages
  - It must be loaded with the LOADNSHR option. To do this, a virtual machine must have a NAMESAVE entry in its directory for the saved segment. The NAMESAVE statement is not required if the segment was defined with the LOADNSHR operand

    To avoid the problems associated with loading a saved segment in a virtual machine, install the application in a recently IPLed virtual machine large enough to contain the pages of the saved segment. Otherwise, perform the steps that follow. It is best that you perform these steps when you first install the application that resides in the saved segment:

    1. Define or redefine a new skeleton file for the saved segment as exclusive write (EW):

       ```
       defseg name range ew
       ```

       If the user ID doing the redefinition does not have NAMESAVE directory privileges (to load a nonshared copy), create a dummy saved segment. This dummy segment will be purged when the real segment is saved. This defines a saved segment as nonshared (EW). If you are defining a segment space or a member saved segment, enter:

       ```
       defseg name range ew space spacename
       ```

       You must define all the members of a segment space. However, any existing members can be defined with the SAME (SAMERANGE) parameter if you are not changing their page ranges:

       ```
       defseg name same space spacename
       ```

    2. Create an active segment by entering the SAVESEG command for the saved segment you are installing:

       ```
       saveseg name
       ```

    3. Set up another skeleton file by entering another DEFSEG command for the saved segment. This step is necessary, because any subsequent SAVESEG commands that happen after the installation procedure is done require a skeleton file. The saved segment you define now is the one that will be attached to your users, so you should define it with the appropriate page range type (SR in this example):

```
defseg name range sr
```

For a segment space or a member saved segment, enter:

```
defseg name range sr space spacename
```

You must define all the members of a segment space. However, they can be defined with the SAME (samerange) parameter:

```
defseg name same space spacename
```

4. Now you can run your installation procedure for saved segments. If your installation procedure is not set up to load the saved segments, use the CMS SEGMENT command to perform the load. The SEGMENT command loads the saved segment and gives the virtual machine addressability to its pages.

SAVESEG Command Functional Description

The SAVESEG command saves a saved segment that was previously defined with the DEFSEG command. The SAVESEG command copies the data from the virtual storage page ranges associated with the saved segment to the spool file that was created by the DEFSEG command. The spool file associated with the saved segment changes from a *skeleton* to an *active* file. If this is a member saved segment, the file associated with the corresponding segment space changes from skeleton to active if all other members are active.

The following SAVESEG commands correspond to the DEFSEG commands described under “Using the DEFSEG Command” on page 12. The DEFSEG commands previously specified were:

```
defseg ppw 700-7ff sr
defseg ppx 800-820 sr space grp1
defseg ppy 821-830 sr space grp1
defseg ppu 800-820 sr space grp2
defseg ppy same space grp2
```

For the examples under “Results of Entering the SAVESEG Command” on page 16, assume the respective page ranges have been properly initialized and the SAVESEG commands are entered from an installation exec. The results of each command are described.

Results of Entering the SAVESEG Command

1. *Assume that class S files for PPU, GRP2, PPY, PPX, GRP1, and PPW currently exist.*
2. Enter the following command:

   ```
   saveseg ppw
   ```

   The SAVESEG command determines whether PPW is a member saved segment:

   a. SAVESEG determines from the class S file for PPW that PPW is not a member saved segment but is a DCSS.

   b. The pages associated with PPW (as defined by the previous DEFSEG command) and their keys are copied to a system data file.

   c. The file for PPW is changed from class S to class A.

   d. If a class A file already exists for PPW, the class of the existing file is changed from A to P (pending purge). If no one is currently using the existing file, the file is purged. A class P file is purged when the last virtual machine using the file purges it from the virtual machine's address space.

   e. Because PPW is a DCSS, if PPW was defined with the RSTD (restricted) parameter on the DEFSEG command, the class would change from S to R.

   Because PPW is a DCSS, it can be attached to a virtual machine after its class becomes A or R.

   Class S files now exist for PPU, GRP2, PPY, PPX, and GRP1, and a class A file exists for PPW.

3. Enter the following command:

```
saveseg ppx
```

The SAVESEG command determines whether PPX is a member saved segment:

a. SAVESEG determines from the class S file for PPX that PPX is a member saved segment.
b. The pages associated with PPX (as defined by the previous DEFSEG command) and their keys are copied to a system data file.
c. The file for PPX is changed from class S to class A.
d. If a class A file already existed for PPX, the class of the existing file is changed from A to P (pending purge). If no one is currently using the existing file, the file is purged. A class P file is purged when the last virtual machine using the file purges it from the virtual machine address space.
e. Because PPX is a member saved segment, the associated segment space directory is updated. For each segment space entry associated with PPX, the respective class S file is processed. PPX has only the GRP1 segment space entry. The following processing occurs for GRP1:
   i) The status of PPX is changed to *saved*.
   ii) The status of other members of GRP1 (if any) is checked. The count indicating how many members are associated with this segment space is used to determine how many entries to check.

   If all the members have a saved status, the class of the segment space is changed from S to A or R. In our example, the PPY member still has a status of *not saved*. So, the class of the file associated with GRP1 remains S.

Class S files now exist for PPU, GRP2, PPY, and GRP1, and class A files exist for PPX and PPW.

4. Enter the following command:

```
saveseg ppy
```

The SAVESEG command determines whether PPY is a member saved segment:

a. SAVESEG determines that PPY is a member saved segment.
b. The pages associated with PPY (as defined by the previous DEFSEG command) and their keys are copied to a system data file.
c. The file for PPY is changed from class S to class A.
d. If a class A file already existed for PPY, the class of the existing file is changed from A to P (pending purge). If no one is currently using the existing file, the file is purged. A class P file is purged when the last virtual machine using the file purges it from the virtual machine address space.
e. Because PPY is a member saved segment:
   i) For each segment space entry associated with PPY, the respective class S file is processed. PPY has two segment space entries: GRP1 and GRP2. The following processing occurs for GRP1:
      a) The status of PPY is changed to *saved*.
      b) The status of other members of GRP1 (if any) is checked. The count indicating how many members are associated with this segment space is used to determine how many entries to check. If all the members have a saved status, the class of the segment space is changed from S to A. In our example, all members have a saved status, and the class of the file associated with GRP1 is changed to A.

      Because PPY is a member, if PPY or any other member of GRP1 was defined with the RSTD (restricted) parameter on the DEFSEG command, the class of GRP1 would change from S to R.
      c) If a class A file already existed for PPY, the class of the existing file is changed from A to P (pending purge). If no one is currently using the existing file, the file is purged. A class P file is

purged when the last virtual machine using the file purges it from the virtual machine address space.

Because PPY is a member saved segment, it can be attached to a virtual machine after one of its associated segment spaces becomes class A or R.

ii) Next, the following processing occurs for GRP2:

a) The status of PPY is changed to saved.

b) The status of other members of GRP2 (if any) is checked. The count indicating how many members are associated with this segment space is used to determine how many entries to check.

If all the members have a saved status, then the class of the segment space is changed from S to A. In our example, the member PPU has a status of not saved. So, the class of the file associated with GRP2 remains as S.

Class S files now exist for PPU and GRP2, and class A files exist for GRP1, PPY, PPX, and PPW.

5. Enter the following command:

```
saveseg ppu
```

The SAVESEG command determines if PPU is a member saved segment:

a. SAVESEG determines that PPU is a member saved segment.

b. The pages associated with PPU (as defined by the previous DEFSEG command) and their keys are copied to a system data file.

c. The file for PPU is changed from class S to class A.

d. If a class A file already existed for PPU, the class of the existing file is changed from A to P (pending purge). If no one is currently using the existing file, the file is purged. A class P file is purged when the last virtual machine using the file purges it from the virtual machine address space.

e. Because PPU is a member saved segment:

i) For each segment space entry associated with PPU, the respective class S file is processed. PPU has only one segment space entry: GRP2. The following processing occurs for GRP2:

a) The status of PPU is changed to saved.

b) The status of other members of GRP2 (if any) is checked. The count indicating how many members are associated with this segment space is used to determine how many entries to check.

If all the members have a saved status, then the class of the segment space is changed from S to A. In our example, all members have a saved status, and the class of the file associated with GRP2 is changed to A. Because PPU is a member, if PPU or any other member of GRP2 was defined with the RSTD (restricted) parameter on the DEFSEG command, the class of GRP2 would change from S to R.

c) If a class A file already existed for PPU, the class of the existing file is changed from A to P (pending purge). If no one is currently using the existing file, the file is purged. A class P file is purged when the last virtual machine using the file purges it from the virtual machine address space.

PPU can be attached to a virtual machine after its class becomes A or R.

Class A files now exist for GRP2, PPU, GRP1, PPY, PPX, and PPW.

Keeping Backup Copies of Saved Segments

VM retains saved segments in the event of a system cold start. However, because VM uses system spooling space to store saved segments, and because you may not always be able to recover spooling space after a CP abend, you should always keep backup copies of saved segments on tape. The CP

SPXTAPE command enables you to do this. For more information on SPXTAPE, see *z/VM: CP Commands and Utilities Reference* and *z/VM: System Operation*.

Purging Saved Segments from the System

Use the PURGE NSS command to purge unwanted files that contain saved segments. PURGE NSS is a class E CP command.

Note: Do not use PURGE NSS if you want to purge the saved segment only from your virtual machine. From CMS, use the SEGMENT PURGE command or macro. If you used DIAGNOSE code X'64' to load the segment, use DIAGNOSE code X'64' to purge it. (See *z/VM: CP Programming Services* for information on DIAGNOSE code X'64'.)

Example—Purging a Saved Segment

To purge the sample program XAPROG, enter:

```
purge nss name xaprog
```

After this command is entered, CP purges all copies of XAPROG *unless* XAPROG is currently in use by a virtual machine. If XAPROG is currently in use, CP places the file in a *pending purge* state and purges it as soon as XAPROG is no longer being used.

If you use PURGE NSS with the ASSOCIATES operand, you purge a saved segment and remove references to it in associated saved segments. If this saved segment is the last referred to by an associated segment, the associated segment is also purged.

Purging a segment space with the ASSOCIATES operand removes the space's name from all associated members' lists of spaces. Purging a member with this operand removes it from all spaces to which it belongs. Any member or space system data file becomes class P (pending purge) if it is currently in use. This also holds for any associated file being purged because no other members (or spaces) are associated with this file.

To determine whether a file is in pending-purge state, enter the QUERY NSS command. If the file is in pending-purge state, CP's response shows that the file is class P. CP purges class P files if:

- All virtual machines using the saved segment log off or re-IPL
- All virtual machines using the saved segment release it (with a DIAGNOSE code X'64' PURGE or a SEGMENT PURGE)

  Note: A saved segment that was loaded with DIAGNOSE code X'64' must be purged with DIAGNOSE code X'64'. A saved segment loaded with a SEGMENT LOAD command must be purged with a SEGMENT PURGE command.
- The system is IPLed
- All virtual machines using the saved segment load a new saved segment that overlays the address range of the first saved segment.

Displaying Information about Saved Segments

To display information about saved segments (and named saved systems), use the QUERY NSS ALL command with the MAP option. The following is an example of the QUERY NSS ALL MAP command. For usage notes associated with the QUERY NSS command, see *z/VM: CP Commands and Utilities Reference*.

```
query nss all map
```

In response to this command, CP displays information similar to the following:

```
FILE FILENAME FILETYPE MINSIZE   BEGPAG ENDPAG TYPE CL #USERS PARMREGS VMGROUP
spid filename filetype N/A       nnnnn  nnnnn  type c  nnnnn  N/A      N/A
```

For DCSSs that contain pages above 2047 MB, the response format is:

```
FILE FILENAME FILETYPE BEGPAG        ENDPAG        TYPE CL #USERS
spid fn       DCSSG    nnnnnnnnnnnn  nnnnnnnnnnnn  type c  nnnnn
```

Note: The format for saved segments and named saved systems below 2047 MB will be displayed first in the response, followed by the format for DCSSs above 2047 MB.

The fields in the response contain the following information:

FILE
identifies the spool ID of the file.

FILENAME
identifies the name of the saved segment or named saved system (the file name of the system data file).

FILETYPE
identifies the file type of the system data file:

Type
Meaning

NSS
Named saved system.

DCSS
DCSS below 2047 MB.

DCSSG
DCSS above 2047 MB.

DCSS-S
Segment space for which members are defined.

DCSS-M
Member of a segment space.

CPNSS
CP system service named saved system.

CPDCSS
CP system service saved segment.

MINSIZE
for NSS files only, specifies the minimum storage size of the virtual machine into which the NSS can be loaded. This field does not apply to saved segments, for which N/A is displayed.

BEGPAG
specifies the beginning page number of a page range of the saved segment or named saved system. For a segment space, this field shows the beginning page number of the entire segment space.

ENDPAG
specifies the ending page number of a page range of the saved segment or named saved system. For a segment space, this field shows the ending page number of the entire segment space.

TYPE
indicates the type of virtual machine access (ER, EW, EN, SR, SW, SN, or SC) allowed to a page range of the saved segment or named saved system. For more information about these codes, see the DEFSEG command in *z/VM: CP Commands and Utilities Reference*. For a segment space, the access type might not apply to the entire range, so a dash (–) is displayed.

CL
indicates the current class (state) of the system data file:

A
Unrestricted available state. This means the system data file has been defined and saved. To determine whether the saved segment or named saved system is in use, examine the #USERS field in the response.

P
Pending purge state. This means the PURGE NSS command (or DIAGNOSE code X'64') has been issued for the name of this saved segment or named saved system but virtual machines are still accessing it. No new users can access this saved segment or named saved system. The file will be purged when the last virtual machine releases the saved segment or named saved system, or during the next system IPL or RESTART.

R
Restricted access available state. This means the system data file has been defined (with the RSTD option) and saved. Access to a restricted saved segment or named saved system requires a NAMESAVE directory statement. To determine whether the saved segment or named saved system is in use, examine the #USERS field in the response.

S
Skeleton state. This means the file has been defined by a DEFSEG or DEFSYS command, and the SAVESEG or SAVESYS command can now be run to complete this system data file.

#USERS
indicates the number of users attached to the saved segment or named saved system.

PARMREGS
for NSS files only, identifies the registers in which parameters are passed to the virtual machine at IPL. This field does not apply to saved segments, for which N/A is displayed.

VMGROUP
for NSS files only, indicates whether the NSS is part of a virtual machine group. This field does not apply to saved segments, for which N/A is displayed.

You can also use QUERY NSS NAME *name* MAP and QUERY NSS *spoolid* MAP to display information about saved segments.

Displaying Which Users Have Loaded a Saved Segment

To display which user IDs have loaded a specified saved segment, use the QUERY NSS USERS command. If the saved segment is a segment space, the response lists those users of the space and each of its members. If the saved segment is a member, users with the member loaded and those with its associated space(s) loaded are listed. For example:

```
query nss users tstspace
```

where TSTSPACE is the name of a segment space with two members, MEMBER01 and MEMBER02. In response to this command, CP displays information similar to the following:

```
FILE     FILENAME     FILETYPE    CLASS
0466     TSTSPACE     DCSS-S      A

NONE

FILE     FILENAME     FILETYPE    CLASS
0465     MEMBER01     DCSS-M      A

USERA   USERB   USERC   USERD   USERF   USERG

FILE     FILENAME     FILETYPE    CLASS
0467     MEMBER02     DCSS-M      A

USER1    USER2        USER3
```

In the example above, USER1, USER2, and USER3 have explicitly loaded MEMBER02. (MEMBER01 is in the virtual storage of virtual machines because it is in the same space as MEMBER02. Nevertheless, these virtual machines have not explicitly loaded MEMBER01.) USERA, USERB, USERC, USERD, USERF, and USERG have explicitly loaded MEMBER01.

Note that if there is a pending purge version, its users are also shown.

For further examples of the QUERY NSS command, see “Examples of Segment Spaces” on page 24.

Installing Applications in Saved Segments

This section tells you how to install an application in a saved segment. It discusses segment packing and describes how to overlay segment spaces.

Tips for Installing Your Applications in Saved Segments

Below are some recommendations that may help you when you are preparing to install an application in a saved segment.

- Storage management determines the default load address, which may vary.
- When you install CMS at the default locations, it uses segments F, 10, 11, 12, and part of 13. All of 13 is set aside when CMS is set up as a named saved system (NSS). You cannot combine an NSS and a saved segment within the same architected segment.

  If you install CMS at the default locations, you should leave the segment below the CMS shared pages free of saved segments. CMS may use some of the D segment when CMS is defined in the default segments.
- Consider writing an exec that lists the DEFSEG commands used to build the saved segments for a product. For example, you could write an exec that enters the new DEFSEG commands, installs the products, and enters the SAVESEG commands.

Fitting Applications below the 16 MB Line

If your installation has a large number of applications that must all run below the 16 MB line, the following suggestions may help you when you set up your saved segments. Note, however, that these tips may not work in every environment.

To fit your applications below the 16 MB line, consider the following ideas:

- If you have an architected segment defined to hold exclusive code (for example, type EW) and the segment has unused space, convert an application that normally runs in a shared segment so that it now operates from an exclusive segment. Then, pack this application into the architected segment where your other EW applications reside. To do this, you must have room in the architected segment for this application. Converting this application to EW may affect performance somewhat, but it will help you ease your storage constraints.

Using Segment Packing to Conserve Storage Space

An *overlay* is two or more saved segments defined in the same address range. If possible, you should avoid overlaying saved segments. The more overlays you have, the more system overhead may increase.

To avoid overlays and provide more efficient use of storage, VM allows you to define *segment spaces* into which you can pack multiple applications. A segment space is on a MB boundary, but each application is stored on a page boundary. Thus, you can store many more programs in a given area without wasting storage. After you take advantage of z/VM and can address licensed programs above the 16 MB line, you do not need to store licensed programs so tightly; 1 MB segments will be sufficient.

You can mix shared and nonshared code within the same segment space as long as you do not mix them within the same 1 MB segment. Thus, if you have a licensed program that requires both shared and exclusive code, you can store both parts in the segment space (but you must store them in separate 1 MB segments).

Figure 6 on page 23 shows programs A, B and C, all stored in one segment space that spans three 1 MB segments (from the beginning of MB 4 to the end of MB 6). In this case, programs A, B and C are all considered *member saved segments* of this segment space. Note that the boundaries of the segment space are rounded to MB boundaries. Program C requires both shared and exclusive code.

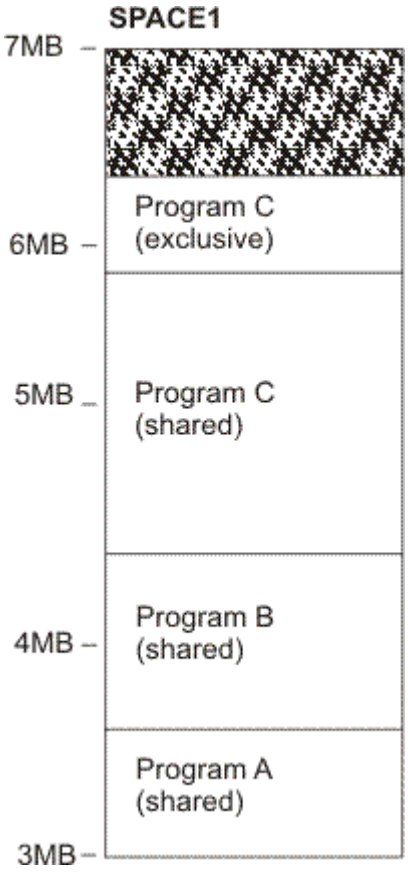


Figure 6. Using a Segment Space to Store Applications

Using a DCSS Compared with Using a Segment Space

When to Use a Segment Space

The following are some reasons for packing programs into segment spaces, instead of defining a DCSS for each program:

- If more than one member segment will fit in a 1 MB segment space
- If your system is constrained for virtual storage below the 16 MB line
- If your system has a *family* of related programs (for example, SQL and QMF™) that are normally used at the same time. You may benefit from defining them in the same segment space, because when a member is loaded for the first time, all other members in the segment space are also brought into storage. Thus, you may reduce I/O operations if you can keep related code or data in the same segment space. If you have very few QMF users (QMF uses SQL) and many SQL users, you might not want to include QMF in the same segment space as SQL. The layout of the segments depends on how they will be used in your environment.

For some examples of segment space definitions, see "Defining a Segment Space" on page 82.

When to Use a DCSS

In the following situations, you may benefit from defining a DCSS instead of a member:

- If the program size is an exact multiple of 1 MB or slightly smaller than a multiple of 1 MB, you will eliminate any overhead caused by using segment spaces.
- If you can fit all your segments below the 16 MB line without packing them into segment spaces, you can eliminate the overhead.

- If your segments reside above the 16 MB line, it is unlikely that your system is constrained because of the large number of segments.
- If you need to include pages above 2047 MB, you can do this only with a DCSS.

For examples of DCSS definitions, see “Defining Overlaid DCSSs” on page 82.

Tips for Using Segment Spaces

Here are some practical tips to help you install applications in segment spaces:

- If you have a program that spans beyond a MB boundary, it may *not* be worth adding additional programs to round out the unfilled last MB. When the smaller program is invoked by a user, the whole segment space gets loaded. Although the larger program is not being used, it may cause an overlay with another DCSS or segment space. For example, GDDM spans beyond 2 MB. If you install GDDM, you must evaluate whether you should create a segment space with GDDM and some other product that does not go beyond the third MB boundary, or define a DCSS for GDDM and put the other product somewhere else. If the other product fits in the remainder of the third MB and that product and GDDM are frequently used together, you should combine them in the same segment space. Because GDDM and ISPF are often used together, you may want to pack them into a segment space.
- Both shared and exclusive (nonshared) pages cannot exist in the same architected segment. A given segment must be either shared or exclusive. Nevertheless, a segment space, and even a member segment, can have both shared and exclusive code if the shared pages are not in the same architected 1 MB segment as the exclusive pages.
- With the SAMERANGE operand on the DEFSEG command, you can make a member part of another (or the same) segment space without redefining the page ranges and saving the member again. An example of when the SAMERANGE operand is useful is if CMSDOS, CMSBAM, CMSVSAM, and CMSAMS are all in the same segment space and CMSDOS requires service. You will need to save CMSDOS again, but you can use SAMERANGE for the other saved segments rather than saving them all again.

  Note, however, that the SAMERANGE operand cannot be used if the member contains SW, SN, EW, or EN pages and you are defining it in a second segment space.

Problems with Large Segment Spaces

Some possible consequences of creating large segment spaces are:

- Because the whole segment space gets loaded when a member is loaded, part of the segment space may overlay another saved segment that is loaded in a user's virtual machine.
- Large segment spaces limit the size of the virtual machines that use them. Although users can load a saved segment in their virtual machine if they use the CMS SEGMENT command or macro, the saved segment cannot span from below the virtual machine size to above it. CMS uses the uppermost segment in a user's virtual machine. If a segment space spans from the page X'400' to X'AFF', users of any member of that segment space would need a virtual machine that is 4 MB or less if they are not using the SEGMENT command or macro. If they use the SEGMENT command or macro, the virtual machine could be 4 MB or less, or greater than 12 MB, but it cannot be between 4 MB and 12 MB.

Examples of Segment Spaces

The products you choose to combine in a segment space and the location you load them at will vary depending on your users' requirements. The page ranges shown may not be the actual page ranges of these programs.

To see a sample storage mapping for a given set of applications, see “Setting Up Your Storage Layout” on page 88. The following are examples of what CP might return in response to QUERY NSS commands for varying configurations of applications:

1. GDDMXAL contains GDDM/GKS, GDDM/IMD, and GDDM/IVU:

```
query nss map name gddmxal
FILE FILENAME FILETYPE MINSIZE BEGPAG ENDPAG TYPE CL #USERS PARMREGS VMGROUP
1112 GDDMXAL  DCSS-S     N/A   00600  006FA   --  A  00000   N/A      N/A
1113 ADMGK000 DCSS-M     N/A   00600  00658   SR  A  00000   N/A      N/A
1114 ADMIM000 DCSS-M     N/A   00660  006C4   SR  A  00000   N/A      N/A
1115 ADMIV110 DCSS-M     N/A   006D0  006FA   SR  A  00000   N/A      N/A
Ready; T=0.01/0.01 12:57:46
```

2. SQLDCS1 contains some of the SQL segments and QMF:

```
query nss map name sqldcs1
FILE FILENAME FILETYPE MINSIZE BEGPAG ENDPAG TYPE CL #USERS PARMREGS VMGROUP
1069 SQLDCS1  DCSS-S     N/A   00700  008C0   --  A  00000   N/A      N/A
1070 QMF220E  DCSS-M     N/A   00700  0084F   SR  A  00000   N/A      N/A
1071 SQLRMGR  DCSS-M     N/A   00850  00860   SR  A  00000   N/A      N/A
1072 SQLISQL  DCSS-M     N/A   00861  008C0   SR  A  00000   N/A      N/A
Ready; T=0.01/0.01 12:58:42
```

3. SQLDCS2 contains SQL segments that are not used by QMF:

```
query nss map name sqldcs2
FILE FILENAME FILETYPE MINSIZE BEGPAG ENDPAG TYPE CL #USERS PARMREGS VMGROUP
1073 SQLDCS2  DCSS-S     N/A   00900  00AA6   --  A  00000   N/A      N/A
1074 SQLSQLDS DCSS-M     N/A   00900  009D0   SR  A  00000   N/A      N/A
1075 SQLXDRS  DCSS-M     N/A   009D1  00AA6   SR  A  00000   N/A      N/A
Ready; T=0.01/0.01 12:58:53
```

   You should consider defining both SQLDCS1 and SQLDCS2 as shown because QMF may use SQLRMGR and SQLISQL, but will not use the SQL segments in SQLDCS2.

4. GAMDCSS contains GAM/SP (Graphics Access Method/SP) segments and GDDM/graPHIGS. In this case, the segment space is still class S because the GDDM/graPHIGS segment, AFMASS00, has not yet been saved. Although the GAM segments have been saved, they cannot be used until member AFMASS00 has been saved.

```
query nss map name gamdcss
FILE FILENAME FILETYPE MINSIZE BEGPAG ENDPAG TYPE CL #USERS PARMREGS VMGROUP
1146 GAMDCSS  DCSS-S     N/A   00800  008F3   --  S  00000   N/A      N/A
1147 CMSGAM   DCSS-M     N/A   00800  0080F   SR  A  00000   N/A      N/A
1148 GAMBUF   DCSS-M     N/A   00810  00811   SW  A  00000   N/A      N/A
1149 AFMASS00 DCSS-M     N/A   00812  008F3   SR  S  00000   N/A      N/A
Ready; T=0.01/0.02 16:29:09
```

Overlaying Your Applications

If segment packing does not sufficiently reduce your installation's storage constraints, you may need to define overlaid saved segments or overlaid segment spaces.

When you overlay two segment spaces, the second segment space that is loaded into storage replaces the entire address range of the first segment space, even if it is not defined at exactly the same location as the first segment space. Also, all of the first segment space is removed from the user's address space. An example is the storage layout in Figure 7 on page 26. If you load SPACE3 while SPACE2 is loaded, all of SPACE2 is removed from the guest's address space even though only part of SPACE2 is overlapped. Similarly, if you load OVVM while SPACE2 is loaded, all of SPACE2 is removed from the user's address space.

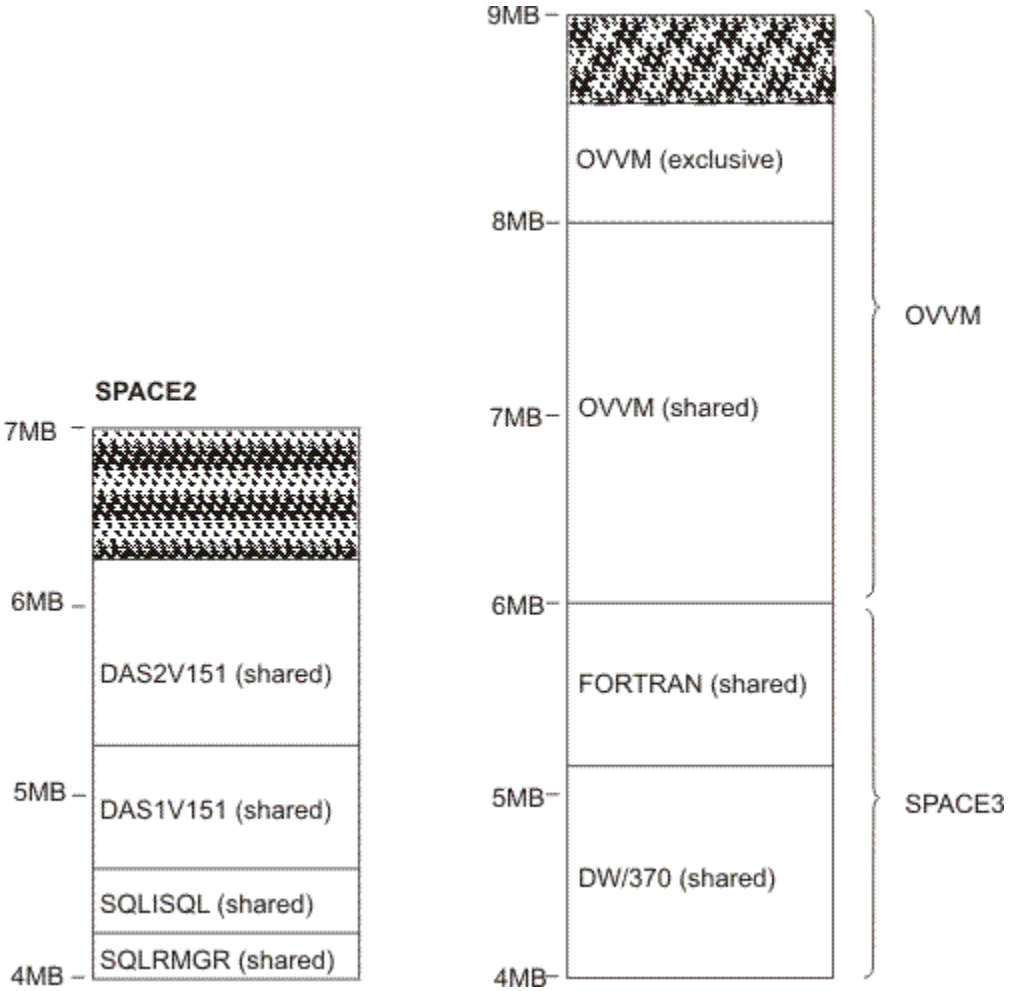


Figure 7. Using Segment Spaces to Overlay Applications

DAS1V151 and DAS2V151 are saved segments associated with Application Systems (AS). SQLISQL and SQLRMGR are saved segments associated with the SQL user machine.

When you overlay two saved segments, it is not necessary for both saved segments to have the same type of code. That is, if you define program A to overlap program B, A and B do not both have to contain shared code; also, A and B do not both have to contain exclusive code. One program can have shared code, and the other can have exclusive code, provided both programs are not needed at the same time.

Remember that all parts of a particular product need not be packed into one segment space. Given the needs of your installation, other arrangements may be better. Some products require more than one segment, but the functions performed by the code allow you to overlap the segments. For example, suppose you have one set of users that:

- Use SQL/DS with AS
- Use SQL/DS with QMF
- Do not use QMF with AS.

You can pack these products together as shown in Figure 8 on page 27, which shows three packed segment spaces:

- SPACE1 has SQL/DS with AS (6 - 9 MB). Its components are DAS1V151, DAS2V151, SQLISQL, and SQLRMGR.
- SPACE2 has SQL/DS with QMF (6 - 8 MB). Its components are QMF220E, SQLISQL, and SQLRMGR. SQLISQL and SQLRMGR occupy the same addresses as those components in SPACE1.
- SPACE3 is the SQL/S database machine (6 - 8 MB). Its components are SQLXRDS and SQLSQLDS.

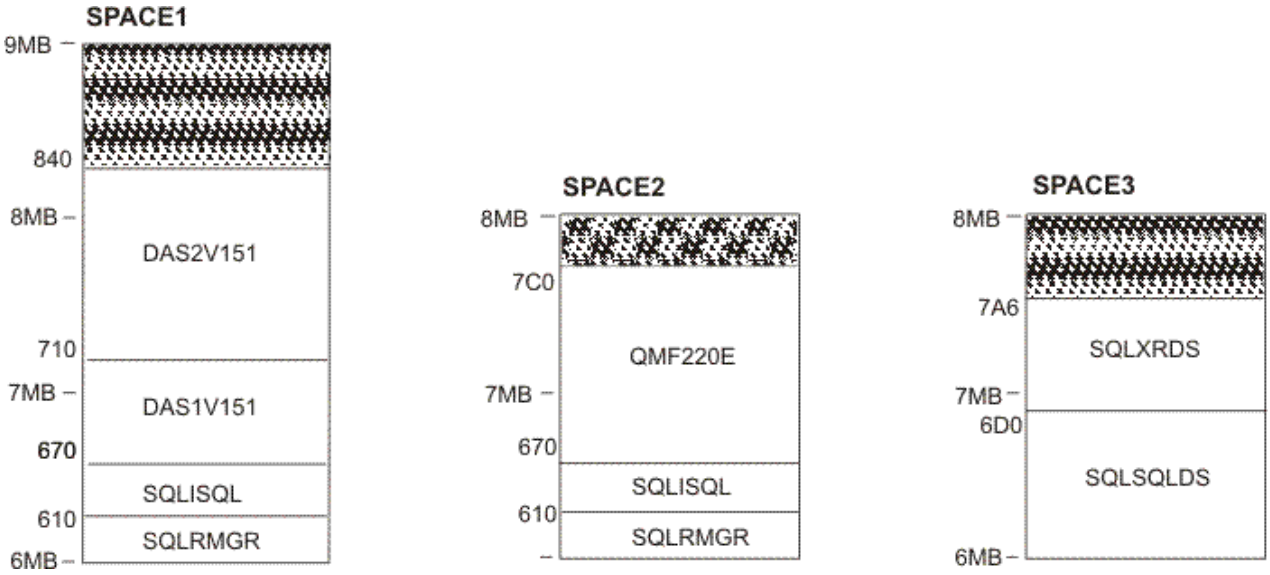


Figure 8. Installing SQL with Overlays

Although you need two segments for the SQL database machine (SQLSQLDS and SQLXDRS) and two segments for the user (SQLISQL and SQLRMGR), the segments for the database machine can overlap the segments for the user, as shown in Figure 8 on page 27.

If you are defining a member in multiple segment spaces, define the member first in the space having more commonly used members or having the highest beginning address. For example, in Figure 8 on page 27, if you have more QMF users than AS users, define SQL in SPACE2 first (and use the SAME operand to define SQL in SPACE1).

For an example of defining the overlays shown in Figure 8 on page 27, see "Defining Overlaid Segment Spaces" on page 83. To see a sample storage mapping that includes overlays, refer to "Setting Up Your Storage Layout" on page 88.

Additional Overlay Possibilities

This section discusses the different ways you can overlay programs. The following methods are discussed:

- Defining overlaid DCSSs
- Defining overlaid segment spaces.

Defining Overlaid DCSSs

One way to overlay program components is to define each component in a separate DCSS and define each DCSS in the same address range. Figure 9 on page 28 shows PPT3, PPT4, and PPT5—components of the program PPT—defined in separate DCSSs. These three DCSSs are all defined in the 7 MB to 8 MB range of architected segments. This type of overlay arrangement may reduce the amount of storage used. Note, however, that the programs PPT3, PPT4, and PPT5 are *mutually exclusive*; that is, they are not used at the same time.

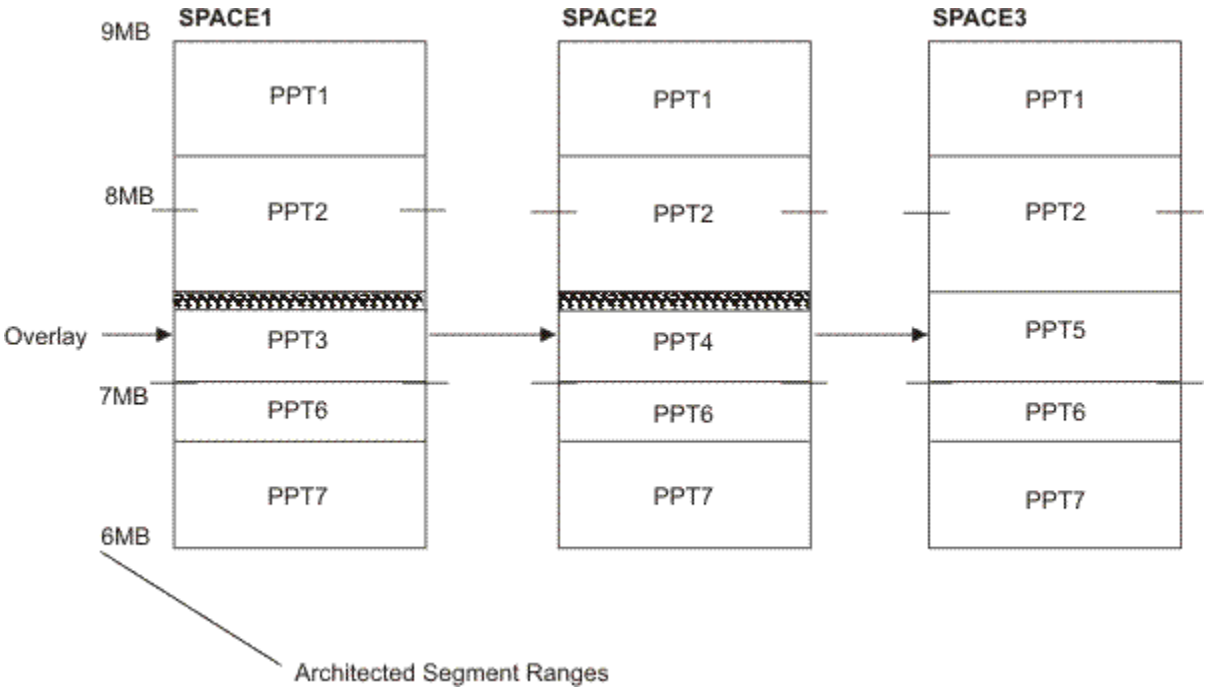


Figure 9. DCSSs as Overlays

Defining Overlaid Spaces with One Unique Member in Each Space

If virtual storage at your installation is extremely constrained, you may want to consider the following overlay structure. You may want to define multiple segment spaces with one unique member in each space. In Figure 10 on page 28, SPACE1, SPACE2, and SPACE3 are each segment spaces. Each segment space contains a set of common members and one unique member.

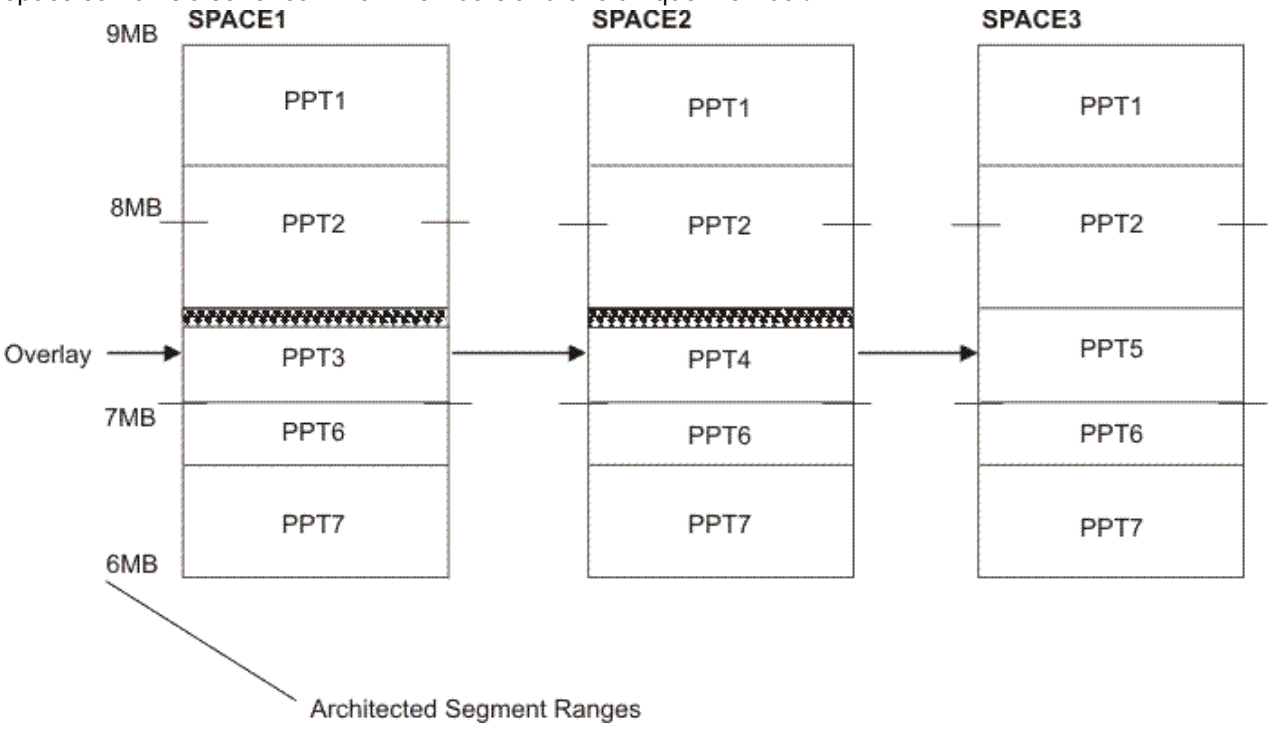


Figure 10. Segment Spaces as Overlays

Note, however, that with this type of arrangement, system overhead increases when a user calls a program not currently in the user's address space. (When a user calls a program not currently in the

user's address space, the entire segment space that contains the called program overlays whatever was executing in the user's address space. Any currently loaded saved segments are overlaid by the new segment space or removed from the user's address space.) Because of this, the set of common members —PPT1, PPT2, PPT6, and PPT7 in this example—must be refreshable programs. In other words, none of these common members can have writable storage. This is an example of why you cannot use the SAME operand of the DEFSEG command to place a member with writeable pages in more than one segment space.

With the above configuration, seven programs are defined in just three architected segments. The procedure for defining SPACE1, SPACE2, and SPACE3 above is described in detail in “Defining Overlaid Segment Spaces” on page 83.

Overlaid Segment Spaces across Several Applications

Segment spaces allow you to define different applications in the same address range. If you have two sets of mutually exclusive applications, you should define a separate segment space for each set. Figure 11 on page 30 shows two overlaid segment spaces—SPACE2 and SPACE4—where SPACE2 and SPACE4 are mutually exclusive. This configuration is similar to the overlay situation shown in Figure 10 on page 28 except that:

- In Figure 11 on page 30, the segment spaces represent multiple applications rather than just one.
- The situation in Figure 10 on page 28 requires that all common code be in each segment space, whereas in Figure 11 on page 30 no code is duplicated.

The configuration in Figure 11 on page 30 allows one set of users to access SPACE1, SPACE2, and SPACE3, for example, while another set of users accesses SPACE1, SPACE3, and SPACE4.

The procedure required to define SPACE2 is described in detail under “Defining a Segment Space” on page 82.

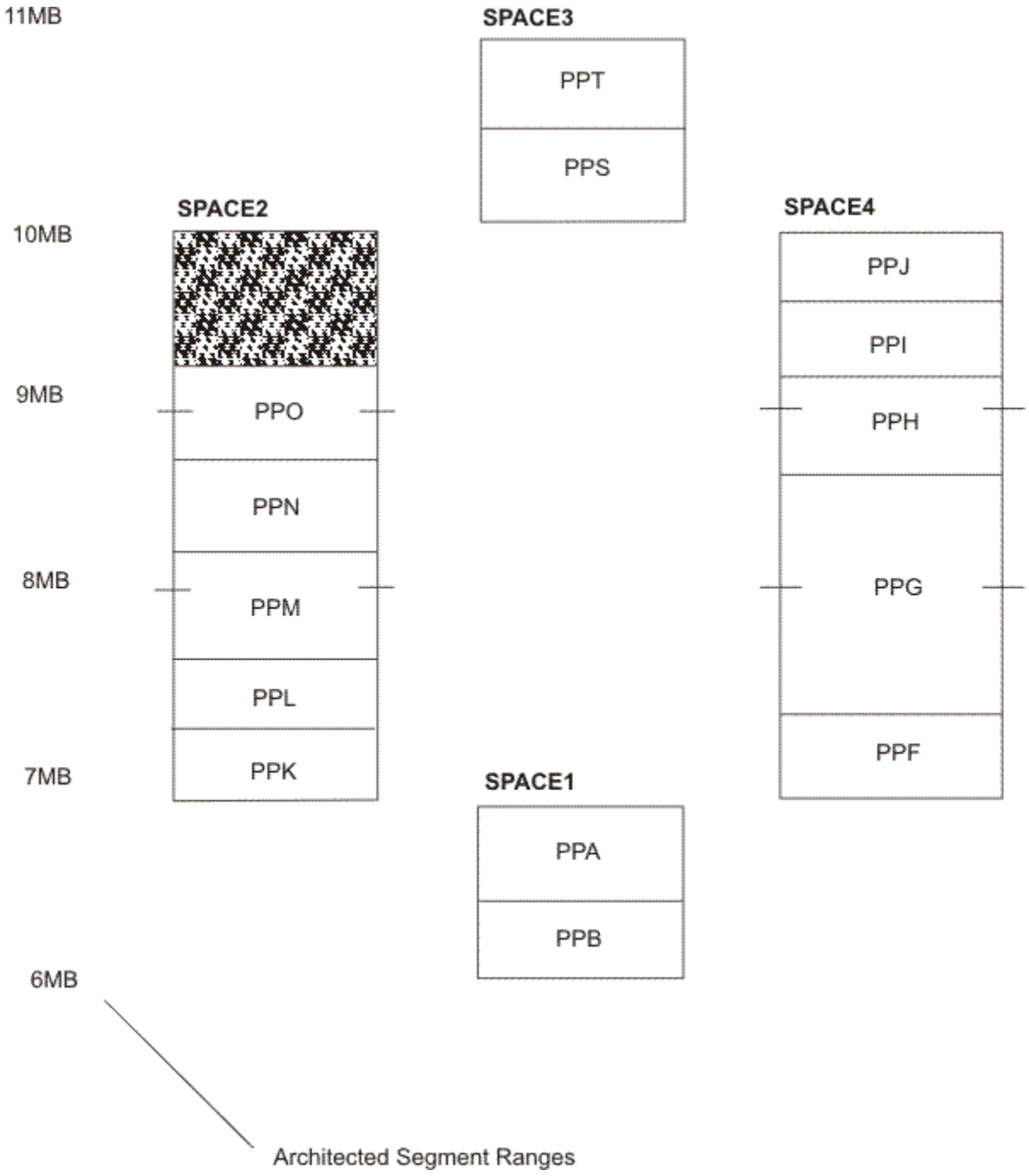


Figure 11. Mutually Exclusive Segment Spaces as Overlays

Redefining Saved Segments

At some point, you may need to redefine some of your saved segments because of service upgrades or a product's ability to take advantage of ESA/390 architecture (the product's ability to be loaded above the 16 MB line) or z/Architecture (the product's ability to be loaded above the 2 GB line). When you redefine a saved segment that was residing below the 16 MB line to now be above the 16 MB line, you may want to change the saved segment from a member to a DCSS. Although some virtual storage within an architected segment will be unused, using a DCSS will give you greater flexibility in terms of product combinations. To include addresses above 2 GB in a saved segment, it must be defined as a DCSS.

For an example of redefining an existing saved segment, see “Replacing an Existing Member of a Segment Space” on page 84.

Table 1 on page 31 through Table 3 on page 31 assist you in redefining saved segments. These tables indicate whether a saved segment definition with the DEFSEG command will be successful (indicated by *yes*) or unsuccessful (*no*) depending on whether a saved segment with the same name as the one being defined exists. The information in each table refers to the characteristics (class and type) of any *existing* saved segments.

Table 1 on page 31 shows, for example, that you cannot define a DCSS if a class A or R segment space with the same name already exists.

Table 1. Defining a DCSS

Type of Existing Saved Segment			
Class	**DCSS**	**Member**	**Space**
A or R	Yes	Yes	No
S	No	No	No
P	Yes	Yes	Yes
None	Yes	Yes	Yes

Table 2 on page 31 shows, for example, that you can define a member if a class P DCSS with the same name already exists.

Table 2. Defining a Member

Type of Existing Saved Segment			
Class	**DCSS**	**Member**	**Space**
A or R	Yes	Yes	No
S	No	No	No
P	Yes	Yes	Yes
None	Yes	Yes	Yes

Table 3 on page 31 shows, for example, that you cannot define a segment space if a class S member with the same name already exists. Also, you can define a segment space if a skeleton with that name exists because you are really just adding another member to that space. This does not create a new system data file for the space; it adds another member to that space's directory.

Table 3. Defining a Segment Space

Type of Existing Saved Segment			
Class	**DCSS**	**Member**	**Space**
A or R	No	No	Yes
S	No	No	Yes
P	Yes	Yes	Yes
None	Yes	Yes	Yes

System Data Files

System data files are provided in order to eliminate the need to preallocate fixed amounts of direct access storage space for certain system functions. System data files hold NSSs, saved segments, printer image libraries, national language support files (such as message repositories), and system trace data. Storing these collections of system data in system data files allows them to be changed and regenerated during system operation.

On z/VM a new NSS is added by defining it in a system data file using the CP DEFSYS command. No regeneration or re-IPL of CP is required.

System data files are managed in a manner similar to spool files. They occupy spooling space, can be created and purged during system operation, are maintained by CP in a set of queues which can be queried, and can be backed up to tape by way of the SPXTAPE command.

Unlike spool files, however, system data files are not associated with virtual devices and do not contain CCWs that control real output devices. Also, because their information pertains to system functions, system data files (with the exception of system trace files) are not associated with particular virtual machines and cannot be sent from one user to another. System trace files are associated with a particular virtual machine and can be transferred to another user.

System Data File Classes

Like spool files, system data files have classes. However, these classes do not associate the files with particular real or virtual devices, but to indicate the state of the system data file.

A system data file's class may be one of the following:

A
Class A system data files are available for use. This class applies to all system data files except UCR files. Class A NSSs or saved segments have been defined and saved, class A image libraries are available for use by real printers, class A message repository files are available for use, and class A system trace files can be processed.

D
Class D applies to UCR and dump files. UCR files are not supported and no longer processed by CP.

I
Class I applies only to UCR files. UCR files are not supported and no longer processed by CP.

R
Class R applies only to NSSs, saved segments, and segment spaces. It indicates that the file is available but its use is restricted to one or more authorized virtual machines. In order to gain access to a restricted system data file, a virtual machine's z/VM directory entry must contain a NAMESAVE statement for the NSS or saved segment.

P
A class P system data file is in the pending purge state, which indicates that it will be purged either when the last virtual machine releases it or during the next system IPL or restart. This class applies only to NSSs, saved segments, and message repository files. CP purges image libraries, UCR files, and system trace files immediately.

S
Class S also applies only to NSSs and saved segments and indicates that the file is in the skeleton state. This means that the NSS or saved segment has been defined by way of a DEFSYS or DEFSEG command, but it has not yet been saved by way of a SAVESYS or SAVESEG command.

W
Class W applies only to system trace files. It indicates that a trace is active and that CP is currently writing trace data to the set of files associated with that trace.

A system data file class is assigned to a system data file when it is created and CP changes the system data file's class to match its state, as needed. For example, when a class E user enters a DEFSYS command to define an NSS, CP assigns class S to the newly-defined NSS file. After the user has IPLed the system to be named and saved and has entered the SAVESYS command, CP assigns class A or R to the completed NSS file. The NSS is now available to be used by one or, if shared, more users. If the class E user enters a PURGE command to remove a shared NSS from the system, but it is still in use by other users, CP marks the file as pending-purge and assigns class P to the file. When the last user releases the NSS from use, the file is purged.

Creating and Deleting System Data Files

The class E user creates a skeleton NSS file by entering the DEFSYS command that specifies a number of parameters including the page ranges to be included in the NSS and the type of access to be permitted for each range of pages. After the skeleton NSS file has been created, the system that is to be saved in the NSS must be IPLed in the class E user's virtual machine. At the point in processing at which the system is to be saved, the user must enter the SAVESYS command and specify the name that was assigned to the system by the DEFSYS command. The NSS is then available to be loaded into the virtual storage of other virtual machines through the IPL command.

In a similar manner, the class E user creates a skeleton saved segment file by entering the DEFSEG command, again specifying the page ranges to be included and their permitted access among other parameters. After the skeleton saved segment file has been created, the data that is to be saved must be loaded into the virtual storage addresses it is to occupy. To save the data in the saved segment, the class E user enters the SAVESEG command. The saved segment is then available to be loaded into the virtual storage of other virtual machines through DIAGNOSE code X'64', or in CMS virtual machines through the CMS SEGMENT LOAD command or macro. If a saved segment is to reside in a virtual machine's address space, you should consider using the SEGMENT RESERVE command to reserve space before you enter the SEGMENT LOAD command.

The definition of NSSs and saved segments can be verified by the class E user by way of the QUERY NSS MAP command. The queue named NSS contains both NSS and saved segment system data files. The class E user can delete all or selected NSS and saved segment files that are not currently attached to virtual machines by way of the PURGE NSS command. NSS and saved segment files can be deleted by spool ID or file name. In addition, by means of the ASSOCIATES operand on the PURGE NSS command, the user can delete all files associated with a segment space or member saved segment.

Image libraries for impact printers and 3800 printers are created using the IMAGELIB service program. This program uses DIAGNOSE code X'74' to load and save a named saved image into a 3800 or impact printer image library. The image library is then available to be used by a real printer. The spooling operator can specify that a real printer is to use the image library by entering its name on the IMAGE operand of the class D START command.

The QUERY IMG command can be used to display all of the existing image libraries. All or selected image libraries that are not in use can be deleted by way of the PURGE IMG command. Image library files can be deleted by spool ID or file name.

UCR files were previously created through the CP OVERRIDE command. However, support for the user class restructure (UCR) function and the OVERRIDE utility have been removed. If any UCR files exist on the system, the contents of those files will not be processed by CP. The QUERY UCR command can be used to display the queue of UCR files. The PURGE UCR command can be used to delete all or selected UCR files.

Message repository files are created through the CMS LANGGEN command. This command uses DIAGNOSE code X'CC' to load and save in a message repository system data file the messages contained in a compiled message repository. The QUERY NLS command can be used to display the queue of message repository files. The PURGE NLS command can be used to delete all or selected message repository files.

System trace files are created when CP or virtual machine trace data recording has been started by way of the TRSAVE or TRSOURCE command and the destination of the information is DASD. The QUERY TRFILES command can be used to display the queue of system trace files. The PURGE TRFILES command can be used to delete all or selected system trace files.

System Data File Attributes

When a system data file is created, it is assigned certain attributes, some of which can be specified by the user. System data file attributes are the following:

- **Owner ID.** The owner ID indicates either the queue on which the system data file resides (*NSS, *IMG, *UCR, or *NLS) or, for system trace files, the user ID of the virtual machine to which the system trace file belongs. By default, the file owner of a system trace file is the user who enters the TRSAVE or

TRSOURCE command that creates the file. This default can be overridden by using the TO operand of the TRSAVE command. All system data files other than system trace files are owned by the system.

- **Spool ID.** This is a number between 1 and 9999 that is automatically assigned to a system data file by CP when a new system data file is created. Spool IDs are assigned consecutively until 9999 (the maximum for each system data file queue) is reached. Assignment then begins again at 1. The spool ID of each system data file is unique to each queue and is used as a part of the identifier of a spool file. In order to identify a specific system data file in a command, the system data file queue and spool ID must be given.
- **System data file class.** See "System Data File Classes" on page 32.
- **System data file type.** This field indicates the queue (NSS, IMG, UCR, NLS, or TRF) on which the system data file resides.
- **System data file size.** This is the number of logical records in the system data file, which can be a value up to 999 million.
- **Creation date and time.**
- **File name and file type.** The interpretation of the file name and file type of a system data file depends on its type. For a system data file in the NSS queue, the file name must be specified by the user when it is created. The file type is assigned by CP and indicates the type of system data file.

  CP assigns a file type of NSS to all NSSs. However, when the MAP or USERS operand is used on the QUERY NSS command, CP displays CPNSS as the file type for a CP system service NSS that contains CP writable pages and displays NSS as the file type for all other NSSs.

  CP assigns a file type of DCSS to all saved segments. However, when the MAP or USERS operand is used on the QUERY NSS command, CP displays a file type that identifies the type of saved segment, as follows:

  – DCSS identifies a DCSS that contains only page addresses below 2047 MB.
  – DCSSG identifies a DCSS that includes page addresses above 2047 MB.
  – DCSS-S identifies a segment space.
  – DCSS-M identifies a member saved segment.
  – CPDCSS identifies a CP system service saved segment that contains CP writable pages, such as the monitor saved segment.

  The file name of an image library must be specified on the IMAGELIB command. CP assigns a file type of IMG to all image libraries.

  The file name of a message repository system data file is the language identifier of the message repository specified on the CMS LANGGEN command. A message repository's file type is NLS.

  The file name of a system trace file is the file name assigned by the user who enters the TRSAVE command. If the file name is not specified, the default file name for a CP system trace file is CPTRACE. The default file name for a file that contains trace data defined by way of the TRSOURCE command is the trace identifier.

  The file type of a system trace file is one of the following:

  CP
  For CP trace table recording

  VM
  For a virtual machine user trace

  VMG
  For a virtual machine group trace

  IO
  For an I/O trace

  DATA
  For a data trace.

Where they can be specified by the user, the file name and file type can each be from one to eight alphanumeric characters. Duplicate names are not permitted for system data files of the same file type and class.

- **Originating user.** This is the user ID of the virtual machine that created the system data file. For system trace files, the originator depends on the type of data it contains. For CP trace table recording and I/O traces, virtual machine group traces, and data traces, the file originator is SYSTEM. For virtual machine user traces, the file originator is the user ID specified by the FOR operand on the TRSOURCE command that defined the trace.

NSS and saved segment files have some additional attributes, as follows:

- **Minimum storage size (NSS only).** This field indicates for an NSS the minimum storage size of the virtual machine in which an NSS can be loaded. This field does not apply to a saved segment.
- **Beginning and ending pages.** These fields indicate the beginning and ending page numbers of a range of pages within an NSS or saved segment. A different type of virtual machine access can be assigned to each page range defined in an NSS or saved segment.
- **Access type.** This field holds a code that describes the virtual machine access for pages in a given page range. Access can be shared or exclusive, read-only or read/write, with or without data saved. A code that allows write access by CP and shared read-only access by virtual machines with no data saved is also provided for CP system service NSSs and saved segments. Because segment spaces are defined by the members they contain rather than by page ranges specified directly by the user, this field does not apply to segment spaces.
- **Number of users.** This field shows the number of users attached to the NSS or saved segment.
- **Registers used for parameters (NSS only).** This field shows the user-specified range of general registers in which parameters will be passed to a virtual machine when an NSS is IPLed in the virtual machine. This field does not apply to a saved segment.
- **Virtual machine group (NSS only).** This field shows whether the VMGROUP attribute was specified on the DEFSYS command when the NSS was created. This field does not apply to a saved segment.

Table 4 on page 35 summarizes some of the system data file attributes.

The QUERY NSS, QUERY IMG, QUERY UCR, QUERY NLS, and QUERY TRFILES commands can be entered to display on the virtual operator's console one line of attribute information about each system data file that currently exists in the NSS, IMG, UCR, NLS, and TRF queues. Each line contains the following:

- The file's owner ID, spool ID, class, and type
- The number of logical records in the system data file
- The date and time of the file's creation
- The file's file name and file type
- The user ID of the user that created the file.

Table 4. System Data File Attributes

Type of System Data File	System Data File Queue	Owner ID	Valid File types
Saved segment	NSS queue	*NSS	CPDCSS DCSS DCSSG DCSS-S DCSS-M
Image library	IMG queue	*IMG	IMG
NSS	NSS queue	*NSS	CPNSS NSS

Table 4. System Data File Attributes (continued)

Type of System Data File	System Data File Queue	Owner ID	Valid File types
UCR file	UCR queue	*UCR	User-specified
Message repository	NLS queue	*NLS	NLS
System trace	TRF queue	User-specified	CP VM VMG IO DATA

The MAP operand can be used with the QUERY NSS command to obtain information about the other attributes of an NSS or saved segment file. Information about the virtual machines that are actively using an NSS or saved segment can be obtained by way of the QUERY NSS USERS command.

When the QUERY NSS command is issued with the MAP operand for a saved segment specified by the NAME operand, CP displays information about the attributes of the specified saved segment. When the specified file is a segment space, CP displays information about the segment space and all of its members. When the specified file is a member saved segment, CP displays information about the member saved segment and all of the segment spaces of which it is a member.

The QUERY NSS USERS command can be entered for a particular NSS or saved segment to determine its active users. When the file is an NSS, DCSS, or member saved segment, CP identifies the file and lists the user IDs of the virtual machines that are currently using it. When the file is a segment space, CP identifies both the segment space and its member saved segments and lists the current users of each.

System Data File Commands

Several CP commands are provided to control system data files, most of which require privilege classes higher than class G. System commands can be grouped into those that are used to create, delete, query, and back up system data files. General users can load NSS files into their virtual machines (by way of the IPL command), can attach saved segments to their virtual machines (by way of DIAGNOSE code X'64' or the CMS SEGMENT command or macro), and choose to receive messages in a language installed in a message repository file (by way of the SET CPLANGUAGE command). General users can also process, query, and delete the system trace files of which they are the owners but cannot create them or back them up.

The privilege class required to manipulate a system data file depends on its type. Privilege class E is required to create, delete, and query NSS, saved segment, and message repository files. Image library files can be controlled by class A, B, C, D, or E users. Privilege class A, B, or C is required to enter commands for UCR files. Classes A and C are required to create a system trace file. Class G users can delete only system trace files they own and can query only system trace files they own or originated. Class A, C, D, and E users can query and delete all system trace files.

The commands used to create and delete system data files are discussed in “Creating and Deleting System Data Files” on page 33. The QUERY command can be entered to determine the number of system data files that exist (QUERY SDF) and obtain the attributes and status of system data files (QUERY NSS, QUERY IMG, QUERY UCR, QUERY NLS, and QUERY TRFILES).

System data files can be backed up by either a spooling operator or a system analyst (privilege class D or E, respectively) using the same spool-to-tape facility used to back up spool files, the SPXTAPE command. If current copies of all system data files are maintained on tape, the files can be quickly reloaded rather than rebuilt if they are inadvertently deleted. All or selected system data files can be dumped to tape by way of the SPXTAPE DUMP command and restored to the system by way of the SPXTAPE LOAD command. System data files can be selected by user ID (system trace files only), queue (IMG, NLS, NSS, TRF, or UCR), spool ID number or range, class, file name pattern, file type pattern, or combinations of these

attributes. A file name or file type pattern can be either a complete file name or file type or a string containing wild cards (* and %).

A class G user can use the SPXTAPE command to process the user's own system trace files.

All of the SPXTAPE operands can be used when backing up system data files. However, SPXTAPE cannot dump a class W system trace file. Note that the migration of system data files between z/VM and VM/SP and VM/SP HPO is not supported.

System Data File Recovery

Like spool files, system data files are recovered by CP when CP is restarted after a real machine termination. During system operation, information in real storage about existing system data files is checkpointed (written to auxiliary storage) when appropriate. This information is then used during a warm start, force start, cold start, or automatic software re-IPL to reconstruct information in real storage about closed system data files (system data file queues). System data files are not recovered during a clean start.

The CP system residence volume contains a user-allocated warm start save area. The warm start save area holds copies of the spool file index pages, each of which contains the auxiliary storage addresses of the spool file map blocks for up to 1022 spool files and system data files. The first spool file map block for each system data file contains a copy of the spool file block (which describes the system data file) and the auxiliary storage addresses of the pages that contain the system data file's data. Auxiliary storage addresses for system data files are cleared during a clean start.

The maximum number of spool file index pages contained in the warm start save area (and hence the maximum number of spool files system data files allowed in the system) depends on the device type of the system residence volume (which affects the number of 4 KB pages per cylinder) and the number of cylinders allocated for this purpose. Each 4 KB page allows for 1022 spool files or system data files; a maximum of 9 cylinders may be allocated to the warm start save area. Accordingly, if the system residence volume is a 3380 (with 150 4 KB pages per cylinder), a warm start area of 9 cylinders allows for over 1.3 million spool files and system data files. The maximum number of spool file index pages possible is proportionally reduced if the warm start save area has fewer than the maximum number of cylinders or pages. The warm start save area is formatted with 4 KB pages so that the CP paging I/O facility can be used for reading and writing spool file index pages.

During system operation, when a system data file is created or deleted, the auxiliary storage address of its associated spool file map block is placed or cleared to zeros in a spool file index page, a copy of which is written in the warm start save area. Specifically, the spool file index page associated with a system data file is checkpointed when a system data file is opened or deleted.

When the warm start save area is full, no more spool IDs can be assigned to either system data files or spool files. The operator is notified that the maximum system spool limit is exceeded. The operator can start additional printers or card punches to handle the classes of the queued spool output files, delete existing spool input files, if possible, or dump spool files and system data files to tape to make room available.

When the status or characteristics of a system data file change during system operation, the changed spool file block in storage is checkpointed by updating the copy of it kept in the spool file map block on auxiliary storage. Specifically, the spool file block is checkpointed when one of the following occurs:

- A system data file is closed.
- The characteristics of a closed system data file are changed (for example, when the file is pending purge, or when part of an image library is replaced by the IMAGEMOD command).

Chapter 2. Planning and Defining CMS Logical Saved Segments

This chapter:

- Discusses planning for your CMS saved segment layout.
- Describes how to create the files that define the contents of physical and logical saved segments.

Overview of Physical and Logical Saved Segments

A ***saved segment*** is an area of virtual storage that is assigned a name, loaded with data or programs, then saved in a system data file in spool space. Using saved segments is a way of using storage that is not yours.

Segment spaces, member saved segments, and discontinuous saved segments (DCSSs) reside on CP-owned volumes and must be defined to CP before being used. A segment space, which begins and ends on a megabyte boundary, contains one or more member saved segments, which begin and end on page boundaries. A DCSS also begins and ends on a megabyte boundary, but does not contain members.

Defining frequently used data or programs as saved segments provides several advantages:

- Several users can access the same saved segment, which helps you use real storage more efficiently.
- Saved segments need not be in the address range of a virtual machine (this can also help you use storage more efficiently).
- Space for saved segments can be reserved within a virtual machine's address space, which helps you make sure that the segment is always available.

For information about defining CP saved segments, see Chapter 3, “Using VMSES/E to Define, Build, and Manage Saved Segments,” on page 57. For information about the concepts of CP saved segments, or for information about defining CP saved segments if you are not using VMSES/E to manage your saved segments, see Chapter 1, “Planning and Defining CP Saved Segments,” on page 1.

Note: VMSES/E does not support saved segments that contain pages above 2047 MB.

A ***physical saved segment*** is a member saved segment or DCSS that may contain one or more ***logical saved segments*** that CMS recognizes. Defining logical saved segments provides further advantages:

- Each logical saved segment can contain different types of program objects, such as modules, text files, execs, callable services libraries, language information, and user-defined objects, or file directory information for one minidisk.

  You can use logical saved segments to package your entire application. For example, you may want to create a logical segment definition file that defines the parts of your application. You could then send it to the system administrator, who will create the logical saved segment and make it available for others to use. For more information, see “Defining the Contents of a Logical Saved Segment” on page 44.
- You can use a member saved segment or DCSS more efficiently by defining it as a physical saved segment containing many different logical saved segments.
- Users can access specific logical saved segments rather than all the contents of a physical saved segment.

Note:

1. CMS recognizes only logical saved segments. Loading a physical saved segment will **not** cause the logical saved segments it contains to be loaded and recognized by CMS. You must explicitly load each logical saved segment you need.

2. You cannot define logical saved segments in a DCSS that contains pages above 2047 MB. CMS does not support saved segments above 2047 MB.

Using Logical Saved Segments

The following list summarizes what you need to do to access code or data from within a logical saved segment.

1. Create the code or data that you want to define as a saved segment.
2. Define the space in CP for the physical saved segment:
   - If you are using VMSES/E to manage your saved segments, see "Changing, Adding, and Deleting Saved Segment Definitions" on page 62.
   - If you are not using VMSES/E, use the CP DEFSEG command to define the saved segment. The DEFSEG command creates a skeleton system data file for the saved segment.

     Note: To use the DEFSEG command to define a saved segment, you need CP class E command privileges.

   For information about the concepts of defining CP saved segments, see Chapter 1, "Planning and Defining CP Saved Segments," on page 1.
3. Build the physical and logical saved segments. See "Creating Physical and Logical Saved Segments" on page 42.
4. Use the SEGMENT RESERVE command to reserve space within your virtual machine's address space for the saved segment (optional).
5. Use the SEGMENT LOAD command or SEGMENT LOAD macro to load the logical saved segment into storage.

   For information on how to use the SEGMENT command, see the *z/VM: CMS Application Development Guide*. For information on how to use the SEGMENT macro, see the *z/VM: CMS Application Development Guide for Assembler*.
6. Use predefined interfaces to access the code or data contained in the saved segment.

Saved Segment Design Considerations

There are a number of questions you need to consider when you create the data or code that you want to store in a saved segment:

- How will users access the data or code in the saved segment? Do you want to create a command interface that provides access to the saved segment?
- Will the saved segment reside within the virtual machine's address space? If so, you should consider reserving space within the virtual machine's address space for the saved segment.
- Will the saved segment work in an XA or XC virtual machine, or any combination? Segments that perform I/O must make sure to provide code for the specific virtual machine mode (or use architecture-independent interfaces).

  Note: In order to execute code in a saved segment which is designed to work in a 370 virtual machine, the CP 370 Accommodation Facility must be enabled.
- Will the saved segment be loaded above the 16 MB line? Saved segments that reside above the 16 MB line must be 31-bit capable, so EXEC 2 execs, CMS message information (message repository, parser table, and translation tables), and minidisk file directory information cannot be included in a saved segment that resides above 16 MB. For example, Table 5 on page 41 lists the saved segments that are supplied with the base z/VM product (unless noted otherwise) and whether or not they can be loaded above the 16 MB line.

Table 5. Location for Loading Saved Segments

Segment Name	Description	Can be Loaded Above 16 MB
CMSBAM	Contains the DOS Basic Access Method for reading disk directories and accessing data on disk or tape.	NO
CMSDOS	Contains DOS Simulation code which is used by DOS application programs to access DOS or FBA formatted directories. This segment is also necessary to install the VSAM product in its segment. The DOS segment provides a point of data interchange for both true DOS and OS simulated applications which use VSAM datasets.	NO
CMSFILES	Contains DMSDAC and DMSSAC. It is ONLY used by SFS server machines. The segment can overlay any segment not being used by the server.	YES
CMSPIPES	Contains PIPES and RXSOCKETS. The SYSPROF EXEC loads both pipes and RXSOCKETS. If the segment is not loaded, CMS will NUCXLOAD the PIPES module.	YES
CMSVMLIB	CMSVMLIB contains VMLIB and REXX RTL segments. VMLIB contains the CSL routines. If the segment is not loaded, the CSL routines will be loaded into virtual storage by CMS. This may cause constraints on your virtual storage. The REXX RTL segment is used for executing the compiled REXX execs that CMS sends on the system disk.	YES
GUICSLIB	Contains the CMS GUI (graphical user interface) CSL routines.	YES
HELPSEG[1]	Contains a directory of all the help files (a saved file directory).	NO
INSTSEG	Contains commonly used EXECs and XEDITs for example, PEEK, RDRLIST, and FILELIST.	YES
MONDCSS	Contains the segment for monitor data.	YES
NLSUCENG	Contains the upper case English message repository.	YES
PERFOUT	Contains the Performance Toolkit Data Collector segment.	YES
SCEE	Contains the Language Environment® segment.	NO
SCEEX	Contains the Language Environment segment.	YES

Table 5. Location for Loading Saved Segments (continued)

Segment Name	Description	Can be Loaded Above 16 MB
SVM (DMSSVM)	Consists of the DMSSVM5 MODULE and the DMSSVM5C MODULE. Both of these modules are serviced through replacement parts. The SVM segment is intended for internal use and is not documented for customer use. The official IBM recommendation is to use the segment for performance reasons. Otherwise, the code will be NUCXLOADED in the virtual machine when needed. SVM is used by CMS and various program products.	YES

[1] This segment is not supplied with the base z/VM product.

Creating Physical and Logical Saved Segments

If the member saved segment or DCSS you are going to use as a physical saved segment is defined in CP, you can create the logical saved segments in CMS. The process for creating logical saved segments depends on whether you are using VMSES/E to manage your saved segments:

- If you are using VMSES/E:
  1. Create the code or data that you want to reside in each logical saved segment. See "Types of Program Objects Allowed in a Logical Saved Segment" on page 42.
  2. Define the contents of each logical saved segment in a logical segment definition file. See "Defining the Contents of a Logical Saved Segment" on page 44.
  3. Use the VMFSGMAP EXEC to modify the definition for the physical saved segment and add entries to identify the logical saved segments to be included.
  4. Use the PUT2PROD EXEC to build the saved segments.

     If required, PUT2PROD (as indicated by a message from VMFBLD), copies the SYSTEM SEGID file from the build disk to the CMS system disk and resaves the CMS named saved system.

  For information about using VMFSGMAP and VMFBLD, see Chapter 3, "Using VMSES/E to Define, Build, and Manage Saved Segments," on page 57.
- If you are not using VMSES/E:
  1. Create the code or data that you want to reside in each logical saved segment. See "Types of Program Objects Allowed in a Logical Saved Segment" on page 42.
  2. Define the contents of the physical saved segment (that is, what logical saved segments it contains) in a physical segment definition file. See "Defining the Contents of a Physical Saved Segment" on page 43.
  3. Define the contents of each logical saved segment in a logical segment definition file. See "Defining the Contents of a Logical Saved Segment" on page 44.
  4. Use the SEGGEN command to build the saved segments. See "Using the SEGGEN Command to Build the Saved Segments" on page 52.
  5. If required, copy the system segment identification file to the CMS system disk and resave the CMS named saved system. See "System Segment Identification File" on page 52.

Types of Program Objects Allowed in a Logical Saved Segment

As previously mentioned, logical saved segments can contain different types of program objects. You can include the following:

- Modules—CMS MODULE files that are treated as nucleus extensions or subcommand processors
- Text files—CMS TEXT files or TXTLIB members that are treated as nucleus extensions or subcommand processors
- Execs—EXEC2, REXX, or alternate format exec files (may have a file type of EXEC or XEDIT)
- Callable services libraries—A set of routines that can be called from a program, such as those in VMLIB or routines that you create
- Language information—System national language information
- User-defined objects
- Minidisk directory—File directory information for minidisks. (If a logical saved segment contains a minidisk directory, it cannot contain any other objects.)

Defining the Contents of a Physical Saved Segment

Before loading and saving a physical saved segment, you need to define the contents of the physical segment in a **physical segment definition file**. You must give this file the same name as the DCSS or member saved segment you have already defined; the recommended file type is PSEG.

Note: Do not create this file if you are using VMSES/E to manage your saved segments. When you use the VMFBLD EXEC to build the saved segments, VMFBLD creates the PSEG file.

The physical segment definition file has the following attributes:

- It is a fixed length file (it can be a variable length file).
- It contains a **logical segment record** for each logical saved segment to be included within the physical segment.
- All records (except comment records) can be continued by placing any nonblank character in column 72; columns 73 and above are ignored.
- Blank lines and comment lines are included by starting the line with an asterisk (*) in column one.
- The SEGGEN command uses a default file type of PSEG for the file.

Logical Segment Record

Each logical segment record in the file specifies a file name that corresponds to the file name of the logical segment definition file; you cannot specify duplicate file names within the file. The default file type for a logical segment record is LSEG.

Each logical segment record has the following format:

lfn lft lfm
is the file name, file type, and file mode of a logical segment definition file. If the file type is not specified, LSEG is assumed. If the file mode is not specified, then all accessed directories and disks are searched.

PI

PROFILE ***profile***
indicates that an exec is to be executed **before** the files are loaded into the logical saved segment. The *profile* is the file name of the exec. If the exec returns with a nonzero return code that equals 28 or less, the segment is not saved and processing continues for the remaining logical segments. If the return code is greater than 28, SEGGEN processing is terminated.

For example, you could use the PROFILE option to call an exec that accesses disks or directories, and then compiles programs and builds a module.

EPIFILE ***epifile***
indicates that an exec is to be executed **after** the files have been loaded into a logical saved segment. The *epifile* is the file name of the exec. If the exec returns with a nonzero return code, the segment is not saved. If the return code is greater than 28, SEGGEN processing is terminated.

For example, you could use the EPIFILE option to call an exec that performs any cleanup work, such as erasing files and releasing any unneeded disks or directories.

PI end

comments
is any sequence of characters. They are treated as comments and are not processed.

Defining the Contents of a Logical Saved Segment

You must define the contents of the logical saved segment in a **logical segment definition file**. The logical segment definition file contains records describing each object to be contained in the logical saved segment. You may include eight different types of objects in this file:

- MODULE
- TEXT
- EXEC
- LIBRARY
- LANGUAGE
- DISK
- USER
- SKIP

Each type of object has its own record format.

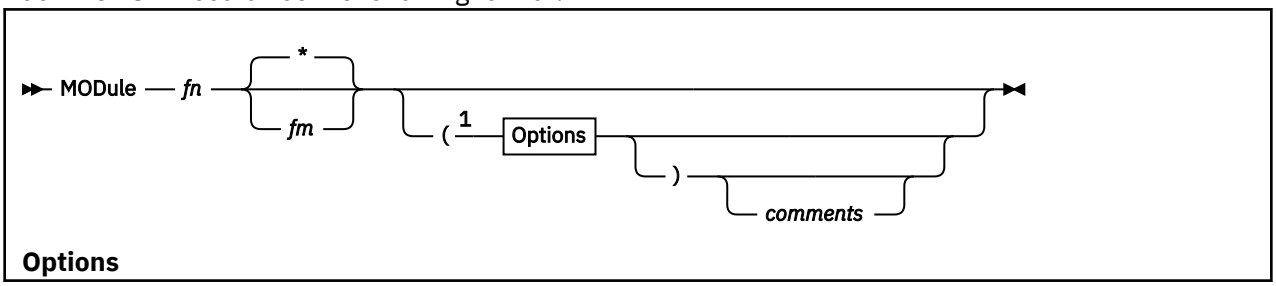
The logical segment definition file has the following attributes:

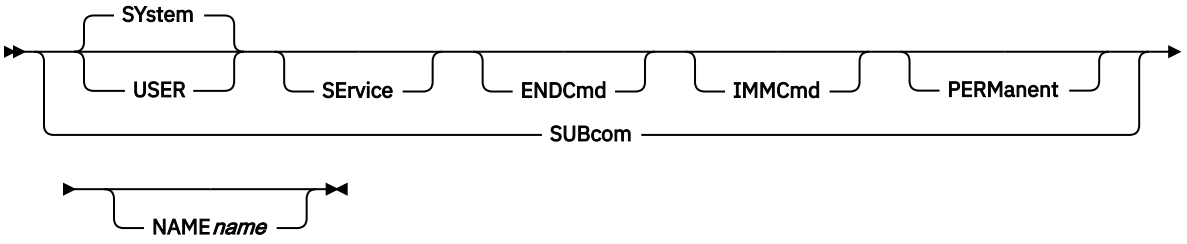
- It is a fixed length file (it can be a variable length file).
- All records (except comment records) can be continued by placing any nonblank character in column 72; columns 73 and above are ignored.
- Blank lines and comment lines are included by starting the line with an asterisk (*) in column one.
- The file name and file type must match a logical segment record in the physical segment definition file; the default file type is LSEG.

MODULE Record

The MODULE record defines a CMS MODULE file that is to be included in the logical saved segment as a nucleus extension or subcommand processor. Nucleus extension command names and subcommand environment names must be unique within a logical saved segment. Programs that execute in shared physical saved segments must be reentrant.

Each MODULE record has the following format:





Notes:

[1] You can enter Options in any order between the parentheses.

fn
specifies the file name (file type of MODULE) of the CMS MODULE file to be included in the logical saved segment.

fm
is the accessed directory or disk of the MODULE file. If the file mode is not specified, all accessed directories and disks are searched.

SYstem
indicates that the routine is to be entered disabled for interrupts and in key zero. It does **not** indicate that the routine will survive ABEND processing; this is determined by the SYSTEM or USER option on the SEGMENT LOAD command. SYSTEM is the default.

USER
indicates that the routine is to be entered enabled for interrupts and in user key.

SEservice
indicates that nucleus extension service calls are accepted.

ENDCmd
indicates that the nucleus extension receives control at normal end-of-command processing.

IMMCmd
indicates that this nucleus extension can be invoked as an immediate command.

PERManent
indicates that this nucleus extension must be named explicitly on a NUCXDROP command. It is not dropped by a `NUCXDROP *` command.

SUBcom
specifies that this MODULE is to be a subcommand processor. It will be entered in key zero with interrupts disabled. If SUBCOM is specified, all other options are ignored. If SUBCOM is not specified, the MODULE is treated as a nucleus extension. Note that all subcommand processors are loaded with the SYSTEM bit on in their SCBLOCKs, and thus survive processing.

NAME *name*
specifies the nucleus extension command name or subcommand environment name for this program. If NAME is not specified, the file name of the MODULE is used for the name.

comments
is any sequence of characters. They are treated as comments and are not processed.

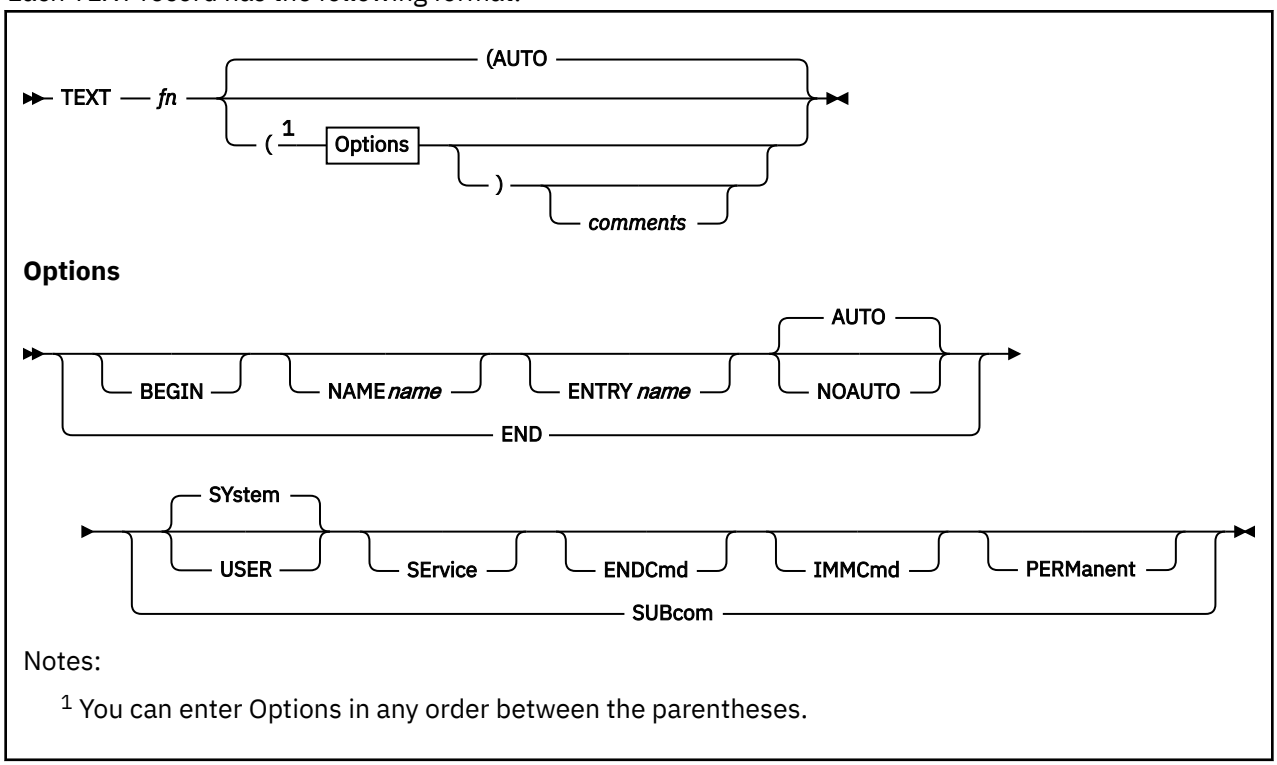
TEXT Record

The TEXT record defines a TEXT file that is to be included in the segment as a nucleus extension or subcommand processor. Consider the following when including TEXT files:

- Each TEXT file will be processed independently for external symbol resolution. Only TEXT files that are part of a BEGIN-END block will be treated as a unit during external symbol resolution.
- Nucleus extension command names and subcommand environment names must be unique within a logical saved segment.

- Programs that execute in shared saved segments must be reentrant.

Each TEXT record has the following format:



Notes:

[1] You can enter Options in any order between the parentheses.

fn
specifies the file name of the CMS TEXT file to be included in the logical saved segment. All accessed file modes are searched. If a GLOBAL TXTLIB command has been issued, *fn* may indicate the name of a TXTLIB member.

BEGIN
indicates that this TEXT file is the first in a series of TEXT files to be considered as one nucleus extension or subcommand processor. The next record in the logical segment definition file must be a TEXT record. The sequence continues until a TEXT record with the END option is encountered. This BEGIN-END block of TEXT files is taken as a single entity and treated as a single nucleus extension or subcommand processor. External symbol resolution will be performed and a single load module will be created.

END
indicates the last of a series of TEXT records which are to be treated as a single nucleus extension or subcommand processor. No other options can be specified if END is specified.

NAME *name*
specifies the nucleus extension command name or subcommand environment name for this program. If NAME is not specified, the file name of the TEXT file is used for the name. The NAME option on the same record as the BEGIN option specifies the name of the single BEGIN/END nucleus extension or subcommand processor. Within a BEGIN-END block of TEXT records, NAME is only valid with the SUBCOM option; otherwise, it is ignored.

ENTRY *name*
specifies the entry point (EXTERNAL definition) to be treated as the nucleus extension or subcommand processor entry point. If ENTRY is not specified, the first CSECT in the TEXT file is used as the entry point. The ENTRY option on the same record as the BEGIN option specifies the name of the entry point to be used for the single BEGIN/END nucleus extension or subcommand processor. Within a BEGIN-END block of TEXT records, ENTRY is only valid with the SUBCOM option; otherwise, it is ignored.

AUTO
indicates that any TEXT file referred to in an EXTERNAL symbol in this TEXT file will be automatically loaded. This is the default.

NOAUTO
indicates that no other TEXT files will be loaded to resolve EXTERNAL references.

SYstem
indicates that the routine is to be entered disabled for interrupts and in key zero. It does **not** indicate that the routine will survive ABEND processing; this is determined by the SYSTEM or USER option on the SEGMENT LOAD command. SYSTEM is the default.

USER
indicates that the routine is to be entered enabled for interrupts and in user key.

SErvice
indicates that nucleus extension service calls are accepted.

ENDCmd
indicates that the nucleus extension receives control at normal end-of-command processing.

IMMCmd
indicates that this nucleus extension can be invoked as an immediate command.

PERManent
indicates that this nucleus extension must be named explicitly on a NUCXDROP command. It is not dropped by a `NUCXDROP * ` command.

SUBcom
specifies that this program is to be a subcommand processor. It will be entered in key zero with interrupts disabled. If SUBCOM is not specified, the program is treated as a nucleus extension. Note that all subcommand processors are loaded with the SYSTEM bit on in their SCBLOCKs, and thus survive end-of-command processing.

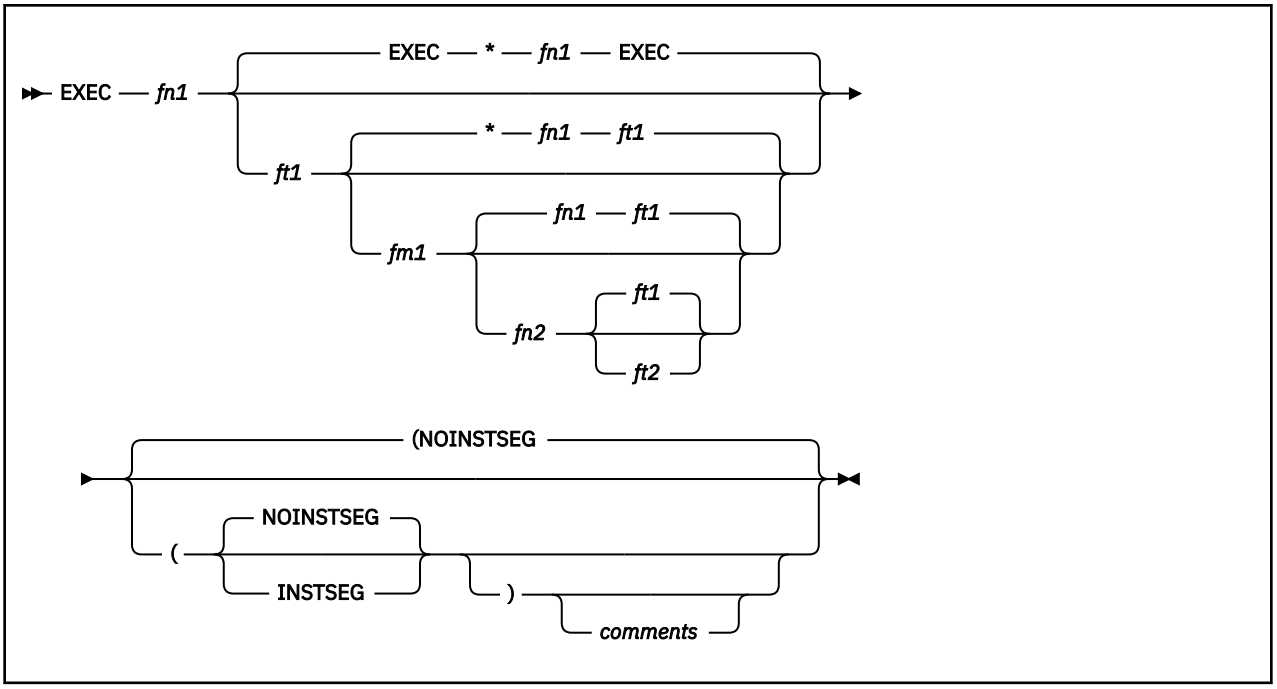
SUBCOM is also valid on a TEXT record within a BEGIN-END block. Each record that includes the SUBCOM option defines a subcommand processor entry point within the nucleus extension or subcommand processor defined by the BEGIN-END block. The default subcommand environment name is the file name and the default entry point is the first CSECT in the TEXT file.

comments
is any sequence of characters. They are treated as comments and are not processed.

EXEC Record

The EXEC record defines an EXEC 2, REXX, or alternate format exec that is to be included in the logical saved segment. Execs within a logical saved segment must have unique names. Also, remember that EXEC 2 execs cannot be included in a logical saved segment that resides above 16 MB.

Each EXEC record has the following format:



fn1 ft1 fm1
specifies the file name, file type, and file mode of the EXEC 2, REXX, or alternate format exec to be included in the logical saved segment. The default file type is EXEC. If the file mode is not specified, all accessed file modes are searched.

fn2 ft2
specifies the file name and file type by which the exec will be known when it resides within the logical saved segment. If this name is not specified, the present file name and file type of the exec file will continue to be used.

INSTSEG
specifies that the exec is to be considered part of the CMS installation segment, and as such, is affected by the SET INSTSEG command and the INSTSEG parameter of the IPL command.

NOINSTSEG
specifies that the exec is not to be considered part of the CMS installation segment, and as such, is not affected by the SET INSTSEG command or the INSTSEG parameter of the IPL command. This is the default.

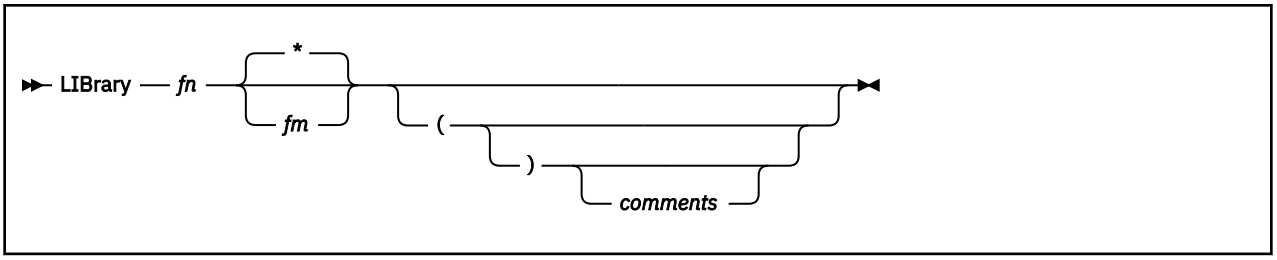
comments
is any sequence of characters. They are treated as comments and are not processed.

LIBRARY Record

The LIBRARY record identifies a callable services library that is to be included in the logical saved segment. Libraries within a logical saved segment must have unique names. You must first create the callable services library using the CSLGEN command with the SEG operand. The resulting library file has a file type of CSLSEG.

For more information on VMLIB Callable Services Library (CSL) and creating your own CSL, see *z/VM: CMS Application Development Guide* and the *z/VM: CMS Application Development Guide for Assembler*.

Each LIBRARY record has the following format:



fn
specifies the file name of the callable services library (CSL) file to be included in the logical saved segment. The file must have the file type of CSLSEG.

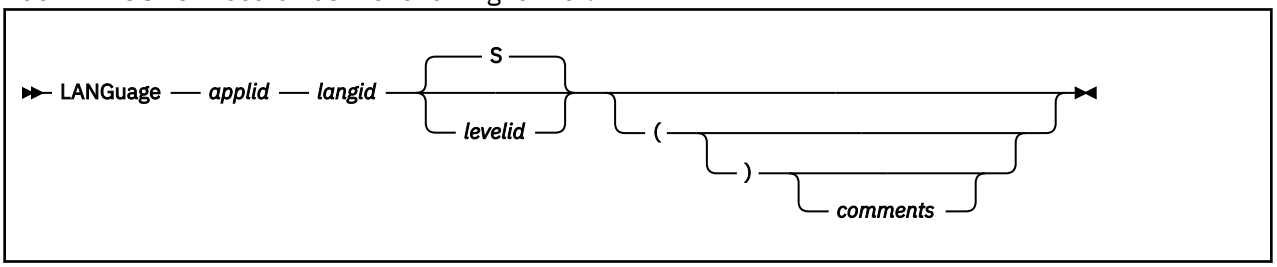
fm
is the accessed directory or disk of the CSLSEG file. If the file mode is not specified, all accessed file modes are searched.

comments
is any sequence of characters. They are treated as comments and are not processed.

LANGUAGE Record

The LANGUAGE record identifies system national language information that is to be included in the logical saved segment. Only one DMS language segment may be included in a logical saved segment. Note that the name of a logical saved segment that contains a DMS language segment must have the name NL*xy*. The *x* is a single character representing the level identifier (*levelid*) of the language segment. The level identifier is defined when the CMS nucleus is created. The default *levelid* is *S*. The *y* is the language identifier (*langid*) for the language segment. Language segments within a logical saved segment must have unique names.

Each LANGUAGE record has the following format:



applid
is the application identifier for a text file created by the LANGMERG command.

langid
is the language identifier for a text file created by the LANGMERG command.

levelid
is one character (A-Z, 0-9) that identifies the version of the logical saved segment being built. If the *levelid* is not specified, it defaults to S.

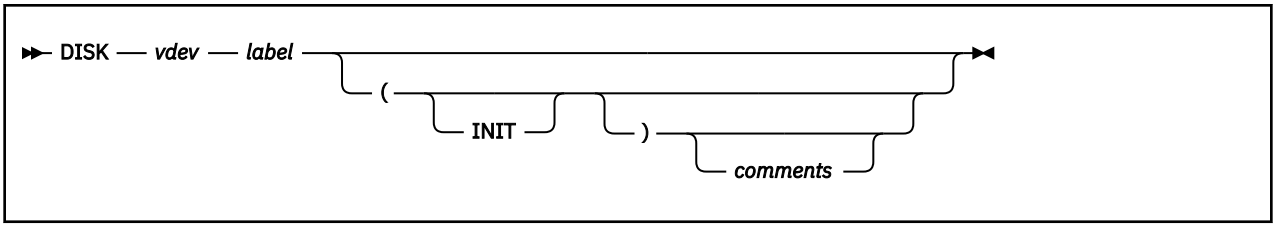
comments
is any sequence of characters. They are treated as comments and are not processed.

The single text file created by LANGMERG has the file ID of *applid*NLS TXT*langid*.

DISK Record

The DISK record identifies a minidisk for which file directory information is to be placed in a logical saved segment. When this logical saved segment is loaded, the file directory information is then available to CMS users who access the disk as read/only. When a DISK record is included in a logical segment definition file, it must be the only record in the file (other than comment or SKIP records).

The DISK record has the following format:



vdev
is the virtual device number of the minidisk for which file directory information is to be included in the logical saved segment. The disk must be linked read/write at the specified device address at the time the SEGGEN command is entered. The *vdev* can consist of up to 4 digits.

label
is the label of the CMS minidisk. The *label* may consist of up to six characters and cannot contain any embedded blanks.

INIT
indicates to SEGGEN to issue the SAVEFD command with the INIT option before saving the actual disk directory. The INIT option of the SAVEFD command initializes the disk for a subsequent SAVEFD command.

Usage Notes:

1. SKIP records may not come before a DISK record in a logical segment definition file.
2. No other records, other than SKIP records, can be in the same logical segment as a DISK record.
3. The file mode letter Z is used when saving the minidisk directory; therefore, any directory or disk accessed as Z is released.

For more information on the SAVEFD command and sharing file directory information, see the *z/VM: CMS Commands and Utilities Reference*.

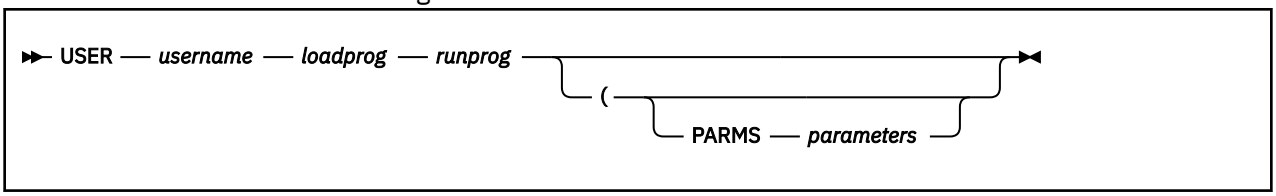
USER Record

The USER record identifies user objects to be included in a logical saved segment. As part of the objects you need to provide:

- A program that is called during SEGGEN processing to load the user object into the segment
- A program that is called when the logical saved segment is loaded or purged.

User program objects are the last to be loaded when the logical saved segment is loaded and the first to be purged when the logical saved segment is purged.

Each USER record has the following format:



username
specifies a 1- to 8-character name of the user object.

PI

loadprog
specifies the MODULE name of the program to be called during SEGGEN processing to load the user objects into the segment.

runprog
specifies the MODULE name of the program to be called when the logical saved segment is loaded or purged.

PARMS *parameters*
is any sequence of characters following the keyword PARMS to be passed to both the load program and runtime program as part of the SGMTEXIT control block. The parameter list is passed as a string with a maximum of 64 KB. It includes any leading or trailing blanks and a right parenthesis if one is included. If you do not specify PARMS, you can include comments after a right parenthesis.

Usage Notes:

1. The address of a control block (SGMTEXIT) is passed to both the load program and the runtime program in register 1. The SGMTEXIT control block contains the logical segment name, the user object starting and ending addresses, a code indicating the action taking place (SEGGEN, shared LOAD, nonshared LOAD, or PURGE), and the address and length of the parameters specified on the USER record. See the description of the SGMTEXIT macro in *z/VM: CMS Macros and Functions Reference* for the format of the SGMTEXIT control block.
2. When the segment is built, the SEGGEN command calls *loadprog* to load the user object into the segment. SEGGEN calls *loadprog* by issuing a CMSCALL macro with register 1 pointing to the SGMTEXIT control block. (Remember, CMSCALL goes through normal CMS command resolution.) The SGMTEXIT control block passed to this program by register 1 contains the starting address of the location in storage where the object is to be loaded. The largest possible ending address will be in the SGMEND field of the SGMTEXIT control block. The user object must not exceed this address. The load program must change the SGMEND field to reflect the ending location of the user object. Also, the SGMFUNC field will contain -1 to indicate that this is a SEGGEN call.

   When the program completes, a return code of 0 indicates a successful completion. A return code other than 0 indicates a user load error, and skips to the next user object.
3. Prior to calling the *loadprog*, SEGGEN does a SEGMENT RESERVE which results in the storage keys for the segment being set to X'F'. If the *loadprog* tries to put anything into the segment using the addresses passed in the SGMTEXIT it will fail with a protection exception. The *loadprog* can get around this problem in several ways, however it is recommended that you GENMOD the *loadprog* with the 'SYSTEM' option so that it will run in key 0.
4. When the logical saved segment is loaded (made active), any user objects that it contains are activated last. The user objects are activated in LIFO order in the order in which they are defined in the logical segment definition file. The program specified on the USER record as *runprog* will be called to activate the user object.

   You or the system administrator should make sure that *runprog* is available to users when they load and/or purge the segment by putting the routine on the S-disk, or by putting it in the same segment as the USER information (using a MODULE or TEXT record in the logical segment definition file). If the routine is not available when either a SEGMENT LOAD or SEGMENT PURGE command or macro is issued, an error results, and in the case of SEGMENT LOAD, the segment is not loaded.
5. When *runprog* is called, register 1 contains the address of the SGMTEXIT control block, and the SGMFUNC field of the SGMTEXIT control block indicates that this is the load function. A 0 indicates the segment is being loaded in shared mode, and a 4 indicates nonshared mode. This program is also called when the logical saved segment is being purged. An 8 in the SGMFUNC field of the SGMTEXIT control block indicates that the segment is being purged.

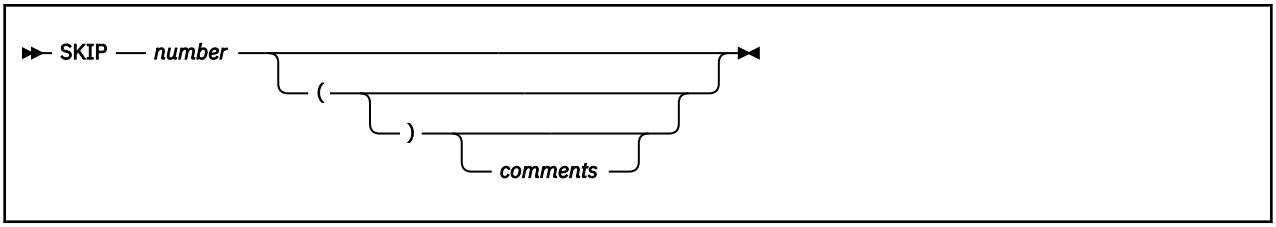
   When the program completes, a return code of zero indicates a successful completion. If a return code other than 0 is issued and you are in LOAD, all the objects are removed and the logical segment is purged. If a return code other than 0 is issued and you are in PURGE, the remaining objects are removed.

PI end

SKIP Record

The SKIP record specifies an amount of space in the segment to be left unused.

The SKIP record has the following format:



number
specifies the number of pages to be skipped. A *number* value of zero means skip to the start of the next page. A *number* value of one means skip to the start of the next page and then skip the next page.

For example, if you have a USER object that needs to start on a page boundary, specifying `skip 0` tells CMS to skip to the start of the next page.

Using the SEGGEN Command to Build the Saved Segments

Use the SEGGEN command to build and save a physical saved segment that is composed of one or more logical saved segments.

Note: If you are using VMSES/E to manage your saved segments, use the PUT2PROD EXEC (which calls VMFBLD, which calls SEGGEN). See "Building or Deleting (Purging) Saved Segments" on page 76.

SEGGEN uses the segment definitions that are in the physical segment definition file and one or more logical segment definition files, one for each logical saved segment to be included in the physical saved segment.

The physical saved segment built by SEGGEN has a directory at the end (highest addresses) of the segment. In addition, each logical saved segment in the physical saved segment also has a directory at the end of the particular logical saved segment.

The SEGGEN command updates or creates a system segment identification file that associates each logical saved segment with its physical saved segment. SEGGEN also generates two types of load map files, one for the physical saved segment and one for each logical saved segment within the physical saved segment.

Note that if errors occur during SEGGEN processing, temporary files may be left on your read/write disk.

See the *z/VM: CMS Commands and Utilities Reference* for information on the format of the SEGGEN command and the load map files.

System Segment Identification File

The process of building logical saved segments using the SEGGEN command updates or creates a **system segment identification file**. Each entry in this file associates a logical saved segment with its physical saved segment. The default file name and file type are SYSTEM SEGID.

Attention: The system segment identification file should be changed only by the SEGGEN command. Modification by any other means may cause unpredictable results.

The system segment identification file must be named SYSTEM SEGID and must reside on the CMS system disk so it is available to CMS at initialization time. This allows CMS to recognize the logical saved segment name specified on the SEGMENT macro or SEGMENT command. If two or more entries for the same logical saved segment name are found in the SYSTEM SEGID file, the one closest to the end of the file is the default logical saved segment used at initialization time. If a logical saved segment is associated with more than one physical saved segment, you can change the default association by using the SEGMENT ASSIGN command.

After SEGGEN completes, you may have to copy the system segment identification file to the CMS system disk. You *do not* need to copy the file if you have modified the contents of an existing logical saved segment (added, deleted, or changed data). You *must* copy the file if you have:

- Created a new logical or physical saved segment
- Deleted an existing logical or physical saved segment

- Changed the relationship between the logical and physical saved segments (for example, moved or copied a logical saved segment from one physical saved segment to another).

Note: If you intend to use logical saved segments on a pre-Release 2.2 CMS, you must always copy the SYSTEM SEGID file to that CMS system disk.

There are special instructions for copying the system segment identification file to the system disk. See Step 5 in the following example.

Building Physical and Logical Saved Segments—An Example

This section shows how to build a physical saved segment containing several logical saved segments *if you are not using VMSES/E to manage your saved segments*. If you are using VMSES/E, see Chapter 3, "Using VMSES/E to Define, Build, and Manage Saved Segments," on page 57.

The following example shows how to build a physical saved segment named PSEG3 that contains these three logical saved segments:

- USERDISK—Contains a minidisk directory
- NLSABC—Contains system language information and objects for an application named ABC
- USERSEG—A user segment that loads user programs and data.

Step 1. Create the Code or Data

The USERDISK logical saved segment will contain the file directory information for a CMS minidisk with a virtual device number of 100 and a disk label of MLU100.

The NLSABC logical saved segment will contain the following objects for an application named ABC:

- ABCNLS TXTABC—the text file from the LANGMERG command
- ABCLIB CSLSEG—the callable services library file from the CSLGEN command
- ABC MODULE
- RUNABC EXEC

The USERSEG logical saved segment contains user data and a runtime module for user data:

- USERSEG—user defined data
- USERRUN MODULE—program that is called when the logical saved segment is loaded or purged

Step 2. Define the Physical Saved Segment Contents

The physical segment definition file for PSEG3 is named PSEG3 PSEG and has the following records:

```
LSEGMENT USERDISK LSEG
LSEGMENT NLSABC LSEG
LSEGMENT USERSEG LSEG
```

Step 3. Define the Logical Saved Segment Contents

Define each logical saved segment in a logical segment definition file:

USERDISK LSEG A:

```
DISK 100 MLU100 (INIT)
```

NLSABC LSEG A:

```
LANGUAGE ABC ABC
LIBRARY ABCLIB
MODULE ABC
SKIP 2
EXEC RUNABC
```

The SKIP record is put in to leave room for objects that will be added later.

USERSEG LSEG A:

```
MODULE USERRUN
USER USERSEG USERLOAD USERRUN
```

Step 4. Enter the SEGGEN Command

To build the PSEG3 physical saved segment you would enter the SEGGEN command:

```
seggen pseg3 pseg (map gen
```

Note that PSEG, MAP, and GEN are the defaults and therefore do not need to be specified.

Upon successful completion, SEGGEN updates or creates the SYSTEM SEGID file. For PSEG3, the following entries would be added to the SYSTEM SEGID file:

```
PSEGMENT  PSEG3     00600000  00100000  12/04/91  10:33:40
LSEGMENT  USERDISK  00600000  000001E0
LSEGMENT  NLSABC    00600220  0000367C
LSEGMENT  USERSEG   006038E0  000000F8
```

In addition, SEGGEN creates load maps for PSEG3 and each logical saved segment. They are written on the first accessed read/write disk. Building PSEG3 creates the following load maps:

PSEG3 PSEGMAP A5:

```
00600000 USERDISK
00600220 NLSABC
006038E0 USERSEG
SPACE UNUSED: 000FC5D0 BYTES
```

USERDISK LSEGMAP A5:

```
DISK     00600000                    100  MLU100
```

NLSABC LSEGMAP A5:

```
ABCAMENG 00600220                  A 0191 MLU191
ABCLIB   006004F0                  A 0191 MLU191
ABC      00600B20                  A 0191 MLU191
         00600B20 ABC      NUCEXT
RUNABC   00603000                  A 0191 MLU191
                  RUNABC   EXEC
```

USERSEG LSEGMAP A5:

```
USERRUN  00603970                  A 0191 MLU191
         00603970 USERRUN NUCEXT
USERSEG  00603A20
```

You can use the NOGEN option if you want to test the build process without actually creating the saved segment. This way you can verify that all the files are correct and all the objects are available.

Step 5. Copy the SYSTEM SEGID File to the System Disk and Resave CMS

Because you have created new logical saved segments, you must copy the SYSTEM SEGID file to the CMS system disk. You must also copy the file to the test CMS system disk. This prevents the file from being backleveled during the application of service. The z/VM service procedure always updates files on the test CMS system disk and then merges them to the production CMS system disk. Adding or updating a file to the system disk invalidates the current shared S-STAT (the S-disk file directory). Therefore, after you copy the file to the system disk, you must resave the CMS saved system to update the S-STAT.

Note: Only authorized user IDs with the necessary privilege class can do the following functions. Contact your system support personnel.

1. Access the production CMS system disk (usually MAINT 190) as a read/write file mode, such as T. Enter:

```
access 190 t
```

2. Access the test CMS system disk (usually MAINT*vrm* 490) as a read/write file mode, such as V. Enter:

```
access 490 v
```

3. Copy SYSTEM SEGID file to the two CMS system disks. The file that you place on the CMS system disks must be named SYSTEM SEGID and the file mode number must be 2. Enter:

```
copyfile system segid fm = = t2 (replace olddate
copyfile system segid fm = = v2 (replace olddate
```

4. Erase or rename the SYSTEM SEGID file on the build disk, so the latest copy of SYSTEM SEGID is on the system disk.
5. Use the SAMPNSS EXEC to create a system data file for the CMS saved system. Enter:

```
sampnss cms
```

6. Resave CMS. Enter:

```
ipl 190 clear parm savesys cms
```

Chapter 3. Using VMSES/E to Define, Build, and Manage Saved Segments

The VMSES/E component of z/VM provides functions to help you define, build, and manage your saved segments. The following topics are discussed in this chapter:

Note: VMSES/E does not support saved segments that contain pages above 2047 MB.

Overview of VMSES/E Saved Segment Support

Managing saved segments requires a system view rather than a product view. A typical z/VM system consists of the z/VM product plus a number of application products. z/VM uses saved segments for various functions, and many of the applications that run on z/VM also use saved segments. Some of the saved segments that reside on a z/VM system might have requirements for the same storage space. In addition, some saved segments might have dependencies on other saved segments.

VMSES/E saved segment support allows you to:

- View a segment map that shows all the segment spaces, member saved segments, discontinuous saved segments (DCSSs), and saved systems known to VMSES/E (defined in a saved segment data file) or defined on your system.

  Notes:

  1. Saved systems are displayed in the segment map only to help you plan the layout of your saved segments. You cannot use VMSES/E segment map functions to define or build saved systems.
  2. Logical saved segments are not displayed in the map. However, the displayable definition record for a member saved segment or DCSS known to VMSES/E indicates whether it is a physical saved segment that contains logical saved segments, and the physical saved segment is displayed in the map.
- Customize saved segment definitions to meet the requirements of your installation. You can add, change, and delete definitions. Then you can view the results of your changes in the segment map before you actually build the saved segments.
- Use the VMSES/E build process to automate the building of saved segments. VMSES/E keeps track of the saved segments that need to be built.

Product-Supplied Saved Segment Information

Saved segment contents and build parameters are defined by individual products. If a product is in VMSES/E format (that is, if the product supplies a product parameter file to control installing, building, and servicing the product), the contents of a saved segment are defined in a product build list. This build list, called a product saved segment build list, is defined in the product parameter file and is used by the VMFBLD EXEC to build the saved segment.

To use the VMSES/E saved segment support, the product also supplies a default definition of the saved segment. This default definition is contained in a saved segment definitions section of the product parts (*prodid* PRODPART) file. The definition includes information such as:

- The storage location and range
- Whether the saved segment:
  - Is a member of a segment space
  - Contains logical saved segments
  - Can be loaded above the 16 MB line
- Whether there are any requisite saved segments
- The build parameters (a pointer to the product parameter file and the product saved segment build list).

A product that is not in VMSES/E format can use the VMSES/E saved segment mapping and build support. The product can supply information that you use to define its saved segments to VMSES/E. (See "Adding Saved Segment Definitions for a Product Not in VMSES/E Format" on page 75.) In a definition for a non-VMSES/E product saved segment, the build parameters, instead of pointing to a product parameter file and build list, are the actual parameters for building the saved segment.

Saved Segment Product Parameter File

To automate the process of building the saved segments in a z/VM system, which consists of the z/VM product plus other products, including those saved segments defined by products not in VMSES/E format, z/VM provides a special product parameter file for building and mapping saved segments.

The name of the supplied saved segment product parameter file is SEGBLD. The information in the ESASEGS "component" of SEGBLD PPF specifies the service structure, including the build support, for all the saved segments on the z/VM system.

System Saved Segment Build List

A z/VM system configuration generally includes saved segments from many products, some of which might not be in VMSES/E format. Therefore, VMSES/E supports another kind of build list for saved segments, called a system saved segment build list. The system saved segment build list contains the name of each saved segment in the z/VM system. If the product that defines the saved segment is in VMSES/E format, the system saved segment build list also contains the name of the product saved segment build list. The name of the system saved segment build list is identified in the build section of the saved segment product parameter file. The name identified in the supplied saved segment product parameter file is SEGBLIST.

Note: Do not use more than one system saved segment build list to define the saved segments in a single z/VM system. You can, however, use different system saved segment build lists to define alternate saved segment layouts on the same z/VM system.

Saved Segment Data File

The saved segment data file contains build information for the saved segments identified in the system saved segment build list. The saved segment data file has the same file name as the system saved segment build list, and the file type is SEGDATA.

The SEGDATA file contains a series of saved segment definitions. Each definition identifies the name of the saved segment, its storage range, the names of the segment spaces in which it is a member (if any), whether it contains logical saved segments, whether it can be loaded above 16 MB, the names of requisite saved segments (if any), the build parameters, and other information. VMSES/E supplies the VMFSGMAP EXEC for viewing and customizing these definitions.

VMFSGMAP EXEC

The VMFSGMAP EXEC is the VMSES/E tool that allows you to display and customize your saved segment layout. VMFSGMAP generates a segment map that shows all the saved segments defined in the SEGDATA file. The segment map also shows all the saved segments and saved systems that are defined on your system (in system data files created by the DEFSEG, DEFSYS, SAVESEG, and SAVESYS commands) that are not included in the SEGDATA file.

Note: Saved systems are displayed in the segment map only for planning purposes. They are not defined in the SEGDATA file, and you cannot use VMFSGMAP to customize them.

Viewing the segment map allows you to see what saved segments are defined, where they are defined, and how they relate to each other. For example, you can see what saved segments are defined in the same storage area, what member saved segments are defined in multiple segment spaces, and so on. "Viewing the Segment Map" on page 60 describes the format and contents of the segment map.

From the segment map, you can select a specific saved segment and display its definition (the information used to build it). You can change the storage range, the segment space information, or other parts of the definition. You can get default information from the PRODPART file. You can add or delete definitions. You can then view the results of your changes in the segment map. You can make and view all kinds of changes to your saved segment layout before you actually build or delete any of the saved segments. "Viewing a Saved Segment Definition" on page 61 describes the format and contents of a saved segment definition. "Changing, Adding, and Deleting Saved Segment Definitions" on page 62 describes how to use VMFSGMAP to do various saved segment management tasks.

In an Single System Image (SSI) cluster, VMFSGMAP has to be run on only one of the members or systems on which the product is installed.

PUT2PROD EXEC

The PUT2PROD EXEC is the VMSES/E tool that automates the building of saved segments. PUT2PROD calls VMFBLD, which processes the system saved segment build list and uses the build information supplied by the definitions in the SEGDATA file.

In an SSI cluster, PUT2PROD must be run on every member or system on which the product is installed.

Resource Requirements for Building and Managing Saved Segments

Saved segment build and management tasks are normally done using the MAINT*vrm* virtual machine. If you plan to use some other user ID, it must have:

- Access to the VMSES/E Software Inventory disks: MAINT*vrm* 51D and PMAINT 41D.
- Access to the VMSES/E code on MAINT*vrm* 5E5.
- The ability to link and access the minidisks and SFS directories specified in the saved segment product parameter file, the product-level product parameter file, and the saved segment definitions in the SEGDATA file. Minidisks must already be linked if you are not using the LINK build list option.
- The ability to get write access to the build target disk specified in the saved segment product parameter file.
- CP authority to issue the DEFSEG, QUERY NSS, PURGE NSS, and SAVESEG commands.
- A virtual machine with enough storage to contain the saved segment with the highest storage range and the additional storage used by CMS.
- A NAMESAVE directory control statement entry for each restricted saved segment.
- A read/write A-disk.

Viewing the Segment Map

To begin the VMFSGMAP session, enter the VMFSGMAP command with the name of the saved segment product parameter file (SEGBLD), the name of the z/VM saved segments "component" in the saved segment product parameter file (ESASEGS), and the name of the system saved segment build list (SEGBLIST), which is also the name of the associated SEGDATA file:

```
vmfsgmap segbld esasegs segblist
```

Note: If you enter only VMFSGMAP, this command will use the default PPF and build list names as shown here.

Entering the VMFSGMAP command places you into an XEDIT session. The initial display is the Segment Map panel. VMFSGMAP divides your system storage into 4 MB ranges: 0-3 MB, 4-7 MB, and so on. If a saved segment or saved system is wholly or partially defined in one of these 4 MB ranges, VMFSGMAP displays a heading for the storage range on the panel, followed by a segment map record for each saved segment or saved system defined in the range. A segment map record identifies the status, name, and type of the saved segment or saved system and pictorially shows the amount of storage used.

If a saved segment is defined in the SEGDATA file and on the system, the segment map record contains the version from the SEGDATA file and indicates by a status code (column 1) how the definitions compare.

Figure 12 on page 60 shows an example of the Segment Map panel. This example shows that the first 4 MB range of storage that contains a saved segment or saved system is X'004-007' MB. Following the heading for the X'004-007' MB range is a map record for the CMSPIPES saved segment. The map record supplies the following information about CMSPIPES:

- Its status is 'P', for planned, which means it is defined in the SEGDATA file but not on the system.
- Its type is DCSS, and it occupies the 1 MB storage range from 7-8 MB.
- Only the first four 64 KB "segments" of the DCSS contain data, and the permitted access is read-only.

For information about the syntax of segment map records, see the description of the VMFSGMAP EXEC in the *z/VM: VMSES/E Introduction and Reference*.

```
                            VMFSGMAP - Segment Map
                                                          Lines 1 to 19 of 28

Meg             004-MB          005-MB          006-MB          007-MB
St  Name   Typ 0123456789ABCDEF0123456789ABCDEF0123456789ABCDEF0123456789ABCDEF
P CMSPIPES DCS 4.............5.............6.............RRRR------------

Meg             008-MB          009-MB          00A-MB          00B-MB
St  Name   Typ 0123456789ABCDEF0123456789ABCDEF0123456789ABCDEF0123456789ABCDEF
P DOSBAM   SPA 8.............9.............====------------===============
P CMSAMS   MEM 8.............9.............WWW.............B...RRRRRR......
P CMSBAM   MEM 8.............9.............A.............BRRR............
P CMSDOS   MEM 8.............9.............A.............R...............
P CMSVSAM  MEM 8.............9.............A..W..........B.........RRRRRR
P CMSFILES DCS 8.............-----RRRRRRRRRRRRRRRRRRRRRRRRRRB.............
P CMSVMLIB DCS RRRRRRRR--------9.............A.............B.............
P DOSINST  DCS 8.............R---------------A.............B.............

               00C-MB          00D-MB          00E-MB          00F-MB
  Name     Typ 0123456789ABCDEF0123456789ABCDEF0123456789ABCDEF0123456789ABCDEF

 F1=Help     F2=Chk Obj   F3=Exit      F4=Chg Obj   F5=File       F6=Save
 F7=Bkwd     F8=Fwd       F9=Retrieve  F10=Add Obj  F11=Del Obj   F12= Class
====>
```

Figure 12. Example of the VMFSGMAP EXEC Segment Map Panel

Pressing the PF8/20 (Fwd) key scrolls down one page in the file, as shown in Figure 13 on page 61.

```
                            VMFSGMAP - Segment Map
                                                        Lines 20 to 28 of 28

================================= 16-MB Line ==================================

Meg            010-MB          011-MB          012-MB          013-MB
St  Name   Typ 0123456789ABCDEF0123456789ABCDEF0123456789ABCDEF0123456789ABCDEF
P SVM      DCS 0.............1.............2.............RRRRRRRRRRRRRRRR
==============================  End Segment Map ==============================









 F1=Help     F2=Chk Obj   F3=Exit      F4=Chg Obj   F5=File       F6=Save
 F7=Bkwd     F8=Fwd       F9=Retrieve  F10=Add Obj  F11=Del Obj   F12= Class
====>
```

Figure 13. Example of the VMFSGMAP EXEC Segment Map Panel, Continued

Viewing a Segment Space

If you want to view only a particular segment space and its members, rather than the full segment map, you can enter the VMFSGMAP command with the SPACE option and the name of the segment space:

```
vmfsgmap segbld esasegs segblist (space spacename
```

You cannot use the SPACE option to view other types of saved segments. Furthermore, if you plan to move or add member saved segments, it is recommended that you enter the VMFSGMAP command without the SPACE option, so you can see how your changes relate to other saved segments as displayed in the full segment map.

Viewing a Saved Segment Definition

On the Segment Map panel, you can move the cursor to the map record for a particular saved segment and press the PF4/16 (Chg Obj) key to display the definition record for the saved segment in the Change Segment Definition panel. For example, on the Segment Map panel shown in Figure 12 on page 60, if you move your cursor to the map record for CMSPIPES and press PF4/16, the Change Segment Definition panel shown in Figure 14 on page 62 is displayed.

```
                          Change Segment Definition


 OBJNAME....:  CMSPIPES
 DEFPARMS...:  700-73F SR
 SPACE......:
 TYPE.......:  PSEG
 OBJDESC....:  CMS PIPES SEGMENT
 OBJINFO....:
 GT_16MB....:  YES
 DISKS......:
 SEGREQ.....:
 PRODID.....:  2VMVMA10 CMS
 BLDPARMS...:  PPF(ESA CMS DMSSBPIP)









F1=Help      F2=Get Obj  F3=Exit       F4=Add Line  F5=Map      F6=Chk Mem
F7=Bkwd      F8=Fwd      F9=Retrieve   F10=Seginfo  F11=Adj Mem  F12=Cancel
====>
```

Figure 14. Example of the VMFSGMAP EXEC Change Segment Definition Panel

The fields in the saved segment definition supply information about where and how the saved segment is built. For example, the definition in Figure 14 on page 62 provides the following information:

- The OBJNAME field indicates that the name of the saved segment is CMSPIPES.
- The DEFPARMS field indicates that CMSPIPES is stored in pages X'700-73F', and the access allowed is shared read-only.
- The SPACE field is blank, indicating that CMSPIPES is a DCSS.
- The TYPE field indicates that CMSPIPES is a physical saved segment containing at least one logical saved segment.
- The GT_16MB field indicates that CMSPIPES may be loaded above the 16 MB line.
- The DISKS field indicates that no additional minidisks or SFS directories need to be accessed to build CMSPIPES.
- The SEGREQ field indicates that no saved segments need to be built before CMSPIPES is built.
- The PRODID field indicates that the default information for CMSPIPES is contained in the CMS section of the 2VMVMA10 PRODPART file.
- The BLDPARMS field indicates that the build information for CMSPIPES is defined in the DMSSBPIP product saved segment build list and in the CMS component section of the ESA product parameter file.

  Note: In the definition for a saved segment defined by a product not in VMSES/E format, this field contains the actual build parameters used by VMFBLD.

For information about the syntax of the fields in a saved segment definition record, see the description of the VMFSGMAP EXEC in the *z/VM: VMSES/E Introduction and Reference*.

Changing, Adding, and Deleting Saved Segment Definitions

This section outlines how to use the VMFSGMAP EXEC to make the following types of changes to the saved segment definitions in the SEGDATA file:

- “Changing the Range of a DCSS” on page 63
- “Changing the Range of a Member Saved Segment” on page 63
- “Renaming a DCSS or Member Saved Segment” on page 64
- “Changing the Name of a Segment Space” on page 65

Changing the Range of a DCSS

1. If you are not already in a VMFSGMAP session, enter the following command to display the full Segment Map panel:

   ```
   vmfsgmap segbld esasegs segblist
   ```

   For detailed information about VMFSGMAP panels, see the description of the VMFSGMAP EXEC in the *z/VM: VMSES/E Introduction and Reference*.
2. On the Segment Map panel, move the cursor to the map record for the DCSS to be changed.
3. Press PF4/16 (Chg Obj) to display the Change Segment Definition panel.
4. Move the cursor to the DEFPARMS field in the definition record and change the range.
5. Press PF5/17 (Map) to return to the Segment Map panel, which is refreshed to show the changes, or press PF12/24 (Cancel).
6. Review the updated map to make sure that the new range is appropriate in relation to the other saved segments in the map.
7. If the new range is not acceptable, repeat the previous steps to readjust the range, or press PF3/15 (Exit) to discard the changes and exit the map.
8. If the new range is satisfactory, press PF6/18 (Save) to record the changes and remain in the map to do other tasks, or press PF5/17 (File) to record the changes and exit the map.
9. After completing the VMFSGMAP session, use the PUT2PROD EXEC to actually build or delete the saved segments. See "Building or Deleting (Purging) Saved Segments" on page 76.

Some segment names are contained in the CMSSEGS BLDDATA and DOSBAM BLDDATA files. CMSSEGS BLDDATA is a list of CMS segments and DOSBAM BLDDATA is a list of segments in the DOSBAM segment space. These lists must be modified to reflect any segment changes you make.

Changing the Range of a Member Saved Segment

1. If you are not already in a VMFSGMAP session, enter the following command to display the full Segment Map panel:

```
vmfsgmap segbld esasegs segblist
```

For detailed information about VMFSGMAP panels, see the description of the VMFSGMAP EXEC in the *z/VM: VMSES/E Introduction and Reference*.

2. On the Segment Map panel, move the cursor to the map record for the member saved segment to be changed.
3. Press PF4/16 (Chg Obj) to display the Change Segment Definition panel.
4. Move the cursor to the DEFPARMS field in the definition record and change the range.
5. Press PF6/18 (Chk Mem) to check for overlaps with other members in all the segment spaces identified in the SPACE field. Definition records for overlapping members are added to the panel.
6. If there are overlaps, press PF11/23 (Adj Mem) to automatically adjust the ranges of the affected members to remove the overlaps.

   Note: If you prefer, you can manually adjust the ranges of the affected members. However, if you manually adjust the range of a member, you should press PF6/18 (Chk Mem) again to make sure that your change has not caused a new overlap.
7. After you complete your changes on the Change Segment Definition panel, press PF5/17 (Map) to return to the Segment Map panel, which is refreshed to show all the changes, or press PF12/24 (Cancel) to discard your changes and return to the Segment Map panel.
8. Review the updated map to make sure that the new member ranges are appropriate in relation to the other saved segments in the map.
9. If the new saved segment layout is not acceptable, repeat the previous steps to adjust the member ranges, or press PF3/15 (Exit) to discard the changes and exit the map.
10. If the new saved segment layout is satisfactory, press PF6/18 (Save) to record the changes and remain in the map to do other tasks, or press PF5/17 (File) to record the changes and exit the map.
11. After completing the VMFSGMAP session, use the PUT2PROD EXEC to actually build or delete the saved segments. See "Building or Deleting (Purging) Saved Segments" on page 76.

Renaming a DCSS or Member Saved Segment

To rename a DCSS or member saved segment, first make a copy of the saved segment and give it the new name, then delete the version with the old name:

1. If you are not already in a VMFSGMAP session, enter the following command to display the full Segment Map panel:

   ```
   vmfsgmap segbld esasegs segblist
   ```

   For detailed information about VMFSGMAP panels, see the description of the VMFSGMAP EXEC in the *z/VM: VMSES/E Introduction and Reference*.
2. On the Segment Map panel, move the cursor to the map record for the DCSS or member saved segment to be renamed.
3. Press PF10/22 (Add Obj) to display the Add Segment Definition panel.
4. Move the cursor to the OBJNAME field in the definition record and change the question marks to the new name you have selected, which must not already be in use for any other segment space, member saved segment, or DCSS.
5. Press PF5/17 (Map) to return to the Segment Map panel, which is refreshed to include the new saved segment, or press PF12/24 (Cancel) to discard your changes and return to the Segment Map panel.
6. Move the cursor to the map record for the old version of the saved segment.
7. Press PF11/23 (Del Obj) to delete the saved segment from the map. The display is refreshed to show the change.
8. If you want to stop the rename and reinstate the old name, press PF3/15 (Exit) to discard the changes and exit the map.

9. If you want to continue the rename, press PF6/18 (Save) to record the changes and remain in the map to do other tasks, or press PF5/17 (File) to record the changes and exit the map.
10. After completing the VMFSGMAP session, use the PUT2PROD EXEC to actually build or delete the saved segments. See "Building or Deleting (Purging) Saved Segments" on page 76.

Changing the Name of a Segment Space

1. If you are not already in a VMFSGMAP session, enter the following command to display the full Segment Map panel:

   ```
   vmfsgmap segbld esasegs segblist
   ```

   For detailed information about VMFSGMAP panels, see the description of the VMFSGMAP EXEC in the *z/VM: VMSES/E Introduction and Reference*.
2. On the Segment Map panel, move the cursor to the map record for the segment space.
3. Press PF4/16 (Chg Obj) to display the Change Segment Definition panel. The panel contains definition records for all the members of the segment space.
4. Move the cursor to the SPACE field in each definition record and change the name of the segment space.
5. Press PF5/17 (Map) to return to the Segment Map panel, which is refreshed to show the changes, or press PF12/24 (Cancel) to discard your changes and return to the Segment Map panel.
6. If you want to stop the rename and reinstate the old name, press PF3/15 (Exit) to discard the changes and exit the map.
7. If you want to continue the rename, press PF6/18 (Save) to record the changes and remain in the map to do other tasks, or press PF5/17 (File) to record the changes and exit the map.
8. After completing the VMFSGMAP session, use the PUT2PROD EXEC to actually build or delete the saved segments. See "Building or Deleting (Purging) Saved Segments" on page 76.

Changing Multiple Members of a Segment Space

1. If you are not already in a VMFSGMAP session, enter the following command to display the full Segment Map panel:

   ```
   vmfsgmap segbld esasegs segblist
   ```

   For detailed information about VMFSGMAP panels, see the description of the VMFSGMAP EXEC in the *z/VM: VMSES/E Introduction and Reference*.
2. On the Segment Map panel, move the cursor to the map record for the segment space.
3. Press PF4/16 (Chg Obj) to display the Change Segment Definition panel. The panel contains definition records for all the members of the segment space.
4. Make your changes to the member definitions.
5. If you change the range of any member, press PF6/18 (Chk Mem) to check for overlaps with other members in all the segment spaces to which the changed member belongs. Definition records for overlapping members are added to the panel.
6. If there are overlaps, press PF11/23 (Adj Mem) to automatically adjust the ranges of the affected members to remove the overlaps.

   Note: If you prefer, you can manually adjust the ranges of the affected members. However, if you manually adjust the range of a member, you should press PF6/18 (Chk Mem) again to make sure that your change has not caused a new overlap. This is especially critical for members that exist in other segment spaces.
7. After you complete your changes to the Change Segment Definition panel, press PF5/17 (Map) to return to the Segment Map panel, which is refreshed to show all the changes, or press PF12/24 (Cancel) to discard your changes and return to the Segment Map panel.

8. Review the updated map to make sure that the saved segment layout is acceptable.
9. If the new layout is not acceptable, repeat the previous steps to adjust member definitions until the layout is acceptable, or press PF3/15 (Exit) to discard the changes and exit the map.
10. If the new layout is satisfactory, press PF6/18 (Save) to record the changes and remain in the map to do other tasks, or press PF5/17 (File) to record the changes and exit the map.
11. After completing the VMFSGMAP session, use the PUT2PROD EXEC to actually build or delete the saved segments. See "Building or Deleting (Purging) Saved Segments" on page 76.

Adding a DCSS or Member Saved Segment

1. If you are not already in a VMFSGMAP session, enter the following command to display the full Segment Map panel:

   ```
   vmfsgmap segbld esasegs segblist
   ```

   For detailed information about VMFSGMAP panels, see the description of the VMFSGMAP EXEC in the *z/VM: VMSES/E Introduction and Reference*.
2. If you want to use an existing saved segment definition as a model for the new saved segment, move the cursor to the map record for the existing saved segment. Otherwise, move the cursor so it is not located on any map record.
3. Press PF10/22 (Add Obj) to display the Add Segment Definition panel.
4. Complete the definition record for the saved segment you want to add. The following is the minimum information required to define a saved segment:
   - Name (OBJNAME field)
   - Storage range (DEFPARMS field)
   - Whether it is a member of any segment spaces (SPACE field)
   - Whether it contains CMS logical saved segments (TYPE field)
   - Whether it can be loaded above 16 MB (GT_16MB field).

   Note: If not specified, the build parameters (BLDPARMS field) default to 'UNKNOWN', which means that during build processing for this saved segment VMFBLD can only issue the DEFSEG command to define the saved segment to CP. The user must issue the function that loads and saves the saved segment.

   If you want to get information for a saved segment that is already defined, enter the name of the existing saved segment in the OBJNAME field and press PF2/14 (Get Obj). VMFSGMAP gets the definition record, if it exists, from the SEGDATA file. If the saved segment is defined on the system (defined in a system data file), VMFSGMAP updates the DEFPARMS and SPACE fields in the displayed definition record with the system data. You must change the name in the OBJNAME field of the definition record before you file the update, because each saved segment defined to CP must have a unique name. Also, if the information you obtain is for a saved segment defined above 16 MB, and you want the new saved segment to be defined above 16 MB, make sure that the GT_16MB field is set to 'YES'.

   If you want to get the default information for a saved segment known to VMSES/E (defined in a product PRODPART file), enter the name of the saved segment in the OBJNAME field and press PF10/22 (Seginfo). If this saved segment already exists in the SEGDATA file or on the system, you must change the name in the OBJNAME field of the definition record before you file the update. Each saved segment defined to CP must have a unique name.
5. If you are adding a member saved segment, press PF6/18 (Chk Mem) to check for overlaps with other members in all the segment spaces specified in the SPACE field. Definition records for overlapping members are added to the panel.

   If there are overlaps, press PF11/23 (Adj Mem) to automatically adjust the ranges of the affected members to remove the overlaps.

Note: If you prefer, you can manually adjust the ranges of the affected members. However, if you manually adjust the range of a member, you should press PF6/18 (Chk Mem) again to make sure that your change has not caused a new overlap.

6. Press PF5/17 (Map) to return to the Segment Map panel, which is refreshed to show all the changes, or press PF12/24 (Cancel) to discard your changes and return to the Segment Map panel.
7. Review the updated map to make sure that the new saved segment layout is acceptable.
8. If the new layout is not acceptable, make the appropriate changes to the saved segment definitions, or press PF3/15 (Exit) to discard the changes and exit the map.
9. If the new layout is satisfactory, press PF6/18 (Save) to record the changes and remain in the map to do other tasks, or press PF5/17 (File) to record the changes and exit the map.
10. After completing the VMFSGMAP session, use the PUT2PROD EXEC to actually build or delete the saved segments. See "Building or Deleting (Purging) Saved Segments" on page 76.

Merging Existing Saved Segments into the SEGDATA File

A saved segment that is mapped from the system (defined in a system data file) but is not defined in the SEGDATA file is indicated on the Segment Map panel by a status code of 'M' in the first column of the saved segment map record. To merge all of these existing saved segments into the SEGDATA file:

1. If you are not already in a VMFSGMAP session, enter the following command to display the full Segment Map panel:

   ```
   vmfsgmap segbld esasegs segblist
   ```

   For detailed information about VMFSGMAP panels, see the description of the VMFSGMAP EXEC in the *z/VM: VMSES/E Introduction and Reference*.
2. Move the cursor to the command line and enter the SEGMERGE subcommand. Saved segment definitions that already exist in the SEGDATA file are not changed. In addition, the system saved segment build list is compared with the SEGDATA file, and entries are added to the build list where necessary.
3. Review the updated map to make sure that the new saved segment layout is acceptable.
4. If the new layout is not acceptable, make the appropriate changes to the saved segment definitions, or press PF3/15 (Exit) to discard the changes and exit the map.
5. If the new layout is satisfactory, press PF6/18 (Save) to record the changes and remain in the map to do other tasks, or press PF5/17 (File) to record the changes and exit the map.
6. After completing the VMFSGMAP session, use the PUT2PROD EXEC to actually build or delete the saved segments. See "Building or Deleting (Purging) Saved Segments" on page 76.

Copying a DCSS

1. If you are not already in a VMFSGMAP session, enter the following command to display the full Segment Map panel:

   ```
   vmfsgmap segbld esasegs segblist
   ```

   For detailed information about VMFSGMAP panels, see the description of the VMFSGMAP EXEC in the *z/VM: VMSES/E Introduction and Reference*.
2. On the Segment Map panel, move the cursor to the map record for the DCSS to be copied.
3. Press PF10/22 (Add Obj) to display the Add Segment Definition panel.
4. Move the cursor to the OBJNAME field and change the question marks to a name that is not already used for a saved segment. Change any other parts of the definition that need to be modified.
5. Press PF5/17 (Map) to return to the Segment Map panel, which is refreshed to include the new DCSS, or press PF12/24 (Cancel) to discard your changes and return to the Segment Map panel.

6. Review the updated map to make sure that the new DCSS definition is appropriate in relation to the other saved segments in the map.
7. If the range of the new DCSS is not acceptable:
   a. Move the cursor to the map record for the new DCSS.
   b. Press PF4/16 (Chg Obj) to display the Change Segment Definition panel.
   c. Move the cursor to the DEFPARMS field and change the range.
   d. Press PF5/17 (Map) to return to the Segment Map panel, which is refreshed to show your changes, or press PF12/24 (Cancel) to discard your changes and return to the Segment Map panel.

   If the range is still not acceptable, repeat the previous sequence, or press PF3/15 (Exit) to discard the changes and exit the map.
8. If the new DCSS definition is satisfactory, press PF6/18 (Save) to record the changes and remain in the map to do other tasks, or press PF5/17 (File) to record the changes and exit the map.
9. After completing the VMFSGMAP session, use the PUT2PROD EXEC to actually build or delete the saved segments. See "Building or Deleting (Purging) Saved Segments" on page 76.

Copying or Moving a Member Saved Segment into Another Segment Space

1. If you are not already in a VMFSGMAP session, enter the following command to display the full Segment Map panel:

   ```
   vmfsgmap segbld esasegs segblist
   ```

   For detailed information about VMFSGMAP panels, see the description of the VMFSGMAP EXEC in the *z/VM: VMSES/E Introduction and Reference*.
2. On the Segment Map panel, move the cursor to the map record for the member saved segment to be copied or moved.
3. Press PF4/16 (Chg Obj) to display the Change Segment Definition panel.
4. Move the cursor to the SPACE field of the definition record:
   - If the member is to be copied into another segment space, add the name of the new segment space.
   - If the member is to be moved to another segment space, change the old segment space name to the new segment space name.
5. Press PF6/18 (Chk Mem) to check for overlaps with other members in all the segment spaces to which the changed member belongs. Definition records for overlapping members are added to the panel.
6. If there are overlaps, press PF11/23 (Adj Mem) to automatically adjust the ranges of the affected members to remove the overlaps.

   Note: If you prefer, you can manually adjust the ranges of the affected members. However, if you manually adjust the range of a member, you should press PF6/18 (Chk Mem) again to make sure that your change has not caused a new overlap.
7. After you complete all your changes, press PF5/17 (Map) to return to the Segment Map panel, which is refreshed to show the changes, or press PF12/24 (Cancel) to discard your changes and return to the Segment Map panel.
8. Review the updated map to make sure that the saved segment layout is acceptable.
9. If the layout is not acceptable, make the appropriate changes to the saved segment definitions, or press PF3/15 (Exit) to discard the changes and exit the map.
10. If the layout is satisfactory, press PF6/18 (Save) to record the changes and remain in the map to do other tasks, or press PF5/17 (File) to record the changes and exit the map.

11. After completing the VMFSGMAP session, use the PUT2PROD EXEC to actually build or delete the saved segments. See "Building or Deleting (Purging) Saved Segments" on page 76.

Copying a Segment Space

1. If you are not already in a VMFSGMAP session, enter the following command to display the full Segment Map panel:

   ```
   vmfsgmap segbld esasegs segblist
   ```

   For detailed information about VMFSGMAP panels, see the description of the VMFSGMAP EXEC in the *z/VM: VMSES/E Introduction and Reference*.
2. On the Segment Map panel, move the cursor to the map record for the segment space to be copied.
3. Press PF4/16 (Chg Obj) to display the Change Segment Definition panel. The panel displays definition records for all the members of the segment space.
4. Move the cursor to the SPACE field in each definition record and add the new segment space name.
5. Press PF6/18 (Chk Mem) to check for overlaps with other members in all the segment spaces to which the changed member belongs. Definition records for overlapping members are added to the panel.
6. If there are overlaps, press PF11/23 (Adj Mem) to automatically adjust the ranges of the affected members to remove the overlaps.

   Note: If you prefer, you can manually adjust the ranges of the affected members. However, if you manually adjust the range of a member, you should press PF6/18 (Chk Mem) again to make sure that your change has not caused a new overlap. This is especially critical for members that exist in other segment spaces.
7. After you complete your changes on the Change Segment Definition panel, press PF5/17 (Map) to return to the Segment Map panel, which is refreshed to show all the changes, or press PF12/24 (Cancel) to discard your changes and return to the Segment Map panel.
8. Review the updated map to make sure that the new saved segment layout is acceptable.
9. If the layout is not acceptable, press PF3/15 (Exit) to discard the changes and exit the map.
10. If the layout is satisfactory, press PF6/18 (Save) to record the changes and remain in the map to do other tasks, or press PF5/17 (File) to record the changes and exit the map.
11. After completing the VMFSGMAP session, use the PUT2PROD EXEC to actually build or delete the saved segments. See "Building or Deleting (Purging) Saved Segments" on page 76.

Converting a DCSS to a Member of a Segment Space

1. If you are not already in a VMFSGMAP session, enter the following command to display the full Segment Map panel:

   ```
   vmfsgmap segbld esasegs segblist
   ```

   For detailed information about VMFSGMAP panels, see the description of the VMFSGMAP EXEC in the *z/VM: VMSES/E Introduction and Reference*.
2. On the Segment Map panel, move the cursor to the map record for the DCSS you want to convert.
3. Press PF4/16 (Chg Obj) to display the Change Segment Definition panel.
4. Move the cursor to the SPACE field and enter the name of the segment space to which you are adding this saved segment as a member.
5. Update the range or other parts of the definition record, as necessary.
6. Press PF6/18 (Chk Mem) to check for overlaps with other members of the segment space. Definition records for overlapping members are added to the panel.

7. If there are overlaps, press PF11/23 (Adj Mem) to automatically adjust the ranges of the affected members to remove the overlaps.

   Note: If you prefer, you can manually adjust the ranges of the affected members. However, if you manually adjust the range of a member, you should press PF6/18 (Chk Mem) again to make sure that your change has not caused a new overlap.
8. After you complete your changes on the Change Segment Definition panel, press PF5/17 (Map) to return to the Segment Map panel, which is refreshed to show the changes, or press PF12/24 (Cancel) to discard your changes and return to the Segment Map panel.
9. Review the updated map to make sure that the new saved segment layout is acceptable.
10. If the new layout is not acceptable, make the appropriate changes to the saved segment definitions, or press PF3/15 (Exit) to discard the changes and exit the map.
11. If the new layout is satisfactory, press PF6/18 (Save) to record the changes and remain in the map to do other tasks, or press PF5/17 (File) to record the changes and exit the map.
12. After completing the VMFSGMAP session, use the PUT2PROD EXEC to actually build or delete the saved segments. See “Building or Deleting (Purging) Saved Segments” on page 76.

Converting a Member of a Segment Space to a DCSS

1. If you are not already in a VMFSGMAP session, enter the following command to display the full Segment Map panel:

   ```
   vmfsgmap segbld esasegs segblist
   ```

   For detailed information about VMFSGMAP panels, see the description of the VMFSGMAP EXEC in the *z/VM: VMSES/E Introduction and Reference*.
2. On the Segment Map panel, move the cursor to the map record for the member saved segment you want to convert.
3. Press PF4/16 (Chg Obj) to display the Change Segment Definition panel.
4. Move the cursor to the SPACE field and delete all the segment space names.
5. Update the range or other parts of the definition record, as necessary.
6. Press PF5/17 (Map) to return to the Segment Map panel, which is refreshed to show your changes, or press PF12/24 (Cancel) to discard your changes and return to the Segment Map panel.
7. Review the updated map to make sure that the new saved segment layout is acceptable.
8. If the new layout is not acceptable, make the appropriate changes to the saved segment definitions, or press PF3/15 (Exit) to discard the changes and exit the map.
9. If the new layout is satisfactory, press PF6/18 (Save) to record the changes and remain in the map to do other tasks, or press PF5/17 (File) to record the changes and exit the map.
10. After completing the VMFSGMAP session, use the PUT2PROD EXEC to actually build or delete the saved segments. See “Building or Deleting (Purging) Saved Segments” on page 76.

Deleting a DCSS

1. If you are not already in a VMFSGMAP session, enter the following command to display the full Segment Map panel:

   ```
   vmfsgmap segbld esasegs segblist
   ```

   For detailed information about VMFSGMAP panels, see the description of the VMFSGMAP EXEC in the *z/VM: VMSES/E Introduction and Reference*.
2. On the Segment Map panel, move the cursor to the map record for the DCSS you want to delete.

3. Press PF11/23 (Del OBJ) to delete the record from the map, or:
   a. Press PF4/16 (Chg Obj) to display the Change Segment Definition panel.
   b. Move the cursor to the DEFPARMS field and enter the word 'DELETED'.
   c. Press PF5/17 (Map) to return to the Segment Map panel, which is refreshed to show your changes, or press PF12/24 (Cancel) to discard your changes and return to the Segment Map panel.

   Whichever method you use to delete the DCSS, a record for the deleted DCSS is added to the end of the map. If the deleted DCSS is defined on the system, the record is:

   ```
   D objname DCS DELETED
   ```

   If the deleted DCSS is defined only in the SEGDATA file, the record is:

   ```
   P objname DCS DELETED
   ```

4. If you want to stop the deletion and reinstate the saved segment, press PF3/15 (Exit) to discard the changes and exit the map.
5. If you want to continue the deletion, press PF6/18 (Save) to record the changes and remain in the map to do other tasks, or press PF5/17 (File) to record the changes and exit the map.
6. After completing the VMFSGMAP session, use the PUT2PROD EXEC to actually build or delete the saved segments. See “Building or Deleting (Purging) Saved Segments” on page 76.

Deleting a Member Saved Segment

1. If you are not already in a VMFSGMAP session, enter the following command to display the full Segment Map panel:

   ```
   vmfsgmap segbld esasegs segblist
   ```

   For detailed information about VMFSGMAP panels, see the description of the VMFSGMAP EXEC in the *z/VM: VMSES/E Introduction and Reference*.
2. On the Segment Map panel, move the cursor to the map record for the member saved segment you want to delete.
3. You can do the deletion from the Segment Map panel or the Segment Definition panel:
   - From the Segment Map panel:
     a. Press PF11/23 (Del Obj) to delete the record from the map. The member is deleted only from the segment space that immediately precedes it in the map. The cursor then moves to next occurrence of the member in the map, if any. To completely delete the member, you must delete every occurrence. If you delete the last member of a segment space, the segment space is also deleted.
   - From the Segment Definition panel:
     a. Press PF4/16 (Chg Obj) to display the Change Segment Definition panel.
     b. If you want to just delete this member from certain segment spaces, move the cursor to the SPACE field and remove the names of those spaces. If you remove the name of a space that contains no other members, the space is also deleted. If only one space was specified, removing that name converts the member saved segment to a DCSS.

        If you really want to delete the member entirely, move the cursor to the DEFPARMS field and enter the word 'DELETED'.
     c. Press PF5/17 (Map) to return to the Segment Map panel, which is refreshed to show your changes, or press PF12/24 (Cancel) to discard your changes and return to the Segment Map panel.

Whichever method you use, if you completely delete the member saved segment (that is, delete it from every segment space), it becomes a deleted DCSS, and a record is added to the end of the map. If the saved segment is defined on the system, the record is:

```
D objname DCS DELETED
```

If the saved segment is defined only in the SEGDATA file, the record is:

```
P objname DCS DELETED
```

4. If you want to stop the deletion and reinstate the saved segment, press PF3/15 (Exit) to discard the changes and exit the map.
5. If you want to continue the deletion, press PF6/18 (Save) to record the changes and remain in the map to do other tasks, or press PF5/17 (File) to record the changes and exit the map.
6. After completing the VMFSGMAP session, use the PUT2PROD EXEC to actually build or delete the saved segments. See "Building or Deleting (Purging) Saved Segments" on page 76.

Deleting a Segment Space

1. If you are not already in a VMFSGMAP session, enter the following command to display the full Segment Map panel:

   ```
   vmfsgmap segbld esasegs segblist
   ```

   For detailed information about VMFSGMAP panels, see the description of the VMFSGMAP EXEC in the *z/VM: VMSES/E Introduction and Reference*.
2. On the Segment Map panel, move the cursor to the map record for the segment space you want to delete.
3. Press PF11/23 (Del Obj) to delete the segment space from the map. Members of this segment space that do not belong to any other segment space are also deleted from the map. Members of this segment space that belong to other segment spaces are not changed, except that the name of the deleted segment space is removed from the SPACE field in the definition record for each member.

   Note: If a member of this segment space does not belong to any other segment space, but you do not want the saved segment deleted, you can convert the member to a DCSS. See "Converting a Member of a Segment Space to a DCSS" on page 70.
4. If you want to stop the deletion and reinstate the segment space, press PF3/15 (Exit) to discard the changes and exit the map.
5. If you want to continue the deletion, press PF6/18 (Save) to record the changes and remain in the map to do other tasks, or press PF5/17 (File) to record the changes and exit the map.
6. After completing the VMFSGMAP session, use the PUT2PROD EXEC to actually build or delete the saved segments. See "Building or Deleting (Purging) Saved Segments" on page 76.

Retrieving a Deleted DCSS or Member Saved Segment

When you use VMFSGMAP to delete a DCSS or member saved segment from the segment map, the definition record is not deleted from the SEGDATA file, only marked 'DELETED' in the DEFPARMS field. Therefore, you can use VMFSGMAP to retrieve the deleted saved segment and add it back into the map.

Note: Segment spaces are created dynamically by CP and are not defined by definition records in the SEGDATA file. Therefore, you cannot use this method to retrieve a deleted segment space. Assuming that the members of the deleted segment space still exist as members of other spaces or as DCSSs, you must add the name of the space to the definition record for each saved segment that is to be a member of the space. See "Copying or Moving a Member Saved Segment into Another Segment Space" on page 68 and "Converting a DCSS to a Member of a Segment Space" on page 69.

1. If you are not already in a VMFSGMAP session, enter the following command to display the full Segment Map panel:

```
vmfsgmap segbld esasegs segblist
```

For detailed information about VMFSGMAP panels, see the description of the VMFSGMAP EXEC in the *z/VM: VMSES/E Introduction and Reference*.

2. On the Segment Map panel, move the cursor to the record for the deleted saved segment, which is located at the end of the map.
3. Press PF4/16 (Chg Obj) to display the Change Segment Definition panel.
4. Make any necessary changes to complete the definition. In the DEFPARMS field, remove the word 'DELETED', leaving the range of the saved segment. In the SPACE field, enter the names of any segment space of which this saved segment is to be a member. If you are defining the saved segment above 16 MB, make sure that the GT_16MB field is set to 'YES'. However, remember that there could be product restrictions on loading the saved segment above 16 MB.
5. Press PF5/17 (Map) to return to the Segment Map panel, which is refreshed to include the retrieved saved segment, or press PF12/24 (Cancel) to discard your changes and return to the Segment Map panel.
6. If you want to stop the retrieval of the saved segment, press PF3/15 (Exit) to discard the changes and exit the map.
7. If you want to continue with the retrieval, press PF6/18 (Save) to record the changes and remain in the map to do other tasks, or press PF5/17 (File) to record the changes and exit the map.
8. After completing the VMFSGMAP session, use the PUT2PROD EXEC to actually build or delete the saved segments. See "Building or Deleting (Purging) Saved Segments" on page 76.

Changing and Adding Definitions for Physical and Logical Saved Segments

A physical saved segment is a DCSS or member saved segment that contains CMS logical saved segments. A logical segment definition file (default file type LSEG) must exist for each logical saved segment to be loaded into the DCSS or member saved segment. VMSES/E-format products that define their saved segments as logical saved segments should supply the LSEG files. If you need to define any new logical saved segment definition files, or if you need to change the definition of an existing logical saved segment, see Chapter 2, "Planning and Defining CMS Logical Saved Segments," on page 39.

Note: Do not create a physical segment definition file (default file type PSEG) to identify the LSEG files. When PUT2PROD calls the VMFBLD EXEC to build the saved segments, VMFBLD creates the PSEG file before calling the SEGGEN command to build the physical and logical saved segments.

1. If you are not already in a VMFSGMAP session, enter the following command to display the full Segment Map panel:

```
vmfsgmap segbld esasegs segblist
```

For detailed information about VMFSGMAP panels, see the description of the VMFSGMAP EXEC in the *z/VM: VMSES/E Introduction and Reference*.

2. On the Segment Map panel, move the cursor to the map record for the DCSS or member saved segment in which you want to define, update, or remove CMS logical saved segments.

   Note: If the physical saved segment does not already exist, follow the directions in "Adding a DCSS or Member Saved Segment" on page 66 to create a new DCSS or member saved segment definition to work with.
3. Press PF4/16 (Chg Obj) to display the Change Segment Definition panel.
4. Change the definition of the saved segment as appropriate:
   a. Make sure that the first range defined in the DEFPARMS field is large enough to contain all of the CMS logical saved segments to be defined in the physical saved segment. You cannot define logical saved segments in noncontiguous storage.
   b. Make sure that the TYPE field contains the keyword PSEG, indicating that the saved segment is a physical saved segment containing logical saved segments.

c. Add or change the information in the BLDPARMS field to indicate the logical saved segments to be included in the physical saved segment. Use PF4/16 (Add Line) to add lines to the field for new entries. An entry with the PPF keyword points to a product parameter file where one or more logical saved segments are defined. An entry with the PROD keyword contains the definition for one logical saved segment. The BLDPARMS field may contain combinations of these entries.

   For example, suppose you want to combine the CMSPIPES logical saved segment, the CMSFILES logical saved segment, and your own logical saved segment called MYSEG into a single physical saved segment called CMSSEG1. Currently CMSPIPES and CMSFILES are contained in separate physical saved segments, each with its own definition. Create a new definition for CMSSEG1. Make sure that the range you define in the DEFPARMS field is large enough. Enter the following information into the BLDPARMS field:

```
BLDPARMS...:  PPF(ESA CMS DMSSBPIP)
              PPF(ESA CMS DMSSBSFS)
              PROD(LSEG MYSEG)
```

   Make sure that you also delete the existing definitions for CMSPIPES and CMSFILES.

5. Press PF5/17 (Map) to return to the Segment Map panel, which is refreshed to show your changes, or press PF12/24 (Cancel) to discard your changes and return to the Segment Map panel.
6. Review the updated segment map to make sure that the new saved segment layout is correct.
7. If the new layout is not acceptable, repeat the previous steps to make adjustments, or press PF3/15 (Exit) to discard the changes and exit the map.
8. If the new layout is satisfactory, press PF6/18 (Save) to record the changes and remain in the map to do other tasks, or press PF5/17 (File) to record the changes and exit the map.
9. After completing the VMFSGMAP session, use the PUT2PROD EXEC to actually build or delete the saved segments. See “Building or Deleting (Purging) Saved Segments” on page 76.

Adding Saved Segment Definitions for a VMSES/E-Format Product

1. If you are not already in a VMFSGMAP session, enter the following command to display the full Segment Map panel:

```
vmfsgmap segbld esasegs segblist
```

   For detailed information about VMFSGMAP panels, see the description of the VMFSGMAP EXEC in the *z/VM: VMSES/E Introduction and Reference*.
2. On the Segment Map panel, move the cursor so it is not located on any map record.
3. Press PF10/22 (Add Obj) to display the Add Segment Definition panel containing a skeleton saved segment definition record.
4. Move the cursor to the PRODID field and enter the *prodid* (and the *compname*, if one is identified) for the product that defines the saved segment or saved segments you want to add.
5. If you want to add the definition record for only one particular saved segment defined by the product, move the cursor to the OBJNAME field and enter the name of the saved segment. Otherwise, to add all of the saved segments defined by the product, leave the OBJNAME field as question marks (????????).
6. Press PF10/22 (Seginfo). The requested saved segment definitions are obtained from the product's PRODPART file and added to the panel.
7. Make any necessary changes to the saved segment definitions.
8. Press PF5/17 (Map) to return to the Segment Map panel, which is refreshed to include the new saved segments, or press PF12/24 (Cancel) to discard your changes and return to the Segment Map panel.
9. Review the updated segment map to make sure that the new saved segment layout is acceptable.
10. If the new layout is not acceptable, make any necessary changes to the saved segment definitions, or press PF3/15 (Exit) to discard the changes and exit the map.

11. If the new layout is satisfactory, press PF6/18 (Save) to record the changes and remain in the map to do other tasks, or press PF5/17 (File) to record the changes and exit the map.
12. After completing the VMFSGMAP session, use the PUT2PROD EXEC to actually build or delete the saved segments. See "Building or Deleting (Purging) Saved Segments" on page 76.

Adding Saved Segment Definitions for a Product Not in VMSES/E Format

You can use the VMSES/E functions to manage saved segments associated with products that are not packaged in VMSES/E format as long as you make these saved segments known to VMSES/E. Use the following procedure to create the saved segment definition records:

1. Find the installation information for the product that contains details about the saved segments associated with the product.
2. If you are migrating from a 370 VM environment, and the product runs in both your 370 VM system and in z/VM, convert the DMKSNT or override file definitions for the saved segments associated with the product to DEFSEG statements. For more information, see the VM/ESA® V2.4 *Conversion Guide and Notebook*.
3. Enter the following command to display and edit the segment map:

   ```
   vmfsgmap segbld esasegs segblist
   ```

   For detailed information about VMFSGMAP panels, see the description of the VMFSGMAP EXEC in the *z/VM: VMSES/E Introduction and Reference*.
4. If the saved segment you want to add is already defined on the system, move the cursor to the map record for that saved segment. Otherwise, move the cursor so it is not located on any map record.
5. Press PF10/22 to display the Add Segment Definition panel. If the cursor was on a map record when you pressed PF10/22, VMFSGMAP uses the system information to fill in as many fields of the saved segment definition record as it can detect. If the cursor was not on a map record, VMFSGMAP displays a skeleton saved segment definition record.
6. Complete the definition record for the saved segment you are adding:
   - In the DISKS field, lists the minidisks and SFS directories that must be accessed to build the saved segment.
   - In the BLDPARMS field, identify the routine that the VMFBLD EXEC is to call to build the saved segment, or specify 'UNKNOWN' if you want VMFBLD to only issue the DEFSEG command.

   For a complete explanation of the content and syntax of the fields in the saved segment definition record, see the description of the VMFSGMAP EXEC in the *z/VM: VMSES/E Introduction and Reference*.
7. If you defined this saved segment as a member in any segment spaces, press PF6/18 (Chk Mem) to check for overlaps with the other members in all the segment spaces identified in the SPACE field. Definition records for overlapping members are added to the panel.
8. If there are overlaps, press PF11/23 (Adj Mem) to automatically adjust the ranges of the affected members to remove the overlaps.

   Note: If you prefer, you can manually adjust the ranges of the affected members. However, if you manually adjust the range of a member, you should press PF6/18 (Chk Mem) again to make sure that your change has not caused a new overlap.
9. Press PF5/17 (Map) to return to the Segment Map panel, which is refreshed to show the new saved segment and any changes that you made to member saved segments, or press PF12/24 (Cancel) to discard your changes and return to the Segment Map panel.
10. Review the segment map to make sure that the new saved segment layout is acceptable.
11. If the new saved segment conflicts with existing saved segments, you might have to adjust the existing saved segments. See "Changing the Range of a DCSS" on page 63 or "Changing the Range of a Member Saved Segment" on page 63.
12. If you want to stop the addition of the new saved segment, press PF3/15 (Exit) to discard the changes and exit the map.

13. If the new layout is satisfactory and you have more saved segments to define, press PF6/18 (Save) to record the changes and remain in the map. Then go back to step “4” on page 75 and repeat the process for the next saved segment. If you have no more saved segments to define, press PF5/17 (File) to record the changes and exit the map.
14. After completing the VMFSGMAP session, use the PUT2PROD EXEC to actually build or delete the saved segments. See “Building or Deleting (Purging) Saved Segments” on page 76.

Building or Deleting (Purging) Saved Segments

After you finish using the VMFSGMAP EXEC to view and modify saved segment definitions, use the PUT2PROD EXEC to update the system by building or deleting the saved segments.

Notes:

1. VMFBLD processing includes the purging of saved segments that are marked 'DELETED' in the SEGDATA file. Therefore, in this discussion "build" and "built" also mean "delete" and "deleted", where appropriate.
2. You cannot use VMFBLD to build a saved segment unless it is known to VMSES/E (identified in the system saved segment build list and the SEGDATA file). See “Adding Saved Segment Definitions for a VMSES/E-Format Product” on page 74 or “Adding Saved Segment Definitions for a Product Not in VMSES/E Format” on page 75.
3. If a saved segment is known to VMSES/E, but is defined by a product not in VMSES/E format, VMFBLD calls the function specified in the BLDPARMS field of the saved segment definition. This could be a product-defined function or a user-defined function. If the BLDPARMS field contains the word 'UNKNOWN', VMFBLD only issues the DEFSEG command to define the storage requirements to CP. You must issue the function that loads and saves the saved segment.

Displaying the Saved Segment Build Status

Before you build any saved segments, you can display what saved segments are identified to be built by running the VMFBLD EXEC with the STATUS option. Enter the VMFBLD command with the name of the saved segment product parameter file (SEGBLD), the name of the z/VM saved segments "component" in the saved segment product parameter file (ESASEGS), and the name of the system saved segment build list (SEGBLIST):

```
vmfbld ppf segbld esasegs segblist ( status
```

Now you can use the VMFVIEW EXEC to view the build message log. Enter:

```
vmfview build
```

Saved segments to be built are identified as 'SERVICED'. Saved segments to be purged are identified as 'DELETE'. Figure 15 on page 77 shows an example of the display showing segments with a status of SERVICED.

```
    ===> VMFVIEW - Message Log Browse of $VMFBLD  $MSGLOG  A1 <===
You are viewing ¬ST: messages from the LAST run.
Number of messages shown = 11 <===> Number of messages not shown = 57
*************************************************************************
****    PPFNAME: SEGBLD     COMPNAME: ESASEGS    BLDID: VM          ****
*************************************************************************
****            Date: 10/24/09           Time: 13:20:50            ****
*************************************************************************
BD:VMFBLD2180I There are 9 build requirements remaining
BD:VMFBLD2180I Build Requirements:
BD:            Bldlist   Object             Status
BD:VMFBLD2180I SEGBLIST  CMSPIPES.SEGMENT   SERVICED
BD:VMFBLD2180I           INSTSEG.SEGMENT    SERVICED
BD:VMFBLD2180I           SVM.SEGMENT        SERVICED


 1=Help       2=All        3=Quit       4=Exception  5=Status     6=Build
 7=Backward   8=Forward    9=OutCompRq 10=Non-Stat  11=Requisite 12=Severe
====>
```

Figure 15. Example of the VMFVIEW EXEC Display Showing Saved Segments to Be Built

Using the PUT2PROD EXEC to Build or Delete Saved Segments

After you have verified what saved segments are identified to be built, use the PUT2PROD EXEC to do the builds.

The format of the PUT2PROD EXEC depends on whether you want to automatically build all your saved segments, build only those saved segments that are identified to be built, or build an individual saved segment.

In an SSI cluster, the PUT2PROD command must be run on each member.

Notes:

1. For segment deletion the segment status must be reset to DELETE on each additional member on which the product is installed. To change the status of the segment to DELETE, before you specify the PUT2PROD command enter the following VMFSIM command for each segment that was deleted:

   ```
   vmfsim modify vm srvblds d tdata :bldlist segblist :object segname.segment :stat
   delete
   ```

2. To build only those saved segments identified as "SERVICED," enter the PUT2PROD command with those segments listed. For example:

   ```
   put2prod segments csmpipes instseg svm
   ```

3. To build a specific saved segment, whether or not it is identified to be built, enter the PUT2PROD command with the name of the saved segment. For example, to build only the CMSPIPES saved segment listed in the SEGBLIST system saved segment build list, enter:

   ```
   put2prod segments cmspipes
   ```

4. To build all the saved segments identified in system saved segment build list SEGBLIST, whether or not they are identified to be built, enter the VMFBLD command with the ALL option:

   ```
   put2prod segments all
   ```

Checking the Saved Segment Build Messages

After the PUT2PROD EXEC completes its processing, you should check the build message log for error, warning, and informational messages. For example, the log might identify saved segments that VMFBLD

could not build, or indicate that the system segment identification file has been updated. To retrieve the build messages, run the VMFVIEW EXEC:

```
vmfview put2prod
```

Saved Segments That VMFBLD Cannot Build

During build processing, if VMFBLD encounters a definition record in the SEGDATA file that contains the word 'UNKNOWN' in the BLDPARMS field, VMFBLD cannot build the saved segment. VMFBLD can only issue the DEFSEG command to define the saved segment's storage range to CP. VMFBLD then issues a message to warn you about the situation and continues processing the next saved segment.

A situation like this might occur for a saved segment that requires some kind of interactive response from you during the build. After VMFBLD completes its processing, you must manually issue the routine or routines that load and save the saved segment. To find out what saved segments you have to build manually, look at the VMFVIEW panel for occurrences of message VMF2005W. Then follow the instructions in the product documentation to load and save each saved segment.

Copying the SYSTEM SEGID File to the CMS System Disk

The system segment identification file, SYSTEM SEGID, associates logical saved segments to the physical saved segments in which they reside. This file must reside on the CMS system (S) disk. The VMFBLD EXEC copies the SYSTEM SEGID file to the build disk before issuing the SEGGEN command, and SEGGEN updates the file on the build disk. If the file does not already exist, SEGGEN creates it on the build disk.

PUT2PROD copies the SYSTEM SEGID file to the system disk if necessary.

The SYSTEM SEGID file *is not* copied if you:

- Modified the contents of an existing logical saved segment (added, deleted, or changed data)
- Moved an existing physical saved segment *without* changing its contents (in other words, you did not add or delete a logical saved segment).

The SYSYTEM SEGID file *is* copied if you:

- Created a new logical or physical saved segment
- Deleted an existing logical or physical saved segment
- Changed the relationship between the logical and physical saved segments (for example, if you moved or copied a logical saved segment from one physical saved segment to another).

Restoring Saved Segments That Have Been Backed Up on Disk by the CP DCSSBKUP Utility

You can use the VMSES/E saved segment support to restore saved segments that were backed up on disk by the CP DCSSBKUP utility. To define a restored saved segment layout, you need to perform the following steps:

1. Create a $PPF override file that contains the :APPID value, :BLDID value, and system saved segment build list name for the restored layout.

   Figure 16 on page 79 shows an example of an override file called SEGRSAVE $PPF that identifies:

   - A component name of ESARSAVE
   - An :APPID value of VMRSAVE
   - A :BLDID value of VMRSAVE
   - A system saved segment build list (and SEGDATA file) name of SEGRSAVE.

```
*============================================================
*  Override file for saved segment layout ESARSAVE
*            NOTE: All tags must be in upper case.
*============================================================
*============================================================
* Start of Product Header - List of Segment Build Components
*============================================================
:OVERLST. ESARSAVE
*============================================================
* End Product Header
*============================================================
*
* Segment Build Overrides
*
:ESARSAVE. ESASEGS SEGBLD
*============================================================
* Control Parameters
*============================================================
:CNTRLOP.
* TAG      VALUE(S)
*--------- ---------
:APPID.    VMRSAVE              * File name of service
                                * apply status table
:BLDID.    VMRSAVE              * File name of service
                                * build status table
:ECNTRLOP.
*============================================================
* BUILD  section
*============================================================
:BLD. REPLACE
* BUILDLIST EXEC     TARGET   DESCRIPTION
* --------- -------- -------  ------------
SEGRSAVE    VMFBDSEG BUILD    * Segment build test layout
:EBLD.

:END.
*
*============================================================
```

Figure 16. Example of a $PPF Override File for Restoring Saved Segments Backed Up by the CP DCSSBKUP Utility

2. Create the system saved segment build list and SEGDATA file for the restored layout by copying the original system saved segment build list and SEGDATA file to the new name:

```
copyfile segblist exc00000 d segrsave = =
copyfile segblist segdata d segrsave = =
```

3. Run the VMFPPF EXEC to create a new PPF file containing the overrides:

```
vmfppf segrsave esarsave
```

4. Use the VMFSGMAP EXEC to process the alternate system saved segment build list and SEGDATA file:

```
vmfsgmap segrsave esarsave segrsave
```

5. Change the BLDPARMS field in each saved segment definition to invoke the CP DCSSRSAV utility:

```
BLDPARMS...: PROD( DCSSRSAV &SEGNAME )
```

6. Access the disk that contains the saved segment files as file mode A. If the files are not all on the same disk, copy them over.
7. Run the VMFBLD EXEC to restore the saved segments:
   - To restore all of the saved segments, enter:

     ```
     vmfbld ppf segrsave esarsave segrsave ( all
     ```

   - To restore a specific saved segment, enter:

     ```
     vmfbld ppf segrsave esarsave segrsave segname ( all
     ```

Appendix A. Defining CP Saved Segments—Examples

This appendix gives examples of using the DEFSEG and SAVESEG commands to define and save CP saved segments. These examples are provided to help you understand the function of the DEFSEG command.

However, if you are using VMSES/E to manage saved segments on your system, you do not use the DEFSEG and SAVESEG commands directly. Instead, you use the VMFSGMAP EXEC to modify or create definitions for the saved segments and enter the DEFSEG information into the DEFPARMS and SPACE fields of the definitions. You then use the PUT2PROD EXEC to build the saved segments, which calls VMFBLD to issue the DEFSEG and SAVESEG commands for you. See “Changing, Adding, and Deleting Saved Segment Definitions” on page 62.

The examples in this appendix show the DEFSEG syntax for:

- Defining a saved segment with both shared and exclusive page ranges
- Defining overlaid saved segments
- Defining segment spaces
- Defining overlaid segment spaces
- Adding a member to an existing segment space
- Replacing an existing member of a segment space
- Setting up a typical saved segment layout.

Page ranges shown for program products are approximate. Use the actual number of pages required by the release, modification, and service level you wish to run.

The following command sequence shows how to define and save each saved segment illustrated in Figure 2 on page 4:

1. Define each saved segment:

   ```
   defseg ppd 500-550 sr
   defseg ppc 600-650 sr
   defseg ppb 700-7cf sr
   defseg ppa 800-830 sr
   ```

   You should consider including these DEFSEG commands in an exec; if you have to make changes to your storage layout, it will be relatively easy to change the exec accordingly.
2. Install the programs PPD, PPC, PPB, and PPA with the appropriate installation execs.
3. For each saved segment you have defined, enter the SAVESEG command to save the saved segment (if the installation execs have not already done so):

   ```
   saveseg ppd
   saveseg ppc
   saveseg ppb
   saveseg ppa
   ```

Defining a Saved Segment with Both Shared and Exclusive Page Ranges

Some licensed programs, such as OfficeVision®, have both shared and exclusive code and therefore require a shared segment and an exclusive segment. You must define these types of licensed programs to have both shared and exclusive page ranges.

For example, to define OfficeVision to have a range of addresses that is shared and a range that is exclusive:

1. Enter:

```
defseg ovvm 600-7ff sr 800-845 ew
```

In the previous example, the address range 600–7FF is defined as *shared read-only*; the address range 800–845 is defined as *exclusive read-write*.

2. Install OfficeVision with the appropriate installation exec.
3. If it has not already been done by the installation exec, enter the SAVESEG command for OfficeVision:

```
saveseg ovvm
```

In the previous example, OfficeVision is defined in a DCSS. If OfficeVision and another licensed program are always used together, you may want to define both in the same segment space, rather than each in a separate DCSS. This reclaims some of the storage that goes unused when you define a DCSS.

Defining Overlaid DCSSs

The following sequence of commands shows how to define and save each overlaid DCSS illustrated in Figure 9 on page 28.

1. Define each DCSS in the same 1 MB address range:

```
defseg ppt3 700-750 sr
defseg ppt4 700-750 sr
defseg ppt5 700-780 sr
```

2. Install the program PPT3 with the appropriate installation exec.
3. Enter the SAVESEG command to save the DCSS (if the installation execs have not already done so):

```
saveseg ppt3
```

4. Repeat steps 2 and 3 for program PPT4.
5. Repeat steps 2 and 3 for program PPT5.

Defining a Segment Space

Example 1

The following sequence of commands shows how to define and save the segment space shown residing in SPACE2 in Figure 11 on page 30:

1. Define each member of the segment space SPACE2:

```
defseg ppk 700-750 sr space space2
defseg ppl 751-7a0 sr space space2
defseg ppm 7a1-820 sr space space2
defseg ppn 821-8a0 sr space space2
defseg ppo 8a1-920 sr space space2
```

You should consider including these DEFSEG commands in an exec. If you have to make changes to your storage layout, it is relatively easy to change the exec accordingly.

2. Install the programs PPK, PPL, PPM, PPN, and PPO with the appropriate installation execs.
3. For each member you have defined, enter the SAVESEG command to save the segment (if the installation execs have not already done so):

```
saveseg ppk
saveseg ppl
saveseg ppm
saveseg ppn
saveseg ppo
```

Example 2

The following example defines a segment space for two applications, PPK and PPL, each of which has both exclusive code and shared code. Remember that exclusive code must be in a separate architected segment from shared code.

1. Define each member of SPACE1:

```
defseg ppk 400-47f ew 500-575 sr space space1
defseg ppl 480-4ff ew 580-5ff sr space space1
```

   The architected segment from 400 to 4FF contains exclusive write code, and the architected segment from 500 to 5FF contains shared read code.

2. Install PPK and PPL with the appropriate installation execs.
3. Enter the corresponding SAVESEG commands:

```
saveseg ppk
saveseg ppl
```

Notes:

1. The ending address of SPACE2 is rounded up to a 1 MB boundary. Therefore, SPACE2 ranges from X'700' to X'9FF'.
2. Any unused space in a segment space can be considered a growth area. Such an area can contain, for example, a new release of a program (currently residing in a segment space) that takes up more storage space than the previous release. For example, if the next release of PPO in Figure 11 on page 30 is larger than the previous release of PPO, you could define it:

```
defseg ppo 8a1-930 sr space space2
```

   and use DEFSEG commands with the SAMERANGE operands for the other members. Thus, only PPO needs to be a saved segment to make the new version of SPACE2 active. The other members are used with their old definitions.

   To avoid having to reinstall other members when one member has grown, you should distribute the growth area (the unused virtual storage space) between all the members. In this example, all members need to be redefined at higher page ranges and reinstalled if PPK grows in size.
3. The segment space SPACE2 is not active until you enter the last SAVESEG command (SAVESEG PPO in this case). Therefore, you cannot use the programs in SPACE2 until you enter the last SAVESEG command.

Defining Overlaid Segment Spaces

The following command sequence shows how to define and save the overlaid segment spaces SPACE1, SPACE2, and SPACE3 illustrated in Figure 8 on page 27.

1. Define each member of SPACE1:

```
defseg sqlrmgr 600-60f sr space space1
defseg sqlisql 610-66f sr space space1
defseg das1v151 670-70f sr space space1
defseg das2v151 710-83f sr space space1
```

   SQLRMGR and SQLISQL are saved segments associated with the SQL user machine; DAS1V151 and DAS2V151 are saved segments associated with AS.

2. Install AS with the appropriate installation exec.
3. Enter the SAVESEG command for the AS segments (if this has not already been done by the installation exec):

```
saveseg das1v151
saveseg das2v151
```

4. Define each member of SPACE2:

```
defseg qmf220e 670-7bf sr space space2
defseg sqlrmgr same space space2
defseg sqlisql same space space2
```

QMF220E is the saved segment associated with QMF.

Note: Do not use the SAME operand for members that have the SW, EW, EN, or SN attribute. Programs in multiple segment spaces must be refreshable.

5. Install QMF with the appropriate installation exec.
6. Define each member of SPACE3:

```
defseg sqlsqlds 600 6cf space space3
defseg sqlxrds  7d0 7a5 space space3
```

SQLSQLDS and SQLXRDS are saved segments associated with the SQL service machine.

7. Install SQL with the appropriate installation exec.
8. Enter the SAVESEG command for the remaining members of SPACE2 and SPACE3 (if the installation exec has not already done so):

```
saveseg qmf220e
saveseg sqlrmgr
saveseg sqlisql
saveseg sqlsqlds
saveseg sqlxrds
```

After you enter the final SAVESEG command, both segment spaces become active, and the SQL, QMF, and AS applications become available to users.

Adding a Member to an Existing Segment Space

To add a member called **PPZ** to the segment space SPACE2 illustrated in Figure 11 on page 30 and defined under “Defining a Segment Space” on page 82:

1. Define the member PPZ:

```
defseg ppz 921-9d0 sr space space2
```

2. Define the rest of the segment space using the SAMERANGE option on the DEFSEG command:

```
defseg ppk same space space2
defseg ppl same space space2
defseg ppm same space space2
defseg ppn same space space2
defseg ppo same space space2
```

3. Install PPZ with the appropriate installation exec.
4. Enter the SAVESEG command for PPZ (unless your installation exec has already done so):

```
saveseg ppz
```

After you enter this SAVESEG command, the program PPZ is available to virtual machines not currently attached to any of the other programs in SPACE2.

Replacing an Existing Member of a Segment Space

When a new release of a licensed program becomes available, you may want to replace your old copy of the program with the new version.

Example 1—Replacing a Member

Suppose you want to replace the version of the program PPL (as shown in Figure 11 on page 30) with a new release of PPL. To do this:

1. Define the new version of PPL as a member of SPACE2:

```
defseg ppl 751-7a0 sr space space2
```

2. Define the rest of SPACE2 using the SAME option on the DEFSEG command:

```
defseg ppk same space space2
defseg ppm same space space2
defseg ppn same space space2
defseg ppo same space space2
defseg ppz same space space2
```

3. Install program PPL with the appropriate installation exec.
4. Enter the SAVESEG command for PPL (unless your installation exec has already done so):

```
saveseg ppl
```

Note that in this example the new version of PPL occupies the same page range as the old version. Because of this, the other members of SPACE2 did not have to be saved again but were merely redefined with the SAME operand on the DEFSEG command.

You may want to keep both releases of PPL available to your users. If so, you should consider defining the two versions of PPL in different segment spaces. Make sure you do not use the same name for both versions.

How System Data Files are Affected

Example 1 works in most cases. The following, more detailed examples, show how system data files are affected when you replace an existing member.

Example 2—Replacing a Member

The segment space SPACE1 has been created, and the system data file environment looks like that illustrated in Figure 17 on page 85. In this figure and others like it in this appendix, the uppermost block shown under the name of the member or segment space is its spool ID. For members, the other blocks indicate the segment space that contains this member. For segment spaces, the other blocks indicate the members of the segment space. In Figure 17 on page 85, for example, SPACE1 has the spool ID 0001 and has M1 (spool ID 0002), M2 (spool ID 0003), and M3 (spool ID 0004) as its members.

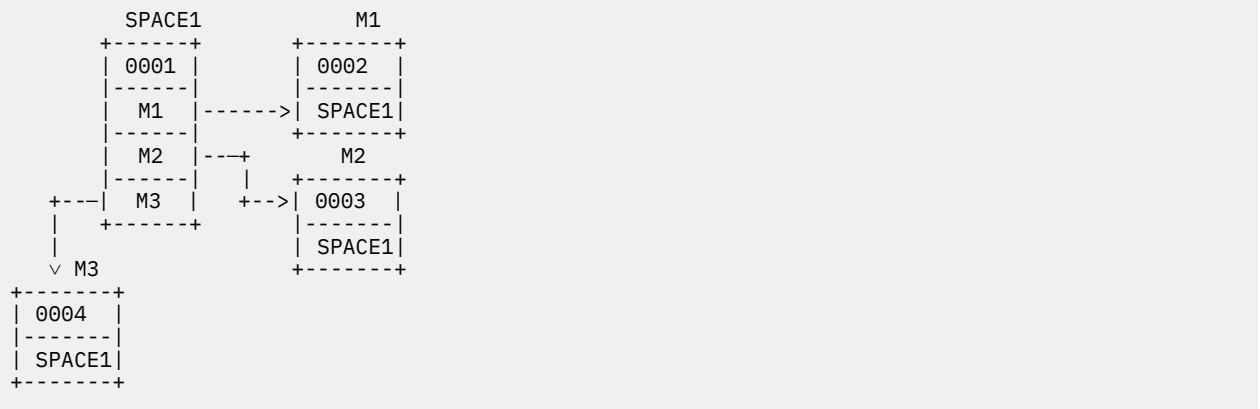


Figure 17. Initial Setup of a Segment Space

To replace M3, define the new version of M3 and SPACE1. You can do this by entering several DEFSEG commands as follows:

```
defseg m3 rangeinfo... space space1
defseg m1 same space space1
defseg m2 same space space1
```

Figure 18 on page 86 shows the situation after these DEFSEG commands have been entered. Note that M3 and SPACE1 have new spool IDs (005 and 006, respectively) which are class S files.

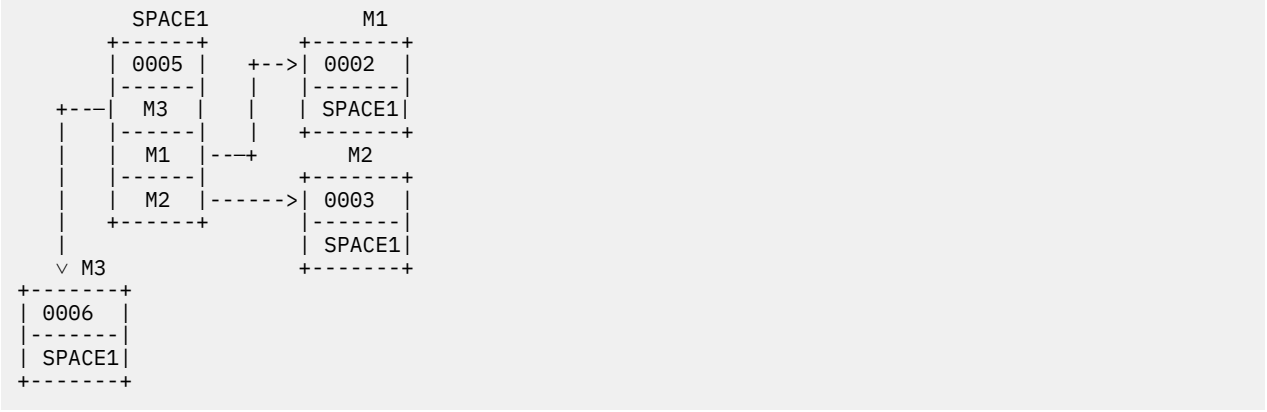


Figure 18. New Version of a Segment Space (DEFSEGs Complete)

Last, you need to enter the SAVESEG command for M3 based on the installation procedures for M3. The SAVESEG M3 command converts spool files 0005 and 0006 to class A files, causing the old class A versions (spool files 0001 and 0004) to be purged.

Example 3—Creating a New Member for an Overlay

This example explains how to create a new version of one member of an overlay.

After the initial segment spaces have been created, the environment looks like that illustrated in Figure 19 on page 86. *L* refers to segment spaces, and *M* refers to members:

- L1 (spool ID 0010) points to M4 (spool ID 0011) and M5 (spool ID 0012).
- L2 (spool ID 0013) points to M4 (spool ID 0011) and M6 (spool ID 0014).
- L3 (spool ID 0015) points to M4 (spool ID 0011) and M7 (spool ID 0016).

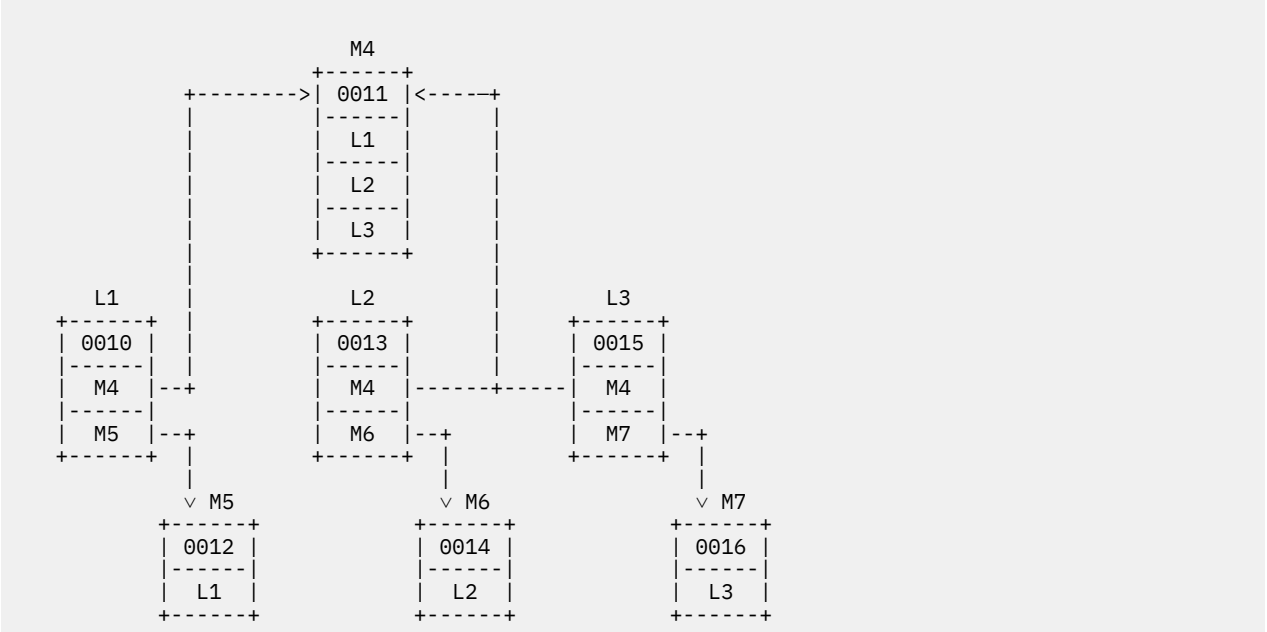


Figure 19. Replacing One Member of an Overlay—Initial Setup

To replace M6, define the new version of M6 and L2 by entering several DEFSEG commands as follows:

```
defseg m6 rangeinfo... space l2
defseg m4 same space l2
```

Figure 20 on page 87 shows the situation after these define commands have been entered. L2 (now spool ID 0017) points to M4 (spool ID 0011) and M6 (now spool ID 00148). Note that spool files 0017 and 0018 are class S files.

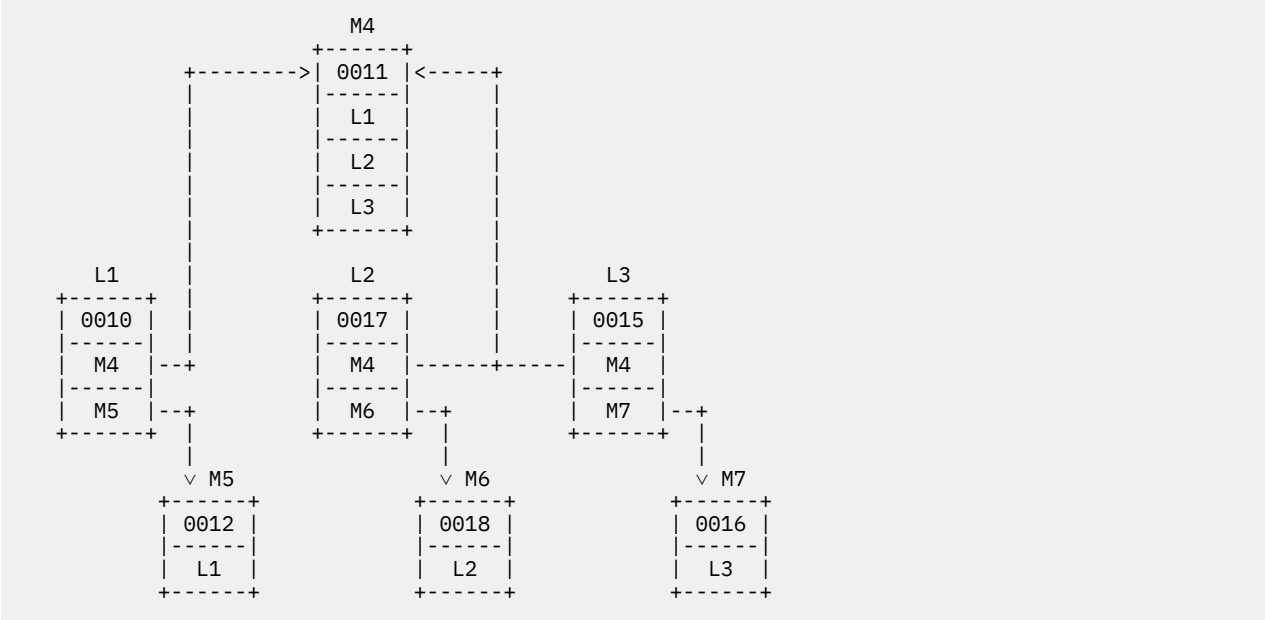


Figure 20. Replacing One Member of an Overlay (DEFSEGs Complete)

Last, you must enter the SAVESEG for M6 as given in the installation procedures for M6. The SAVESEG M6 command converts spool files 0017 and 0018 to class A files, purging the old class A versions.

Example 4— Creating a New Version of a Member

This example shows how to create a new version of a member that several segment spaces share.

After the initial segment spaces have been created, the environment looks like that illustrated in Figure 21 on page 87.

- L10 (spool ID 0101) points to M9 (spool ID 0111) and M10 (spool ID 0112).
- L11 (spool ID 0102) points to M8 (spool ID 0100) and M11 (spool ID 0114).
- L12 (spool ID 0103) points to M8 (spool ID 0100) and M12 (spool ID 0116).

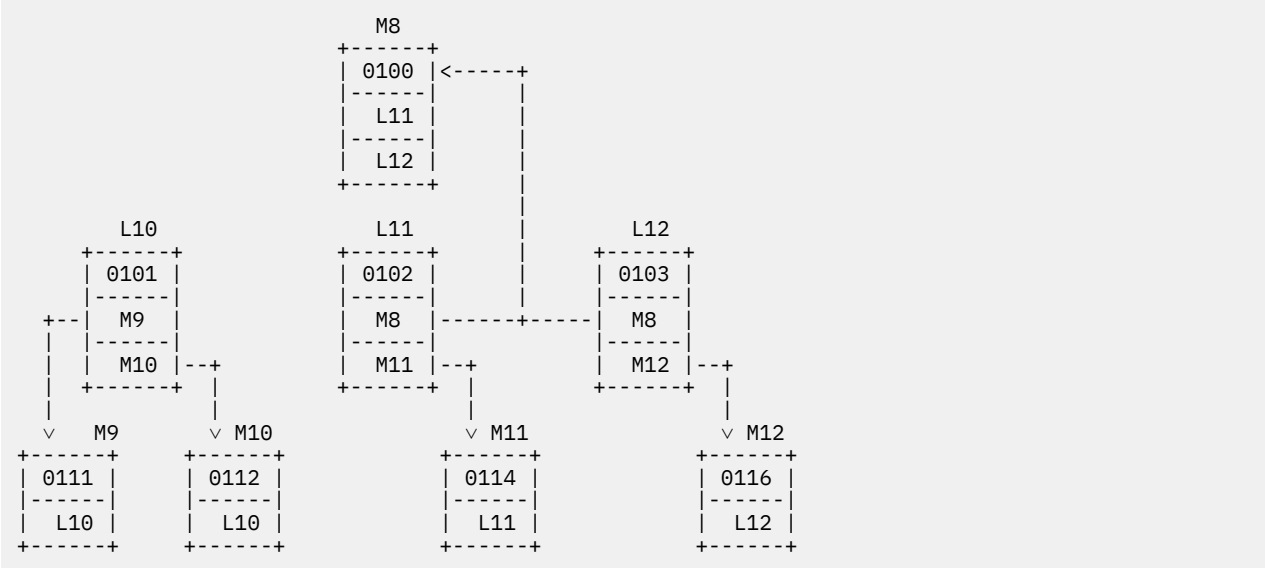


Figure 21. Replacing a Shared Member—Initial Setup

To replace M8, define the new version of M8, L11, and L12 by entering several DEFSEG commands as follows:

```
defseg m8 rangeinfo... space l11
defseg m11 same space l11
defseg m8  same space l12
defseg m12 same space l12
```

Figure 22 on page 88 shows the situation after these define commands have been entered. Note that spool files 0117, 0118, and 0119 are class S files.

- L11 (now spool ID 0117) points to M8 (now spool ID 0118) and M11 (spool ID 0114).
- L12 (now spool ID 0119) points to M8 (now spool ID 0118) and M12 (spool ID 0116).

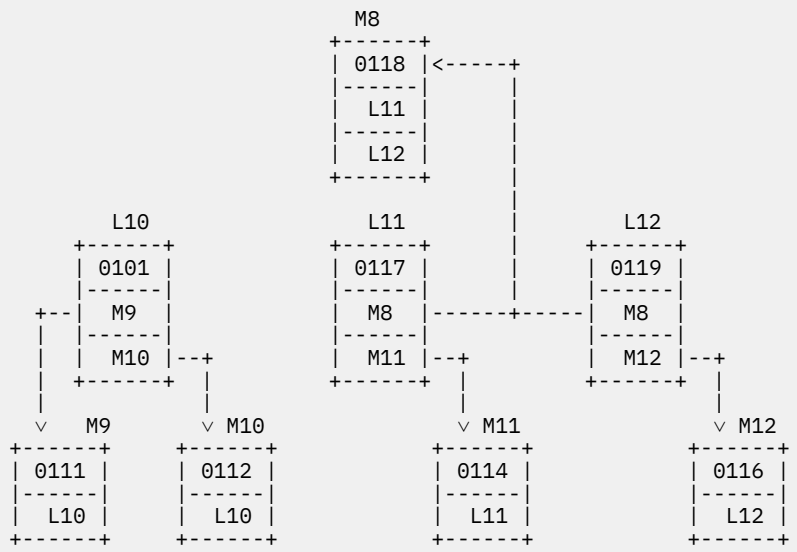


Figure 22. Replacing a Shared Member (DEFSEGs Complete)

Last, you must enter the SAVESEG command for M8 based on the installation procedures for M8. The SAVESEG M8 command converts spool files 0117, 0118, and 0119 to Class A files, purging the old class A versions.

Setting Up Your Storage Layout

The following examples are storage layouts for a given group of applications. Please note that these mappings are only examples and may not work successfully for every installation. They should, however, provide you with some ideas on how to set up your own saved segment environment.

Example 1—A Sample Storage Layout

The applications shown in Figure 23 on page 89 are CMS, OfficeVision, GDDM, GDDM/PGF, DisplayWrite®/370 (DW/370), SQL, Query Management Facility™ (QMF), FORTRAN, VMAS, Document Composition Facility (DCF) and APL2®.

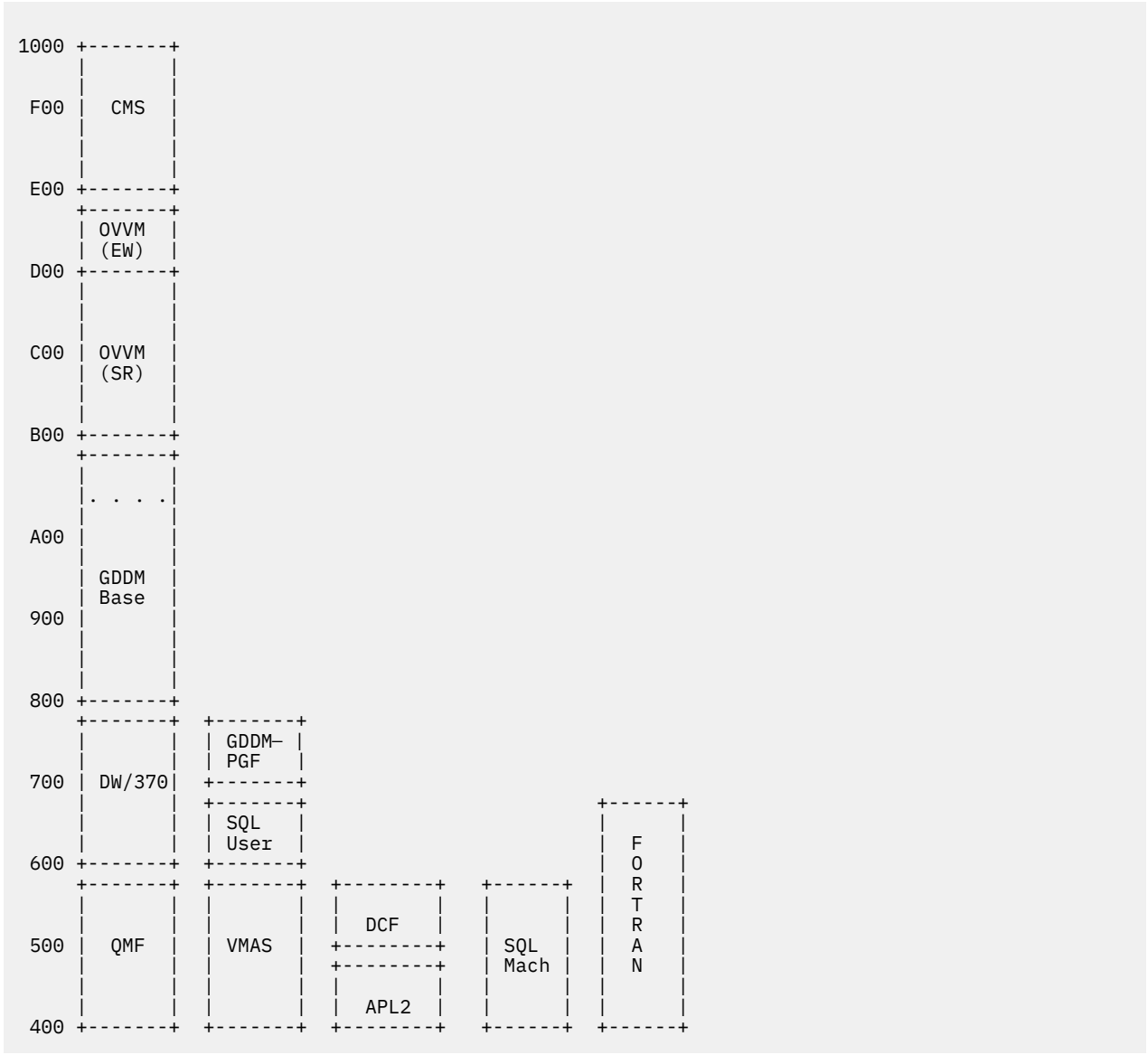


Figure 23. A Typical Saved Segment Environment—Example 1

In Figure 23 on page 89, the two segments that make up the SQL user machine are SQLISQL and SQLRMGR. The two segments that make up the SQL service machine are SQLSQLDS and SQLXRDS.

Example 2—A Sample Storage Layout

The applications shown in Figure 24 on page 90 are CMS, OfficeVision, The Information Facility Program Offering (TIF), Group Control System (GCS), GDDM, SQL, DW/370, QMF, DCF, and VTAM®.

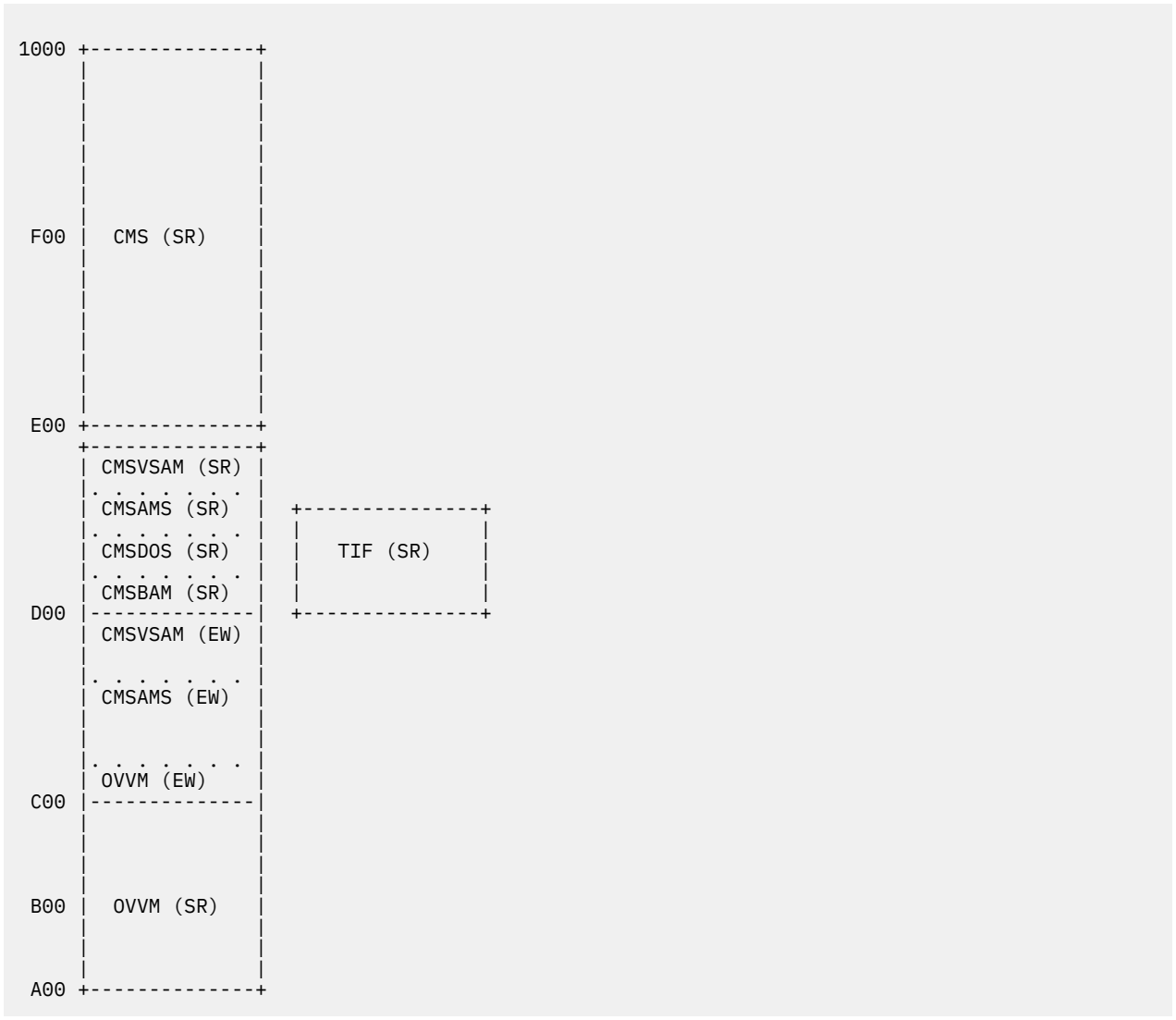


Figure 24. A Typical Saved Segment Environment—Example 2 (1 of 2)

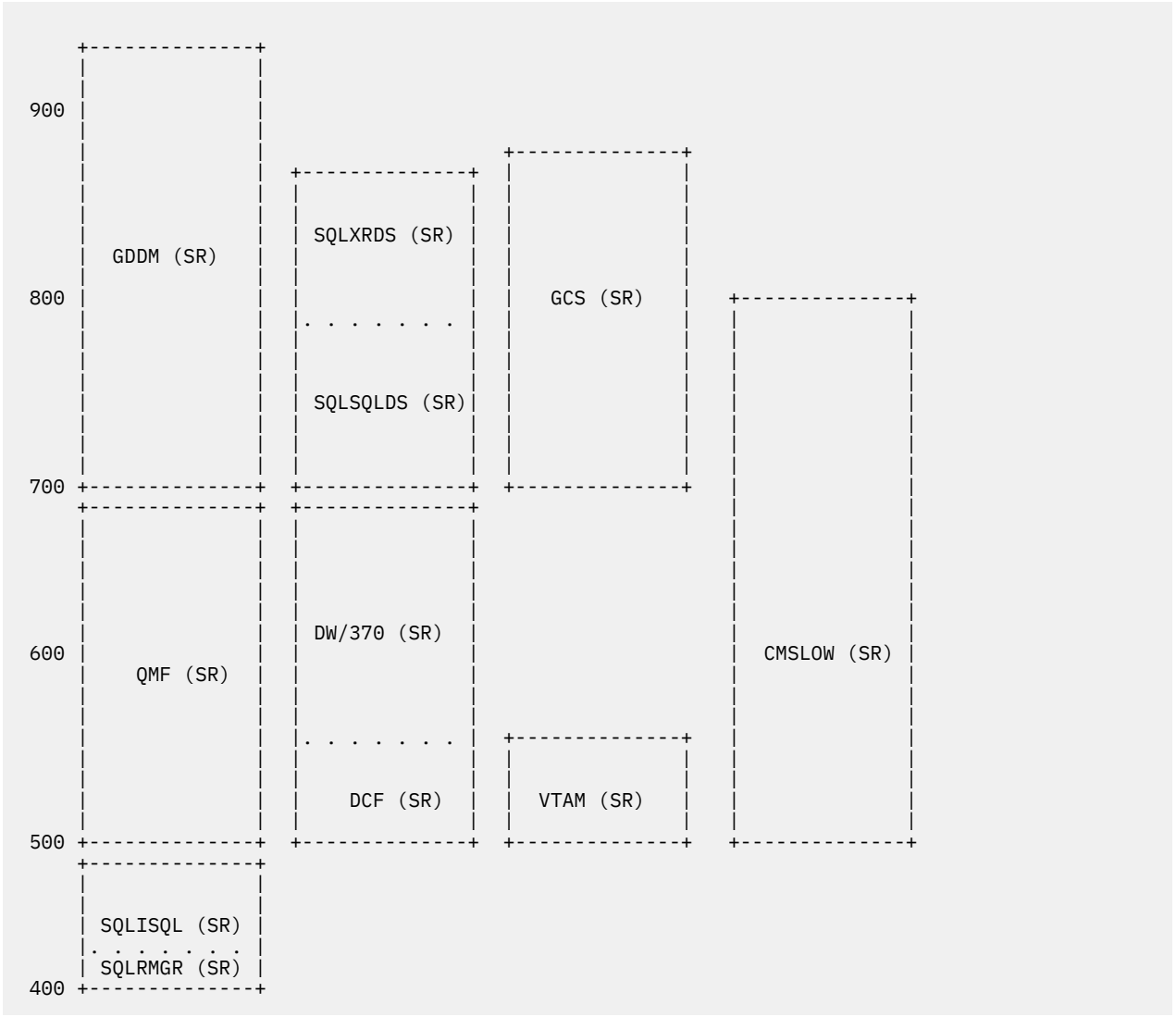


Figure 25. A Typical Saved Segment Environment—Example 2 (2 of 2)

In Figure 24 on page 90:

- To make segment D (the default location of CMS) available for other applications, CMS is defined at a secondary location as a named saved system called *CMSLOW*.
- The two segments that make up the SQL user machine are SQLISQL and SQLRMGR. The two segments that make up the SQL service machine are SQLSQLDS and SQLXRDS.

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.

The performance data and client examples cited are presented for illustrative purposes only. Actual performance results may vary depending on specific configurations and operating conditions.

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 may contain 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 may contain 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 primarily documents information that is NOT intended to be used as Programming Interfaces of z/VM.

This publication also documents intended Programming Interfaces that allow the customer to write programs to obtain the services of z/VM. This information is identified where it occurs, either by an introductory statement to a chapter or section or by the following marking:

PI

<...Programming Interface information...>

PI end

Trademarks

IBM, the IBM logo, and ibm.com® are trademarks or registered trademarks of International Business Machines Corp., in the United States and/or other countries. Other product and service names might be trademarks of IBM or other companies. A current list of IBM trademarks is available on IBM Copyright and trademark information (https://www.ibm.com/legal/copytrade).

The registered trademark Linux® is used pursuant to a sublicense from the Linux Foundation, the exclusive licensee of Linus Torvalds, owner of the mark on a worldwide basis.

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.

IBM Online Privacy Statement

IBM Software products, including software as a service solutions, ("Software Offerings") may use cookies or other technologies to collect product usage information, to help improve the end user experience, to tailor interactions with the end user, or for other purposes. In many cases no personally identifiable information is collected by the Software Offerings. Some of our Software Offerings can help enable you to collect personally identifiable information. If this Software Offering uses cookies to collect personally identifiable information, specific information about this offering's use of cookies is set forth below.

This Software Offering does not use cookies or other technologies to collect personally identifiable information.

If the configurations deployed for this Software Offering provide you as customer the ability to collect personally identifiable information from end users via cookies and other technologies, you should seek your own legal advice about any laws applicable to such data collection, including any requirements for notice and consent.

For more information about the use of various technologies, including cookies, for these purposes, see:

- The section entitled **IBM Websites** at IBM Privacy Statement (https://www.ibm.com/privacy)
- Cookies and Similar Technologies (https://www.ibm.com/privacy#Cookies_and_Similar_Technologies)

Bibliography

This topic lists the publications in the z/VM library. For abstracts of the z/VM publications, see *z/VM: General Information*.

Where to Get z/VM Information

The current z/VM product documentation is available in IBM Documentation - z/VM (https://www.ibm.com/docs/en/zvm).

z/VM Base Library

Overview

- *z/VM: License Information*, GI13-4377
- *z/VM: General Information*, GC24-6286

Installation, Migration, and Service

- *z/VM: Installation Guide*, GC24-6292
- *z/VM: Migration Guide*, GC24-6294
- *z/VM: Service Guide*, GC24-6325
- *z/VM: VMSES/E Introduction and Reference*, GC24-6336

Planning and Administration

- *z/VM: CMS File Pool Planning, Administration, and Operation*, SC24-6261
- *z/VM: CMS Planning and Administration*, SC24-6264
- *z/VM: Connectivity*, SC24-6267
- *z/VM: CP Planning and Administration*, SC24-6271
- *z/VM: Getting Started with Linux on IBM Z*, SC24-6287
- *z/VM: Group Control System*, SC24-6289
- *z/VM: I/O Configuration*, SC24-6291
- *z/VM: Running Guest Operating Systems*, SC24-6321
- *z/VM: Saved Segments Planning and Administration*, SC24-6322
- *z/VM: Secure Configuration Guide*, SC24-6323

Customization and Tuning

- *z/VM: CP Exit Customization*, SC24-6269
- *z/VM: Performance*, SC24-6301

Operation and Use

- *z/VM: CMS Commands and Utilities Reference*, SC24-6260
- *z/VM: CMS Primer*, SC24-6265
- *z/VM: CMS User's Guide*, SC24-6266
- *z/VM: CP Commands and Utilities Reference*, SC24-6268

- *z/VM: System Operation*, SC24-6326
- *z/VM: Virtual Machine Operation*, SC24-6334
- *z/VM: XEDIT Commands and Macros Reference*, SC24-6337
- *z/VM: XEDIT User's Guide*, SC24-6338

Application Programming

- *z/VM: CMS Application Development Guide*, SC24-6256
- *z/VM: CMS Application Development Guide for Assembler*, SC24-6257
- *z/VM: CMS Application Multitasking*, SC24-6258
- *z/VM: CMS Callable Services Reference*, SC24-6259
- *z/VM: CMS Macros and Functions Reference*, SC24-6262
- *z/VM: CMS Pipelines User's Guide and Reference*, SC24-6252
- *z/VM: CP Programming Services*, SC24-6272
- *z/VM: CPI Communications User's Guide*, SC24-6273
- *z/VM: ESA/XC Principles of Operation*, SC24-6285
- *z/VM: Language Environment User's Guide*, SC24-6293
- *z/VM: OpenExtensions Advanced Application Programming Tools*, SC24-6295
- *z/VM: OpenExtensions Callable Services Reference*, SC24-6296
- *z/VM: OpenExtensions Commands Reference*, SC24-6297
- *z/VM: OpenExtensions POSIX Conformance Document*, GC24-6298
- *z/VM: OpenExtensions User's Guide*, SC24-6299
- *z/VM: Program Management Binder for CMS*, SC24-6304
- *z/VM: Reusable Server Kernel Programmer's Guide and Reference*, SC24-6313
- *z/VM: REXX/VM Reference*, SC24-6314
- *z/VM: REXX/VM User's Guide*, SC24-6315
- *z/VM: Systems Management Application Programming*, SC24-6327
- *z/VM: z/Architecture Extended Configuration (z/XC) Principles of Operation*, SC27-4940

Diagnosis

- *z/VM: CMS and REXX/VM Messages and Codes*, GC24-6255
- *z/VM: CP Messages and Codes*, GC24-6270
- *z/VM: Diagnosis Guide*, GC24-6280
- *z/VM: Dump Viewing Facility*, GC24-6284
- *z/VM: Other Components Messages and Codes*, GC24-6300
- *z/VM: VM Dump Tool*, GC24-6335

z/VM Facilities and Features

Data Facility Storage Management Subsystem for z/VM

- *z/VM: DFSMS/VM Customization*, SC24-6274
- *z/VM: DFSMS/VM Diagnosis Guide*, GC24-6275
- *z/VM: DFSMS/VM Messages and Codes*, GC24-6276
- *z/VM: DFSMS/VM Planning Guide*, SC24-6277

- *z/VM: DFSMS/VM Removable Media Services*, SC24-6278
- *z/VM: DFSMS/VM Storage Administration*, SC24-6279

Directory Maintenance Facility for z/VM

- *z/VM: Directory Maintenance Facility Commands Reference*, SC24-6281
- *z/VM: Directory Maintenance Facility Messages*, GC24-6282
- *z/VM: Directory Maintenance Facility Tailoring and Administration Guide*, SC24-6283

Open Systems Adapter

- Open Systems Adapter/Support Facility on the Hardware Management Console (https://www.ibm.com/docs/en/SSLTBW_2.3.0/pdf/SC14-7580-02.pdf), SC14-7580
- Open Systems Adapter-Express ICC 3215 Support (https://www.ibm.com/docs/en/zos/2.3.0?topic=osa-icc-3215-support), SA23-2247
- Open Systems Adapter Integrated Console Controller User's Guide (https://www.ibm.com/docs/en/SSLTBW_2.3.0/pdf/SC27-9003-02.pdf), SC27-9003
- Open Systems Adapter-Express Customer's Guide and Reference (https://www.ibm.com/docs/en/SSLTBW_2.3.0/pdf/ioa2z1f0.pdf), SA22-7935

Performance Toolkit for z/VM

- *z/VM: Performance Toolkit Guide*, SC24-6302
- *z/VM: Performance Toolkit Reference*, SC24-6303

The following publications contain sections that provide information about z/VM Performance Data Pump, which is licensed with Performance Toolkit for z/VM.

- *z/VM: Performance*, SC24-6301. See z/VM Performance Data Pump.
- *z/VM: Other Components Messages and Codes*, GC24-6300. See Data Pump Messages.

RACF® Security Server for z/VM

- *z/VM: RACF Security Server Auditor's Guide*, SC24-6305
- *z/VM: RACF Security Server Command Language Reference*, SC24-6306
- *z/VM: RACF Security Server Diagnosis Guide*, GC24-6307
- *z/VM: RACF Security Server General User's Guide*, SC24-6308
- *z/VM: RACF Security Server Macros and Interfaces*, SC24-6309
- *z/VM: RACF Security Server Messages and Codes*, GC24-6310
- *z/VM: RACF Security Server Security Administrator's Guide*, SC24-6311
- *z/VM: RACF Security Server System Programmer's Guide*, SC24-6312
- *z/VM: Security Server RACROUTE Macro Reference*, SC24-6324

Remote Spooling Communications Subsystem Networking for z/VM

- *z/VM: RSCS Networking Diagnosis*, GC24-6316
- *z/VM: RSCS Networking Exit Customization*, SC24-6317
- *z/VM: RSCS Networking Messages and Codes*, GC24-6318
- *z/VM: RSCS Networking Operation and Use*, SC24-6319
- *z/VM: RSCS Networking Planning and Configuration*, SC24-6320

TCP/IP for z/VM

- *z/VM: TCP/IP Diagnosis Guide*, GC24-6328
- *z/VM: TCP/IP LDAP Administration Guide*, SC24-6329
- *z/VM: TCP/IP Messages and Codes*, GC24-6330
- *z/VM: TCP/IP Planning and Customization*, SC24-6331
- *z/VM: TCP/IP Programmer's Reference*, SC24-6332
- *z/VM: TCP/IP User's Guide*, SC24-6333

Prerequisite Products

Device Support Facilities

- Device Support Facilities (ICKDSF): User's Guide and Reference (https://www.ibm.com/docs/en/SSLTBW_2.5.0/pdf/ickug00_v2r5.pdf), GC35-0033

Related Products

XL C++ for z/VM

- *XL C/C++ for z/VM: Runtime Library Reference*, SC09-7624
- *XL C/C++ for z/VM: User's Guide*, SC09-7625

z/OS

IBM Documentation - z/OS (https://www.ibm.com/docs/en/zos)

Index

T

U

V

Z

Product Number: 5741-A09

Printed in USA

SC24-6322-74