IBM

Virtual Machine/Extended Architecture™
System Product

SC23-0457-0

Guide to Saved Segments

VM/XA™ SP Release 2

IBM

Virtual Machine/Extended Architecture™ System Product

SC23-0457-0

Guide to Saved Segments

VM/XA™ SP Release 2

First Edition (June 1989)

This edition applies to Release 2 of the Virtual Machine/Extended Architecture System Product (VM/XA SP) Licensed Program 5664-308. Changes are made periodically to the information herein; before using this publication in connection with the operation of IBM systems, consult the latest *IBM System/370, 30xx, 4300, and 9370 Processors Bibliography*, GC20-0001, for the editions that are applicable and current.

Publications are not stocked at the address given below. Requests for IBM publications should be made to your IBM representative or to the IBM branch office serving your locality.

A form for readers' comments is provided at the back of this publication. If the form has been removed, comments may be addressed to International Business Machines, Department 52Q/MS 511, Neighborhood Road, Kingston, N.Y. 12401. IBM may use or distribute whatever information you supply in any way it believes appropriate without incurring any obligation to you.

Preface

Saved segments allow you to make efficient use of storage and improve virtual machine performance.

Audience

This manual is for:

- System administrators responsible for planning the migration of programs that operate in a saved segment from a VM/SP (or VM/SP HPO) environment to a VM/XA SP environment.
- System programmers responsible for installing licensed programs or other applications in saved segments.
- Application programmers responsible for developing programs that operate from saved segments.

Prerequisite Knowledge

Before reading this book, you should be somewhat familiar with VM/XA SP, and you should read *VM/XA SP General Information*, GC23-0362.

Related Publications

VM/XA System Product:

Licensed Program Specifications, GC23-0366

Features Summary, LY27-8058

CP Command Reference, SC23-0358

Real System Operation, SC23-0371

Planning and Administration, GC23-0378

CP Diagnosis Reference, LY27-8054

CP Programming Services, SC23-0370

CMS Application Program Development Guide, SC23-0355

CMS Application Program Development Reference, SC23-0402

CMS Application Program Conversion Guide, SC23-0403

Contents

Chapter 1. Introduction to Saved Segments

This chapter gives you an overview of saved segments. You may not need to read this if you are already familiar with segment terminology.

This chapter tells you:

- What a segment is
- What a saved segment is
- Why saved segments are useful
- How to create saved segments
- The different types of saved segments you can define.

A **segment** (also called an "architected segment") is a 1-megabyte portion of real storage defined by 370-XA architecture.

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

Why Use Saved Segments?

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

- Because several users can access the same saved segment, 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 be outside its virtual storage. This allows the virtual storage of the virtual machine to be used 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 it is loaded using a DIAGNOSE X'64' rather than an IPL.

Using Saved Segments – An Overview

The following list 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. To do this:
   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 issue 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 a 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 Chapter 3, "System Programmer Considerations" on page 21.
3. Load the saved segment into a virtual machine. To use a saved segment in CMS:
   - Use the SEGMENT LOAD command or SEGMENT LOAD macro to load the saved segment into storage. (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 issue the SEGMENT LOAD command.)
   - Or, use the CP interface, DIAGNOSE X'64', to load the code or data into the saved segment.

   The CMS SEGMENT command and macro provide a CMS interface to the DIAGNOSE X'64' instruction.

   For more information on accessing saved segments, see "Using Saved Segments from Your Virtual Machine" on page 55.

Notes:

1. An application programmer generally:
   - Creates the code or data that resides in a saved segment
   - Provides code in the form of an installation EXEC which 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 which invokes either SEGMENT LOAD or DIAGNOSE X'64'.
2. 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

The types of saved segments in Virtual Machine/Extended Architecture™ System Product (VM/XA™ SP) are **discontiguous saved segments, saved segment spaces,** and **member saved segments.**

A **discontiguous saved segment (DCSS)** is a saved segment that occupies one or more architected segments. A DCSS begins and ends on a megabyte boundary, and is accessed by name. Figure 1 shows several DCSSs defined in the 5-megabyte to 9-megabyte 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.

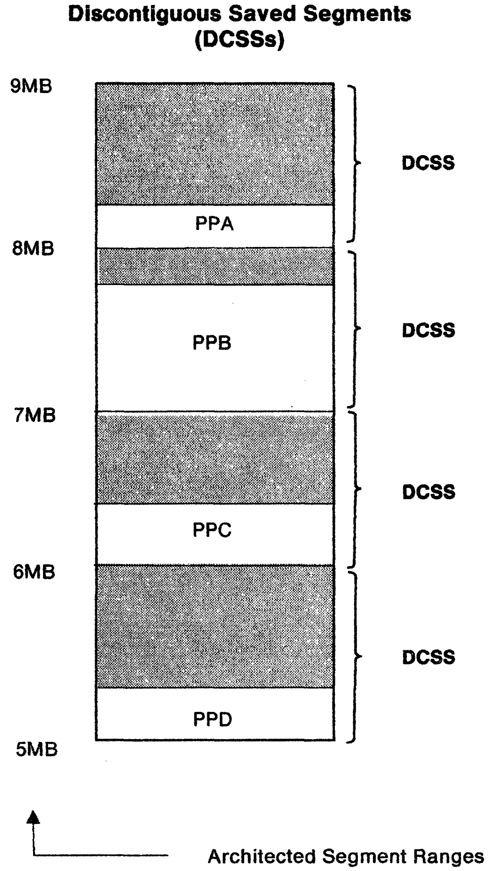


Figure 1. Discontiguous Saved Segments

A **segment space** is a saved segment composed of up to 64 member saved segments referred to by a single name. A segment space occupies one or more architected segments. It begins and ends on megabyte boundaries. A user with access to a segment space has access to all of its members.

A **member saved segment** is a saved segment that begins and ends on a page boundary. It belongs to up to 64 segment spaces and is accessed either by its own

Virtual Machine/Extended Architecture and VM/XA are trademarks of the International Business Machines Corporation.

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 2 shows a segment space defined in the 5-megabyte to 8-megabyte range of architected segments. This segment space contains several member saved segments, which are used to hold applications.

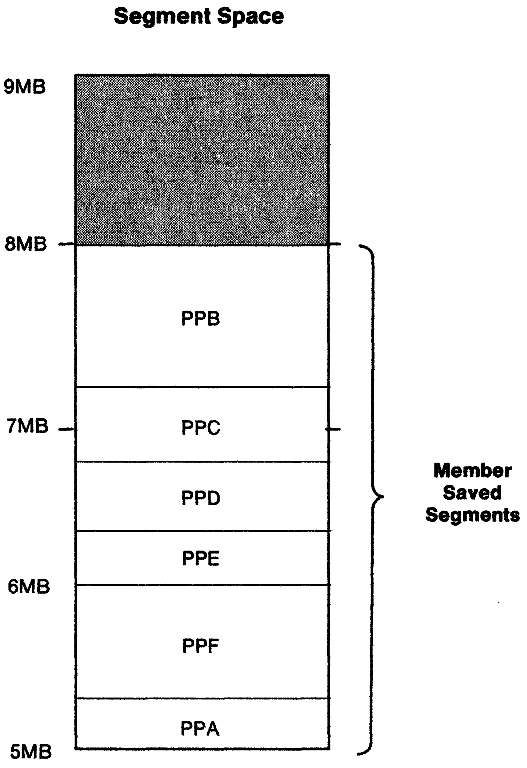


Figure 2. Member Saved Segments

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 **discontiguous saved segment** (DCSS). A DCSS is at least one megabyte in size. The beginning address of a DCSS is rounded down to a megabyte boundary, and the ending address is rounded up to a megabyte boundary. *Only one application can reside in each DCSS.*

By including the SPACE operand on the DEFSEG command, you define a **segment space**. Like a DCSS, the beginning address of a segment space is rounded down to a megabyte boundary, and the ending address is rounded up to a megabyte boundary. A segment space is at least one megabyte in size.

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

- Segment spaces allow different, nonoverlapping saved segments to occupy the same architected segment.
- A segment space is composed of up to 64 **member saved segments** (also called members). A member is a licensed program or application, or a component thereof, that you run under VM/XA SP in a segment space. A member begins and ends on a page boundary, and is able to span a megabyte 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. Saved segment packing is most useful for programs that are used by System/370 mode virtual machines, which are restricted to 16MB of virtual storage. To avoid the CP overhead involved when segments are packed, programs used by 370-XA mode virtual machines, which can address up to 999MB of virtual storage, should be installed in DCSSs above the 16MB line. Figure 3 shows the relationship between DCSSs and a packed segment space.

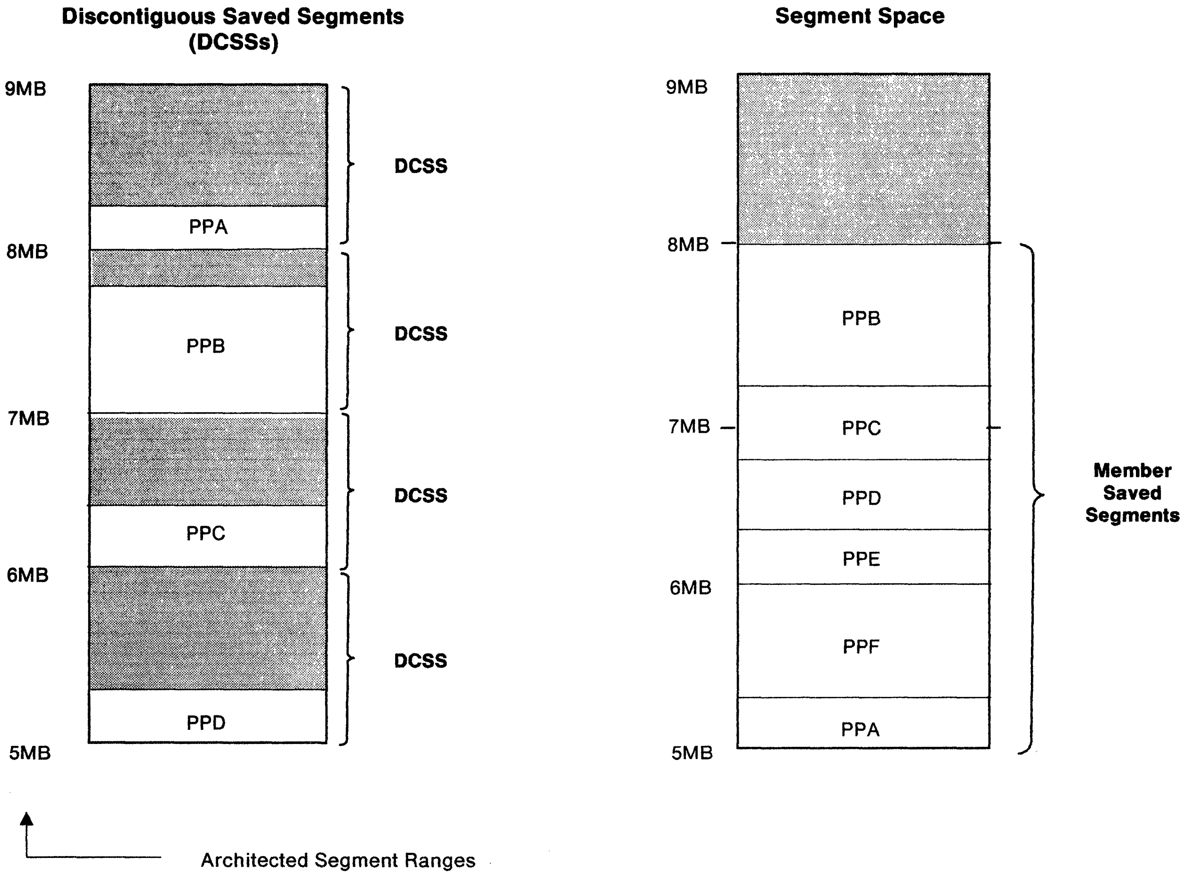


Figure 3. Relationship between DCSSs and a Packed Segment Space

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-megabyte architected segment. *Each 1-megabyte 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 DCSS with Both Shared and Exclusive Page Ranges” on page 66. 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.

Chapter 2. VM/XA SP Segment Support

This chapter explains segment and saved segment support in VM/XA SP. It also discusses the differences between the way saved segments work in 370-XA architecture and the way they work in System/370 architecture.

This chapter tells you:

- What aspects of saved segments are unique to 370-XA architecture and to VM/XA SP
- What you should be aware of when you migrate your applications from a System/370 environment to a 370-XA environment.

What Did 370-XA Architecture Introduce?

370-XA architecture introduced a larger size of architected segments.

Architected Segment Size

The differences in the size of segments in 370-XA architecture can be summarized as follows:

```
1 370-XA segment     = 1MB  (256 pages)

1 System/370 segment = 64KB (16 pages)

1 370-XA segment     = 16 System/370 segments
```

This size difference can cause problems when you migrate from a System/370 environment to a 370-XA environment. One of these problems is that only 16 architected segments are available in VM/XA SP to a System/370 mode virtual machine, whereas 256 architected segments are available in VM/SP. Therefore, there are fewer segments available for your System/370 applications in VM/XA SP.

What Did VM/XA SP Introduce?

The changes introduced to segments by VM/XA SP are:

- Dynamic segment definition
- CP segment packing
- The SEGMENT command and macro (new with CMS 5.5).

Dynamic Saved Segment Definition

VM/XA SP allows you to define (or redefine) a saved segment without IPLing the system. You define a saved segment with the DEFSEG command, and save it with the SAVESEG command.

The ability to define a saved segment with out re-IPLing allows you to install a new version of an application while the old version is still being used. Once the new version is installed and saved, users who access the application receive the new version. When all users accessing the old version release it, the old version is purged.

Classes of Saved Segments: Each saved segment you define is maintained in a system data file that has a specific ID and class. The class associated with this file can be:

A, indicating an **active** saved segment.

R, indicating the saved segment is active and **restricted**.

S, indicating the saved segment is a **skeleton** and is not active.

P, indicating the saved segment is in a **pending purge** state, meaning the saved segment is about to be purged.

When you define a saved segment with the DEFSEG command, a class S (skeleton) system data file is created. The file only becomes an active file (class A or R) when you issue a corresponding SAVESEG command. The SAVESEG command copies the code or data included within the page ranges specified on the DEFSEG command to the associated system data file. The file then is considered active. Only active saved segments can be loaded by a virtual machine.

Member and Segment Space Directories: The system data file associated with a member saved segment contains a segment space directory identifying the segment spaces to which the member belongs. Similarly, a segment space has its own directory of all its members. For more information on system data files and segment space directories, see “DEFSEG Command Functional Description” on page 32 and “SAVESEG Command Functional Description” on page 36.

For more information on defining saved segments, see “Creating Saved Segments” on page 29.

CP Segment Packing

To alleviate the problem of having fewer architected segments, 370-XA architecture introduced **segment packing**. Segment packing allows you to conserve storage by placing more than one application (a licensed program or other code or data) into an architected segment. Consider the applications PPA, PPB, and PPC. Figure 4 on page 9 shows how these programs are placed in storage in System/370 architecture. It also shows how they would have to be placed in storage in 370-XA architecture if segment packing was not possible. SR indicates a shared-read saved segment.

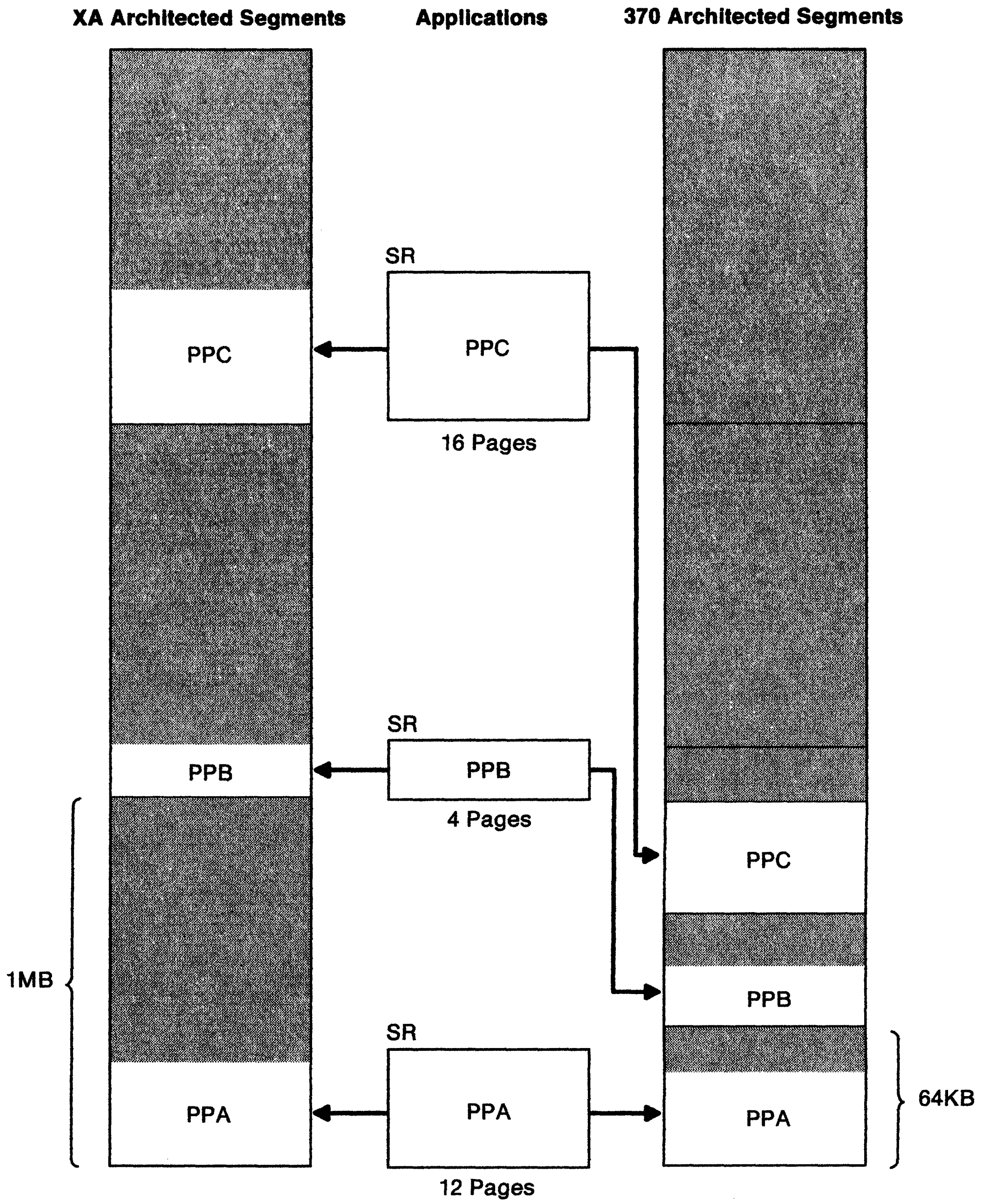


Figure 4. Storing Programs Without Segment Packing

Without segment packing, a great deal of storage would be wasted.

Figure 5 on page 10 shows the same applications placed into 370-XA storage *with* segment packing, which reclaims the storage that otherwise would not be used.

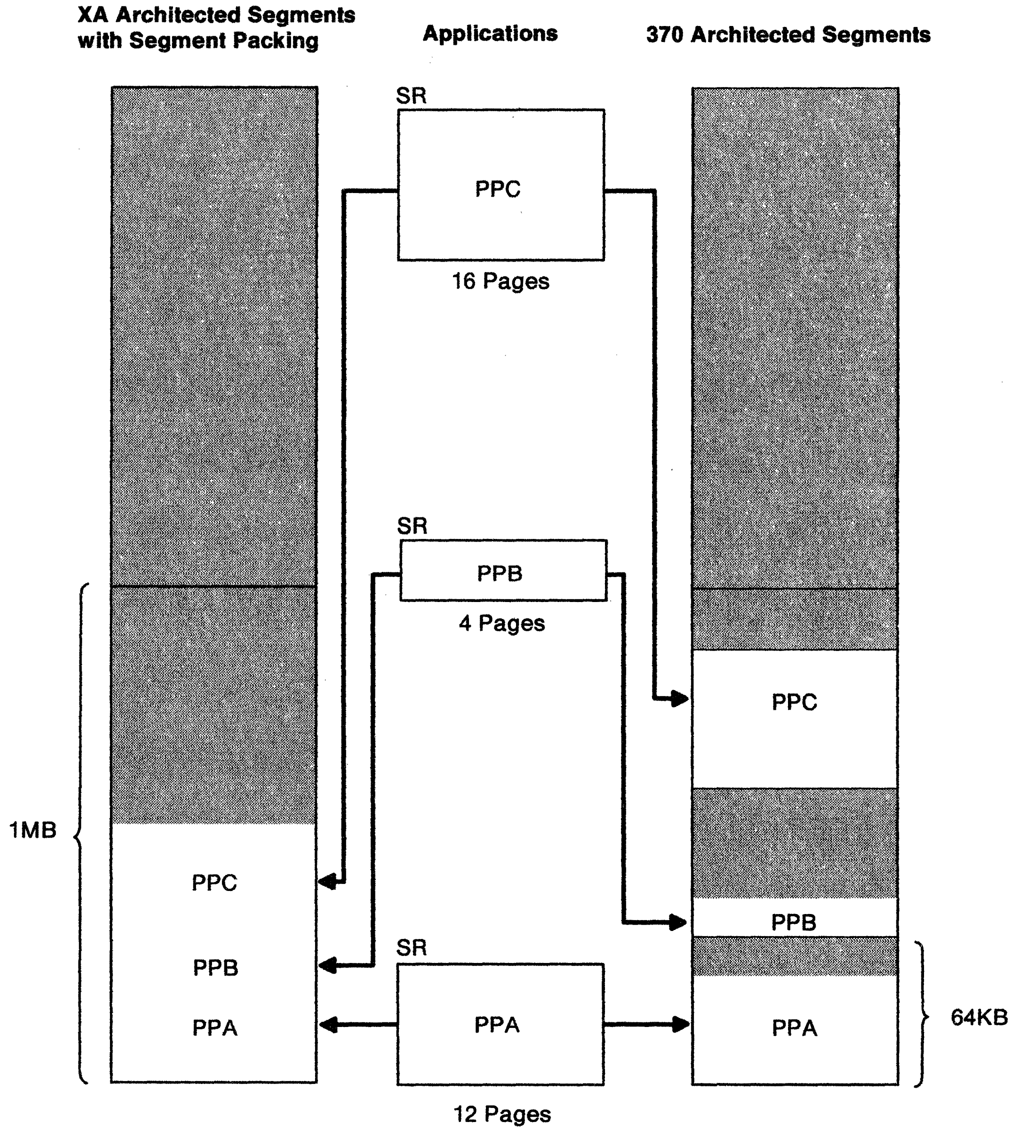


Figure 5. Storing Programs With Segment Packing

For more information on segment packing, see "Using Segment Packing to Conserve Storage Space" on page 45.

The CMS SEGMENT Command and SEGMENT Macro

CMS 5.5 and later releases provides the SEGMENT command, the SEGMENT macro, and the QUERY SEGMENT command to make it easier to load and manage saved segments. The basic formats and functions of these are as follows. (For the complete syntax, see *VM/XA SP CMS Command Reference* or *VM/XA SP CMS Application Program Development Guide.*)

- SEGMENT command:
  - SEGMENT RESERVE — Reserves a space for segments and, optionally, specifies the name of the saved segment to be loaded into the space. This space is a "hole" you can create within a virtual machine's address space. Creating a space for a segment:

    - Allows you to ensure that virtual machines can load saved segments in the storage they specify
    - Eliminates the possibility of saved segments overlaying or being overlaid by portions of CMS.
  - SEGMENT LOAD — Reserves a space for segments (if one is not already reserved) and loads a saved segment into it.
  - SEGMENT PURGE — Purges a saved segment. (SEGMENT PURGE also releases the storage spaces that were created by SEGMENT LOAD.)
  - SEGMENT RELEASE — Releases the storage held by a saved segment.

- SEGMENT macro:
  - SEGMENT FIND — Returns the starting and ending addresses of the saved segment.
  - SEGMENT LOAD — Reserves a space for segments (if one is not already reserved) and loads a saved segment into it.
  - SEGMENT PURGE — Purges a saved segment. (SEGMENT PURGE also releases the storage spaces that were reserved by SEGMENT LOAD.)
- QUERY SEGMENT command — Returns information about spaces reserved for segments and saved segments that were loaded using the SEGMENT command or macro.

For more information on using the SEGMENT command and macro, see "Using Saved Segments from Your Virtual Machine" on page 55.

Migrating Saved Segments to a 370-XA Environment

This section discusses some important considerations for migrating saved segments from a System/370 environment to a 370-XA environment.

Migration Inhibitors

The factors that may cause problems when you migrate your saved segments to 370-XA architecture are:

- The CMS nucleus size
- The way CMS allocates storage
- Applications that require exclusive write segments
- Applications with VM/SP sensitivity.

CMS Nucleus Size

In VM/XA SP, the CMS 5.5 nucleus is approximately 275 pages in size, and ties up two architected segments. Figure 6 shows that CMS resides in all of segment E and part of segment F. (However, you cannot use the rest of segment F for any other products.) In VM/SP HPO, the CMS 4.0 nucleus occupies approximately 126 pages, or half a megabyte. It ties up only the upper half of segment F, as shown below. Therefore, there is more room for applications under VM/SP HPO than there is with VM/XA SP.

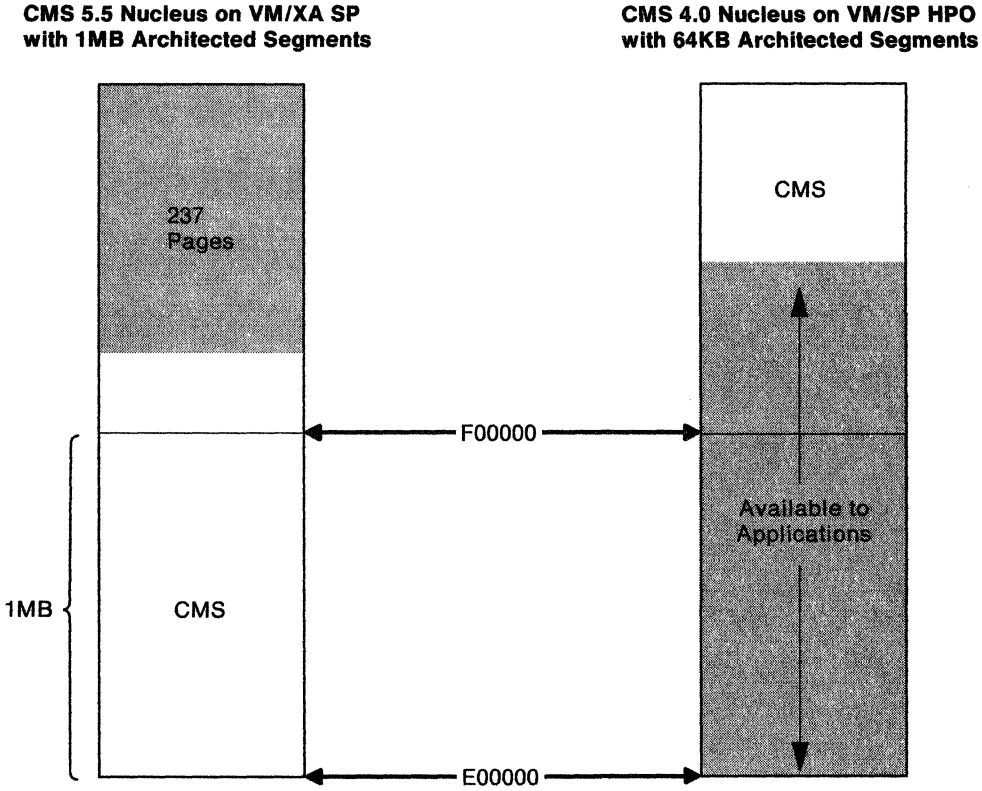
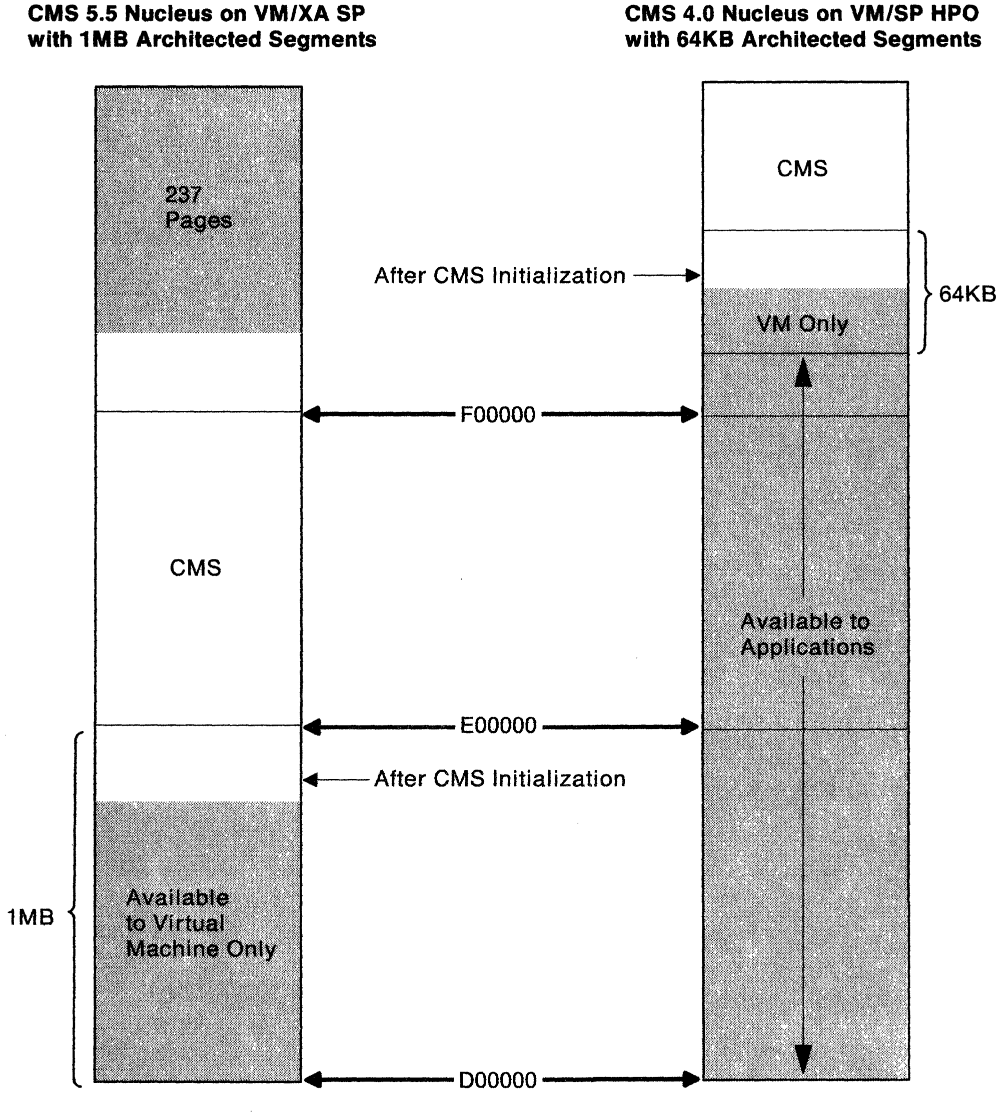

CMS 5.5 Nucleus on VM/XA SP
with 1MB Architected Segments
CMS 4.0 Nucleus on VM/SP HPO
with 64KB Architected Segments
237
Pages
CMS
F00000
Available to
Applications
1MB
CMS
E00000

Figure 6. CMS 5.5 Nucleus Size and Location

CMS Storage Allocation

The way CMS allocates storage at initialization time is another factor to consider when you are migrating to VM/XA SP. When CMS is IPLed from the default location in a virtual machine of 14MB or more, the upper portion of segment D is used to load control blocks, as shown in Figure 7. The remainder of this segment should generally not be used for any saved segments. For information on loading CMS at a lower address so that you *can* use segment D, see "Fitting Applications Below the 16MB Line" on page 44. VM/SP HPO, on the other hand, uses only a 64KB portion of storage for control blocks.



CMS 5.5 Nucleus on VM/XA SP with 1MB Architected Segments
CMS 4.0 Nucleus on VM/SP HPO with 64KB Architected Segments
237 Pages
CMS
After CMS Initialization
64KB
VM Only
F00000
CMS
Available to Applications
E00000
After CMS Initialization
1MB
Available to Virtual Machine Only
D00000

Figure 7. VM/XA SP Storage Allocation

Applications Requiring Exclusive Write Segments

Some applications that execute in a saved segment have to be defined in an exclusive write segment. Since each 1-megabyte architected segment must be defined entirely as shared or entirely as exclusive, an application that requires only one page of exclusive write storage must reside in a separate segment from any applications requiring shared storage. In Figure 8 on page 14, the applications PPA, PPB, and PPC all are defined in shared (SR) saved segments.

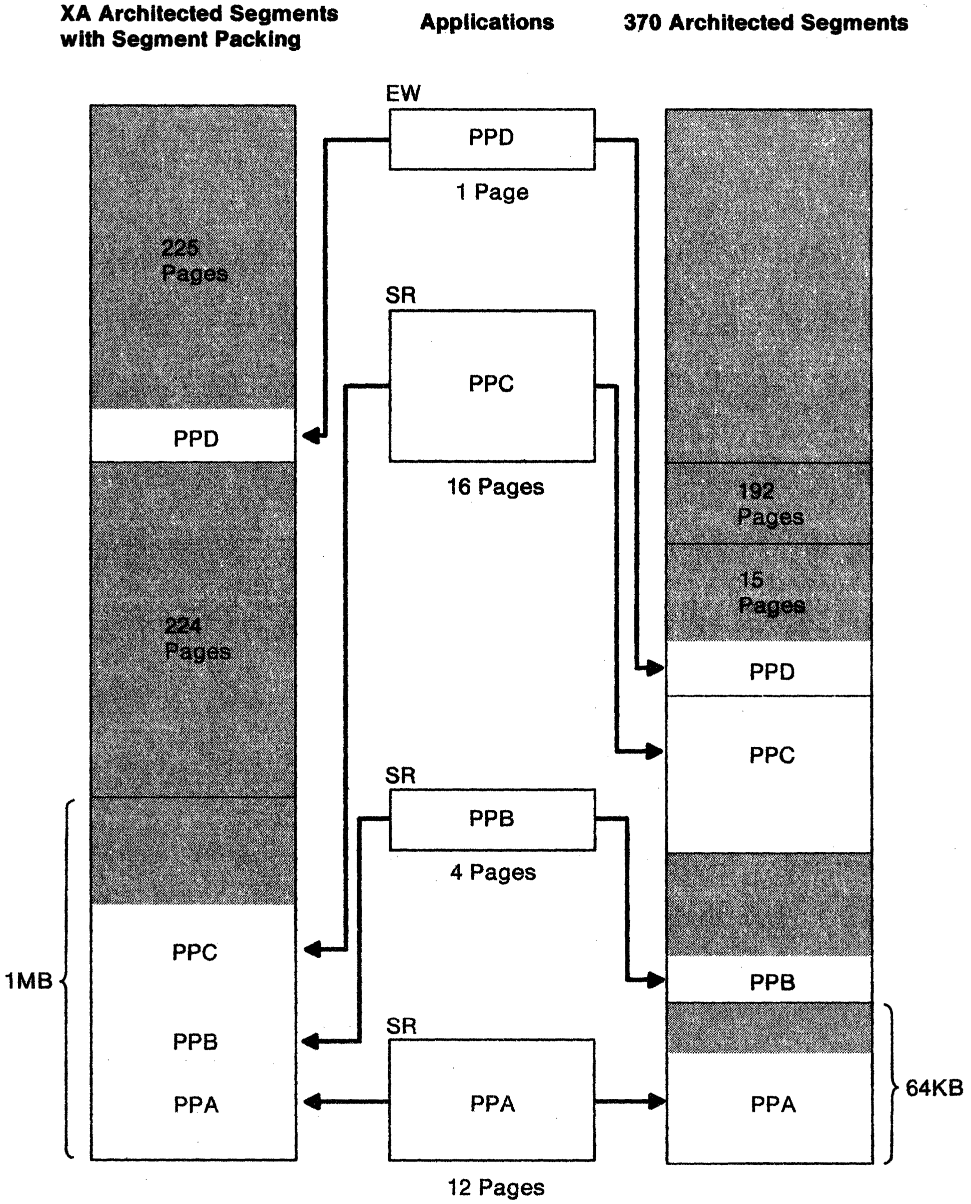


Figure 8. Exclusive Write Segments

These applications therefore can be packed into one architected segment. PPD, however, is defined in an exclusive write (EW) segment, and must be placed in a separate architected segment.

PPD encompasses only one page of storage, but because it requires an exclusive write segment, the remainder of the segment (255 pages) cannot be used for any applications that are defined in a shared segment. However, you can pack other exclusive write saved segments into this segment.

This problem is not as severe in System/370 architecture, since the segment sizes are smaller. When PPD is loaded into a separate segment in System/370 architecture, only 15 pages of storage are unused.

Applications with VM/SP Sensitivity

The installation procedures for some applications require the attachment of a saved segment. To use this type of installation procedure with VM/XA SP, you will need to make sure an active saved segment is available. In VM/XA SP, you can only attach an active saved segment.

For example, the installation EXECs DCSSGEN and SAVEFD, which install the HELP and CMSINST saved segments, use certain VM/SP interfaces. VM/SP and VM/SP HPO allow a skeleton file to be loaded, while VM/XA SP does not. To avoid problems when you install the HELP and CMSINST saved segments:

1. If you are installing the saved segment for the first time, or if you are changing the page ranges associated with the saved segment:

   a. Issue the DEFSEG and SAVESEG commands for the saved segment to create a "dummy" version. This gives you the empty saved segment required by the installation procedures.

   b. Issue the DEFSEG command again. This gives you the skeleton file required by VM/XA SP to create an active saved segment.

2. If you are not changing the page ranges associated with the saved segment, just issue the DEFSEG command to create a skeleton file. (Since the saved segment exists, you already have an active saved segment.)

For examples of updating the HELP and CMSINST saved segments, see "Updating the HELP Saved Segment" on page 74.

Converting DMKSNT Macros to DEFSEG Commands

In VM/SP and VM/SP HPO, you define saved segments during system initialization using a system names table (DMKSNT). In VM/XA SP, you define saved segments while the system is running using CP commands. This section tells you how you can convert your DMKSNT macros to CP DEFSEG commands.

Converting your DMKSNT to DEFSEG commands helps you determine where your existing applications fit in a 370-XA environment. Applications that are overlaid in a System/370 environment can usually be overlaid in a 370-XA environment.

Because of the differences between VM/SP HPO and VM/XA SP and because of other factors — for example, the size of your applications may be larger with VM/XA SP — you must completely remap your installation's shared storage. This remapping, however, is a one-time effort.

Suppose that you have the following entry for the DMKSNT:

```
NAMESYS SYSNAME=SFMASS00

SYSPGCT=144,
SYSPGNM=(2128-2271),
SYSHRSG=(133,134,135,136,137,138,139,140,141),
SYSVOL=volid,
SYSSTRT=(40,07),
SYSSIZE=576K,
VSYSRES=IGNORE
VSYSADR=IGNORE
```

To translate the information in this entry to definitions for the DEFSEG command, follow the steps in this list. It will show you how to obtain the equivalent DEFSEG command for the above DMKSNT entry.

The equivalent command for the preceding entry is:

```
DEFSEG  SFMASS00 800-88F SR
```

See "Using the DEFSEG Command" on page 29 for the full syntax of the DEFSEG command.

1. Extract the name on the SYSNAME parameter to use as *dcssname* on the DEFSEG command. This is the name given to the segment. In the DMKSNT entry, this is entered as:

   ```
   SYSNAME=SFMASS00
   ```

   In VM/XA SP, this same segment name is entered as:

   ```
   DEFSEG  SFMASS00
   ```

2. Convert the decimal value of SYSPGCT to hexadecimal. It is recommended that you use a hexadecimal calculator or conversion table to make the hexadecimal conversions.

   The result tells you how many pages the program needs (in hexadecimal).

   In the example, the result would be X'90', meaning that the program requires X'90' (4KB) pages of storage. If all pages were in the same segment, the page range would be X'0' to X'8F' within the chosen segment. Be sure to count page 0 as the first page. In the DMKSNT entry, this is entered as:

   ```
   SYSPGCT=144
   ```

   In VM/XA SP, X'90' pages are required. The page range may be X'00' to X'8F' within the chosen segment.

3. Divide the decimal value of SYSPGCT by 16.

   The result tells you how many 64KB segments the program needs in VM/SP HPO.

In the example, the result would be 9, meaning that the program requires nine 64KB-segments. In the DMKSNT entry:

```
SYSPGCT=144
```

To convert this for VM/XA SP:

```
144 divided by 16 equals 9
```

The program needs nine 64KB segments. This information is used later in the DEFSEG command definition.

4. Convert both decimal values of the SYSPGNM parameter to hexadecimal values.

   Use the hexadecimal values to define the starting and ending pages and relative location of the segment in VM/SP HPO.

   In the example shown, 2128 is X'850'. The likely place to locate this program might be to start it on page X'0' in segment 8. (If you use segment 8 for CMSVSAM, this new segment will overlay CMSVSAM.) Round *down* to the nearest segment boundary. The ending page is X'8DF' (X'850' plus X'8F' pages). Count page 0 as the first page. In the DMKSNT entry, this is entered as:

   ```
   SYSPGNM=(2128-2271)
   ```

   Converting the values for VM/XA SP:

   ```
   2128 = X'850'     2271 = X'8DF'
   ```

   In VM/XA SP, this is entered as:

   ```
   DEFSEG  SFMASS00 800
   ```

   (You will use the ending page value later in the DEFSEG command definition.)

5. Look for the SYSHRSG parameter. If present, all the 64KB segments listed are to be shared by users. The first letter of the *mode* operand will be S. Count the number of segments listed singly or within a segment range. Compare the count to the result of SYSPGCT.

   If the count equals the result of SYSPGCT, all storage is shared. When the count is less than the result of SYSPGCT, you must figure out which pages and how many pages of storage must be put in an exclusive segment or segments. The first letter of the *mode* operand is an E.

   Another way to determine the first letter of the *mode* operand is to multiply the beginning segment number by 16 (the number of pages in a 64KB segment) and then multiply the ending segment number by 16 and add 15 (pages 0—15 of the last segment). Compare the two results to the decimal values for SYSPGNM:

   - When your results are the same as the values in SYSPGNM, all segments are shared.
   - When your results do not match the values in SYSPGNM, the segments *outside* your results are exclusive.

   In the example, there are nine segments listed as shared; 9 is the result of dividing SYSPGCT by 16, so all segments defined for SFMASS00 will be shared. In the DMKSNT entry, this is entered as:

   ```
   SYSHRSG=(133,134,135,136,
            137,138,139,140,141)
   ```

To convert this for VM/XA SP:

```
9 segments
133 x 16 = 2128
141 x 16 = 2256 + 15 = 2271
```

In VM/XA SP, the equivalent statement is:

```
DEFSEG  SFMASS00 800-88F Sm
```

The *m* in the above example is the second letter of the mode operand (either R, E, or N). This will be discussed below.

If the example had SYSHRSG=(133,134,135,136,137,138), six segments would be shared; the remaining three segments would be exclusive and would have to be placed in the next 1MB segment. Each segment of SYSHRSG represents X'0' to X'F' pages, so the shared portion would be the first X'60' pages (pages X'0' to X'5F') of the lower segment, and X'30' pages would be the first pages in the next segment (pages X'0' to X'2F'). The page range might be X'800' to X'85F' for shared code and X'900' to X'92F' for the exclusive code. In the DMKSNT entry, this is entered as:

```
SYSHRSG=(133,134,135,136,137,138)
```

To convert this for VM/XA SP:

```
6 segments
133 x 16 = 2128
138 x 16 = 2208 + 15 = 2223
```

In VM/XA SP, the equivalent statement is:

```
DEFSEG  SFMASS00 800-85F Sm 900-92F Em
```

6. Look for the PROTECT parameter. When PROTECT=ON or when the parameter is missing (the default is ON), the second letter of the *mode* operand will be R (for read-only). When PROTECT=OFF, the second letter of the *mode* operand will be W (for read/write access). All exclusive segments should have W as the second letter. In the DMKSNT entry:

   ```
   The PROTECT parameter is missing
   ```

   For VM/XA SP, this means that all pages are shared read-only. In VM/XA SP, the equivalent statement is:

   ```
   DEFSEG  SFMASS00 800-88F SR
   ```

   In the preceding example with both shared and exclusive segments, you would assign R to the shared segment because there was no PROTECT parameter, but W to the exclusive segment. In the DMKSNT entry:

   ```
   The PROTECT parameter is missing
   ```

   For VM/XA SP, shared pages are read-only and exclusive pages are read/write.

   In VM/XA SP, the equivalent statement is:

   ```
   DEFSEG  SFMASS00 800-85F SR 900-92F EW
   ```

7. Look for the RCVID operand. If present, add the RSTD operand to the DEFSEG command. When you want to restrict access to the program, you might want to add the RSTD operand. You would also want to add the RSTD operand when a person will load the segment using DIAGNOSE X'64' with the LOADNSHR function code. Access to a restricted segment requires that the user have directory authorization via the NAMESAVE directory statement. If you want to restrict access to the program in the example, you would add RSTD

to the DEFSEG command. Suppose the following were entered in the DMKSNT entry:

```
RCVID=userid
```

In VM/XA SP, the equivalent statement would be:

```
DEFSEG  SFMASS00 800-88F SR RSTD
```

8. The following parameters of the DMKSNT entry are not applicable to a DEFSEG command in VM/XA SP: SYSVOL, SYSSTRT, SYSSIZE, VSYSRES, VSYSADR, SYSCYL, SYSBLOK, USERID, and SAVESEG.

Summary — How Saved Segment Support Differs in VM/XA SP

Although the basic concept is the same, the saved segment support in VM/XA SP differs from the saved segment support in VM/SP HPO. Table 1 summarizes the differences.

Table 1. How Saved Segment Support Differs in VM/XA SP

VM/SP HPO Saved Segment Support	VM/XA SP Saved Segment Support
You define saved segments during system initialization using a system names table (DMKSNT).	You define saved segments while the system is running using CP commands.
Storage is comprised of 64KB segments, each containing 16 pages of 4KB storage.	Storage is comprised of 1MB segments, each containing 256 pages of 4KB storage.
You can store only one licensed program in each 64KB segment. The licensed program must be stored on a 64KB boundary.	You can store several licensed programs in one or more 1MB segments as long as you do not store both shared code and nonshared (exclusive) code in the same 1MB segment. Each licensed program must be stored on a page boundary.
You define the ranges of licensed programs in decimal values. Messages and responses are in decimal.	You define the ranges of licensed programs in hexadecimal values. Messages and responses are in hexadecimal.
To prevent overlaid saved segments, a saved segment must be outside the address space of the virtual machine that loads the saved segment.	To prevent overlaid saved segments, a saved segment must be loaded via the SEGMENT command or macro; or, the saved segment must be outside the address space of the virtual machine that loads the saved segment.
You can attach a saved segment once it is defined in DMKSNT.	You can only attach an active saved segment (one that has been defined, installed, and saved).
You can resave the contents of an existing saved segment without redefining it as long as you are not changing the page ranges.	You must redefine a saved segment before you resave information into it.

When converting to VM/XA SP, you should maintain the DMKSNT file on the VM/SP HPO SYSRES in case you need to back off to VM/SP HPO. The VM/XA SP saved segments and saved systems are kept in system data files and cannot be transferred to VM/SP HPO using SPTAPE. (You can move files between VM/XA SP systems using SPTAPE.)

Chapter 3. System Programmer Considerations

This chapter presents information that a system programmer needs to plan for and install saved segments. It is divided into three sections:

1. "Planning Considerations" on page 22 tells you:
   - Where to and where not to save a saved segment
   - How to plan for saved segments based on virtual machine size
   - How to plan for saved segments based on System/370 mode and 370-XA mode.
2. "Creating Saved Segments" on page 29 tells you how to:
   - Use the DEFSEG and SAVESEG commands to create saved segments
   - Keep backup copies of saved segments
   - Purge saved segments from the system
   - Display information on saved segments.
3. "Installing Applications in Saved Segments" on page 44 tells you how to:
   - Load applications into saved segments
   - Pack segments to conserve storage
   - Overlay applications
   - Redefine saved segments.

In this chapter and those that follow, these acronyms are used to refer to licensed programs (and other applications). Consult the *VM/XA SP Licensed Program Specifications* for more current information on licensed programs.

- APL 2 refers to A Programming Language 2 Release 3 (Program No. 5668-899)
- AS refers to Application System Version 1 Release 5 Modification Level 1 (Program No. 5767-032)
- DW/370 refers to DisplayWrite/370 Version 1 Release 2 (Program No. 5664-370)
- DCF refers to Document Composition Facility Version 1 Release 3 Modification Level 1 (Program No. 5748-XX9)
- FORTRAN refers to VS FORTRAN Version 2 Release 3:
  - Compiler, Library, and Interactive Debug (Program No. 5668-806)
  - Library (Program No. 5668-805).
- GAM/SP refers to Graphics Access Method/SP Release 2 (Program No. 5668-978)
- GCS refers to the group control system virtual machine supervisor
- GDDM refers to GDDM/VMXA Version 2 Release 2 (Program No. 5684-007)
- GDDM/graPHIGS refers to GDDM/graPHIGS Release 3 Modification Level 2 (Program No. 5668-792)

- GDDM/PGF refers to GDDM/Presentation Graphics Facility Version 2 Release 1 (Program No. 5668-812)
- ISPF refers to Interactive System Productivity Facility for VM/XA Version 2 Release 2 (Program No. 5684-014)
- ISPF/PDF refers to Interactive System Productivity Facility/ Program Development Facility for VM/XA Version 2 Release 2 (Program No. 5664-285)
- PROFS® refers to IBM Professional Office System Version 2 Release 2 Modification Level 2 (Program No. 5664-309) including the PROFS Application Support Feature and the PROFS Personal Computer Support Feature in System/370 mode only
- QMF refers to Query Management Facility Version 2 Release 2 (Program No. 5668-AAA)
- SQL refers to SQL/DS Version 2 Release 1 (Program No. 5688-004)
- TIF refers to The Information Facility Program Offering Version 2 (Program No. 5798-DYF).

Note: Later versions, releases, and modifications of the above-listed programs are supported unless explicitly stated otherwise.

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 two 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. A map of your DMKSNT ASSEMBLE file can help you determine which programs overlap in your current environment. For more information, see "Overlaying Your Applications" on page 48.
   - Know how many pages of storage each saved segment requires.
   - Know in what mode the application operates. Under VM/XA SP, these modes are possible:

     System/370 compatibility mode
     The application runs in a System/370 mode virtual machine. This implies the application runs under the 16MB line.

PROFS is a registered trademark of the International Business Machines Corporation.

370-XA toleration mode
The application can run in a 370-XA mode virtual machine without taking advantage of 31-bit addressing. It can also run in a System/370 mode virtual machine. In either case, this type of application runs under the 16MB line, but in a 370-XA mode virtual machine the application can call programs that reside above the 16MB line.

370-XA exploitation mode
The application can run above the 16MB line in a 370-XA mode virtual machine and can exploit 31-bit addressing. It can also run below the 16MB line in a System/370 mode virtual machine or a 370-XA mode virtual machine.

Note: In this book, System/370 mode is also referred to as "370 mode". Also, "XA mode" refers to both 370-XA toleration mode and 370-XA exploitation mode.

- 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 three pages of exclusive-write storage and eight pages of shared-read storage, the program will require parts of two segments.
- Determine if 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-megabyte virtual machine. Based on this, you should install saved segments from the 4-megabyte line on up. (Be aware, however, of where CMS and other system-related saved segments are loaded. This is discussed in "Planning for Segments Based on System/370 or 370-XA Mode" on page 26.)
   - If a set of applications executes in 370-XA toleration or exploitation mode, you may want to have users of these applications run XA mode virtual machines. For an application that runs in 370-XA exploitation mode, consider defining two saved segments: one above the 16MB line (for XA users) and one below (for 370 users). You can only do this if the application can be called using two different names.
   - Specific users may have unique product requirements. For example, a user might have FORTRAN programs that interface with GDDM/VMXA and GDDM/PGF. In this case, all of these programs have to be available to this user, so they should not be defined in saved segments that overlay each other.

By gathering the above information, you develop a set of rules and guidelines that your installation needs to follow. Once 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.

For a sample storage layout for a given set of applications, see "Setting Up Your Storage Layout" on page 76.

Planning Where to Save a Saved Segment

To avoid defining more than one saved segment in the same address range, consider the following:

- The size of a virtual machine that will access a saved segment
- Whether the virtual machine runs in System/370 mode or 370-XA mode.

Planning for Saved Segments Based on the Size of a Virtual Machine

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.

For users with a virtual machine size less than or equal to 16 MB, CMS requires the uppermost megabyte of the virtual machine. Because of this, if a saved segment resides just below the 8MB line, a 2MB or 4MB virtual machine can use it; an 8MB virtual machine can not. A 9MB 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 5MB line, an 8MB virtual machine can use it but a 5MB virtual machine can not.

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

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

Where CMS is Loaded: When you IPL CMS, it initializes an internal page allocation table at a location determined by the size of the virtual machine. CMS cannot reserve a space for a saved segment at the location of this table. The following sections describe how you can avoid collisions between the page allocation table and saved segments.

Saved Segments above 16MB: For virtual machines larger than 16MB, CMS always places the page allocation table at the 16MB line. The formula for determining the size of the table is:

```
table size = VMDSSIZE + 1
```

where:

table size is measured in pages

VMDSSIZE is measured in megabytes

For example, for a 999MB virtual machine, CMS allocates a 1000-page table. For a 64MB virtual machine, CMS allocates a 65-page table.

To determine the address where you can safely define segment spaces, add the storage required for the page allocation table (rounded up to the nearest megabyte) to 16MB. The following table shows the results:

VMDSSIZE (MB)	**Table Size (Pages)**	**Safe Address for Defining Segment Spaces (MB)**
16 through 255	17 through 256	17
256 through 511	257 through 512	18
512 through 767	513 through 768	19
768 through 999	769 through 1000	20

Storage Configuration for a CMS Virtual Machine Greater Than 16MB: Figure 9 illustrates the location of the page allocation table for a virtual machine greater than 16MB.

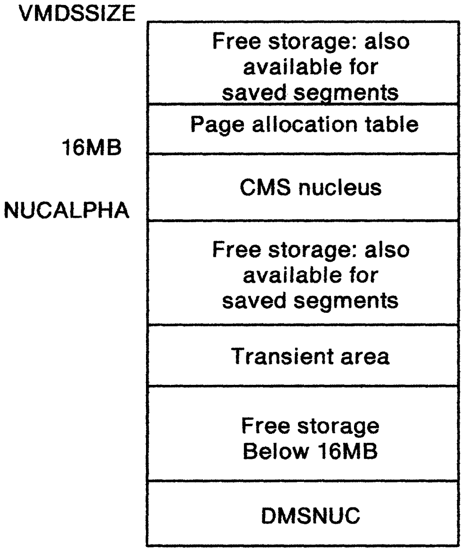


Figure 9. Storage Configuration for a CMS Virtual Machine Greater Than 16MB

Saved Segments below 16MB: If a virtual machine is less than 16MB, the page allocation table extends downward from VMDSSIZE or NUCALPHA, whichever is smaller. (NUCALPHA is the starting address of the CMS nucleus.) If CMS is a saved system starting at the 14MB line and is IPLed in an 8MB virtual machine, the page allocation table would end at the 8MB line. If the same saved system was IPLed in a 2MB virtual machine, the end of the allocation table would be at 2MB. In a 16MB virtual machine, the end of the table would be at 14MB because NUCALPHA is less than VMDSSIZE.

Storage Configuration for a CMS Virtual Machine Less Than 16MB: Figure 10 on page 26 shows the location of the page allocation table for a virtual machine less than or equal to 16MB. The left half of the figure shows the storage configuration when VMDSSIZE is greater than NUCALPHA; the right half shows the storage configuration when VMDSSIZE is less than NUCALPHA.

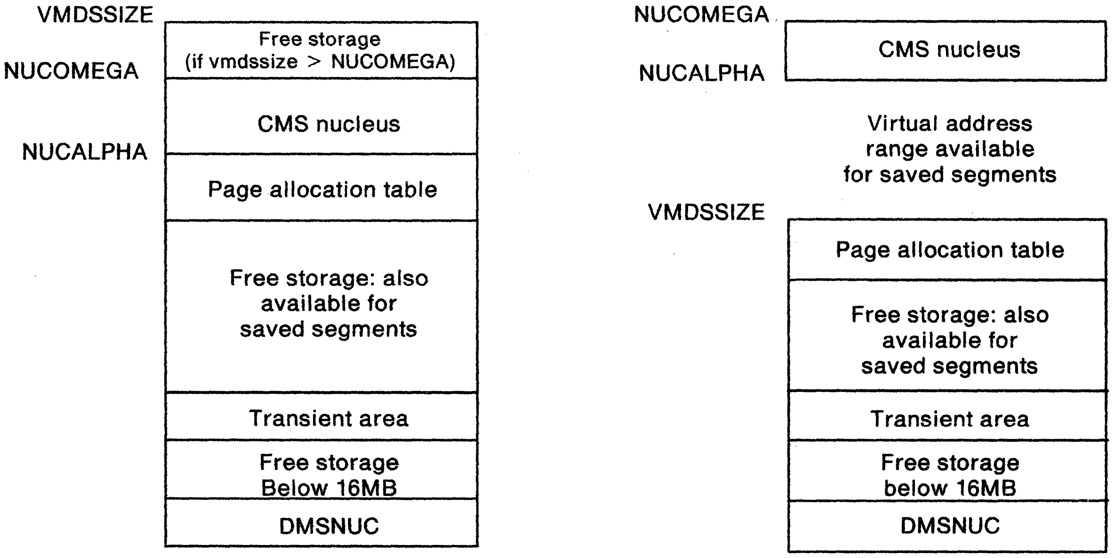


Figure 10. Storage Configuration for a CMS Virtual Machine Less Than 16MB

Planning for Segments Based on System/370 or 370-XA Mode

For programs that operate in System/370 mode, 16 segments are available: segments X'0' through X'F'. For programs that operate in 370-XA mode, 999 segments are available: segments X'0' through X'3E6'.

In Figure 11 on page 27, you can see the default locations for System/370 segments; optional segments have the segment name within parentheses. Three columns on the right are provided for planning the layout of segments for your installation.

For programs that operate in 370-XA mode, 999 segments may be available: segments X'0' through X'3E6'. CMS and the optional segments occupy the same segments as they do in System/370 storage. Unless otherwise defined, CMS occupies segments 0 (EW), E, and F (both SR). A portion of another segment is used for a work area and page allocation table. The location of this megabyte depends upon the size of the user's virtual machine and the location of the CMS nucleus. For a further explanation, see "Saved Segments above 16MB" on page 24 and "Saved Segments below 16MB" on page 25.

Although programs may be 370-XA capable, this does not mean they can be loaded above the 16MB line. Check the installation instructions for each product that you will install in your 370-XA system; note any restrictions about the location at which to load the program.

In Figure 12 on page 28, you can see the locations for default segments in XA mode; optional segments have the segment name within parentheses. The columns on the right are provided for you to plan the layout of segments for your installation. For examples of storage layouts containing various applications, see "Setting Up Your Storage Layout" on page 76.

16M					
(F) F00	↑	↑			
E00	CMS	CMSXA			
D00					
C00	(HELP)(SR) (CMSINST)(SR)	(HELP)(SR) (CMSINST)(SR)			
B00	(CMSVSAM)(SR) (CMSAMS)(SR) (CMSBAM)(SR) (CMSDOS)(SR)	(CMSVSAM)(SR) (CMSAMS)(SR) (CMSBAM)(SR) (CMSDOS)(SR)			
A00	(CMSVSAM)(EW) (CMSAMS)(EW)	(CMSVSAM)(EW) (CMSAMS)(EW)			
900					
800					
700					
600					
500					
400					
300					
200					
100					
(0) 000	CMS	CMSXA			

Note: CMS may use part of the D00 segment for a work area and page allocation table. For more information, see "Tips for Installing Your Applications in Saved Segments" on page 44.

Figure 11. Storage Planning Worksheet for System/370 Mode

(3E6) 3E600					
. .					
. . .					
1200					
1100					
(10) 1000					
(F) F00	↑	↑			
E00	CMS	CMSXA			
D00					
C00	(HELP)(SR) (CMSINST)(SR)	(HELP)(SR) (CMSINST)(SR)			
B00	(CMSVSAM)(SR) (CMSAMS)(SR) (CMSBAM)(SR) (CMSDOS)(SR)	(CMSVSAM)(SR) (CMSAMS)(SR) (CMSBAM)(SR) (CMSDOS)(SR)			
A00	(CMSVSAM)(EW) (CMSAMS)(EW)	(CMSVSAM)(EW) (CMSAMS)(EW)			
900					
800					
700					
600					
500					
400					
300					
200					
100					
(0) 000	CMS	CMSXA			

Note: CMS may use part of the D00 segment for a work area and page allocation table. For more information, see "Tips for Installing Your Applications in Saved Segments" on page 44.

Figure 12. Storage Planning Worksheet for 370-XA Mode

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. Issue 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 issue a corresponding SAVESEG command.
2. Load the application into the area of storage you set aside with the DEFSEG command.
3. Issue 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 Chapter 5, "Defining Saved Segments – Examples" on page 65.

Using the DEFSEG Command

The following is the syntax for the DEFSEG command. (For usage notes associated with the DEFSEG command, see *VM/XA SP CP Command Reference*.)

```
DEFSEG        dcssname {{hexpage1-hexpage2  type}...} [RSTD] [SPACE spacename ]
                       {SAMErange                 }
```

where:

dcssname
is the name of the saved segment (a 1- to 8-character alphanumeric string). When used with SPACE, the *dcssname* specified here is known as the member name and must be different from the SPACE name.

hexpage1-hexpage2
is a range of pages to be saved. When *hexpage1* and *hexpage2* are the same, only that page is saved. The range may be contained in a single 1-megabyte segment, or it may comprise a contiguous area in two or more segments. You may use multiple page ranges to define a 1-megabyte segment. However, the page descriptor codes of all page ranges in the same segment must have the same shared or exclusive attribute.

Define only those pages whose data must be saved. Any page or pages in a segment not explicitly defined in a page range on the DEFSEG command will default to the *no data saved* attribute. If other pages in the segment were defined with the shared attribute (page descriptor codes SR, SW, SN and SC), pages not specified on the command line become shared read-only pages whose data are not saved. No page descriptor code corresponds to this state; it has the same attributes as the state defined by page descriptor code SC, except that CP will not write into these pages.

If other pages in the segment were defined with the exclusive attribute (page descriptor codes ER, EW, and EN), any pages not specified on the command line become exclusive read/write pages whose data are not saved. This corresponds to page descriptor code EN.

The page ranges of a member saved segment cannot overlap with the specified page ranges of another member saved segment associated with the same segment space. Areas not specified (defaulted to *no data saved*) may be taken by another member saved segment. If the member uses these pages but doesn't need the pages' data saved, the range with the EN or SN attribute type must be specified. Also, this member saved segment's range descriptor code cannot conflict with those of an existing member that has ranges defined in the same segment of storage.

The page number is a hexadecimal value less than or equal to X'3E6FF'(999 MB). The following list shows examples of how storage addresses translate into page numbers:

Hexadecimal Storage Address	**Hexadecimal Page Number**
00000xxx	0
00001xxx	1
00022xxx	22
00333xxx	333
04444xxx	4444
3E6FFxxx	3E6FF

type

refers to the page descriptor code of the page range in the saved segment. It indicates the type of virtual machine access permitted to pages in the range.

The first character is either "E" for exclusive access or "S" for shared access. Sharing of storage between virtual machines is based on 1-megabyte segments, so all 256 pages in any one segment must be either exclusive or shared.

The second character of the page descriptor code defines the storage protection characteristics of the page. These character designators are as follows:

"R" indicates that page protection is used to make the page range read-only.

"W" means that the page is not protected, and users have read/write access.

"N" stands for "read/write, no data saved," meaning that the page is treated as a new page when the user references it. The contents of and storage keys for "no data saved" pages are not saved by the SAVESEG command.

"C" stands for "CP-writeable, virtual machine read-only, no data saved," meaning virtual machines have read-only access to these pages, but CP services may change the data in the pages.

The contents of storage keys for "no data saved" pages are not saved by the SAVESEG command.

Valid page descriptor codes are:

EW	Exclusive read/write access
EN	Exclusive read/write access, no data saved
ER	exclusive read-only access
SW	shared read/write access
SN	shared read/write access, no data saved
SR	shared read-only access
SC	CP writeable pages, shared read-only access by virtual machine, no data saved

The user must define all pages in segment 0 in ranges with an exclusive page descriptor code (EW, ER, or EN). The system will reject the DEFSEG command if the user defines any pages in segment 0 with a shared access page descriptor code.

SAMERANGE

specifies that this member saved segment definition is the same as one specified by a previous DEFSEG. This operand is mutually exclusive with the *hexpage1-hexpage2* operands. When you specify SAMERANGE, you also must specify SPACE.

You don't have to specify SAVESEG if you use DEFSEG with the SAME operand for a class A member saved segment. However, if the member saved segment you're referencing is a class S file, you must still use SAVESEG to save the file.

SAMERANGE makes it easier to create overlaying segment spaces by using the same member for both segment spaces.

If you use this operand with a *spacename* that doesn't exist, the member that you specify must be a class S (skeleton) file.

You must have either a class S or class A file with the same *dcssname* when you use this operand. If you have both class A and class S files, the command uses the information in the class S file.

RSTD

indicates a restricted saved segment. The user must have a corresponding NAMESAVE directory statement to access this saved segment. If any member of a segment space is defined with the restricted operand (that is, the SPACE operand is used), the entire segment space is then restricted. The restricted *spacename* must be used on the NAMESAVE directory statement. If you didn't specify SPACE, then the *dcssname* is restricted. If you use the RSTD operand, make sure that you use the NAMESAVE directory statement to allow authorization.

If you used the SAMERANGE operand, you can use RSTD. This will affect the authorization of the segment space that you specified in the DEFSEG definition. The *spacename* must be used on the NAMESAVE directory statement.

SPACE

specifies the definition of a segment space. Use this operand to make the *dcssname* that you are defining a member of this segment space.

spacename

is the name of the segment space (a 1- to 8-character alphanumeric string). You must specify this if you use the SPACE operand. The *spacename* must be different from any saved segment known to CP.

DEFSEG Command Functional Description

This section shows what operations the DEFSEG command performs when it is issued.

When the DEFSEG command is issued, 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 following scenarios show which files are created (or affected) after various DEFSEG commands are issued. The abbreviations used (such as PPW, PPX, and GRP1) are the names of saved segments.

No saved segment files currently exist.

defseg ppw 700-7ff sr is issued.

- A class S (skeleton) SDF with the name PPW is created. The system descriptor record is updated, and a unique spool ID number is assigned to the file.
  - Because the SPACE operand was *not* specified on the DEFSEG command, the file created is a DCSS.
  - 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.

defseg ppx 800-820 sr space grp1 is issued.

- A class S SDF with the name PPX is created. The system descriptor record is updated, and a unique spool ID number is assigned to the file.
  - Because the SPACE operand was specified on the DEFSEG command, the SDF created is a member.
  - 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 since GRP1 is the only segment space associated with the member PPX.
  - An entry for GRP1 is made in the directory indicating that GRP1 is a segment space containing PPX.
- A class S SDF with the name GRP1 is created. The system descriptor record is updated, and a unique spool ID number is assigned to the file.
  - Because the SPACE operand was specified on the DEFSEG command, the SDF created is a segment space.

Figure 13 (Part 1 of 3). Results of Issuing the DEFSEG Command

- 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 megabyte 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 the DIAGNOSE X'64' programming interface.
- A count is maintained indicating how many members are associated with this segment space. In this case, the value is 1.
- An entry for PPX is made in the directory indicating that PPX is a member of this segment space.
- The page range (800-820) 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.

defseg ppy 821-830 sr space grp1 is issued.

- 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:
  - Since a class S SDF already exists for GRP1, this file does not have to be created, but its descriptor record is updated.
  - 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.
  - An entry for PPY is made in the directory indicating 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. The directory now has entries for PPX and PPY.

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

defseg ppu 800-820 sr space grp2 is issued.

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

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

Figure 13 (Part 2 of 3). Results of Issuing the DEFSEG Command

defseg ppy same space grp2 is issued.

- 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.
  - An entry for GRP2 is made in the directory indicating that GRP2 is a segment space containing the member PPY. The class S file for PPY now has entries for GRP1 and GRP2.
- 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.
  - An entry for PPY is made in the directory indicating 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. The directory now has entries for PPU and PPY.

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

Figure 13 (Part 3 of 3). Results of Issuing the DEFSEG Command

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 only specify the SAME operand if you also specify the SPACE operand.
- With type EW, EN, SW, and SN saved segments, you can *not* 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 re-install a read-only (SR) member of a space containing other members with writeable (EW or SW) pages.
- You should issue 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 can not overlay any of the ranges specified for a 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

The following is the syntax of the SAVESEG command. For usage notes associated with the SAVESEG command, see *VM/XA SP CP Command Reference.*

SAVESEG	dcssname

where:

dcssname
is the name (a 1- to 8-character alphanumeric string) of the segment to be saved. This is the file name of a class S (skeleton) SDF previously defined with the DEFSEG command. When a DEFSEG command using the SPACE operand is issued, the *dcssname* specified in the SAVESEG command is interpreted as the member name specified in the DEFSEG command.

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. Once this is done, the SAVESEG command can be issued 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:

1. The virtual machine size must include the pages ranges of the saved segment. For example, for a saved segment defined in the B00—BFF address range, the virtual machine must be at least 12MB. Or,

2. The virtual machine must define or use an existing saved segment containing the required pages. In other words, the saved segment:

   - Must be defined with writeable pages, or

   - 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.

   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. Or, follow the steps below. It is best that you perform these steps when you first install the application that resides in the saved segment:

   a. Purge any **skeleton** files previously defined for the saved segment:

      ```
      purge nss spoolid#
      ```

      If the saved segment is a DCSS, you are done. If the saved segment is a member saved segment, then you must also purge the spool file associated with the segment space which contains the member.

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

      ```
      defseg name range ew
      ```

This defines a DCSS as non-shared (EW). If you are defining a segment space or a member saved segment, issue:

```
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
```

c. Create an active segment by issuing the SAVESEG command for the saved segment you are installing:

```
saveseg name
```

d. Set up another skeleton file by issuing another DEFSEG command for the saved segment. This step is necessary since 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, issue:

```
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
```

e. 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 will change from skeleton to active if all other members are active.

The following SAVESEG commands correspond to the DEFSEG commands described in "DEFSEG Command Functional Description" on page 32. The DEFSEGs 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 following examples, assume the respective page ranges have been properly initialized and the SAVESEG commands are issued from an installation EXEC. The results of each command are described.

A class S file for PPU, GRP2, PPY, PPX, GRP1, and PPW currently exist.

saveseg ppw is issued.

- The SAVESEG command determines if the saved segment is a member:
  - SAVESEG reads the descriptor record from the class S file for PPW. In this case, a flag field indicates PPW is not a member but rather a DCSS.
  - The pages associated with PPW (as defined by the previous DEFSEG command) and their keys are copied to a system data file.
  - The file for PPW is changed from class S to class A.
  - 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 address space.
  - Since 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.

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

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

saveseg ppx is issued.

- The SAVESEG command determines if the saved segment is a segment space:
  - SAVESEG reads the descriptor record from the class S file for PPX. In this case, a flag field indicates PPX is not a segment space but rather a member saved segment.
  - The pages associated with PPX (as defined by the previous DEFSEG command) and their keys are copied to a system data file.
  - The file for PPX is changed from class S to class A.
  - 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.
  - Since 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 descriptor record of GRP1 is read and its directory section is then processed as follows:
      - The status of PPX is changed to saved.
      - 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.

Figure 14 (Part 1 of 3). Results of Issuing the SAVESEG Command

If all the members have a saved status, then 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 exist for PPU, GRP2, PPY, and GRP1, and a class A file exists for PPX, and PPW.

saveseg ppy is issued.

- The SAVESEG command determines if the saved segment is a segment space:
  - SAVESEG reads the descriptor record from the class S file for PPY. In this case, a flag field indicates PPY is not a segment space but rather a member saved segment.
  - The pages associated with PPY (as defined by the previous DEFSEG command) and their keys are copied to a system data file.
  - The file for PPY is changed from class S to class A.
  - 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.
  - Since PPY is a member saved segment, the directory is updated:
    - 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 descriptor record of GRP1 is read and its directory section is then processed as follows:
      - The status of PPY is changed to saved.
      - 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, then the class of the segment space is changed from S to A. In our example, all members have a saved status. So, the class of the file associated with GRP1 is changed to A. Since 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.
      - If a class A file already existed for GRP1, 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.

      Since PPY is a member, it can be attached to a virtual machine once one of its associated segment spaces becomes class A or R.
    - Next, the descriptor record of GRP2 is read and its directory section is then processed as follows:
      - The status of PPY is changed to saved.

Figure 14 (Part 2 of 3). Results of Issuing the SAVESEG Command

        - 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 exist for PPU and GRP2, and class A files exist for GRP1, PPY, PPX, and PPW.

saveseg ppu is issued.

- The SAVESEG command determines if the saved segment is a segment space:
  - SAVESEG reads the descriptor record from the class S file for PPU. In this case, a flag field indicates PPU is not a segment space but rather a member saved segment.
  - The pages associated with PPU (as defined by the previous DEFSEG command) and their keys are copied to a system data file.
  - The file for PPU is changed from class S to class A.
  - 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.
  - Since PPU is a member saved segment, the directory is updated:
    - For each segment space entry associated with PPU, the respective class S file is processed. PPY has only one segment space entry: GRP2. The descriptor record of GRP2 is read and its directory section is then processed as follows:
      - The status of PPU is changed to saved.
      - 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. So, the class of the file associated with GRP2 is changed to A. Since 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.
      - 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 once its class becomes A or R.

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

Figure 14 (Part 3 of 3). Results of Issuing the SAVESEG Command

Keeping Backup Copies of Saved Segments

VM/XA SP retains saved segments in the event of a system cold start. However, because VM/XA SP 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 SPTAPE command enables you to do this. For more information on SPTAPE, see *VM/XA SP CP Command Reference* or *VM/XA SP Real 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 the SEGMENT PURGE macro. Otherwise, use DIAGNOSE X'64' to purge the saved segment. (See "Purging Saved Segments from Your Virtual Machine" on page 57.)

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 will become 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, issue 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 logoff or re-IPL
- All virtual machines using the saved segment release it (with a DIAGNOSE X'64' PURGE or a SEGMENT PURGE).
- The system is IPLed
- All virtual machines using the saved segment load a new saved segment which 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 and include 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 *VM/XA SP CP Command 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 nnnnnnK  begpag endpag  --  c  nnnnn  N/A      N/A
                       nnnnnnM
```

In this display:

FILE
Is the spool ID of the file.

FILENAME
Is the name of the saved segment.

FILETYPE
Is one of the following:

NSS	A named saved system
CPNSS	A CP system service named saved system
CPDCSS	A CP system service saved segment
DCSS	A saved segment defined without the SPACE operand
DCSS-S	A segment space for which members are defined
DCSS-M	A member of a segment space.

MINSIZE
Is the minimum storage size of the virtual machine into which a named saved system can be IPLed. This field does not apply to saved segments.

BEGPAG
Is 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
Is 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
Is the type of access (ER, EW, EN, SR, SW, SN, SC) allowed to a page range of a named saved system. The TYPE field does not apply to segment spaces; therefore, "—" appears in this field.

CL

Describes the current state of the file:

A

Indicates that this system data file is in the unrestricted available state. This means the system data file has been defined and saved. To determine whether the system data file is in use, examine the #USERS field in the response from the QUERY NSS command with the MAP option.

P

Indicates that this system data file is in the pending purge state. This means the PURGE NSS command has been issued for the name of this NSS or DCSS but virtual machines are accessing it. No new users can access this NSS or DCSS. It is purged when the last virtual machine releases the NSS or DCSS, or during the next system IPL or RESTART.

R

Indicates this system data file has restricted access and is in the available state. This means the system data file has been defined (with the RSTD option) and saved. Access to a restricted NSS or DCSS requires a NAMESAVE directory statement. To determine whether the system data file is in use, examine the #USERS field in the response from the QUERY NSS command with the MAP option.

S

Indicates this system data file is in "skeleton" format. This means that a DEFSEG or DEFSYS command has been executed for the system data file. The SAVESEG or SAVESYS can now be executed to complete this system data file.

#USERS

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

PARMREGS

"N/A", since this is not applicable to saved segments

VMGROUP

"N/A", since this is not applicable to saved segments.

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
0465     TSTSPACE     DCSS-S      A

NONE

FILE     FILENAME     FILETYPE    CLASS
0465     MEMBER01     DCSS-M      A

USERA   USERB   USERC   USERD   USERF   USERG

FILE     FILENAME     FILETYPE    CLASS
0465     MEMBER02     DCSS-M      A

USER1    USER2        USER3
```

In the example above, USER1, USER2, and USER3 have loaded (by name) MEMBER02. USERA, USERB, USERC, USERD, USERF, and USERG have loaded MEMBER01.

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

For further examples of the QUERY NSS command, see "Examples of Segment Spaces" on page 47.

Installing Applications in Saved Segments

This section tells you how to install an application in a saved segment. It discusses segment packing, and also 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.

- If you want to load a program at the default load address, load it in System/370 mode, where the default is X'20000'. In 370-XA mode, storage management determines the default load address, which may vary.
- When you install CMS at the default locations (see Figure 11 on page 27), it uses segments E and part of F. All of F 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 for a page allocation table when CMS is defined in the E and F segments. (See "Planning Where to Save a Saved Segment" on page 24.)
- 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 purges the old version, issues the new DEFSEG commands, installs the products, and issues the SAVESEG commands.

Fitting Applications Below the 16MB Line

If your installation has a large number of applications which all must run below the 16MB 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 16MB 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 impact performance somewhat, but it will help you ease your storage constraints.
- Some installations require the use of the segments where CMS normally (by default) resides: segments D, E, and F. You can use these segments if you install CMS at a different (non-default) location. For example, if you install CMS in the 400 and 500 segments, you will free up the D, E, and F segments. CMS may still need part of a third segment, in this case the top of the 300 segment, for a page allocation table and other control blocks. For an example of a saved segment environment where CMS is defined at a non-default location, see Figure 31 on page 78.

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/XA SP allows you to define **segment spaces** into which you can pack multiple applications. A segment space is on a megabyte boundary, but each application is stored on a page boundary. Thus, you can store many more programs in a given area without wasting storage. This is particularly important to you when running application programs in 370 mode because you need to fit all your applications below the 16MB line. Once you exploit VM/XA SP and can address licensed programs above the 16MB line, you do not need to store licensed programs so tightly; 1MB 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 1MB segment. So, if you have a licensed program, such as PROFS, which requires both shared and exclusive code, you can store both parts in the segment space but you must store them in separate 1MB segments.

Figure 15 shows ISPF/PDF, ISPF, and PROFS all stored in one segment space which spans three 1MB segments (from the beginning of megabyte 4 to the end of megabyte 6). In this case, ISPF/PDF, ISPF, and PROFS are all considered **member saved segments** of this segment space. Note that the boundaries of the segment space are rounded to megabyte boundaries.

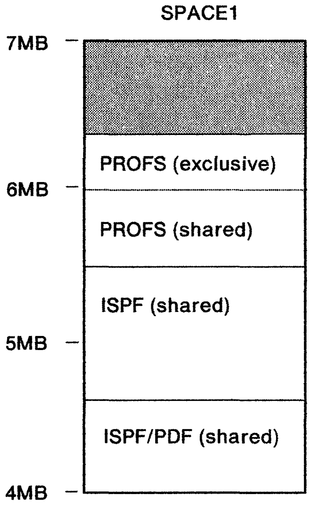


Figure 15. Using a Segment Space to Store Applications

Using a DCSS Versus Using a Segment Space

When to Use a Segment Space: The following are some guidelines for when to pack programs into segment spaces, instead of defining a DCSS for each program:

- If more than one member segment will fit in a 1MB segment space
- If your system is constrained for virtual storage below the 16MB line
- If your system has a "family" of related programs (for example, SQL and QMF) that are usually 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 66.

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

- If the size of a program is exactly on a megabyte boundary or slightly smaller. You will eliminate any overhead caused by using segment spaces.
- If you can fit all your segments below the 16MB line without packing them into segment spaces, you can eliminate the overhead.
- If your segments reside above the 16MB line, it is unlikely that your system is constrained due to the large number of segments.

For examples of DCSS definitions, see "Defining DCSSs" on page 65.

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 megabyte boundary, it may *not* be worth adding additional programs to round out the unfilled last megabyte. 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 2MB. If you install GDDM, you must evaluate whether you should create a segment space with GDDM and some other product which does not go beyond the third megabyte boundary, or whether to define a DCSS for GDDM and put the other product somewhere else. If the other product fits in the remainder of the third megabyte 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 (non-shared) pages can not 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 1MB 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. A 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 resave CMSDOS, but you can use SAMERANGE for the other saved segments rather than resaving them all.

  Note, however, the SAMERANGE operand can not 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 DCSS or segment space that is also 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 4MB or less virtual machine if they are not using the SEGMENT command or macro. If they use the SEGMENT command or macro, the virtual machine could be 4MB or less, or greater than 12MB, but it cannot be between 4MB and 12MB.

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 76.

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 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 DCSSs 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. For example, look at the storage layout in Figure 16 on page 49. 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 SPACE4 (PROFS) while SPACE2 is loaded, all of SPACE2 is removed from the user's address space.

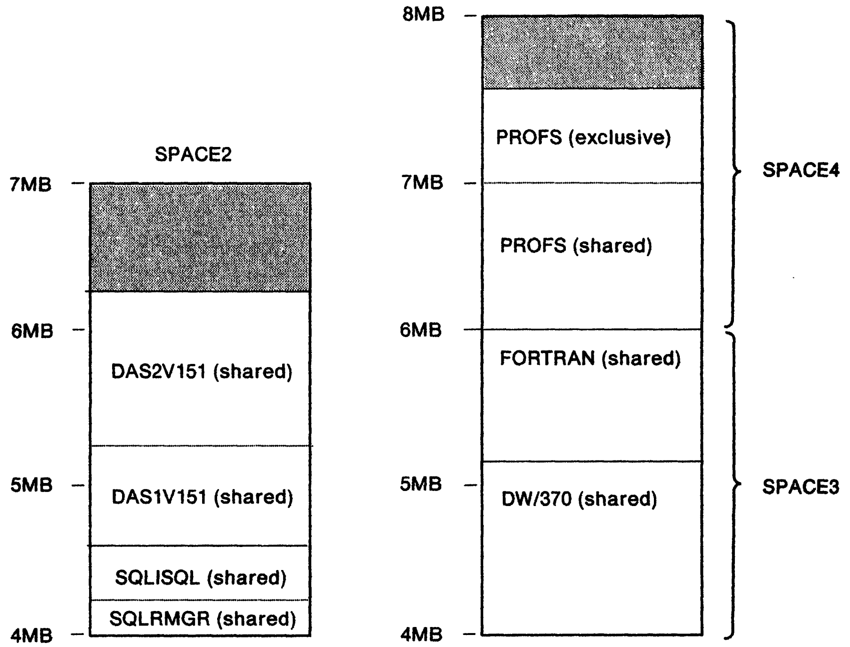


Figure 16. Using Segment Spaces to Overlay Applications

DAS1V151 and DAS2V151 are saved segments associated with 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 QMF
- Use SQL/DS with AS
- Do not use QMF with AS.

You can pack these products together in the following manner:

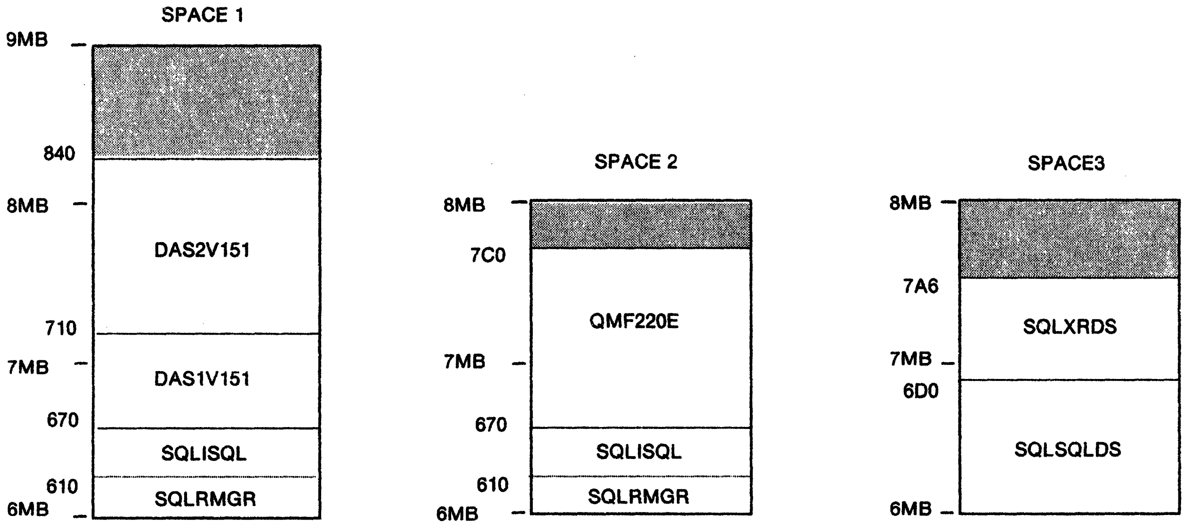


Figure 17. Installing SQL with Overlays

Although you need two segments for the SQL database machine (SQLSQLDS and SQLXRDS) 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 17.

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 17, if you have more QMF users than AS users, define SQL in SPACE2 first (and use the SAME operand to define SQL in SPACE1).

When setting up overlays, a map of your DMKSNT ASSEMBLE file can help you determine which programs overlap in your current environment.

For an example of defining the overlays shown in Figure 17, see "Defining Overlaid Segment Spaces" on page 67. To see a sample storage mapping that includes overlays, refer to "Setting Up Your Storage Layout" on page 76.

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 18 on page 51 shows PPT3, PPT4, and PPT5 — components of the program PPT — defined in separate DCSSs. These three DCSSs are all defined in the 7MB to 8MB 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.

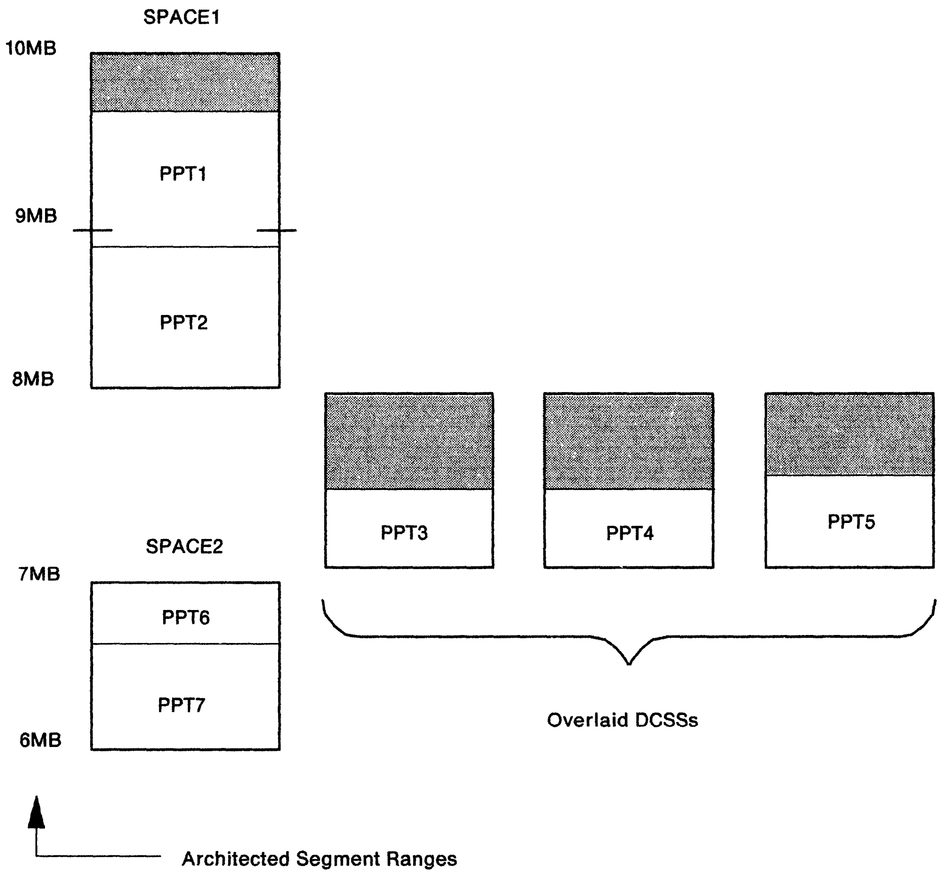


Figure 18. DCSSs as Overlays

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 19 on page 52, SPACE1, SPACE2, and SPACE3 are each segment spaces. Each segment space contains a set of common members plus one unique member.

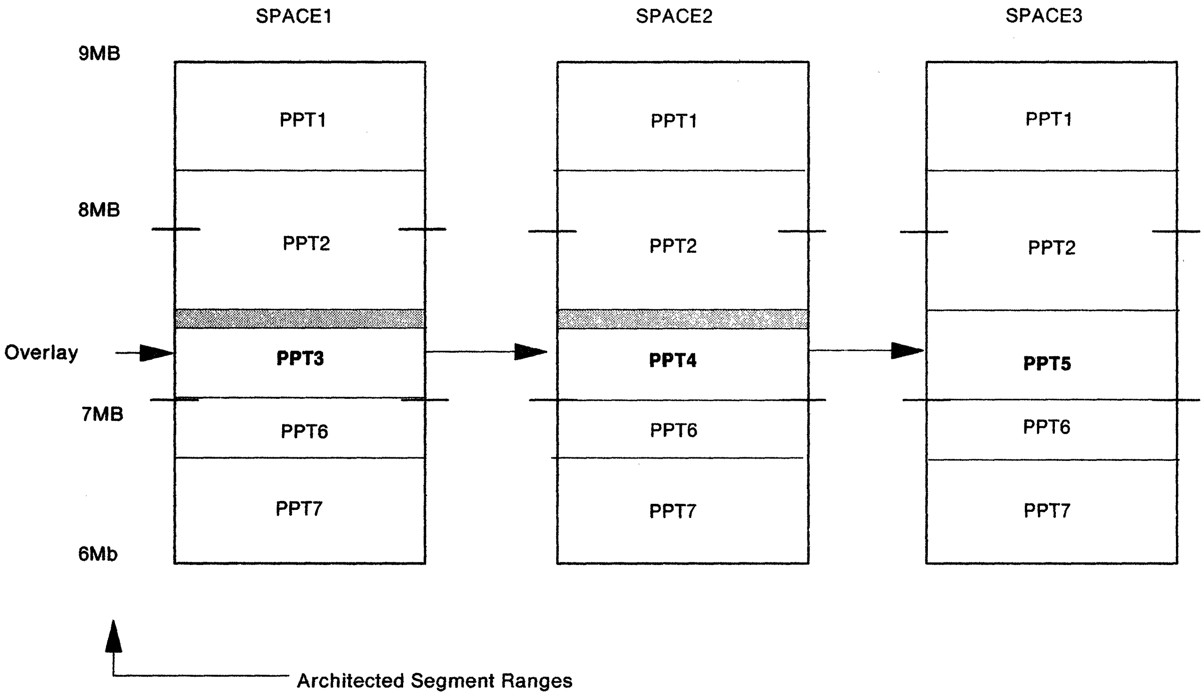


Figure 19. Segment Spaces as Overlays

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 under "Defining Overlaid Segment Spaces" on page 67.

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 application. Figure 20 on page 53 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 19 except that:

- In Figure 20 on page 53, the segment spaces represent multiple applications rather than just one.
- The situation in Figure 19 requires that all common code be in each segment space, while in Figure 20 on page 53 no code is duplicated.

The configuration in Figure 20 on page 53 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 66.

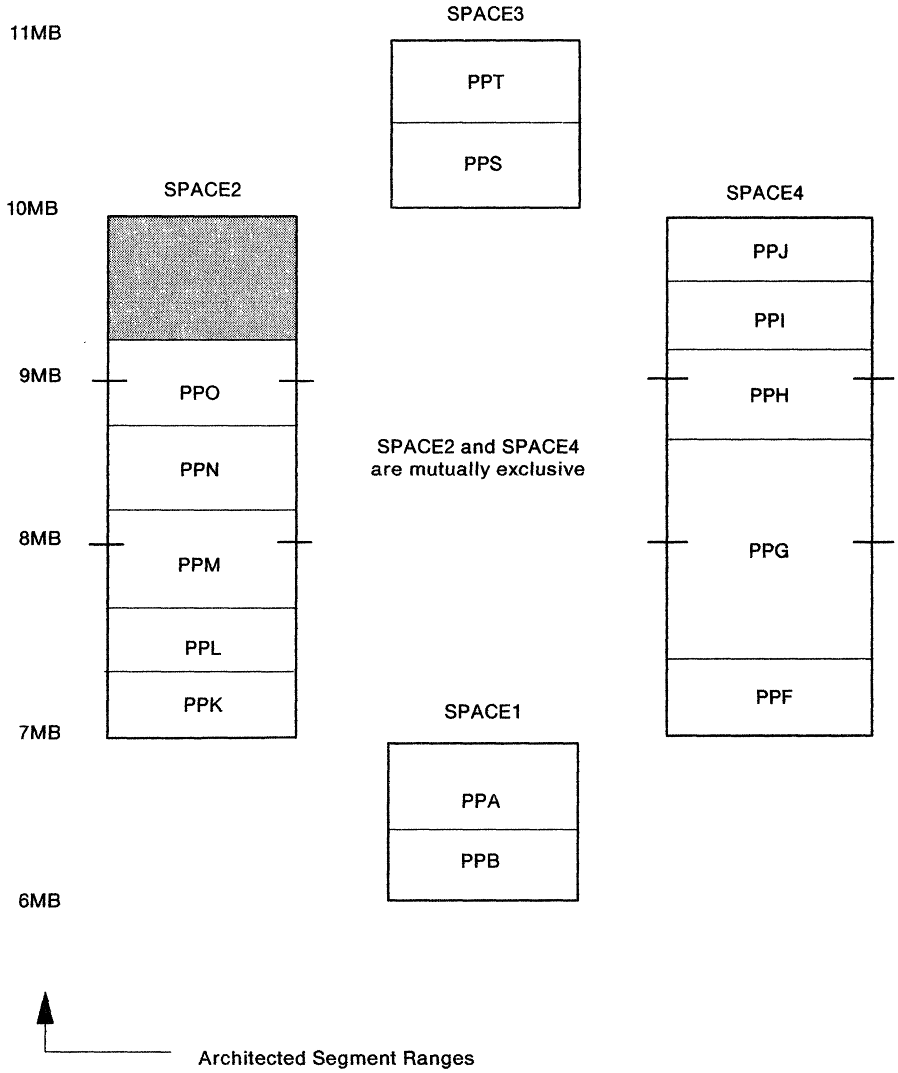


Figure 20. Mutually Exclusive Segment Spaces as Overlays

Redefining Saved Segments

At some point, you may need to redefine some of your saved segments due to service upgrades or a product's ability to exploit 370-XA architecture; that is, the product's ability to be loaded above the 16MB line. When you redefine a saved segment that was residing below the 16MB line to now be above the 16MB 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.

For an example of redefining an existing saved segment, see "Replacing an Existing Member of a Segment Space" on page 69.

The following charts are provided to assist you in redefining saved segments. These charts indicate whether a saved segment definition with the DEFSEG command will be successful (indicated by "yes") or unsuccessful ("no") based on the existence of a saved segment with the same name as that being defined. The information in each chart refers to the characteristics (class and type) of any *existing* saved segments.

Table 2. 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

The chart above shows, for example, that you cannot define a DCSS if a class A or R segment space with the same name already exists.

Table 3. 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

The chart above shows, for example, that you can define a member if a class P DCSS with the same name already exists.

Table 4. 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

The chart above 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.

Chapter 4. Application Programmer Considerations

This chapter presents information that helps application programmers use saved segments.

This chapter tells you how to:

- Determine where saved segments should reside
- What the differences are between the SEGMENT command (and macro) and DIAGNOSE X'64'
- Reserve space for a saved segment
- Load a saved segment
- Purge a saved segment from your virtual machine
- Display information on a saved segment
- Protect a saved segment with a storage protection key.

Using Saved Segments from Your Virtual Machine

Once a saved segment has been defined with the DEFSEG command, the application programmer is responsible for initializing the content of the saved segment from a virtual machine. The content of the saved segment is usually application programs or other data. Then the SAVESEG command can be issued (generally by the system programmer or by an EXEC) to make the saved segment active. This chapter discusses what the application programmer needs to know to use saved segments from a virtual machine.

Where Saved Segments Reside

The storage addresses you assign to a saved segment can be within or outside of the virtual storage currently available to the virtual machine that attaches the saved segment. However, saved segments are designed to execute *outside* the addressing range of a user's virtual machine. When a saved segment is loaded, the virtual machine can reference any area within the range of the saved segment. The virtual machine cannot address areas above or below the saved segment that are also outside the virtual machine storage definition.

Attaching a saved segment that is defined outside a virtual machine is a way to increase the storage of the virtual machine without actually redefining its virtual storage (with the DEFINE command). For example, a 4-megabyte virtual machine that attaches a saved segment defined in the 4- to 5-megabyte range now has access to five megabytes of virtual storage.

The following sections describe how a virtual machine can use the CMS SEGMENT command and macro and DIAGNOSE X'64' to access saved segments and to manage segment spaces.

Saved Segment Restrictions

Before you work with saved segments from a virtual machine, note the following restrictions:

- You cannot save a saved segment from a V=R virtual machine, from a V=F virtual machine, or from a virtual machine for which you have defined multiple virtual processors.
- You cannot load a saved segment into a V=R or V=F virtual machine.

Differences between SEGMENT and DIAGNOSE X'64'

The SEGMENT command and SEGMENT macro may be used instead of DIAGNOSE X'64'. However, existing DIAGNOSE X'64' interfaces have not changed, and applications using them continue to work.

The following list describes the differences between saved segment support (in particular, the SEGMENT command and the SEGMENT macro) and the DIAGNOSE X'64' instruction.

1. The SEGMENT command and the SEGMENT macro can load a saved segment even if the segment resides within the virtual machine. Therefore, programs that use the SEGMENT command or the SEGMENT macro do not need to check the VMDSSIZE against the starting address of the saved segment to determine whether the segment can be used.

   For example, you can use the following code to load a segment:

```
         SEGMENT LOAD,NAME=SEGNAME
         C       R15,=F'12'        Already loaded?
         BE      LOADED            Yes, go process
         BH      LOADERR           Higher return codes are errors,
*                                  lower return code is good load.
```

2. You can use the SEGMENT RESERVE command to create a segment space so that a subsequent SEGMENT LOAD command or a DIAGNOSE X'64' may be safely issued. SEGMENT RESERVE does not actually load a saved segment into storage. It reserves a segment space so that you can safely issue a SEGMENT LOAD command or a DIAGNOSE X'64' knowing that CMS has not used that address range for something else. *DIAGNOSE X'64', on the other hand, does not check when it loads a saved segment to see if existing data would be overlaid unless you specify the LOADNOLY operand.*
3. The SEGMENT macro returns information about the loaded segment in general registers 0 and 1, rather than in the Rx and Ry registers used by DIAGNOSE X'64'. Also, the SEGMENT command and the SEGMENT macro use return codes rather than condition codes to indicate the success or failure of an operation.
4. DIAGNOSE X'64' has additional functions not available with SEGMENT support (such as the load no overlay feature). For more information, see "Using DIAGNOSE X'64'" on page 62.

Reserving Space for Saved Segments

In CMS 5.5, saved segments can be in your virtual machine's address space. For saved segments that are *not* loaded immediately after IPL, consider reserving space for the saved segment. If you do not reserve space for the saved segment, other programs can use the storage. If the required storage is occupied when you try to load a saved segment, the load fails.

You can use the SEGMENT RESERVE command to reserve space. Reserving a segment space:

- Allows you to ensure that your applications can load saved segments in the storage they specify
- Eliminates the possibility of saved segments overlaying or being overlaid by portions of CMS.

To reserve segment spaces for all users, add the SEGMENT RESERVE command to the SYSPROF (system profile) EXEC. To reserve segment space only for your virtual machine, issue the SEGMENT RESERVE command from your virtual machine.

Loading Saved Segments

Use the SEGMENT LOAD command to load a SEGMENT into storage. SEGMENT LOAD reserves a segment space (if one is not already reserved) and loads a saved segment into it.

Example 1: To load a segment named MYSEG that survives abend processing and can be shared by any user, enter:

```
segment load myseg system share
```

Note that SHARE is a default value and can be omitted.

Example 2: To load a segment named MYSEG that does not survive abend processing and cannot be shared by other users, enter:

```
segment load myseg noshare user
```

Purging Saved Segments from Your Virtual Machine

Use the SEGMENT PURGE command to purge a saved segment from a segment space. (SEGMENT PURGE also releases the storage held by segment spaces that were created by SEGMENT LOAD.)

For example, to purge MYSEG, enter:

```
segment purge myseg
```

Releasing Segment Spaces

Use the SEGMENT RELEASE command to release the storage reserved for a saved segment.

Displaying Information about Saved Segments

Use the QUERY SEGMENT command to display information about saved segments that were loaded using the SEGMENT command or macro. For example:

```
query segment profs
```

might return a response similar to the following:

```
Space    Name     Location Length   Loaded    Attribute
PROFS    PROFSEG  00800000 00020000 YES       USER
```

To display information on all the currently defined segment spaces, enter:

```
query segment *
```

In response, CMS returns something similar to the following:

```
Space    Name     Location Length   Loaded    Attribute
PROFS    PROFSEG  00800000 00020000 YES       USER
GDDM     GDDM     00900000 00040000 NO        SYSTEM
```

The SPACE field's value is different from the name fields value only for members of a segment space. In the example above, PROFSEG is a member of the segment space called "PROFS".

Saved Segment Storage Protection

To ensure that a saved segment is protected from inappropriate use, a storage protection key can be assigned to the segment. The key should be set to something other than X'F' (decimal 15) before you save a segment. You can use the CMS SETKEY command to assign the storage protection key for a segment.

When CMS EXECs install a segment, the EXEC may assign a storage protection key to the segment; for example:

- DOSGEN assigns storage protection key X'D' to the CMSDOS segment.
- SAMGEN assigns storage protection key X'F' to the CMSBAM segment.

To change the storage protection key, or to assign a storage protection key, use the CMS SETKEY command. For information on the SETKEY command, see *VM/XA SP CMS Command Reference.*

Tips for Using Saved Segments in a Virtual Machine

Figure 21 on page 59 shows a saved segment environment with four CMS virtual machines (two running in 370-XA mode and two running in System/370 mode) and two segment spaces (SPACE1 and SPACE2). This figure illustrates some important points about loading saved segments:

- User 2 and User 3 can safely load SPACE1 because SPACE1 is defined outside the virtual storage range of these two users.
- SPACE1 is defined within the virtual storage range of User 4. Therefore, User 4 should use the SEGMENT command or macro to load this segment space and thereby ensure access to the program operating in SPACE1. If a virtual machine's address range exceeds the beginning address of a saved segment, and the virtual machine loads the saved segment with DIAGNOSE X'64', the virtual machine may not be able to access the data in the saved segment. See

"Using the SEGMENT Command and Macro" on page 60 for more information.

- User 1 cannot load SPACE1 without overlaying its own page allocation table.
- SPACE2 is defined above the 16MB line and therefore can only be used by User 3 and User 4. This is discussed in more detail in "370-XA Mode Virtual Machine Considerations" on page 60.

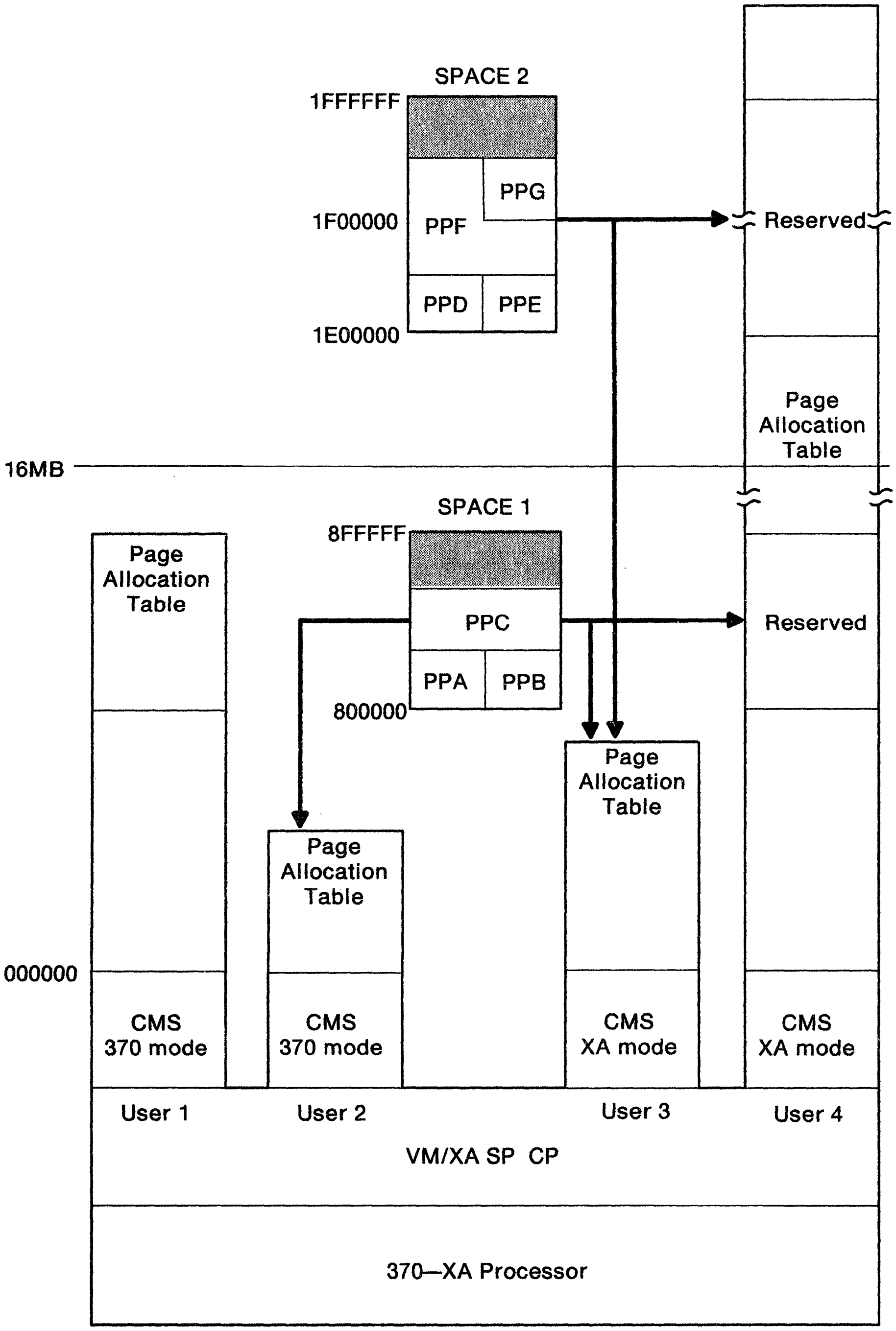


Figure 21. Loading Segment Spaces

Using the SEGMENT Command and Macro

The SEGMENT command and macro help to prevent the overlay situation that can occur in 370-XA mode virtual machines with a large virtual storage size. This interface allows large virtual machines (those with an address range that overlaps a saved segment definition) to reserve space in the virtual machine for the saved segment prior to loading it. Should the space required for the saved segment be currently allocated for some other CMS activity, a return code describing this situation is returned. In this case, you should suspend the loading of the saved segment into the virtual machine until the storage required is available. Once the area has been reserved with the SEGMENT command or macro, the saved segment can be loaded. Once loaded, the area occupied by the saved segment will not be allocated for any other purpose. In Figure 21 on page 59, User 4 can use the SEGMENT interface to avoid an overlay situation when loading SPACE1.

You may want to have SEGMENT RESERVE issued from your PROFILE EXEC or some other EXEC that reserves and loads a saved segment. SEGMENT RESERVE is usually used in conjunction with DIAGNOSE X'64'.

370-XA Mode Virtual Machine Considerations

CMS can now execute in 370-XA mode virtual machines with more than 16MB of virtual storage. However, the majority of the products that operate in a saved segment can only execute from within saved segments defined below 16MB. To be able to execute above the 16MB line, a product must be able to operate in a 370-XA mode virtual machine and execute in 31-bit addressing mode.

If a saved segment is defined above the 16MB line, its services are only available to 370-XA mode virtual machines. A System/370 mode virtual machine is limited to 24-bit addressing and therefore is unable to transfer control to a saved segment operating above the 16MB line. In Figure 21 on page 59, SPACE2 is defined above the 16MB line and is therefore only available to the 370-XA mode virtual machines of User 3 and User 4. It is recommended that you define saved segments for products wishing to exploit 370-XA architecture to have different names than those used by System/370 virtual machines. This allows the existing System/370 user set to still receive services to the extent of the product's System/370 support.

You can only define two saved segments for one application if the application allows you to do this. Some applications (GDDM for example) determine which saved segment to load based on the user's virtual machine mode. If you have an application that does not do this, you may need to set up EXECs that determine which version to load.

Applications with System/370 Architecture Sensitivity

Programs that are coded to expect a 64KB segment need to be modified for VM/XA SP. For example, suppose you have an application called PP that requires a shared segment for some of its code and an exclusive segment for the rest of its code. That is, the program PPES (a component of PP) requires a shared segment, and PPEW (a second component of PP) requires an exclusive write segment, as shown in Figure 22 on page 61.

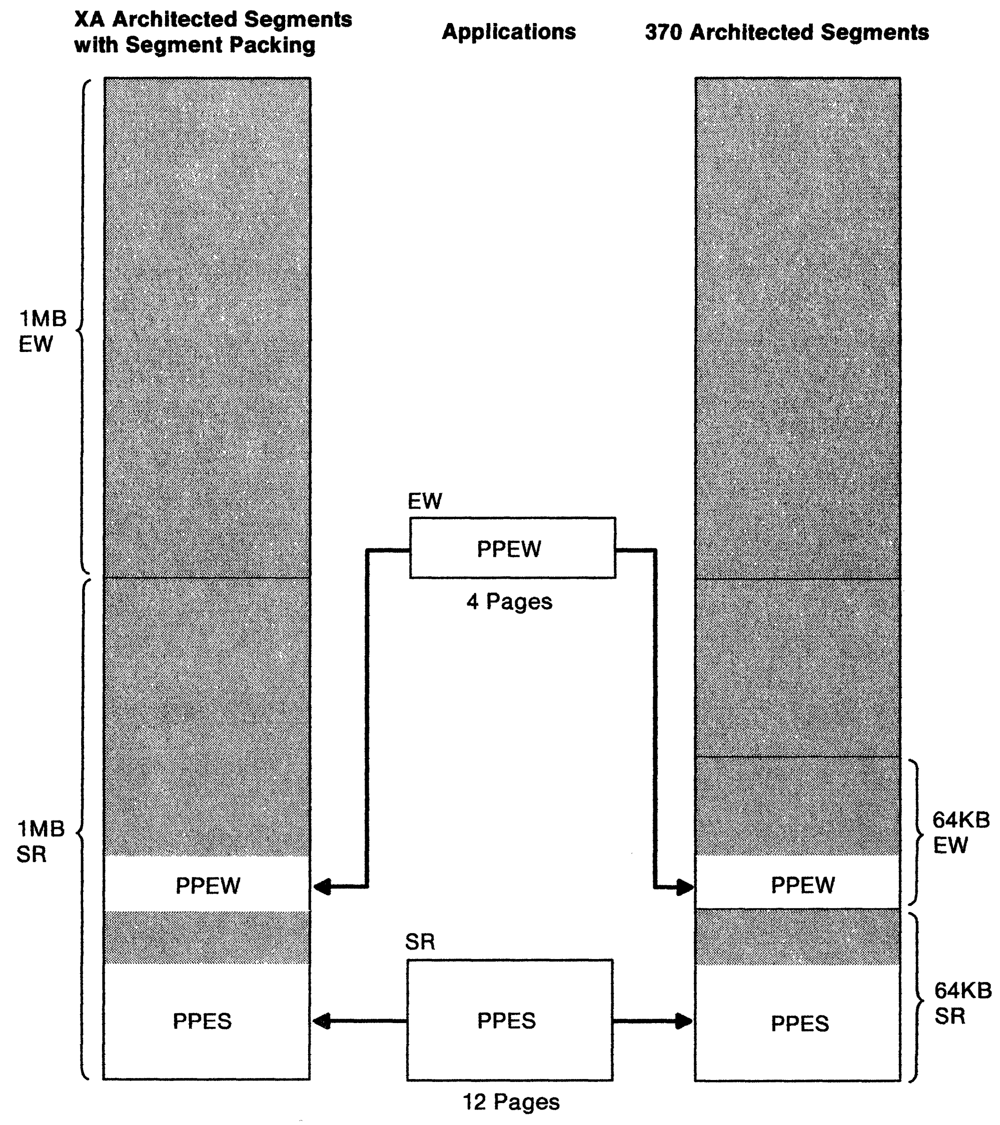


Figure 22. Applications with System/370 Architecture Sensitivity

You plan to load the applications with the following section of a VM/SP installation EXEC:

```
LOAD PPES (ORIGIN A00000
INCLUDE PPEW (ORIGIN A10000
```

When running under VM/SP, this EXEC loads PPES into the saved segment defined as SR (shared). The EXEC then loads PPEW starting at address A10000, placing PPEW in the saved segment defined as EW (exclusive write).

Under VM/XA SP, the EXEC tries to load PPES and PPEW into the same storage locations as it did under VM/SP. Due to the larger segment sizes in VM/XA SP, this would mean that PPEW is placed in the SR saved segment with PPES as shown in Figure 22. Although the DEFSEGs for PPEW and PPES would work, when you load PPEW it would overlay PPES, and you would not have the use of PPES. (Because of the overlay situation, DIAGNOSE X'64' LOADNOLY will not work.)

This EXEC needs to be modified to take into account the fact that segments in VM/XA SP are one megabyte in size.

Using DIAGNOSE X'64'

In VM/XA SP, issuing a DIAGNOSE X'64' against a segment space or a member saved segment produces results that are slightly different from those in VM/XA SF or VM/SP HPO. The following sections highlight these differences:

LOADSR/LOADNSHR/LOADOVLY: When a member of a saved segment is attached (using LOADSR, LOADNSHR, or LOADOVLY) to a virtual machine, the entire storage range occupied by the segment space is addressable by the virtual machine. However, only the beginning and ending addresses related to the member are returned to the virtual machine.

In Figure 23 on page 64, if User 1 issues a DIAGNOSE X'64' LOAD*xxxx* of the saved segment PPM, the following things happen:

- User 1 can reference the range of addresses of the SPACE2 segment space (700000—9FFFFF).
- A CP control block is established indicating that User 1 is attached to the member saved segment PPM.
- The addresses of PPM (7B0000—81FFFF) are returned in the Rx and Ry registers of User 1.

Similarly, if a virtual machine attaches a saved segment by specifying the name of the segment space, the storage range occupied by the saved segment is addressable by the virtual machine. All members that make up the segment space are attached to the virtual machine. The address of the member with the lowest address is returned in the Rx register. The Ry register contains the address of the last byte of the megabyte in which the member with the highest address resides.

In Figure 23 on page 64, if User 1 issues a DIAGNOSE X'64' LOAD*xxxx* of the segment space SPACE3, the following things happen:

- User 1 can reference the range of addresses of the SPACE3 segment space (A00000—AFFFFF).
- CP control blocks are established indicating that User 1 has loaded SPACE3.
- The addresses of SPACE3 (A00000—AFFFFF) are returned in the Rx and Ry registers.

When a virtual machine loads a member of a segment space, the virtual machine can access all the members of the space. However, to access other members predictably, the virtual machine must use DIAGNOSE X'64' or the SEGMENT command or macro for each member.

FINDSEG: For member saved segments, the FINDSEG function returns the beginning and ending addresses of the member. In Figure 23 on page 64, if User 2 issues a DIAGNOSE X'64' FINDSEG of saved segment PPN, the addresses of PPN (820000—8AFFFF) are returned in the Rx and Ry registers.

For segment spaces, the FINDSEG function returns to the Rx register the address of the megabyte that contains the member with the the lowest page definition. The Ry register contains the address of the last byte of the megabyte in which the member with the highest page definition resides. In Figure 23 on page 64, if User 2 issues a DIAGNOSE X'64' FINDSEG of segment space SPACE2, the addresses of of SPACE2 (700000—9FFFFF) are returned in the Rx and Ry registers.

PURGESEG: For member saved segments, the PURGESEG function releases the control block associated with the member that was acquired on the corresponding LOAD*xxxx* function. If the purged member is the last member of a segment space loaded by this user, the entire segment space is removed from the user's address space.

If the storage occupied by the member is beyond the defined virtual machine storage size, that storage is still addressable by the virtual machine provided another member of the same segment space is still attached to the virtual machine.

In Figure 23 on page 64, suppose User 2 has previously attached members PPL and PPO. If User 2 then issues a DIAGNOSE X'64' PURGESEG of the saved segment PPO, the control block indicating that PPO is attached to User 2 is deleted.

If User 2 then issues a DIAGNOSE X'64' PURGESEG of the saved segment PPL, the following things happen:

- The control block indicating that PPL was loaded by User 2 is deleted.
- Since there is no other member of SPACE2 attached to User 2, the page table associated with SPACE2 is deleted from the User 2 virtual machine segment table entries.
- The User 2 virtual machine can now use the 7MB, 8MB, and 9MB ranges for its own activities.

If the storage occupied by the saved segment was beyond the defined virtual machine storage size, that storage is no longer addressable by the virtual machine.

In Figure 23 on page 64, suppose User 2 has previously attached members PPL and PPO. If User 2 then issues a DIAGNOSE X'64' PURGESEG of SPACE2, nothing is purged unless a corresponding LOAD*xxxx* for SPACE2 was issued. (The PURGESEG would then return a condition code of 1.) If a corresponding LOAD*xxxx* function was previously issued only for SPACE2, then:

- The control blocks indicating that SPACE2 was loaded by User 2 are deleted.
- Since there are no members of SPACE2 which were explicitly loaded by User 2, the page table associated with SPACE2 is removed from the User 2 virtual machine segment table entries.
- The User 2 virtual machine can now use the 7-, 8-, and 9MB ranges for its own activities.

If the space was loaded by name and a member was also explicitly loaded, only the blocks indicating that the space is loaded are deleted. User 2 still has addressability to the space until the member is purged.

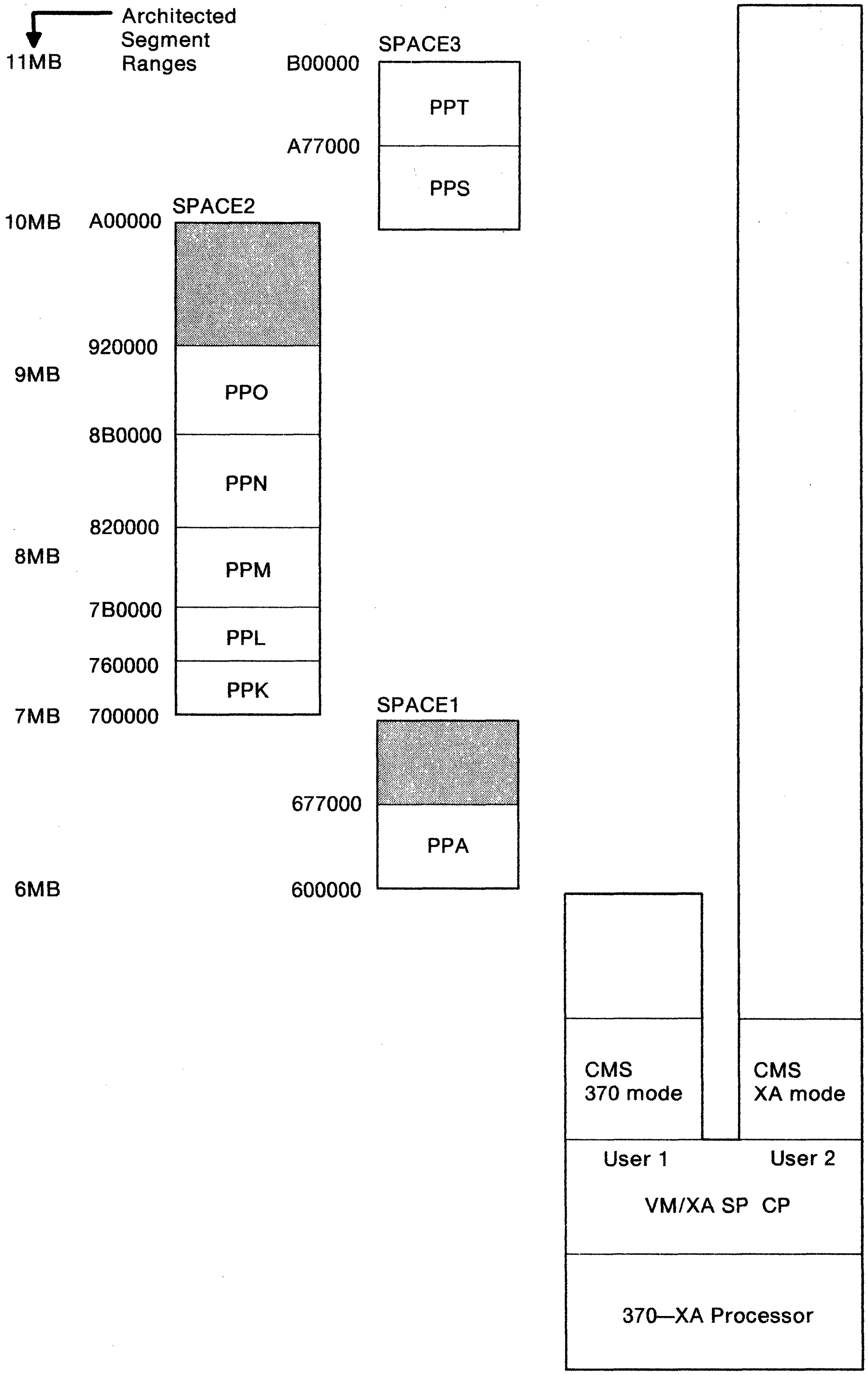


Figure 23. Segment Space Environment

Chapter 5. Defining Saved Segments — Examples

This chapter gives examples of saved segment definitions.

This chapter tells you how to:

- Define a DCSS
- Define a DCSS with both shared and exclusive page ranges
- Define overlaid DCSSs
- Define segment spaces
- Define overlaid segment spaces
- Add a member to an existing segment space
- Replace an existing member of a segment space
- Save a new version of an existing member
- Set up a typical saved segment layout.

Defining DCSSs

The following sequence of commands tells you how to define and save each DCSS shown in Figure 3 on page 5:

1. Define each DCSS:

   ```
   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 DCSS you have defined, issue the SAVESEG command to save the DCSS (if the installation EXECs have not already done so):

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

Defining a DCSS with Both Shared and Exclusive Page Ranges

Some licensed programs, such as PROFS, 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 PROFS to have a range of addresses that is shared and a range that is exclusive:

1. Issue:

   ```
   defseg profs 700-7ff sr 800-845 ew
   ```

   In the above example, the address range 700 – 7FF is defined as shared read-only; the address range 800 – 845 is defined as exclusive read-write.

2. Install PROFS with the appropriate installation EXEC.
3. If it has not already been done by the installation EXEC, issue the SAVESEG command for PROFS:

   ```
   saveseg profs
   ```

In the above example, PROFS is defined in a DCSS. If PROFS 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 tells you how to define and save each overlaid DCSS shown in Figure 18 on page 51.

1. Define each DCSS in the same 1-megabyte 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. Issue 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.

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

Defining a Segment Space

The following sequence of commands tells you how to define and save the segment space shown residing in SPACE2 in Figure 20 on page 53:

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. Then, if you have to make changes to your storage layout, it will be 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, issue 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
   ```

Notes:

1. The ending address of SPACE2 is rounded up to a 1-megabyte 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 20 on page 53 was larger, 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 will need 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 re-install 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 re-installed if PPK grows in size.

3. The segment space SPACE2 is not active until you issue the last SAVESEG command (SAVESEG PPO in this case). Therefore, you cannot use the programs in SPACE2 until you issue the last SAVESEG command.

Defining Overlaid Segment Spaces

The following sequence of commands tell you how to define and save the overlaid segment spaces SPACE1, SPACE2 and SPACE3 as shown Figure 17 on page 50.

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. Issue 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.

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. Issue 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
   ```

   Once you issue 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 shown in Figure 20 on page 53 and defined under "Defining a Segment Space" on page 66:

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. Issue the SAVESEG command for PPZ (unless your installation EXEC has already done so):

   ```
   saveseg ppz
   ```

The program PPZ is available to users who are not currently attached to any of the other programs in SPACE2 as soon as you issue the above SAVESEG command.

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 20 on page 53) 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. Issue 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 resaved 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: The above example will work in most cases. The following are more detailed examples that 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 shown in Figure 24 on page 70. In this figure and others like it in this chapter, 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. For example, in Figure 24 on page 70, SPACE1 has the spool ID 0001 and has M1, M2, and M3 as its members.

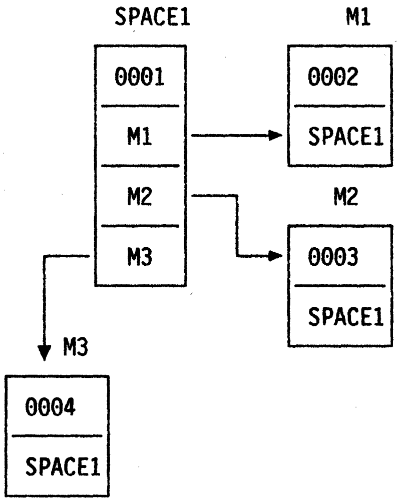


Figure 24. Initial Setup of a Segment Space

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

```
defseg m3 rangeinfo... space spacel
defseg ml same space spacel
defseg m2 same space spacel
```

Figure 25 shows the situation after these DEFINE commands have been issued. Note that M3 and SPACE1 have new spool IDs (005 and 006 respectively) which are class S files.

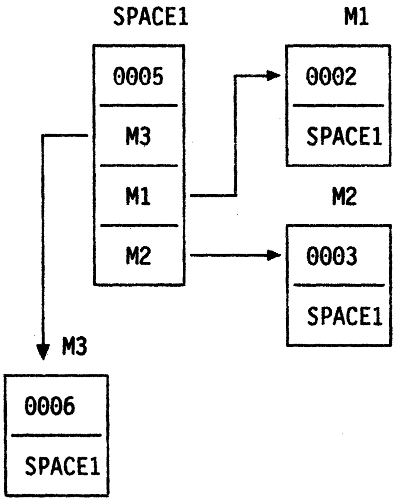


Figure 25. New Version of a Segment Space (DEFSEGs Complete)

Lastly, you need to issue 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, which causes 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 shown in Figure 26 on page 71. (The Ls refer to segment spaces, and the Ms refer to members.)

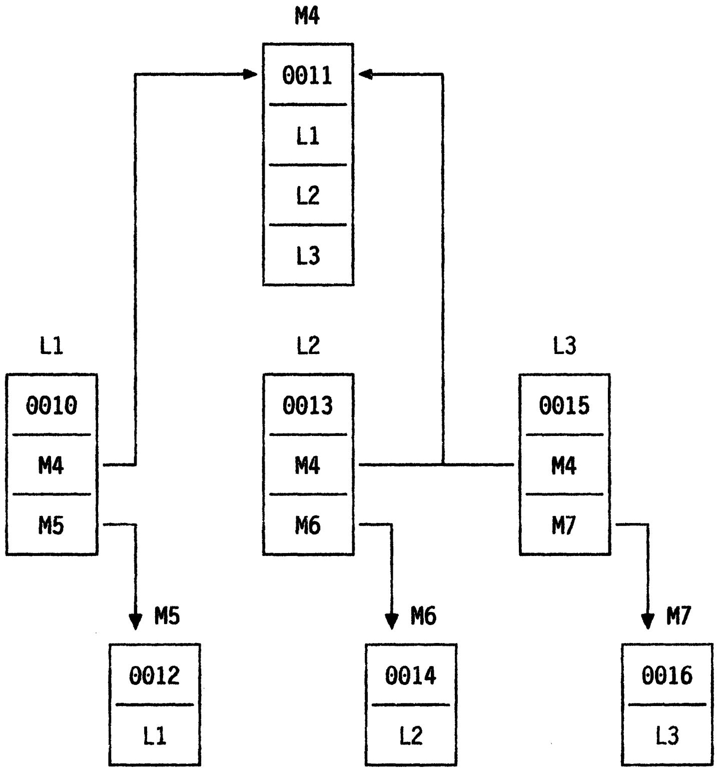


Figure 26. Replacing One Member of an Overlay — Initial Setup

To replace M6, define the new version of M6 and L2. This can be done by issuing several DEFSEG commands as follows:

```
defseg m6 rangeinfo... space 12
defseg m4 same space 12
```

Figure 27 on page 72 shows the situation after these define commands have been issued. Note that spool files 0017 and 0018 are class S files.

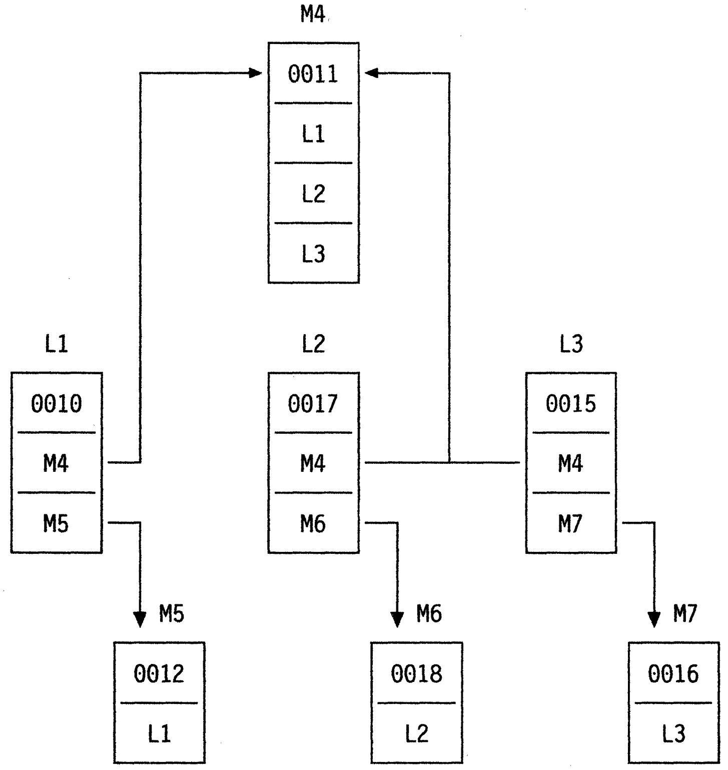


Figure 27. Replacing One Member of an Overlay (DEFSEGs Complete)

Lastly, you must issue 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 and causes the old class A versions to be purged.

Example 4 — Creating a New Version of a Common Member: This example explains how to create a new version of a member which is shared between several segment spaces.

After the initial segment spaces have been created, the environment looks like that shown in Figure 28 on page 73.

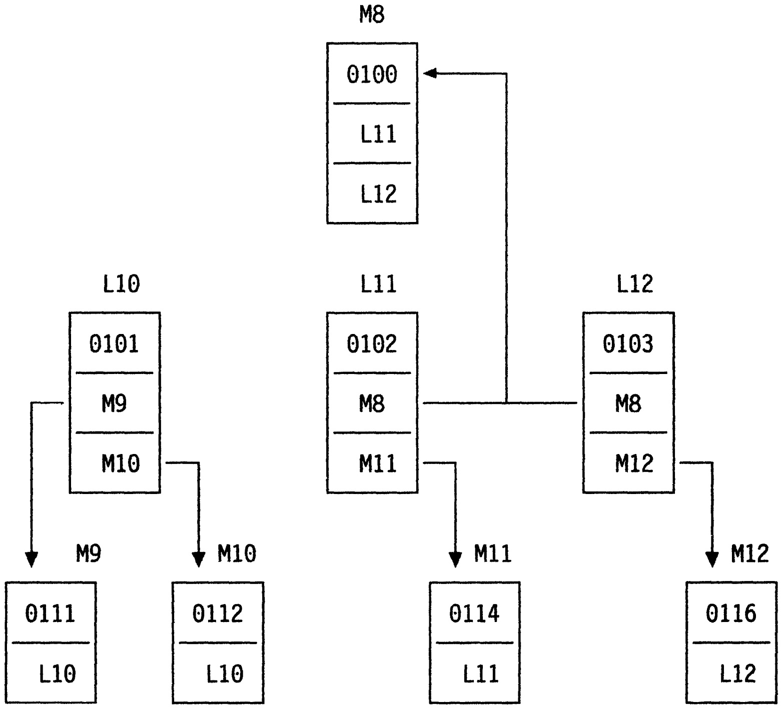


Figure 28. Replacing a Shared Member — Initial Setup

To replace M8, define the new version of M8, L11, and L12. This can be done by issuing several DEFSEG commands as follows:

```
defseg m8 rangeinfo... space 111
defseg m11 same space 111
defseg m8  same space 112
defseg m12 same space 112
```

Figure 29 on page 74 shows the situation after these define commands have been issued. Note that spool files 0117, 0118, and 0119 are class S files.

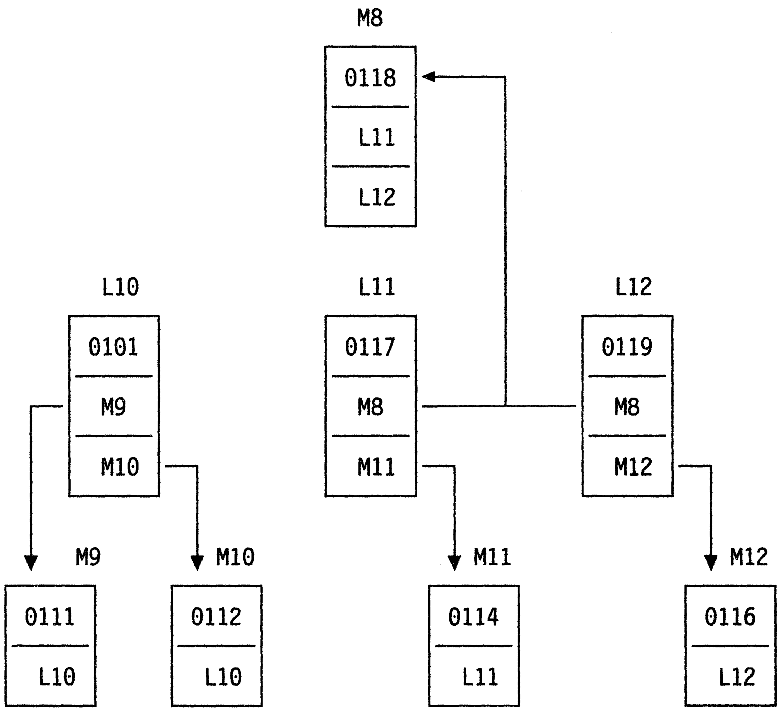


Figure 29. Replacing a Shared Member (DEFSEGs Complete)

Lastly, you must issue 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, and purges the old class A versions.

Updating the HELP Saved Segment

If you are using EXECs which require an active saved segment, consider the following examples.

The installation EXECs DCSSGEN and SAVEFD are used to install the CMSINST and HELP saved segments. These EXECs require that an active saved segment exists in order for a saved segment to be installed successfully.

This section gives two examples of updating the HELP saved segment without having to re-install CMSINST. Follow Example 1 if you are not changing the page ranges of the HELP saved segment. Follow Example 2 if you are installing HELP for the first time or if you are changing its page ranges.

Example 1 (no page ranges changed):

1. Check to see what saved segments currently exist:

   ```
   q nss name tester map
   FILE FILENAME FILETYPE MINSIZE BEGPAG ENDPAG TYPE CL #USERS PARMREGS
   0240 TESTER   DCSS-S     N/A   00C00  00C05   --  A  00000   N/A
   0241 HELP     DCSS-M     N/A   00C03  00C05   SR  A  00000   N/A
   0214 CMSINST  DCSS-M     N/A   00C00  00C02   SR  A  00000   N/A
   ```

   The segment space TESTER exists with two members, HELP and CMSINST. We want to update HELP.

2. Redefine the member containing HELP:

   ```
   defseg help c03-c05 sr space tester
   HCPNSD440I SEGMENT HELP    DEFINED SUCCESSFULLY IN FILEID 0243  .
   ```

3. Redfine the member that is not changing (CMSINST):

   ```
   defseg cmsinst same space tester
   HCPNSD440I SEGMENT CMSINST   DEFINED SUCCESSFULLY IN FILEID 0214  .
   ```

4. Check to see what saved segments now exist:

   ```
   q nss name tester map
   FILE FILENAME FILETYPE MINSIZE BEGPAG ENDPAG TYPE CL #USERS PARMREGS
   0240 TESTER   DCSS-S     N/A   00C00  00C05   --  A  00000   N/A
   0241 HELP     DCSS-M     N/A   00C03  00C05   SR  A  00000   N/A
   0214 CMSINST  DCSS-M     N/A   00C00  00C02   SR  A  00000   N/A
   0242 TESTER   DCSS-S     N/A   00C00  00C05   --  S  00000   N/A
   0243 HELP     DCSS-M     N/A   00C03  00C05   SR  S  00000   N/A
   0214 CMSINST  DCSS-M     N/A   00C00  00C02   SR  A  00000   N/A
   ```

5. Issue SAVEFD to save the data in the skeleton file:

   ```
   savefd save 999 jmu999 help
   DMSACP723I Z (999) R/O
   HCPNSS440I SEGMENT HELP   SAVED SUCCESSFULLY IN FILEID 0234 .
   ```

   In the SAVEFD command above, 999 is the disk where HELP resides, and jmu999 is the label of the disk.

6. Issue Q NSS again to see the results:

   ```
   q nss name tester map
   FILE FILENAME FILETYPE MINSIZE BEGPAG ENDPAG TYPE CL #USERS PARMREGS
   0242 TESTER   DCSS-S     N/A   00C00  00C05   --  A  00000   N/A
   0243 HELP     DCSS-M     N/A   00C03  00C05   SR  A  00000   N/A
   0214 CMSINST  DCSS-M     N/A   00C00  00C02   SR  A  00000   N/A
   ```

Example 2: You are installing HELP for the first time, or you are changing its page ranges.

1. Check to see what saved segments currently exist:

   ```
   q nss name tester map
   FILE FILENAME FILETYPE MINSIZE BEGPAG ENDPAG TYPE CL #USERS PARMREGS
   0244 TESTER   DCSS-S     N/A   00C00  00C05   --  A  00000   N/A
   0245 HELP     DCSS-M     N/A   00C03  00C05   SR  A  00000   N/A
   0214 CMSINST  DCSS-M     N/A   00C00  00C02   SR  A  00000   N/A
   ```

2. Create a "dummy" active version of TESTER with the desired changed pages. First, re-establish the TESTER segment space by redefining HELP, which has changed pages, and CMSINST, which does not.

   ```
   defseg help c03-c05 sr space tester
   HCPNSD440I SEGMENT HELP    DEFINED SUCCESSFULLY IN FILEID 0247  .

   defseg cmsinst same space tester
   HCPNSD440I SEGMENT CMSINST   DEFINED SUCCESSFULLY IN FILEID 0214  .
   ```

   Now, create the active version of HELP:

   ```
   saveseg help
   HCPNSD440I SEGMENT HELP    DEFINED SUCCESSFULLY IN FILEID 0247  .
   ```

   This purges any existing active versions of HELP and TESTER.

```
q nss name tester map
FILE FILENAME FILETYPE MINSIZE BEGPAG ENDPAG TYPE CL #USERS PARMREGS
0246 TESTER   DCSS-S     N/A   00C00  00C05   --  A  00000   N/A
0247 HELP     DCSS-M     N/A   00C03  00C05   SR  A  00000   N/A
0214 CMSINST  DCSS-M     N/A   00C00  00C02   SR  A  00000   N/A
```

3. Create the skeleton files for the real version of TESTER:

```
defseg help c03-c05 sr space tester
HCPNSD440I SEGMENT HELP     DEFINED SUCCESSFULLY IN FILEID 0249  .

defseg cmsinst same space tester
HCPNSD440I SEGMENT CMSINST   DEFINED SUCCESSFULLY IN FILEID 0214  .

q nss name tester map
FILE FILENAME FILETYPE MINSIZE BEGPAG ENDPAG TYPE CL #USERS PARMREGS
0246 TESTER   DCSS-S     N/A   00C00  00C05   --  A  00000   N/A
0247 HELP     DCSS-M     N/A   00C03  00C05   SR  A  00000   N/A
0214 CMSINST  DCSS-M     N/A   00C00  00C02   SR  A  00000   N/A
0248 TESTER   DCSS-S     N/A   00C00  00C05   --  S  00000   N/A
0249 HELP     DCSS-M     N/A   00C03  00C05   SR  S  00000   N/A
0214 CMSINST  DCSS-M     N/A   00C00  00C02   SR  A  00000   N/A
```

4. Issue SAVEFD (rather than the SAVESEG command) to save the new version of HELP:

```
savefd save 999 jmu999 help
HCPNSD440I SEGMENT HELP     DEFINED SUCCESSFULLY IN FILEID 0249  .

q nss name tester map
FILE FILENAME FILETYPE MINSIZE BEGPAG ENDPAG TYPE CL #USERS PARMREGS
0248 TESTER   DCSS-S     N/A   00C00  00C05   --  A  00000   N/A
0249 HELP     DCSS-M     N/A   00C03  00C05   SR  A  00000   N/A
0214 CMSINST  DCSS-M     N/A   00C00  00C02   SR  A  00000   N/A
```

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 used in this example are CMS, PROFS, ISPF, ISPF/PDF, GDDM, GDDM/PGF, DW/370, SQL, QMF, FORTRAN, VMAS, DCF and APL.

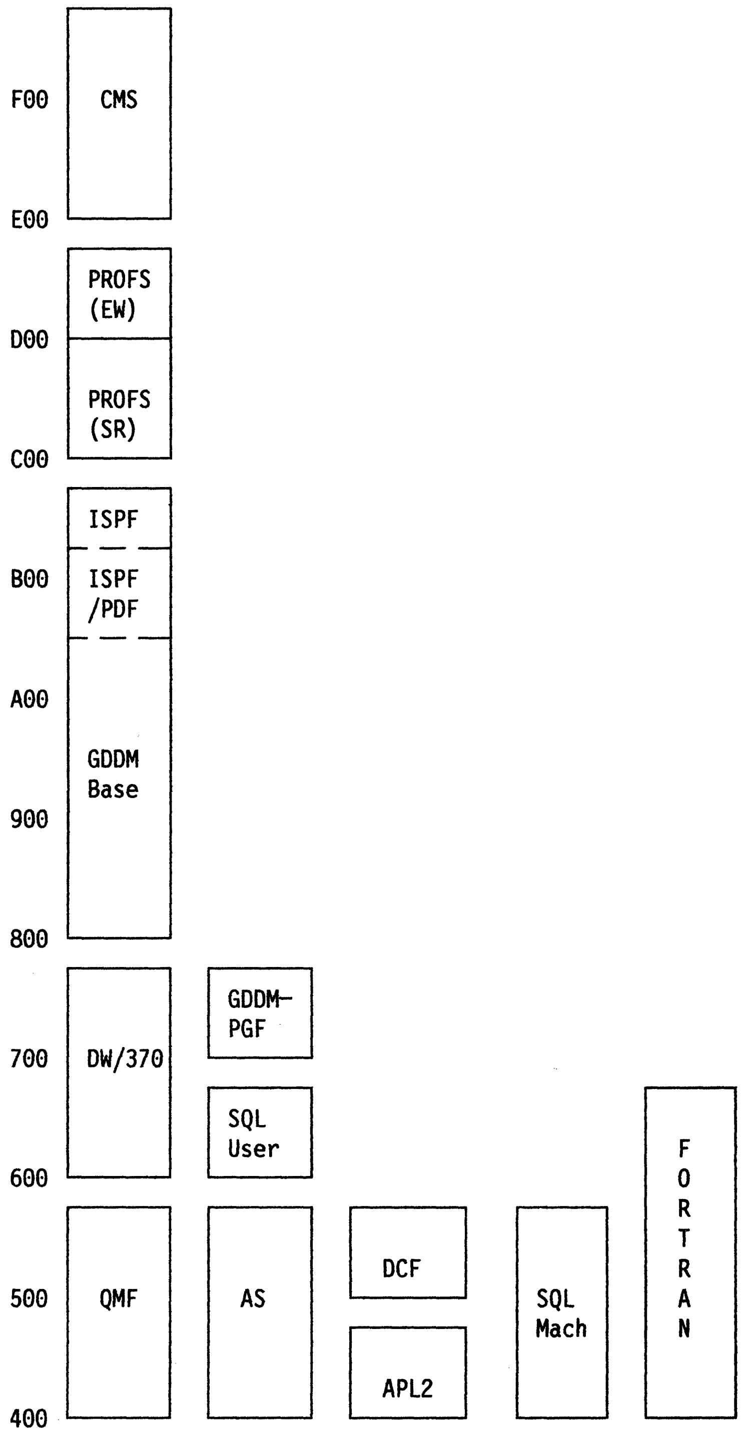


Figure 30. A Typical Saved Segment Environment — Example 1

In Figure 30, 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 used in this example are CMS, PROFS, TIF, GCS, ISPF, ISPF/PDF, GDDM, SQL, DW/370, QMF, DCF, and VTAM.

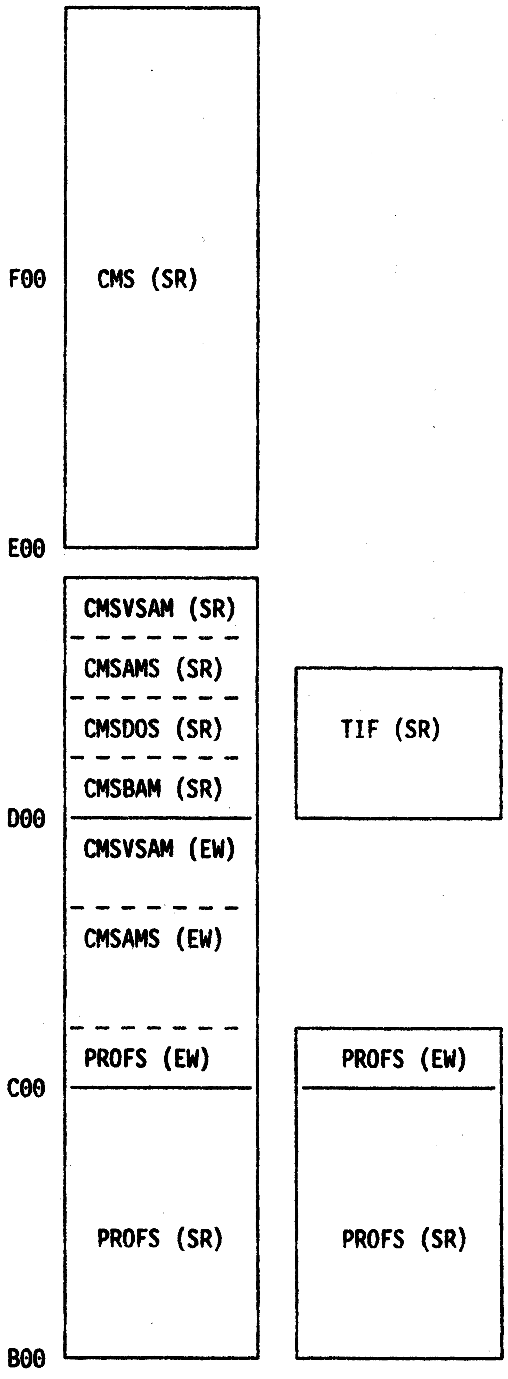


Figure 31 (Part 1 of 3). A Typical Saved Segment Environment — Example 2

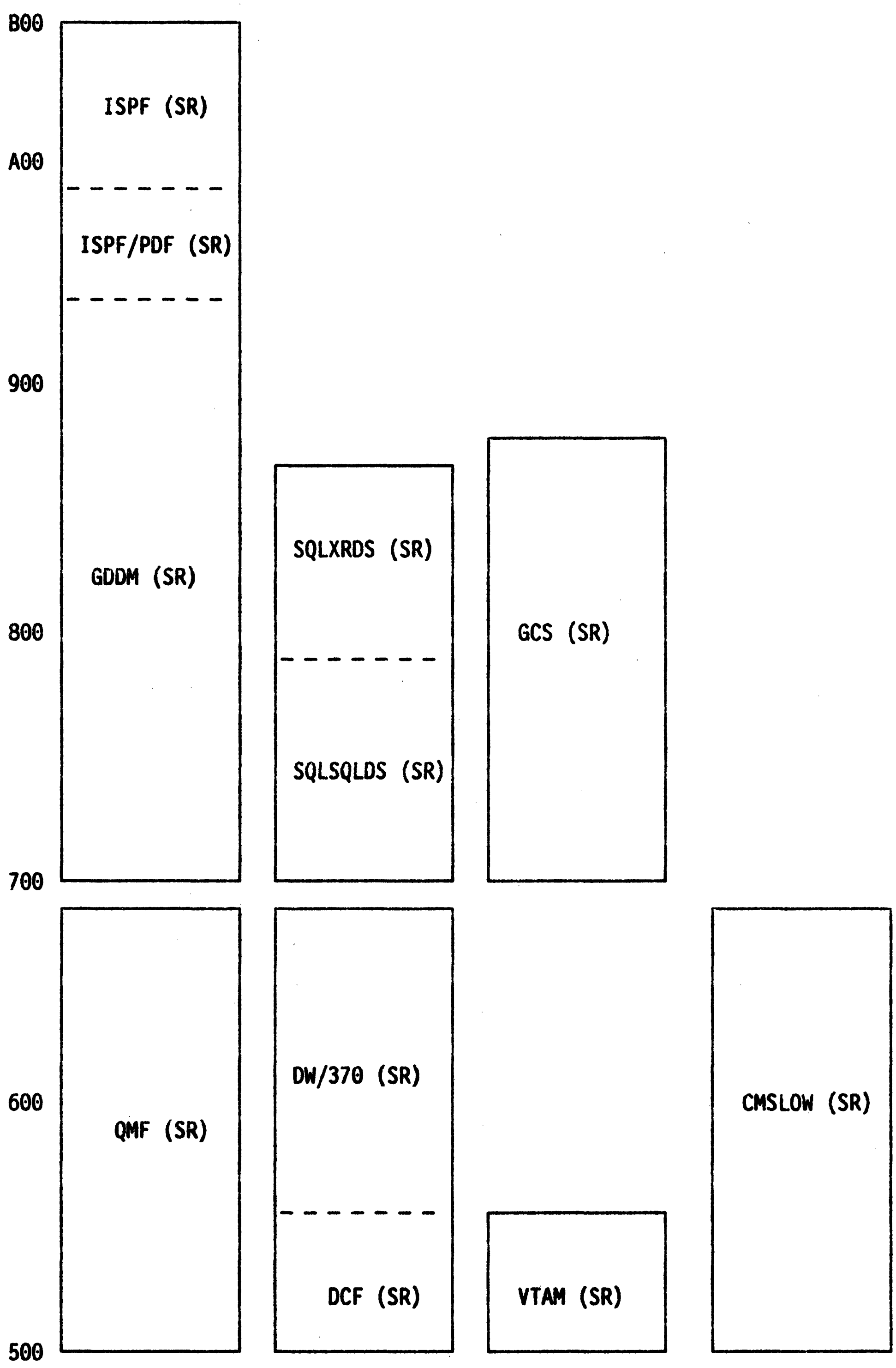


Figure 31 (Part 2 of 3). A Typical Saved Segment Environment — Example 2

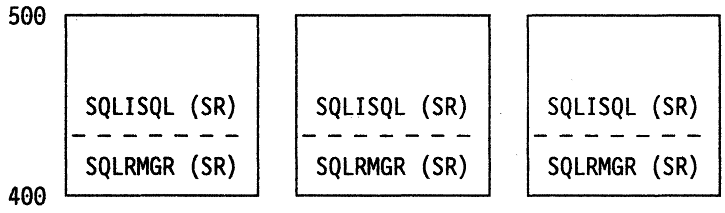


Figure 31 (Part 3 of 3). A Typical Saved Segment Environment — Example 2

In the above figure:

- CMS is defined at a secondary location as a named saved system called CMSLOW in order to make segment D (the default location of CMS) available for other applications.
- The overlayed saved segment containing only PROFS was defined first, so that PROFS users who do not require CMSVSAM, CMSAMS, CMSDOS, and CMSAMS do not load these saved segments.
- 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.

Index

Virtual Machine/
Extended Architecture
System Product
Release 2

READER'S COMMENT FORM

Guide to Saved Segments

Order No. SC23-0457-0

This manual is part of a library that serves as a reference source for systems analysts, programmers, and operators of IBM systems. You may use this form to communicate your comments about this publication, its organization, or subject matter, with the understanding that IBM may use or distribute whatever information you supply in any way it believes appropriate without incurring any obligation to you. Your comments will be sent to the author's department for whatever review and action, if any, are deemed appropriate.

Note: *Copies of IBM publications are not stocked at the location to which this form is addressed. Please direct any requests for copies of publications, or for assistance in using your IBM system, to your IBM representative or to the IBM branch office serving your locality.*

How did you use this publication?

[ ] As an introduction [ ] As a text (student)

[ ] As a reference manual [ ] As a text (instructor)

[ ] For another purpose (explain) ______________________________

____

Is there anything you especially like or dislike about the organization, presentation, or writing in this manual? Helpful comments include general usefulness of the book; possible additions, deletions, and clarifications; specific errors and omissions.

Page Number: Comment:

What is your occupation? ______________________________

Newsletter number of latest Technical Newsletter (if any) concerning this publication: __________

If you wish a reply, give your name and address: ______________________________

____

____

IBM branch office serving you ______________________________

Thank you for your cooperation. No postage stamp necessary if mailed in the U.S.A. (Elsewhere, an IBM office or representative will be happy to forward your comments or you may mail directly to the address in the Edition Notice on the back of the title page.)

Note: Staples can cause problems with automatic mail-sorting equipment.
Please use pressure-sensitive or other gummed tape to seal this form.

SC23-0457-0

Reader's Comment Form

Fold and Tape   Please Do Not Staple   Fold and Tape

NO POSTAGE
NECESSARY
IF MAILED
IN THE
UNITED STATES

BUSINESS REPLY MAIL

FIRST CLASS   PERMIT NO. 40   ARMONK, N.Y.

POSTAGE WILL BE PAID BY ADDRESSEE

International Business Machines Corporation
Department 52Q MS 511
Neighborhood Road
Kingston, New York 12401

Fold and Tape   Please Do Not Staple   Fold and Tape

IBM®

PRINTED IN U.S.A.

IBM®

Program Number
5664-308

File Number
S370-34

SC23-0457-0

Printed in U.S.A.