# VMSES/E Primer: Concepts and Experiences

Document Number GG24-3851-02

June 1994

International Technical Support Organization
Poughkeepsie Center

**Take Note!**

Before using this information and the product it supports, be sure to read the general information under "Special Notices" on page xvii.

**Third Edition (June 1994)**

This edition, GG24-3851-02, is a major revision of GG24-3851-01, and applies to the Virtual Machine Serviceability Enhancements Staged/Extended (VMSES/E) component of Virtual Machine/Enterprise Systems Architecture (VM/ESA) Release 2.2 and Release 1.5 370 Feature, program number 5684-112.

For a list of changes, see "Summary of Changes for VM/ESA Release 2.2 and Release 1.5 370 Feature" on page xxi.

Changes and additions to the text and illustrations are indicated by a vertical line to the left of the change.

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

An ITSO Technical Bulletin Evaluation Form for reader's feedback appears facing Chapter 1. If the form has been removed, comments may be addressed to:

IBM Corporation, International Technical Support Organization
Mail Station P099
522 South Road
Poughkeepsie, New York 12601-5400

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

# Abstract

This document provides an overview of the concepts, functions, and capabilities of VMSES/E (Virtual Machine Serviceability Enhancements Staged/Extended), a component of VM/ESA (Virtual Machine/Enterprise Systems Architecture), and the installation and service tool for the system and related program products. Usage experiences are included.

This document was written for customers and IBM technical personnel involved in planning for, installing, and servicing a VM/ESA Release 2.2 or a VM/ESA Release 1.5 370 Feature system. Knowledge of VM/ESA concepts, in particular CMS commands and file system, is assumed.

The information is based on an early version of VM/ESA Release 2.2 and may not reflect the release as delivered.

VM (237 pages)

# Contents

# Figures

# Tables

# Special Notices

This publication is intended to help IBM technical personnel and customer systems engineers plan for, install, and service a VM/ESA Release 2.2 system (or a VM/ESA Release 1.5 370 Feature system) and related program products. The information in this publication is not intended as the specification of any programming interfaces that are provided by either VM/ESA (Virtual Machine/Enterprise Systems Architecture) Release 2.2 or Release 1.5 370 Feature. See the Publications section of the IBM Programming Announcement for VM/ESA Release 2.2 for more information about what publications are considered to be product documentation.

References in this publication to IBM products, programs or services do not imply that IBM intends to make these available in all countries in which IBM operates. Any reference to an IBM product, program, or service is not intended to state or imply that only IBM′s product, program, or service may be used. Any functionally equivalent program that does not infringe any of IBM′s intellectual property rights may be used instead of the IBM product, program or service.

Information in this book was developed in conjunction with use of the equipment specified, and is limited in application to those specific hardware and software products and levels.

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

IBM Director of Licensing
IBM Corporation
208 Harbor Drive
Stamford, CT USA 06904

The information contained in this document has not been submitted to any formal IBM test and is distributed AS IS. The use of this information or the implementation of any of these techniques is a customer responsibility and depends on the customer′s ability to evaluate and integrate them into the customer′s operational environment. While each item may have been reviewed by IBM for accuracy in a specific situation, there is no guarantee that the same or similar results will be obtained elsewhere. Customers attempting to adapt these techniques to their own environments do so at their own risk.

Reference to PTF numbers that have not been released through the normal distribution process does not imply general availability. The purpose of including these reference numbers is to alert IBM customers to specific information relative to the implementation of the PTF when it becomes available to each customer according to the normal IBM PTF distribution process.

The following terms, which are denoted by an asterisk (*) in this publication, are trademarks of the International Business Machines Corporation in the United States and/or other countries:

| | |
|---|---|
| Common User Access | CUA |
| ESA/370 | ESA/390 |
| IBM | OS/2 |
| PR/SM | QMF |
| SQL/DS | System/370 |
| Virtual Machine/Enterprise Systems Architecture | VM/ESA |
| Virtual Machine/Extended Architecture | VM/XA |

The following terms, which are denoted by a double asterisk (**) in this publication, are trademarks of other companies:

| | |
|---|---|
| UNIX | Unix System Laboratories, Inc., a wholly owned subsidiary of Novell, Inc. |

# Preface

This document is intended as a first approach to VMSES/E (Virtual Machine Serviceability Enhancements Staged/Extended), the Installation and Service Tool and a component of VM/ESA* (Virtual Machine/Enterprise Systems Architecture*).

Discussed are the tasks and information needed to install and service VM/ESA Release 2.2 and Release 1.5 370 Feature systems, and their related program products. How VMSES/E helps to do those tasks, and gathers and maintains information is also documented.

The document also includes actual experiences, suggestions, guidance information, and some sample code. Some parts of the document are written for people with little or no previous experience with VMSES/E.

As the VMSES/E component of VM/ESA Release 1.5 370 Feature is functionally equivalent to VMSES/E on VM/ESA Release 2.1, some of the most recently introduced VMSES/E capabilities are not available. Whenever in doubt, the Release 1.5 370 Feature user should refer to Table 10 on page 203, showing functions available per release.

Where the addition of new or enhanced functions or rewording for improved user understanding caused changes to the text and illustrations, modifications are flagged with a vertical bar on the left, as is this paragraph. Minor editorial corrections are not flagged.

## Who Should Read This Document

This document is intended primarily for IBM technical personnel and customer personnel who are involved in planning for, installing, and servicing VM/ESA Release 2.2 and Release 1.5 370 Feature and IBM program products in the VM/ESA environment.

Readers include IBM systems engineers, IBM customer engineers, and, in general, VM systems specialists, systems programmers, and systems planners.

## How This Document Is Organized

The document contains three parts, subdivided into chapters and appendixes, as follows.

**Part 1, “VMSES/E Concepts”**

This part is an overview of VMSES/E concepts and functions. It includes the following chapters:

- **Chapter 1, “Introduction”**

  This chapter presents, at a very conceptual level, an overview of VMSES/E, and highlights its most important features.

- **Chapter 2, “Functional Overview”**

  This chapter discusses in more detail the functions and data involved in installing, servicing, and managing the software in a VM/ESA system, and how VMSES/E relates to them.

- **Chapter 3, “Software Inventory”**

  This chapter contains further details on the VMSES/E information database, with a more in-depth description of the more important tables.

- **Chapter 4, “Saved Segments”**

  This chapter introduces the support for defining and building saved segments.

**Part 2, “VMSES/E Usage Experiences”**

This part expands the explanations given in Part 1, and presents actual usage experiences as well as examples, suggestions, and some sample code. It includes the following chapters:

- **Chapter 5, “Installation Experiences”**

  This chapter contains information obtained during the installation of VM/ESA and selected program products.

- **Chapter 6, “Service Experiences”**

  This chapter contains information gained in performing several service tasks on a VM/ESA system, including examples of local service application and changes to CP and GCS.

- **Chapter 7, “Exploring the Software Inventory”**

  This chapter provides many examples on the use of the VMSES/E information database.

- **Chapter 8, “Saved Segment Experiences”**

  This chapter provides examples of using the saved segment support functions.

- **Chapter 9, “Multiple Systems and Product Versions”**

  This chapter contains suggestions on how to maintain several systems or product versions using a common database.

**Appendixes**

This part contains the following appendixes:

- **Appendix A, “Comparing VMSES/E to Previous Systems”**

  This appendix discusses the most important differences between VMSES/E and the installation and service tools available in previous VM systems.

- **Appendix B, “Product Packaging and Distribution Media Formats”**

  This appendix describes the various product packaging and tape formats that can be used by VMSES/E.

- **Appendix C, “Removing Service”**

  This appendix provides guidance on how to remove service from the system.

- **Appendix D, "VMFSIM Exploitation Code Examples"**

  This appendix includes several sample coding examples on using the Software Inventory.

- **Appendix E, "Diskette Installation Instructions"**

  This appendix contains detailed instructions for the installation of the accompanying diskette's materials.

## Summary of Changes for VM/ESA Release 2.2 and Release 1.5 370 Feature

Following is a summary of the changes for these releases:

- Information supporting VM/ESA Release 2.2 and VM/ESA Release 1.5 370 Feature has been added in most chapters of the book.
- A new table showing the major new features of VMSES/E per VM/ESA release has been added to Appendix A.

## Summary of Changes for VM/ESA Release 2.1 and Release 2

New and enhanced functions introduced by VM/ESA Release 2.1 and Release 2 are dispersed throughout the document, however the following major changes have been made:

- The syntax diagrams are in "railroad track" format.
- The following chapters and appendixes are new:
  - Chapter 4, "Saved Segment Support"
  - Chapter 8, "Saved Segment Experiences"
  - Appendix E, "Installing the Sample Code Examples"
- The following chapters have been changed:
  - Chapter 5, "Installation Experiences," was completely restructured and a great deal of information added.
  - Chapter 6, "Service Experiences" was expanded to include new sections on the build process, local service, and changing CP.
  - Appendix D, "VMFSIM Exploitation Code Examples," had the source listings removed. On the previous edition of this publication this was Appendix E.
- The following appendix was deleted:
  - Appendix D, "Local Service," was merged with Chapter 5. "Service Experiences."
- Finally, many small editorial corrections have been made to the text.

## Abbreviations

In this document, the following abbreviations are used:

| Abbreviation | Refers to |
|---|---|
| DIRMAINT | Virtual Machine/Directory Maintenance Version 1 Release 4 (program number 5748-XE4) |

| | |
|---|---|
| PVM | VM/Pass-Through Facility Version 2 Release 1.1 (program number 5684-100) or VM/Pass-Through Facility Version 1 (program number 5748-RC1) |
| QMF* | Query Management Facility Version 3 Release 1 (program number 5706-255) |
| RACF | Resource Access Control Facility Version 1 Release 9.2 (program number 5740-XXH) |
| RSCS | RSCS Networking Version 3 Version 3 Release 1.1 (program number 5684-096) |
| SQL/DS* | Structured Query Language/Data System Version 3 Release 4 (program number 5688-103) |
| VM/SP | Virtual Machine/System Product (program number 5664-167) |
| VM/HPO | VM/SP High Performance Option (program number 5664-173) |
| VM/XA* | Virtual Machine/Extended Architecture* System Product Release 2.1 (program number 5664-308) |
| VM/ESA* | Virtual Machine/Enterprise Systems Architecture* (program number 5684-112) |

## Related Publications

The following publications are considered particularly suitable for a more detailed discussion of the topics covered in this document.

- Prerequisite Publications:

  *VM/ESA: VMSES/E Introduction and Reference*, SC24-5444

  *VM/ESA: Service Guide*, SC24-5527

  *VM/ESA: Installation Guide*, (UM98122)

  *VM/ESA: Installation Guide for 370 Release 1.5 370 Feature*, (UM98115)

  *VM/ESA: VMSES/E Introduction and Reference for 370*, SC24-5680

  *VM/ESA: Service Guide for 370*, SC24-5429

- Additional Publications:

  *VM/ESA Release 2.2 Overview and Usage Experiences*, GG24-4219

  *VM/ESA Release 2.1 Usage and Experience*, GG24-4032

  *VM/ESA Release 2.1 Presentation Guide*, GG24-4024

  *VM/ESA Release 2 Usage and Experience*, GG24-3932

  *VM/ESA Release 1.1 Overview and Usage Experiences*, GG24-3744

  *VM/ESA: Planning and Administration*, SC24-5521

  *VM/ESA: SFS and CRR Planning, Administration and Operation*, SC24-5649

  *VM/ESA: Conversion Guide and Notebook for VM/SP, VM/SP HPO and VM/ESA 370 Feature*, SC24-5654

*VM/ESA: Conversion Guide and Notebook for VM/XA SP and VM/ESA*, SC24-5525

*VM/ESA: System Messages and Codes*, SC24-5529

*VM/ESA: CMS Utilities Feature*, SC24-5535

*VM/ESA: CMS Pipelines User's Guide*, SC24-5609

*VM/ESA: CMS Pipelines Reference*, SC24-5592

*CMS Pipelines Tutorial*, GG66-3158

*IBM Online Library Omnibus Edition: VM Collection*, SK2T-2067

To get listings of redbooks online, VNET users may type:

```
TOOLS SENDTO WTSCPOK TOOLS REDBOOKS GET REDBOOKS CATALOG
```

**How to Order Redbooks**

IBM employees may order redbooks and CD-ROMs using PUBORDER. Customers in the USA may order by calling 1-800-879-2755 or by faxing 1-800-284-4721. Visa and Master Cards are accepted. Outside the USA, customers should contact their IBM branch office.

You may order individual books, CD-ROM collections, or customized sets, called GBOFs, which relate to specific functions of interest to you.

## Acknowledgements

This publication is the result of a residency and internal efforts conducted at the International Technical Support Organization, Poughkeepsie and IBM Glendale Programming Laboratory, Endicott.

The author of this document is:

| | |
|---|---|
| **Fernando Nogal** | IBM Portugal |

The following person has contributed to this document:

| | |
|---|---|
| **Scott Vetter** | IBM ITSO Poughkeepsie |

Authors and contributors to previous editions of this document are:

| | |
|---|---|
| **Fernando Nogal** | IBM ITSO Poughkeepsie |
| **Christopher Chambers** | IBM Australia |
| **Jack Dacey** | Skill Dynamics, an IBM Company |
| **Benny Dueholm** | IBM Denmark |
| **Paul Evans** | IBM Canada |
| **Alberto Evandro Favero** | IBM Brazil |
| **Andreas Kalski** | IBM Germany |
| **Artur Rodrigues** | IBM Portugal |
| **Vera Prado Sabag** | IBM Brazil |
| **Brian Troube** | IBM United Kingdom |
| **Jens Olav Tveter** | IBM Norway |

We would also like to acknowledge the professionals who took time to review the current or previous editions of this document, and provided invaluable advice and guidance during its development:

| | |
|---|---|
| **Larry Boulia** | IBM Washington Systems Center |
| **Jay Brodfuehrer** | IBM GPL, Endicott |
| **Kris Buelens** | IBM Belgium |
| **David Chase** | Skill Dynamics, an IBM Company |
| **Scott Deiter** | IBM Kingston |
| **Richard Edwards** | IBM Kingston |
| **Patrick Fadden** | IBM Kingston |
| **Alex Feinberg** | IBM Kingston |
| **Katie Glynn** | IBM GPL, Endicott |
| **D.M. Marshall, Jr.** | IBM GPL, Endicott |
| **Robert Michie** | IBM Kingston |
| **Ken Morgan** | IBM GPL, Endicott |
| **William Noonan** | IBM Kingston |
| **Julie Novak** | IBM Kingston |
| **Michael Rice** | IBM Kingston |
| **Christine Stamm** | IBM GPL, Endicott |

# Part 1. VMSES/E Concepts

This part describes VMSES/E at a conceptual level. The tasks involved in installing and servicing a VM/ESA system and its related program products are discussed, as well as the control information needed to perform these tasks.

The organization and contents of the VMSES/E information base, the Software Inventory, are covered in detail.

# Chapter 1. Introduction

With the introduction of VMSES/E as the installation and service tool, and an integral part of VM/ESA, IBM extended the migration work begun with VMSES. The new and enhanced functions introduced since VM/ESA Release 1.1, and the recent availability of VMSES/E on VM/ESA Release 1.5 370 Feature, place it much closer to the strategic goal of a single tool for the installation of products, service, and packages for the VM/ESA product set.

Today, VMSES/E is fully exploited by the components of the VM/ESA operating system, and by other products as well. VMSES/E has a well-defined set of rules for:

- Packaging installation and service materials.
- Specifying a product's resources, requirements, constituent parts and objects, and building rules.

Furthermore, VMSES/E provides a solid platform upon which specialized functions can be built (actually, added to the ones already supplied with the current release). Existing and future VM products could, thus, take advantage of the VMSES/E capabilities.

Also, other vendors could, in the future, deliver their products in VMSES/E format.

VMSES/E is not only an excellent set of tools that help the system programmer control and manage VM installation and service, but it also provides a reliable and consistent base for future enhancements.

## Installation and Service Tools Requirements

Many VM systems programmers can readily write a list of functions that VMSES/E should have. Obviously, there are also new functions one would like to see, so that the entire task of software management becomes easier and less consuming of resources.

Based on the combined capabilities of earlier tools, such as ITASK, INSTFPP, program product (PP) installation EXECs, VMSERV, and PP service EXECs, the list might look like this:

- Install new VM systems and PPs
- Help tailor products
- Service products
  - Corrective service
  - Preventive service
  - Refresh service
- Keep a record of what has been done

VMSES/E was designed to eventually accomplish all of these tasks, and add many new functions, some of which have been requirements for a long time.

## Highlights

Throughout the remainder of this book the term “product,” unless explicitly stated otherwise, may refer to an IBM program product or a System Control Program (SCP). Also, unless explicitly stated otherwise, “MDISK” or “minidisk” may refer to a CMS minidisk or a Shared File System (SFS) directory. Sometimes the term “disk” is employed to emphasize that either a minidisk or an SFS directory can be used.

The highlights of VMSES/E, listed below, are described in more detail, and examples given, in subsequent chapters:

- A common tool for all VM products.
- Online Help files for all VMSES/E commands.
- A Software Inventory which implements:
  - Automatic creation and maintenance of a database (sets of tables) that contains control and status information, used during product installation and service.
  - Two levels of information: system-level and service-level.
  - Query and Manipulation Facility (Software Inventory Management) providing both user and system access to the software inventory tables.
- An installation function that can:
  - Install new products (add a product to the system).
  - Add a new copy of an already installed product.
  - Migrate installed products, while keeping tailored files unchanged.
  - Build a product (last step in add/migrate).
  - Delete a product.
  - Assist you in planning the addition, migration, and deletion of products.
  - Optionally define (or delete) resources such as user IDs, minidisks, and so on, when installing, migrating, or deleting a product.
- A service process that:
  - Effectively controls back-leveling. No executable code will be loaded from service tapes, unless requested for recommended service upgrade (RSU) service.
  - Allows the database (the Software Inventory) to be independent of any service information included in the update files themselves.
  - Enables selective PTF application.
  - Allows PTF removal (not automated).
  - Has improved performance.
  - Automatically detects object changes.
  - Is restartable.
  - Has a planning, or dry-run, capability (What would happen if I apply PTF UM12345...?).
- Support for managing saved segments, seamless with product servicing.

Appendix A, “Comparing VMSES/E to Previous Systems” on page 197 compares the functions provided by VMSES/E with the ones provided by previous VM systems, as well as a cross-reference of VMSES/E functions per VM/ESA release.

## Design Guidelines

The overall design philosophy of VMSES/E is to treat everything as much as possible in the same way. So, generally speaking, VMSES/E performs the same functions when installing (adding a new copy), adding an additional copy, migrating, or servicing a product. These tasks are all done in three sequential but independent steps, namely:

1. Receive
2. Apply
3. Build

During these steps, a database is updated. This database contains **all** information needed by VMSES/E on the installed products and the service that has been applied to them.

## Software Inventory

The database or, as referred to in VMSES/E terminology, the Software Inventory, is kept at two levels:

- The system-level Software Inventory is a collection of tables describing the installed products. It also has information on system objects, such as saved segments.
- The service-level Software Inventory has information on the service applied to each of these products.

This new database design, along with a complete separation of the receive, apply, and build functions, is the real key to all the benefits of VMSES/E.

For example, tables reflecting receive, apply, and build status, along with a powerful query tool, provide access to a highly detailed level of information about your system.

In addition, tables detailing service requisite information and relationships between products are created. Before VMSES/E, the service requisite information (coreq, prereq, and so on) was in the service files themselves, scattered in hundreds of small files on many minidisks.

Furthermore, the old technique made it difficult to manage the concurrent existence of several software levels. Since service information was scattered, there was no easy way to determine the status of the system. A presumably simple task, such as removing a PTF, would involve a great number of manual operations, including rebuilding all the objects affected, and you were never quite sure that you had remembered everything.

## VM/ESA Release 2.2 Highlights

The most significant functions introduced by VM/ESA Release 2.2 are:

- Installation
  - VMFINS requisite support enhancements
  - File pool flexibility
  - New receive part handler for file uppercasing
- Service
  - PSU/RSU planning tool
  - VMFBLD performance enhancements
  - Building a list of objects (VMFBLD LIST)
  - Test build capability
  - CP load list modification aid
  - Support of local modifications for source files using EXECUPDT
  - Local modifications support enhancements
  - VMFPPF multi-component compile capability
- Saved Systems
  - Enhanced VMFSGMAP displays
  - More flexible segment build

## VM/ESA Release 2.1 Highlights

The most significant functions introduced by VM/ESA Release 2.1 are:

- Installation
  - VMFINS refinement
- Service
  - High Level Assembler support
  - Building of CMS DOS and callable services libraries
  - Generated object build support
  - Local modification support

## VM/ESA Release 2 Highlights

The most significant support aspects introduced by VM/ESA Release 2 are:

- Installation
  - Electronic envelopes
  - User definable access mode for the Software Inventory
  - Expanded PPF override creation opportunities
- Service
  - Expanded role of the Version Vector tables
  - Parts may have versions
  - Object requisites
  - Object definition change detection
  - Building of CMS macro, text, and load libraries
  - General enhancements to object building
  - Building the CP nucleus as a CMS module file

- Software Inventory
  - Panel-driven query
  - Object properties query tool
  - Saved segments information
- Saved Segments
  - Planing, mapping, customizing, and building (includes non-VMSES/E products)
  - Automatic detection of service changes

## VM/ESA Release 1.5 370 Feature Highlights

VMSES/E in VM/ESA Release 1.5 370 Feature is functionally similar to VM/ESA Release 2.1 VMSES/E component. The major difference is the lack of mapping and building capability for saved segments (the 370 feature does not have several commands required by VMSES/E's segment management routines; however, VMSES/E will inform the user when a saved segment needs to be rebuilt). Segment mapping support is provided by the SNTMAP EXEC.

For a summary of the major functions available on each VM/ESA release see Appendix A, "Comparing VMSES/E to Previous Systems" on page 197.

# Chapter 2. Functional Overview

This chapter describes software management in the VM/ESA environment, and VMSES/E in terms of the overall design philosophy and the functions it provides.

You may use this chapter as a road map to VMSES/E and its publications. Although the level of detail is kept low, a few key concepts and control files have to be defined and discussed in order to clarify the control structure. Part 2, "VMSES/E Usage Experiences" on page 81 expands the information introduced here.

The following publications are recommended if you want to gain a deeper insight into the VMSES/E installation and service processes described in this chapter:

- *VM/ESA: VMSES/E Introduction and Reference*
- *VM/ESA: Service Guide*
- *VM/ESA: Installation Guide*
- *VM/ESA: Planning and Administration*
- *VM/ESA: VMSES/E Introduction and Reference for 370*
- *VM/ESA: Service Guide for 370*

## Tasks

Some of the tasks systems programmers most often perform are:

- Installing new products (or new copies of existing ones).
- Applying service to products as required.
- Documenting the software installed on the system, and the service applied to each of the products.
- Managing saved segments.

After a few general remarks on Software Management and VMSES/E architecture, we describe how VMSES/E handles software installation and service. How VMSES/E may assist you in documenting your system is covered in detail in Chapter 3, "Software Inventory" on page 37. VMSES/E support for saved segments is imbedded in several installation and service functions but, in order to provide a coherent view of the whole process and a single place of reference, this support is explained in Chapter 4, "Saved Segments" on page 65, with further information and some examples given in Chapter 8, "Saved Segment Experiences" on page 157.

Detailed information on the Software Inventory can be found in *VM/ESA: VMSES/E Introduction and Reference*.

## Installed Software

Before VM/ESA Release 1.1, there was virtually no means of checking the software products and their service levels, except by use of manually created records.

Since VM/ESA Release 1.1, the system installation process initializes the Software Inventory, and servicing VM/ESA automatically updates it. Likewise, VMSES/E automatically initializes (if needed) and updates the Software Inventory whenever a program product is installed or serviced.

By maintaining all this product and service information, VMSES/E relieves the systems programmer of a time consuming job. Also, features such as automatic requisite checking and planning capability make software management less error prone.

Query and reporting facilities may also assist the systems programmer in preparing system documentation and planning for new products or service.

As the Software Inventory is separate from and independent of the code, all software management functions can be executed without impacting the production system.

Inevitably, when customers order service from the IBM Support Center, the question arises: “Yes we have a PTF that will solve your problem. But have you applied PTF UM12345 and not PTF UM54321?”

Who knows?

VMSES/E does.

## Concepts and Sources of Information

Before we proceed with the functional description of VMSES/E it is necessary to introduce a few concepts and definitions. In this section we also discuss data input to the VMSES/E database, and explicitly name some of the key input files.

References to these definitions and files will be made throughout the rest of this book.

## Definitions and Terms

Of the many concepts and definitions employed by VMSES/E, we list below the ones used throughout the remainder of this publication:

| Term | Definition |
|---|---|
| **Product** | A software program that can be separately ordered. |
| **Component** | A product′s functional unit, installed and serviced separately from other functional units in the same product.<br><br>A product may contain several components; for example, CMS, CP, and VMSES/E are components of the VM/ESA product. |

**Note:** the term product is often used instead of component, since most products have only one component.

**Object** Each component consists of discrete objects. Types of objects include nuclei, load libraries, help panels, modules, and so on. An object, in turn, is built from parts.

**Part** A CMS file provided as input to the build process. Types of parts include macros, source files, text decks, and so on. A part is the smallest serviceable unit of a product (the smallest element to which service can be applied).

**APAR** An *Authorized Program Analysis Report* is the formal means of reporting and tracking the solution of a problem. IBM develops a fix to the problem and gives it the same number as the APAR, so the term APAR may refer to both a problem report and its solution. The fix may require changes to a number of parts and may contain update service, replacement service, or both. For a description of these types of service see "Maintaining Your System" on page 23.

The APAR number is composed of a two-letter prefix and five digits (for example, VM12345).

**PTF** A *Program Temporary Fix* contains the solutions for one or more APARs. Each PTF is identified by a PTF number similar in structure to the APAR number (for example, UM87654).

**Requisites** Interrelations between products (system-level requisites) or between program fixes (service-level requisites).

**System-level requisites** Define relations between products (or components). The most common ones are:

**PREREQ** Identifies a product that must be installed before the given product can be installed.

**REQ** Identifies another product that must be installed for the current one to operate correctly.

**DREQ** Identifies a specific level of another product that must be installed for the current one to operate correctly, and if that other product is deleted then the current one must also be deleted. For example, in every VM/ESA release, REXX lists the CMS component of the same release as a DREQ.

**NPRE** Indicates a mutually exclusive condition. For example, two External Security Managers; installing one would prevent installing the other.

**Service-level requisites** Define dependencies between program fixes for the same or different products. The defined types of dependencies include:

- **PREREQ** PTFs that must be applied before a given PTF is applied.
- **COREQ** PTFs that must be applied at the same time as the given PTF. No order is implied.
- **IFREQ** PTFs in another component or product, which must be applied at the same time as the given PTF if the component or product is installed. No order is implied.
- **SUP** PTFs that have been completely replaced by the given PTF.
- **HARDREQ** A subset of PREREQ PTFs that have real code intersections (PTFs that have updated some of the same source lines as the given PTF) or functional dependencies.

**Usable form** Synonym for "Object." Examples are the VMFINS EXEC, and PIPE MODULE, as opposed to the serviceable parts VMFINS EXC12345 and PIPE MOD12345.

**Back-leveling** A term used to describe the situation in which you apply service to your system, and end with a service level that is lower than your starting point. This situation may occur when, for instance, usable forms are loaded from a service tape.

**prodid** A unique 7- or 8-character identifier assigned to each product or component by IBM. For example, 6VMVMA22 is the prodid for CMS in VM/ESA Release 2.2.

**compname** A unique 1- to 16-character alphanumeric identifier assigned to each component by IBM. For example, CMS is the compname for CMS in VM/ESA Release 2.2.

## Information Sources

As one might guess, most of the product and service information needed by VMSES/E is delivered on the product and service tapes themselves. The rest is supplied by the user, and includes modifications to standard installation procedures and normal product tailoring.

### Parameter Files on the Product Tape

The two most important files shipped with a product are:

**prodid PRODPART** The Product Parts file is the basic product description file. Products and components in VMSES/E or Parameter Driven Installation (PDI) format are shipped with this file, which contains information on requisites, tailorable parts, and resources, including:

- CP directory entries for any required user IDs; for example, servers and sample user entries.

- Saved segment default information.
- Minidisk and SFS directory space requirements.

This information is used to update the system-level Software Inventory.

**prodid $PPF** Products and components in VMSES/E or PDI format are shipped with a Product Parameter File (PPF), which defines how VMSES/E handles this product. The information in the PPF includes:

- Control options.
- Logical files on the product installation and service media.
- Indications for building the usable forms of the product.
- Variables that are used.
- Symbolic names for target disks.
- User-exit definitions. Examples of user-exit use are installation related tailoring, and copying and renaming sample files.

## Parameter Files on the Service Tape

The number and types of parameter files shipped with a VMSES/E service tape depend on the type of service, such as PSU, COR, or other (see “Maintaining Your System” on page 23 for descriptions of service types). Besides the tape descriptor files, the most important ones are:

**ptfnum $PTFPART** The PTF Parts file is shipped for each PTF and contains the following important information:

- Description (PTF number, APAR number, and description). A user memo may also be included.
- Requisite section defining PTF relationships.
- Parts section defining parts being serviced.

Most of the information in this file is used to update the service-level Software Inventory.

**Apply List** List of the PTFs that are to be applied. The file name is defined in the PPF. The file type as shipped is $APxxxxx, where xxxxx depends on the type of service. After the file is loaded on the disk, the file type is changed to $APPLIST.

**Exclude List** List of the PTFs that are not to be applied. The file name is defined in the PPF. The file type as shipped is $EXxxxxx, where xxxxx depends on the type of service. After the file is loaded on the disk, the file type is changed to $EXCLIST.

## Default Overriding

So far we have discussed only the product parameter file as it is supplied with the product. This file contains default control information for product installation and service; however, there are cases where you may need to modify these defaults.

Suppose you want to install an *additional* copy of a program product. If VMSES/E were to use the original PPF and install the product accordingly, the

result would actually be a replacement, since the disk definitions would not have been changed.

Because IBM strongly recommends that the originally shipped $PPF files never be altered, any changes are placed in *override* files. Overrides may also change other overrides; in other words, overrides may be chained.

The original $PPF file is used as a base, and VMSES/E provides a tool to merge the override files into the base and produce a "compiled," or usable form, Product Parameter File (of file type PPF). The procedure is logically similar to the use of the CMS UPDATE function. The following files are related to PPFs:

**prodid $PPF** The original source PPF, as supplied with the product.

**prodid PPF** The usable form of the default PPF, also supplied with the product.

**ppfname $PPF** User overrides to the original, or other override, PPF. You define "ppfname," which can be any valid CMS file name.

**ppfname $PPFTEMP** A temporary file created by the override utility.

**ppfname PPF** The usable form of the modified Product Parameter File.

Further information about these parameter files can be found in Chapter 3, "Software Inventory" on page 37 and in *VM/ESA: VMSES/E Introduction and Reference*.

## Product Database Layout

The VMSES/E installation and service strategy is to separate the various types of files for each component into logical strings having one or more disks each. All components of VMSES/E-enabled products have the same logical disk structure, which simplifies handling. Actual minidisk addresses or SFS directory names are defined in each component's PPF. The following strings, listed in the VMSES/E access order sequence, have been defined:

| Minidisk/String | Contents |
|---|---|
| **5E5** | VMSES/E code. Commonly referred to as the SESDISK. |
| **51D** | System-level Software Inventory, PRODPART, $PPF and PPF files. Commonly referred to as the SIDISK. |
| **TASK** | Any files that you want accessed before the service disks defined for a component; for example, tools disks. |
| **LOCAL** | Sample files, updates, replacement parts, and auxiliary control files for local service. Files on the local string are not changed by IBM during service. |
| **APPLY** | Service control files and service-level Software Inventory information that specify the service level and build status of a component. The information includes PTF apply status, service control information for parts (AUX files and Version Vector tables), and object build status. |
| **DELTA** | This strings holds all service materials received for the component, and related service-level Software Inventory information: PTF-numbered parts, update files, $PTFPART files, Software Inventory PTF information, Apply and Exclude lists, build lists. |

**BUILD** Usable forms.

**BASE** This string contains component materials at the base level, that is, before any service has been applied. These materials include base source and object files, and build lists. The contents of this string are never changed after the initial installation.

**SYSTEM** Running system disks.

Each string may have many minidisks, or SFS directories, or both. In total, when you consider all products and components, there can be a substantial number of disks, but VMSES/E will keep track of them for you. VMSES/E provides a tool, the VMFSETUP EXEC, to access the correct disks for the task you want to perform.

The individual files are distributed among the different strings and individual disks. We shall return to their exact locations later.

If you compare the installation process to building a house, you may think of the Product and Service files as the building blocks, the Control files as the drawings and documentation, and the Running System as the finished house. Taking this analogy one step further, one could use the same building blocks with different drawings to build a different house. This illustrates well VMSES/E capabilities for managing multiple systems using a common base.

## Installation and Service Processes Data Flow

Figure 1 on page 16 schematically shows the data flow within the VMSES/E database during an installation or service process. Note that the boxes in this figure represent groups of files with common features, and that the arrows represent data flow to or from a group.

The numbers indicate the sequence in this 3-step process. The processes shown are:

**RECEIVE**

1. Call the RECEIVE utility
2. Load the tape descriptor files
3. Update the Software Inventory information
4. Load control files and product or service files

**APPLY**

5. Call the APPLY utility
6. Read current maintenance level
7. Write updated level

**BUILD**

8. Call the BUILD utility
9. Read status and control information
10. Read base and updated files
11. Build product and write new build status
12. Write usable forms to a test disk (saved segments are generated as real system data files)

The following sections discuss these processes in more detail.

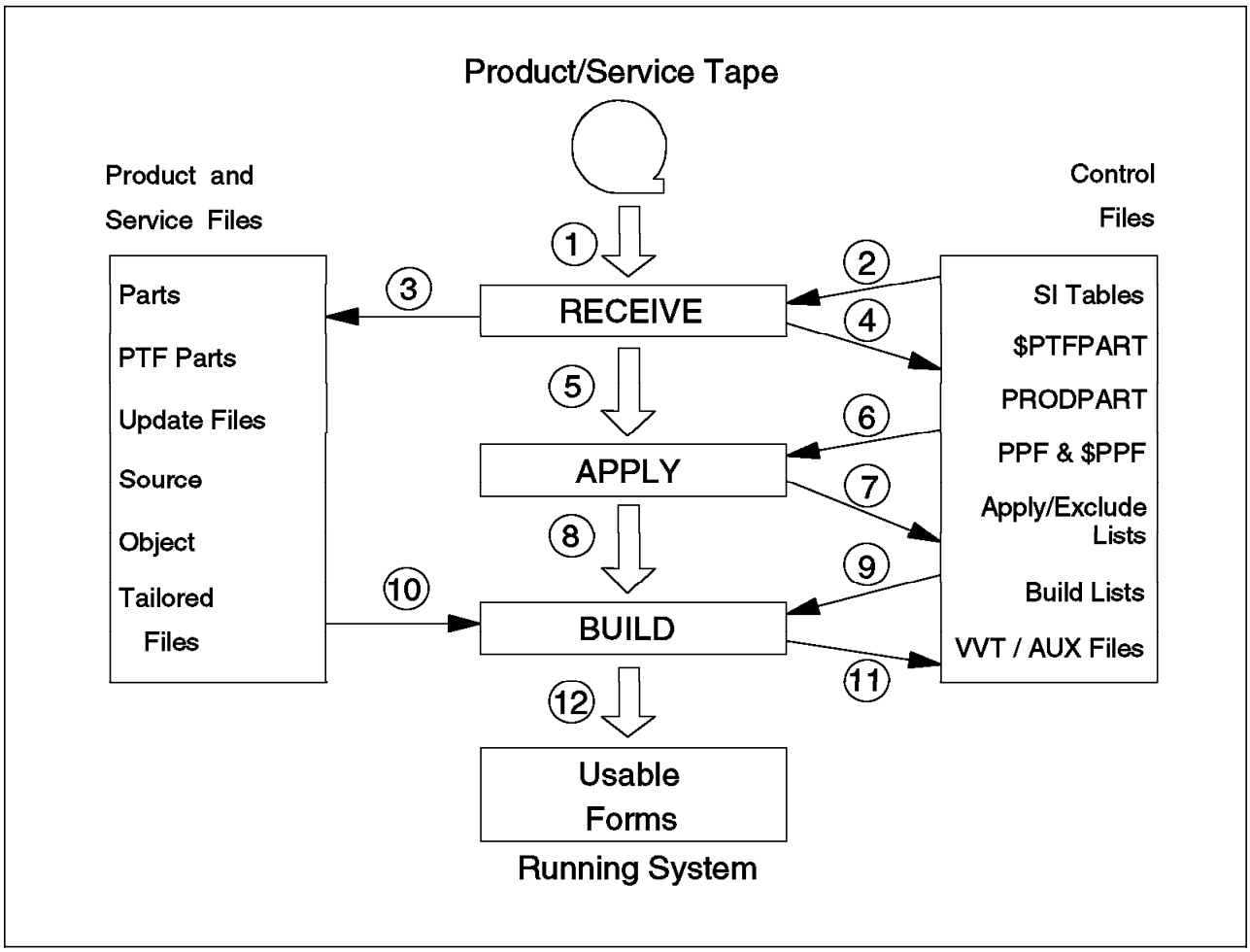



*Figure 1. Data Flow in the VMSES/E Database*

## Information Supporting Media and Tape Formats

The preceding discussion implicitly assumed that data to be fed into the VMSES/E database is supplied on a magnetic tape. From a logical perspective, VMSES/E will treat all input as tape, but the physical medium could be almost anything, such as:

- CD-ROM.
- Electronic envelopes.
- "Telephone" APARs. The IBM Support Center provides the APAR during a telephone conversation.
- Paper. You receive a printout listing a patch or a source update.

In each case, the user can format the information for VMSES/E use.

Please refer to *VM/ESA: VMSES/E Introduction and Reference* for further information.

VMSES/E relies on products and service being delivered in a well-defined format, with appropriate parameter and control files. Do not confuse product format with tape format. The latter is also known as product packaging format. VMSES/E accepts the older installation formats and does certain installation tasks and

Software Inventory updates, but no service processing for these old format products.

### Distribution Tapes

A description of the different tape formats and the processing done by VMSES/E is found in Appendix B, “Product Packaging and Distribution Media Formats” on page 205, and in *VM/ESA: VMSES/E Introduction and Reference*. VMSES/E supports the following tape formats:

- For installation:
  - VMSES/E
  - Parameter Driven Installation (PDI)
  - INSTFPP
- For service:
  - VMSES/E Refreshed Product Tape and RSU
  - PUT
  - COR

**Note:** VMSES/E-formatted products are not serviced by PUT tapes (only COR and RSU). VMSES/E, however, can receive PUT tapes for other products.

### CD-ROM

CD-ROM media is available since VM/ESA Release 2.1 and has the following prerequisite:

- IBM System/370 and System/390 Optical Media Attach/2 Package (5621-264)

CD-ROM supports:

- Initial installation of VM/ESA Release 2.1 and Release 2.2, including the RSU.
- RSU service, for the PSU processes.
- Manuals, in the *IBM Online Library Omnibus Edition: VM Collection* (also available for over 50 VM-related products, previous VM/ESA releases, and VM/ESA Release 1.5 370 Feature).

### Electronic Envelopes

An electronic envelope is a single CMS file with a special internal structure. Inside the envelope there are several CMS files and associated control information, in a format that resembles the one created by the VMFPLC2 command on a tape. Electronic envelopes are beneficial, for example, to sites with multiple or distributed systems: Because the envelope is a CMS file, the SENDFILE command can be used to distribute the software, preserving the sequence of the files.

VMSES/E Release 1.1 supported electronic envelopes for service only. Release 2 extended their use to the installation functions. A product distributed in more than one envelope is supported in a way similar to a product distributed in several magnetic tapes (or cartridges). Electronic envelopes are manipulated with the VMFPLCD command. This command executes for a disk file the same or equivalent functions that the VMFPLC2 command executes on tape files.

## Installing Products

This section provides a closer look at the VMSES/E installation procedures. We have moved down to the disk level and, to a certain extent, also to the file level, but the detailed contents of the files is of no interest at this point. The key concerns are how the contents of the installation media are loaded to the correct disks, and how the product is built.

The installation tool is the VMFINS EXEC. With it, you can perform the following tasks:

- Install a product for the first time
- Add additional copies of a product
- Delete products
- Migrate products to other releases or versions
- Use the planning capability

It was previously stated that VMSES/E handles all installation (and service) tasks in essentially the same way. To understand how this is possible, please consider the following points:

- Installing products and service is a matter of moving files from tape (or another media) to disks, and then building the product according to a set of predefined rules. These rules also define the disks from which to select the parts for the build process, and where to place the built objects.
- The differences between the various “flavors” of installation, and the application of service, are simply which set of disks are used for the load, and which set of rules are selected for the build.

We now know that this information, selecting disks and rules for the build process, is exactly what is found in the PPF. Furthermore, this file governs all VMSES/E activity and that is why these processes are transparent to VMSES/E.

The “secret” is in the parameter files.

Product installation and several install scenarios are described in great detail in *VM/ESA: VMSES/E Introduction and Reference*.

## Installing VM/ESA

The next section contains a description of the VMSES/E installation process valid for any VMSES/E-enabled product, VM/ESA included. However, the installation of VM/ESA Release 2.1, Release 2.2, and Release 1.5 370 Feature do not follow the process described there.

As the initial installation of VM/ESA does not require the full capabilities of VMSES/E, function is traded for speed and simplicity by using a process that, in essence, restores a system image created with the DDR utility. This process is called “flex-DDR.” Note that, however, the system from which the DDR is obtained *was itself installed by using VMSES/E functions*.

Further information on the VM/ESA installation process is provided in the *Installation Guide* packaged directly with VM/ESA.

## Installing a Product for the First Time

The following example illustrates the data flow during the installation of a component, and only that. Products having several components have to separately install each component. For a more detailed description of the installation process, see Chapter 5, “Installation Experiences” on page 83. Also, *VM/ESA: VMSES/E Introduction and Reference* is of particular interest.

For simplicity, we have chosen a first-time installation of a program product, and there is only one minidisk in each of the strings defined in “Product Database Layout” on page 14. This situation is illustrated in Figure 2. Note that VMSES/E does not use data on the SESDISK, TASK, and SYSTEM disks. However, these disks contain functions used by VMSES/E. Also, the 191 disk is used as a work area.

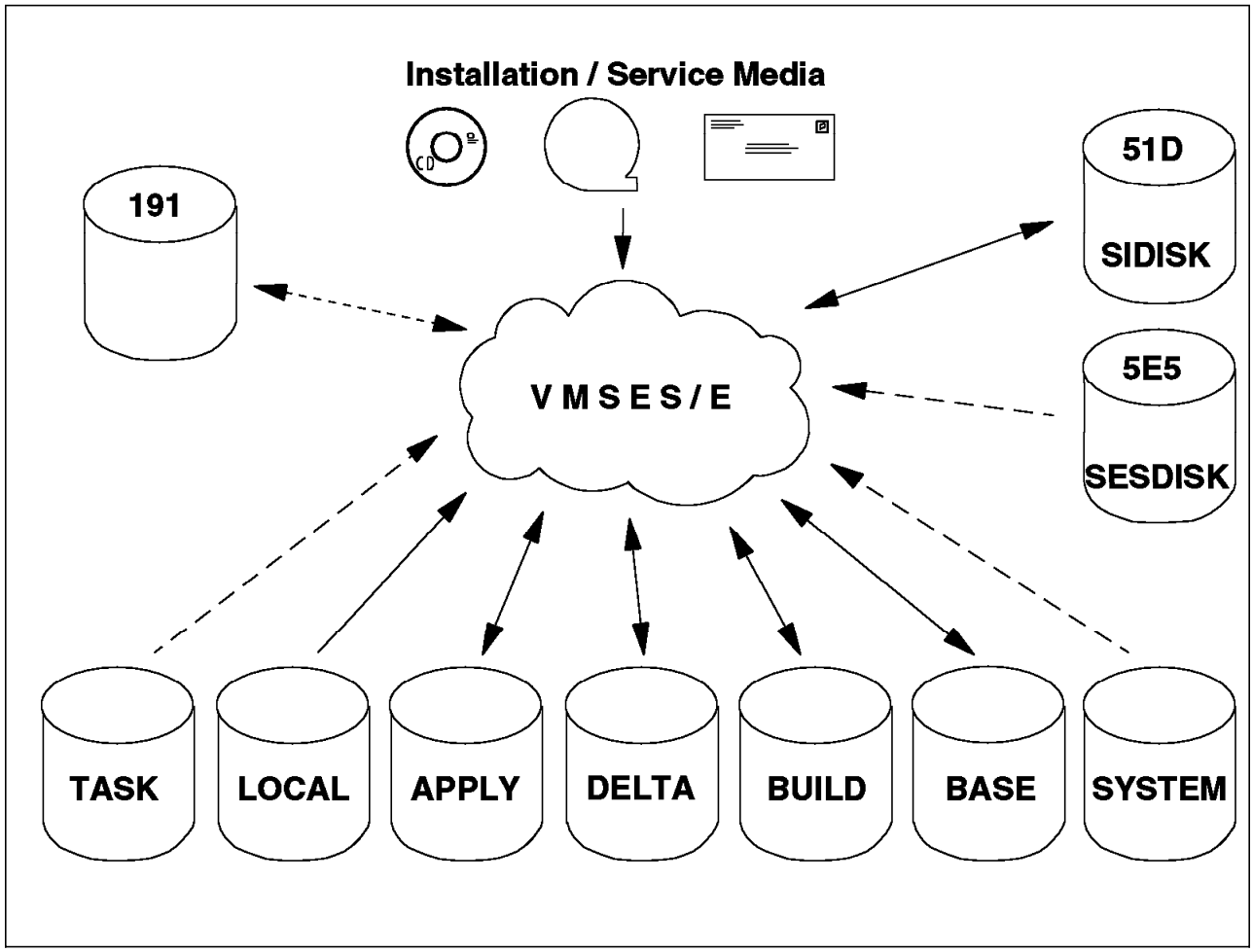



*Figure 2. Installation Example - Minidisk Configuration*

Also, a VMSES/E format product is used; therefore, the installation tape is a Refreshed Product tape (RPT). This type of tape contains the product with service already applied and built up to a certain service level. As we will see, the tape contents include the base files, service refreshed files, and the service updates.

Because the RPT may not have the most recent level of service, another tape containing the IBM recommended service since the last RPT may also be

shipped with the product. This tape is the Recommended Service Upgrade tape, or RSU. It is issued periodically.

The process of installing or maintaining a product using the RPT and RSU tapes is called Product Service Upgrade (PSU). See also “Maintaining Your System” on page 23.

To follow the execution flow, please refer to Figure 1 on page 16.

**RECEIVE**

The first tape file contains installation related information such as the Memo to Users, tape layout description, level identifier, and our two most important sources of information (see “VMSES/E Installation Tape Format” on page 210):

- prodid $PPF
- prodid PRODPART

By calling VMFINS with the appropriate options, receive, the first installation step, may result in the following:

1. The information and parameter files are loaded from the tape.
2. The product part file, prodid PRODPART, and the product parameter file (both source, prodid $PPF, and usable form, prodid PPF) are loaded to the SIDISK.
3. The system-level Software Inventory Requisite and Description Tables are updated.
4. VMFINS prompts you to change the PPF defaults. If you decide to change the defaults you must also supply a name for the override PPF (ppfname $PPF). The Make Override panel is displayed and you can enter your changes. The usable-form PPF, ppfname PPF, is generated on the A-disk, then both this file and the override are moved to the SIDISK.
5. Resource and requisite checking is performed for:
   - Requisite program products
   - Required user IDs
   - Required minidisks and their sizes

   The result of the resource check is stored in the file “prodid PLANINFO,” on the A-disk.

You now have a usable form PPF. Next, review the PLANINFO file and decide what changes have to be made. VMFINS can automatically create updates to the CP Directory (add new user IDs, make changes to existing ones, and add or change minidisks), or you can do it manually.

Figure 3 on page 21 shows the contents of the database after these first steps, which are referred to as the “planning step.”

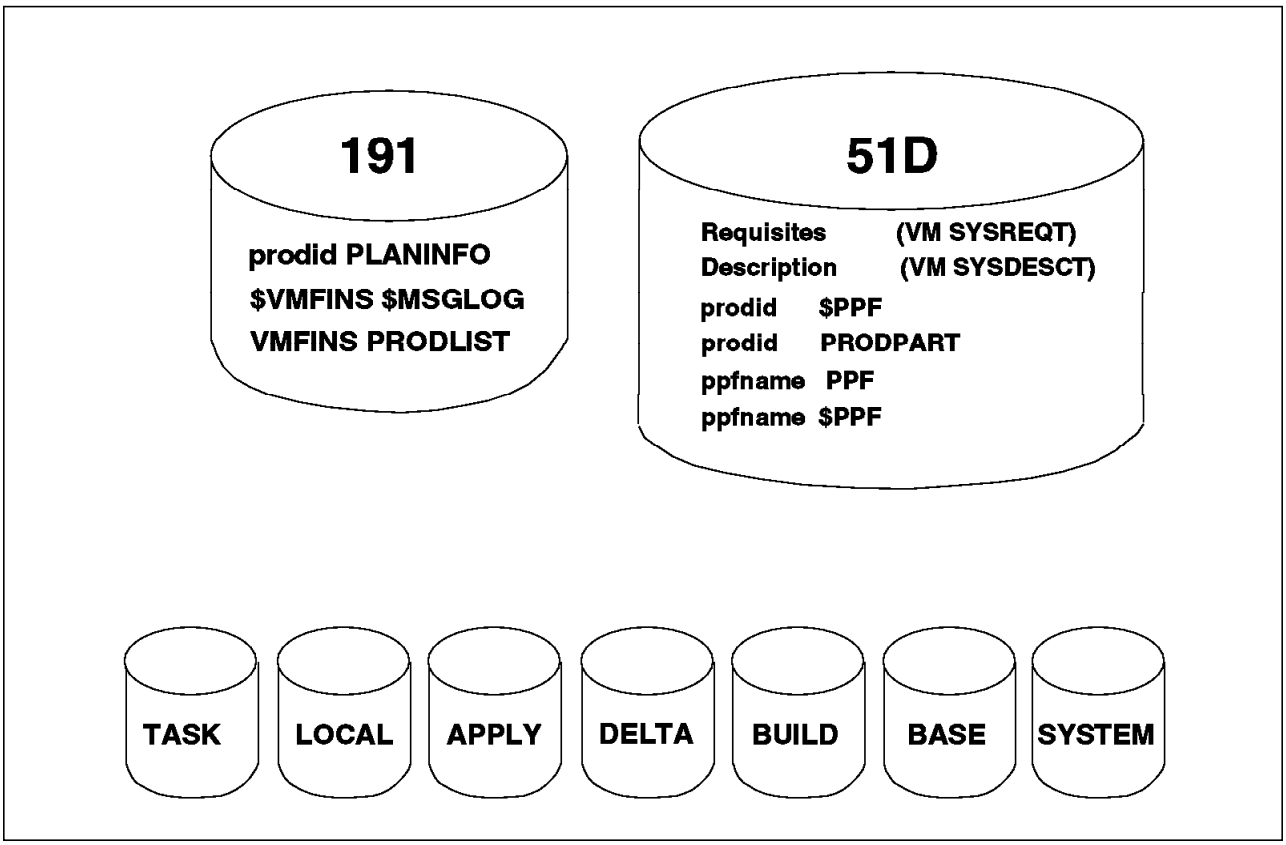



*Figure 3. Files Loaded and Built by the Planning Step*

The newly created PPF contains the minidisk addresses (or SFS directory names) where the product files are to be loaded. So, you may now proceed with the receive step:

6. Files are loaded to the appropriate disks, as specified in the PPF.
7. On each of the involved disks, a file, VMSES PARTCAT, is updated (or created). This file contains a record for each file existing on the disk. This record includes the file's file ID, prodid, status, and the user ID installing the product.

**APPLY**

The apply step, at the system level, is used to indicate whether the product has all its requisites satisfied. If they are, the system-level Apply Status table, on the SIDISK, is updated accordingly. If the apply step fails, then the product cannot be built.

**BUILD**

All that remains to be done now is to build the product. Most of the product's objects come already built on the installation tape, so only objects that were tailored (nuclei, saved segments and saved systems) are normally required to be built. If any parts require assembly, they have to be manually assembled, before starting the build. Some products supply a B*ponum* EXEC to accomplish this. The build process may require link-edit operations, but the build process will, at least, update the System Build Status table. Other functions you perform on the system will then know your product was built. The following steps are performed:

8. The product requisites are once again checked.

9. The product is built on the disks specified in the PPF.
10. The system-level Software Inventory is updated.
11. If the product contains objects that VMSES/E cannot build, VMFINS calls "B*ponum* EXEC," if it exists, to build those objects. The B*ponum* EXEC must be supplied by the product.
12. VMFINS verifies that the product was installed and built correctly, by calling a product supplied EXEC (V*ponum* EXEC) if it exists.

**Note:** ponum is the product order number, as listed in the :PONUM tag in the product's PPF.

If the installation fails, some work files may remain on the A-disk. In that case, you can examine the log files created, correct the problem, and restart the installation.

Figure 4 shows the disks' contents after installation completion. For completeness, files are shown on both the DELTA and APPLY strings because the Refreshed Product Tape, in addition to the base materials, may contain service that is loaded to those strings, though you do not explicitly apply service during the installation process. How this is done will be further detailed in "Receive Install Tape Definitions (RECINS) Section" on page 49 and "Receive Service Media Definition (RECSER) Section" on page 50. Besides the files we have mentioned explicitly here, a number of log files and temporary work files are also created. A detailed description of these files may be found in *VM/ESA: VMSES/E Introduction and Reference*.

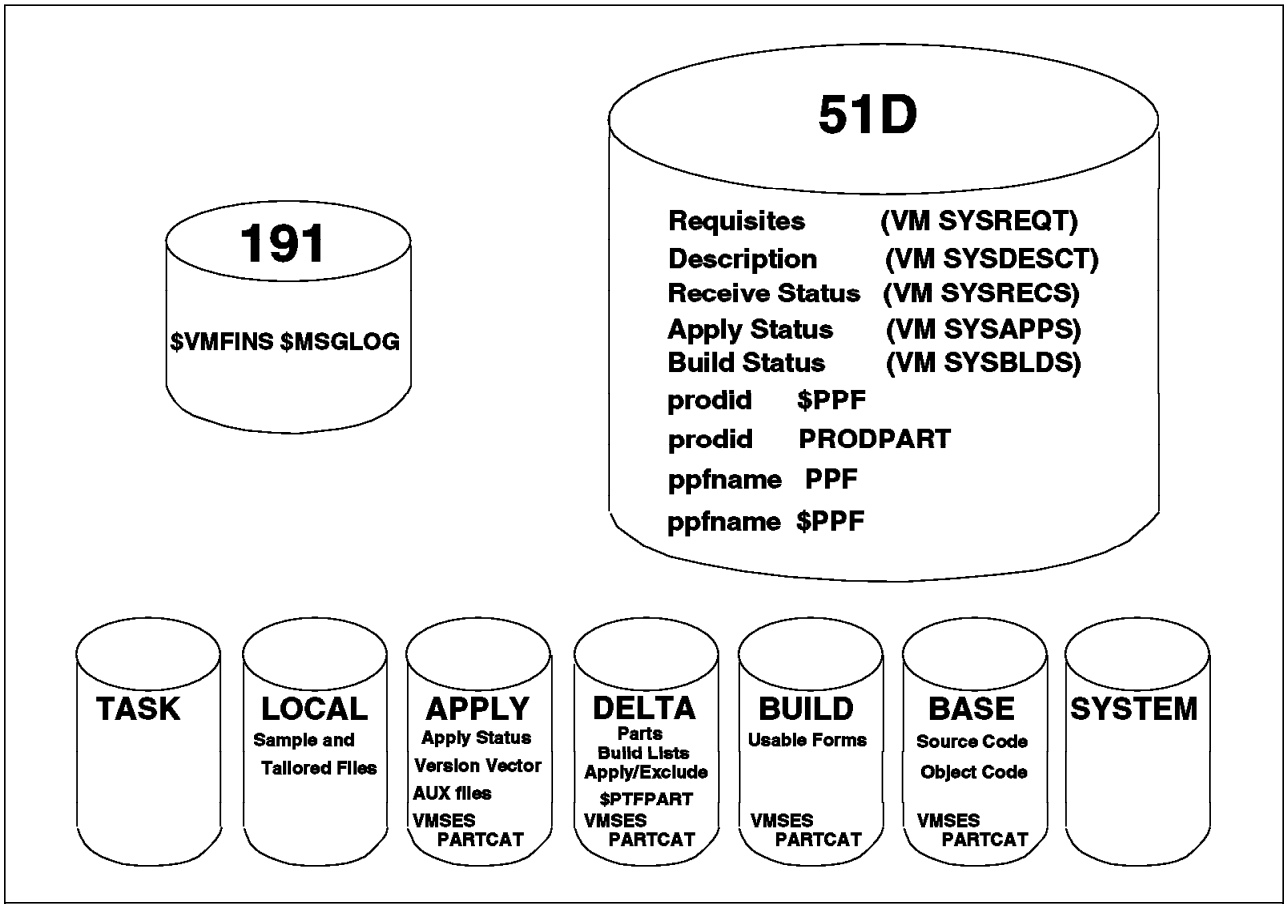



*Figure 4. Files Loaded and Built after Installation Completion*

## Other Installation Options

For the purpose of this discussion, the VMFINS options that allow you to add additional copies of a product, or delete a product, will not add anything new to our understanding of the installation process. The use of VMFINS and its options is discussed in Chapter 5, “Installation Experiences” on page 83, and in *VM/ESA: VMSES/E Introduction and Reference*.

## Maintaining Your System

The task of maintaining software products has three distinctly different purposes:

- The traditional interpretation: fixing software bugs
- Updating your software base and actually adding new function
- Modifying the distributed software to satisfy a local need

However from the systems programmer′s point of view, there is no difference, since small program enhancements (SPEs) are usually shipped as program fixes. A typical example of an SPE is a change that adds support for a new hardware device.

To identify software problem reports, their fixes, and requests for program enhancements, the concepts of APAR and PTF are used.

Service for VM/ESA and program products, as delivered by IBM, can be provided in four ways:

**Product Service Upgrade (PSU)**

This procedure uses Refreshed Product Tapes (RPT), and Recommended Service Upgrade (RSU) tapes, which have all the service pre-installed and pre-built. You can service your system while leaving customized files untouched. Also, both reach-ahead and local service are identified and can be re-applied. This is the recommended method of installing preventive maintenance.

**Note:** VM/ESA Release 1.5 370 Feature, VM/ESA Release 2 and later releases do not use RPTs for preventive service; they only use RSU tapes.

The RSU is cumulative from the beginning of the release (from July 1992, in the case of VM/ESA Release 1.1) but it contains only the fixes to the most pervasive or HIPER types of error. In other words, it contains only a subset of the total service.

**Program Update Service**

IBM will, on a regular basis, produce a Program Update Tape (PUT) that contains service to VM program products. The PUT is cumulative and contains all service up to the current level. However, service for VMSES/E products does **not** use PUTs.

**Corrective Service (COR)**

Service IBM sends you to correct a specific problem that you or another customer have encountered and reported. The service is neither pre-applied nor pre-built, which is one of the major differences with the RSU tape. The service is distributed either via a COR tape or an electronic envelope.

**Local Service**

Any service that is not supplied in one of the above defined formats. It can be service that you originated, or service that IBM sent you to correct a specific problem that you have reported, but that IBM has not yet incorporated into a formal PTF.

A VM/ESA system is maintained by changing its objects and parts. This sounds very simple, but remember:

- A part may affect more than one object.
- A change in one part may require changes to others, since objects are usually made up of many parts.
- Objects are related as well, and together form components.
- Components work together to make up a product.
- Finally, different products are used on the same system and may depend on each other.

To illustrate these dependencies:

A “harmless” change to a tiny macro in a CP MACLIB may introduce errors in CMS, so that a FORTRAN compiler will start to generate code at addresses right on top of CMS itself. And where is the problem? Is it in FORTRAN, CMS, CP, or somewhere else? Since we are not concerned here with problem determination, we will not worry about that. But the example shows how all these dependencies, in fact, make maintenance a very complicated task.

Since a few changed parts may start an avalanche, it is quite evident that very precise service records must be kept at all times. Only by meeting all requisites can a service update be successful. Problems that may occur include the possibility of the requisite list not being complete, or that the fix itself might be in error.

The example and comments made here explain why VMSES/E:

- Has a service-level Software Inventory
- Has a planning (also called test, or dry run) capability
- Uses a multi-level system structure (alternate, intermediate, and production)

## Bypassing VMSES/E - Don′t Do It

If you have used other service methods in previous VM systems you may be tempted to bypass VMSES/E procedures. Our recommendation is **do not do it**.

It may seem, at times, that VMSES/E adds complexity to tasks when compared to previous (manual) procedures. However, consider that VMSES/E is constantly generating, updating, and checking system and product status information.

With VMSES/E, tasks that were very difficult and extremely time consuming to do manually, not to mention of dubious reliability, are now easily performed. Even local modifications are easy to maintain, and Release 2.2 has improved this support. We have found that VMSES/E does simplify the systems programmer′s tasks and saves a significant amount of time.

## VMSES/E Service Concepts and Methods

The objects and parts of a VM/ESA system and its associated products can be maintained in four ways:

- Update the part (called update service)
- Replace the part (called replacement service)
- Local service (user made changes) or IBM relief service
- Directly modify the object code (emergency situations only, not generally recommended)

The methods used to obtain the changed version of an object or part are different in each case but, for all types of service, the ground rule is to keep the original IBM supplied part, or object, untouched. To do this, you need a carefully designed control structure that keeps track of changes, and the sequence in which they should be applied.

### Update Service

This type of service is used for parts for which IBM supplies the source code. The original (base) source file is never changed. Instead, the changes are contained in separate update files. Both the base and the update files contain sequence numbers in the rightmost columns of each record. In addition, the update files contain control statements that use the sequence numbers, causing the records to be changed, deleted, or new ones to be added. Using the CMS UPDATE (or the EXECUPDT) facility, the changes are merged with the base to produce a new, updated source file, but the base code remains unchanged.

The procedure is similar to that used when you create the PPF from the base $PPF and override files.

The order in which updates are applied is crucial. Consider one update that adds a record to a base file, another update that changes its contents, and a third update that deletes it. It is critical which one comes first.

Update files are related to parts or objects, and the sequence in which they are applied is determined by a control file (AUX) structure. There is at least one control file structure for each component of VM/ESA. Except for local service and patches, AUX files are generated by VMSES/E, so you will not have to worry about that.

A discussion of this topic can be found in “Update Control Files” on page 117. Service types and control files are also discussed in *VM/ESA: VMSES/E Introduction and Reference*.

For ASSEMBLE language parts, the updated source file is then assembled to produce the updated text deck. For $SOURCE parts, the updated source file is compiled to produce an executable.

XEDIT can be used to create the update files. When invoked with the CTL option, XEDIT uses the update control structure, creates an updated source file, and displays it for you to edit it. When you issue the FILE command, XEDIT will save in an update file only the records you have added or changed, and associated insert and delete information.

### Replacement Service

Some parts are serviced by replacing the part with a new part that has the changes already incorporated. Files are serviced in this way, for example, when the source files are not available (Object Code Only) or have limited availability (as might be the case for an optional feature). Replacement service also makes it easier to apply and maintain service, and saves space both on tape and on disk.

### Local Service

The VMSES/E receive and apply functions cannot handle local service. These tasks, as well as the corresponding Software Inventory updates, must be done manually.

**Note:** VMSES/E since VM/ESA Release 2.1 supports optional automatic updating of the Software Inventory for local service in Assembler language, and for message repository files. VM/ESA Release 2.2 has extended this support to $SOURCE parts serviced through EXECUPDT. Also, new options (LOGMOD, $SELECT) produce a result functionally similar to the apply step, further automating the local modifications process.

An example of local service application can be found in “Local Service” on page 122. That section also shows how to manually receive an individual PTF (not on a COR tape), so that VMFAPPLY and VMFBLD can take over and update all relevant Software Inventory tables. You can also find other examples in *VM/ESA: Service Guide*.

### Patches

Patches, or ZAPs, are emergency fixes to object code, and are to be considered as temporary solutions until replacement object code, or source code updates, can be provided.

Patch files are similar to update files in the sense that patch files produce changes in the link-edited nuclei, modules, or text decks without changing the original text files.

The use of these facilities is not generally recommended. For more information see *VM/ESA: Service Guide*.

## Multiple System Levels

In a real production environment, life is not simple. Ideally, you never apply service directly to the production-level system and applications, but rather to a test version. After a successful test, the newly installed or serviced program eventually goes into production. So, in effect, you should operate with at least two levels of system software.

After the switch to production, the disks that held the definitions of the old production version may now be used to install a new test version. However, if, in spite of extensive testing, there is a severe problem with the new production version, it would be nice to be able to switch back to the old one. This process is referred to as “backing out” the changes.

To have that ability we need three service levels for the software. VMSES/E has adapted this 3-level structure, and isolates each level on its own disk. The following terms are used to refer to the levels:

- **Production**: this disk contains all service that has been accepted as stable. Accepting service should be the result of a conscious decision, based on the belief that there will be no future need to remove any part of it. This decision is extremely important: the disk is the result of accumulating (also called merging) several service levels, and once a level is merged it is very difficult to undo the merging.
- **Intermediate**: this disk contains service that has been received, applied, and built but is still under test. Once you are *completely satisfied* with the test results, you can merge this level with the production level.
- **Alternate**: this is a work disk, or staging area, for the most recent (highest) level of service. It is used to receive and apply new service and verify the results. Building and testing this service-level, however, should wait until testing of the intermediate level is complete. The reason is that, normally, only two build levels, test and production, are defined.

The use of, and need for, an alternate disk is fairly obvious. Any process can go wrong for reasons ranging from power failures to program or media errors. You want your production system isolated until the new (or serviced) product has been validated.

The intermediate disk is required if you want to preserve the definition of the previously running level of service, while testing new service, which is usually a very good idea. This protects you from problems arising from cross-product dependencies (VMSES/E does not verify those), or one or more PTFs that may be bad. In that case it is necessary to back-out the service, which then becomes a very easy task (see Appendix C, “Removing Service” on page 217 for an example).

When counting levels, three is by no means a magic number. You can define more, if you need. The number of levels is limited only by the availability of CMS disk access modes.

## Recommended Logical Strings and Service Levels

As discussed in “Multiple System Levels” on page 26, you may need more than one service level per logical string. IBM recommends the following:

**DELTA** For VM/ESA Release 1.1, two levels, alternate and production; starting with VM/ESA Release 2 a single level, production, is defined, saving one access mode. Though, strictly, only one disk is needed, having two disks (the alternate is used as a staging area) allows you to more easily recover from a disk-full situation. To simplify this recovery, we do recommend that, for the DELTA string, you use a single SFS directory instead of using minidisks. As the DELTA string does not contain any executable parts, the number of disks (levels) in the string does not affect in any way the running system, or the possibilities and techniques of recovering from bad service.

**APPLY** Three levels are recommended. A newly serviced system should be thoroughly tested before being put into production. This requires a separate (intermediate) service level. However, there may still be problems that testing could not detect. Therefore, another service level is required to allow retaining, for a reasonable amount of time, the service level that was in production before the newly serviced product (or system). Meanwhile, more recent service might have appeared, and you would like to get started on it without mixing it up

with the already existing service. To be able to do so, you need the third (alternate) level.

VMSES/E receive and apply functions work with a single string with multiple disks (levels), and always place their results on the alternate (highest) level disk. The build function uses a different structure: multiple strings with one disk each. The reason is that products often require that their usable forms be placed in several independent disks. Examples include keeping all help files in a help disk, and isolating administrator-specific functions from general user functions by placing them in separate disks (for an example see “Minidisk/Directory Assignment Section” on page 49).

Because build results can be placed on several strings, defining multiple build levels differs from defining multiple delta and apply levels, as explained below:

**BUILD** One level per string is defined, but several strings are employed. Each string is, in a way, specialized. For example, online help panels may be held in a separate string. Since there is only one level per string, multiple product (or system) levels are achieved by building to a test string. Normally, two strings, production and test, are enough. Disks considered less critical might have just the production level. As an example, CMS has four strings defined:

- Test CMS system tools (normally the Maint 493 disk)
- Test CMS system disk (normally the Maint 490 disk)
- CMS system tools
- HELP disk

**Note:** There is no need to define the CMS system disk, as it is always available.

## Applying Preventive Service

Preventive service aims at fixing problems before they happen. This is possible because installation configurations vary enormously, and therefore one site may experience a problem well in advance of others. Problems that potentially may affect a large number of installations are collected in RSU tapes (see “Maintaining Your System” on page 23). With the quality improvements in code, and testing procedures, the fixes are smaller in numbers and more reliable. RSU tapes are available periodically, are cumulative, and should be applied regularly.

**Note:** Most products no longer supply RPTs as part of the PSU process, since the RSU tapes are cumulative.

### PSU Planning

As reach-ahead and local modifications may need to be reapplied, you may decide, based on the amount of new PTFs, to use the RSU in one of two ways:

- As a COR tape (also referred to as “PTFs only”).

  Follow this path if the number of new PTFs is small compared to the amount of reach-ahead service, or there are excluded PTFs.

- As a product refresh.

  Remember that the RSU comes with pre-applied and pre-built service. If there are few reach-ahead PTFs, and no PTFs to exclude, following this path will save you time and effort.

The VMFPSU EXEC, new in VM/ESA Release 2.2, can help you plan for the PSU process. VMFPSU uses information provided on the service tape (a special VVT file of file type VVT$PSU$ is shipped for each component) and compares it to the VVTs on your system. The special VVTs are obtained by executing a VMFINS INSTALL INFO command before calling VMFPSU (the files are placed on the SDISK). For a description of VVTs, see section "Update Control Files" on page 117.

VMFPSU generates a file listing the following:

- New PTFs (supplied with the RSU and not on your system)
- Reach-ahead PTFs (applied on your system but not on the RSU)
- Excluded PTFs (supplied on the RSU but excluded)
- A list of parts with local modifications that are also serviced by the RSU.

Once you have decided which way you are going to use the RSU, you can start preparations to receive the service.

### Preparation for Service

Before service can be applied, the system has to be prepared. In other words, we have to clear some minidisks to get work space. This discussion is valid for the DELTA string, if it has more than one level, and the APPLY string. Assuming a three-disk structure, a one-level or two-level merge can be performed:

- One-level merge

  The contents of the alternate are merged into the intermediate disk, leaving the alternate disk empty. This approach does not affect the production system, but affects the intermediate level. Unless this is exactly what you require, this merge should be done if, **and only if**, the intermediate level is empty.

- Two-level merge

  This should be done only if testing of the intermediate level is completed. First, the contents of the intermediate disk are merged into the production disk, leaving the intermediate disk empty. Next, the alternate disk is merged with the intermediate disk. This leaves us with an empty alternate disk, a new level of intermediate disk, and a new, higher-level production system.

Applying new service may, therefore, have to be deferred until testing the previously applied service is complete. The merge function and the tool provided by VMSES/E are described in "Merge" on page 103. You can also find more information in *VM/ESA: VMSES/E Introduction and Reference*.

### Other PSU Steps

For a discussion of the remaining steps please refer to *VM/ESA: Service Guide*, *VM/ESA: Service Guide for 370* or the product's service guide as appropriate.

## Applying Corrective Service

This section describes the VMSES/E corrective service process. The focus will be on the principles and data flows, using an approach similar to the one used in "Installing Products" on page 18. The minidisk configuration is the same as the one used for the installation example (see Figure 2 on page 19).

## Preparation for Service

Just as when doing preventive service, before you can apply corrective service you must prepare you system. Everything we discussed in “Preparation for Service” on page 29 applies here.

## Receive the Service

The VMFREC EXEC receives service from the service media and places it on the alternate (or only) DELTA disk. In the preparation step we made sure that this disk was empty, if the DELTA string contains more than one level, so all previously received service files are on the production disk. The receive process is as follows:

1. The first tape files loaded contain the Tape Descriptor File (TDF) and memos. The TDF can be seen as a high-level map of the tape. It is used to locate the service files for a specific component.
2. The header service files for the component are loaded. Among them is the Product Contents Directory (PCD) file. This file is a map of the logical tape, and is used to locate the other service files. See “VMSES/E Service Tapes” on page 214.
3. If there are updates to the $PPF, we have a special situation. The receive and apply steps must use the existing PPF. The new PPF will have to be recompiled before the build step can be done. This is a manual step, and the serviced and compiled PPF files must be copied to the SIDISK disk. This situation is detected during the build step by the VMFBLD EXEC, and warnings are issued to the user.
4. The service-level Software Inventory (on the production DELTA disk) and the PPF (on the SIDISK) are scanned for information on:
   - The general layout of the logical tape, how to handle the various parts, file processing sequence, and target disks.
   - The status of service already received.
5. The tape files are located using the PCD, and loaded to the alternate DELTA disk, in the desired sequence.

The information from the Software Inventory is needed to avoid loading parts in usable form, or service files that have been loaded previously. VMFREC uses a set of routines (called “part handlers”) to conditionally load parts of specific file types, thus avoiding the usable forms. This approach effectively eliminates the possibility of back-leveling. The benefit of the check for already installed service is that the alternate DELTA disk, if defined, can be kept relatively small.

6. Finally, the Software Inventory tables are updated to reflect the new receive status.

Figure 5 schematically shows the data flow, and where the files reside after the receive step. Please note that, after Release 2, all VM/ESA components have only one DELTA disk defined. This does not affect the logical data flow, as described above.

Also, please note that VMFREC may be invoked repeatedly, before the service process moves to the apply step. VMFREC appends the Apply and Exclude lists received from the tape to any Apply and Exclude lists already on the DELTA disk. In effect, this merges the several sets of service, which are then applied as a single entity. You should be aware of this fact so you can use it to your own advantage. For example, this can be useful to combine the processing of several COR tapes.

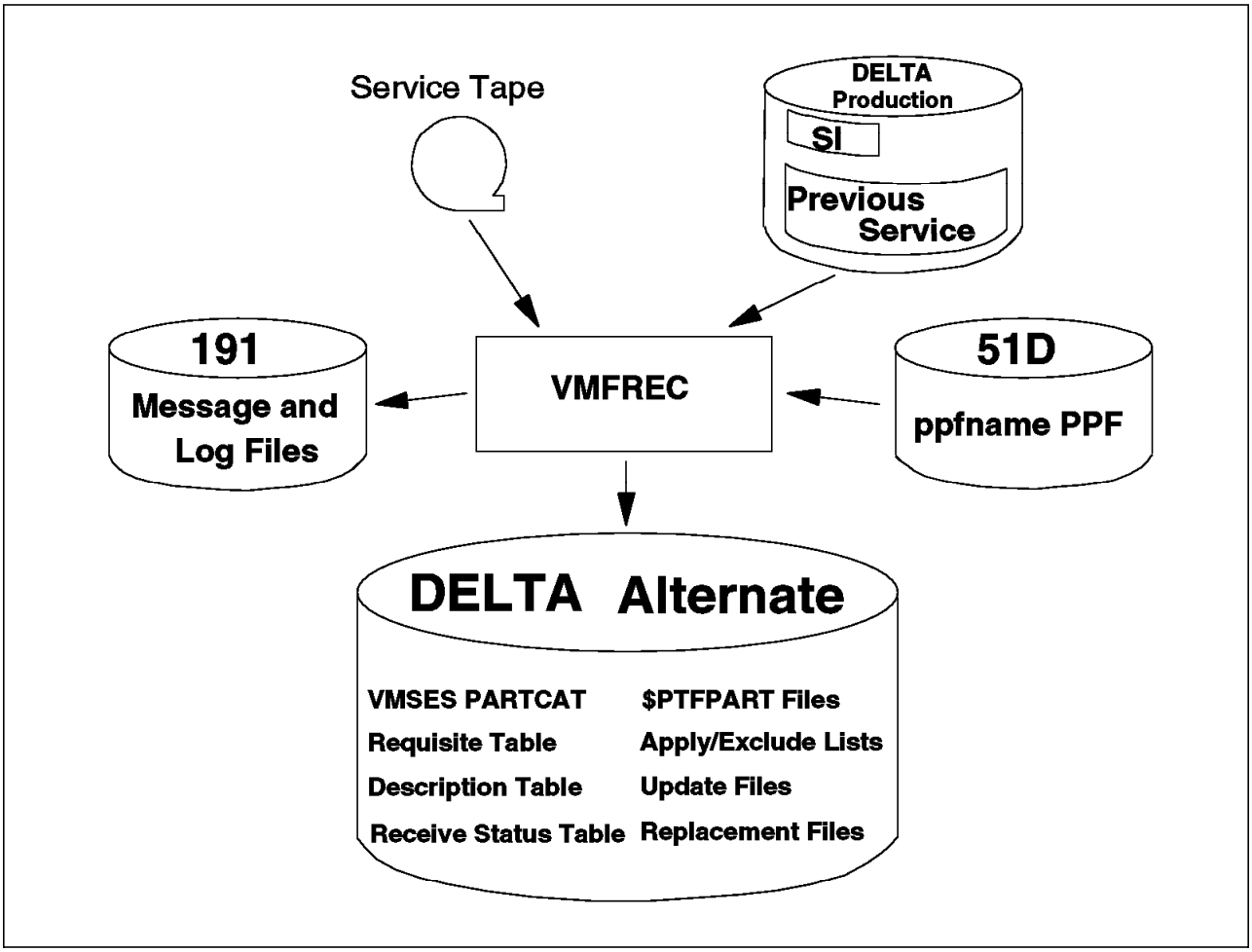



*Figure 5. Receive Step - Files and Data Flow*

### Apply the Service

The apply process produces an input to the subsequent build step. The VMFAPPLY EXEC uses control and status information as input, and produces updated control and status information as output. Therefore, no changes are made to the parts, objects, or usable forms of the product.

As in the receive step, we again assume that the maintenance levels have been consolidated. The alternate disk is empty, the highest maintenance level is on the intermediate disk, and the previous level is on the production disk.

VMFAPPLY combines the control information for the service we placed on the alternate DELTA disk, in the previous step, with the service level information on the intermediate APPLY disk. In sequence, what happens is:

1. VMFAPPLY has to know which new PTFs we are dealing with. So the Apply and Exclude lists, and the corresponding Requisite tables are read from the alternate (or only) DELTA disk.
2. We also have to know the starting point or, in other words, what service level we are updating from. This information is found in the service level tables, on the intermediate APPLY disk.
3. Finally (as always) we get fundamental service information for the particular product or component from the PPF.

All necessary input information is now in storage and the actual PTF processing starts:

4. Using the Apply and Exclude lists as a “shopping list,” each PTF is individually processed. Descriptive information is obtained from the respective $PTFPART files.
5. The $PTFPART files are scanned, checks are made for requisites, and missing service is identified.
6. If no errors are found, VMFAPPLY updates the Version Vector tables, and creates AUX files for the parts serviced by update service. Furthermore, a file with a list of serviced parts (the Select Data File) is updated. This file is later used by the build process to identify objects that have to be rebuilt.
7. If requisites are missing, or VMFAPPLY fails for any reason, the service level tables are not updated and VMFAPPLY stores two files on the A-disk:
   - RETRY $APPLIST, which contains a list of PTFs that passed requisite checking. If you run VMFAPPLY again, using this file as an apply list, these PTFs are applied.
   - appid $MISSING, which lists the parts identified as missing, where “appid” is the name of the service Apply Status table.

Figure 6 schematically shows the data flow, and where the files reside after the apply step.

The Apply process is described in *VM/ESA: VMSES/E Introduction and Reference*.

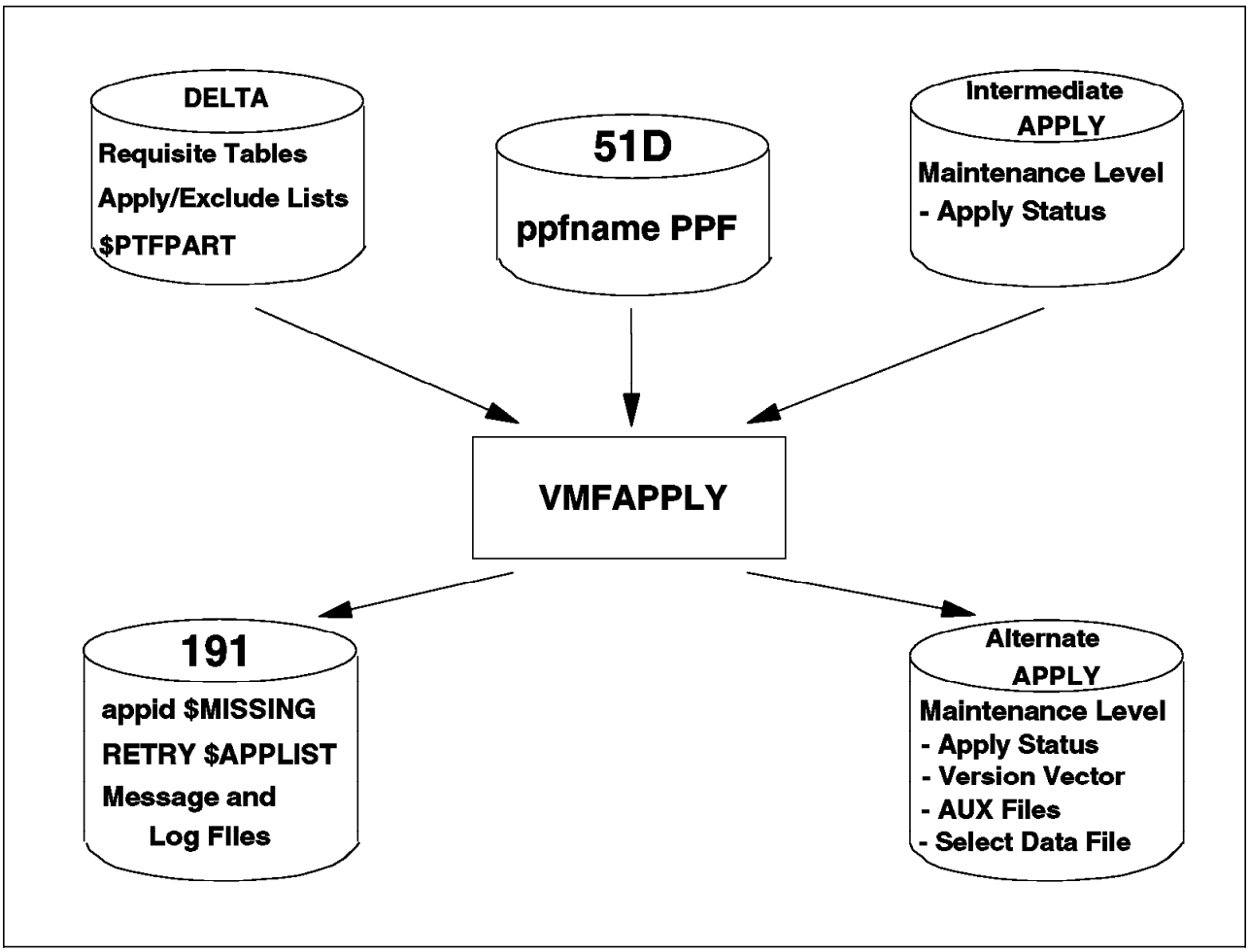



*Figure 6. Apply Step - Files and Data Flow*

### Build the Product

So far we have done nothing in terms of usable forms. The service and update files are on disk, and the apply function has created a “recipe” of how to use them. To build the product from the newly received and serviced parts, the VMFBLD EXEC has to be invoked.

The build process can be divided into three basic steps:

1. Identify the objects to be rebuilt due to the application of service. This includes any saved segments defined by the product.
2. Rebuild these objects, excluding saved segments. Saved segments are built at a later stage.
3. Manually build objects and perform other post-build tasks that cannot be performed by VMFBLD.

The main events are as follows:

1. Start by satisfying any assemble requirements. Though service ships preassembled serviceable parts, any local modifications you may have made require re-assembly of the affected parts. Also, now is the time to take care of any patches and fixes.

2. One of the first tasks VMFBLD performs is to identify whether the PPF has been serviced. If so, VMFBLD prompts you to compile the PPF. If you reply indicating you wish to compile, VMFBLD copies the serviced $PPFs to the A-disk. Compiling has to be a manually initiated operation, for VMSES/E has no way of knowing which user overrides exist and whether they are still valid. The user should do this validity check, perform any required changes to the override $PPF, and invoke VMFPPF to compile the serviced $PPF and its overrides. The files should be copied to the SIDISK. Once the user indicates that all $PPFs are compiled, the build can proceed.
3. The Select Data File created in the apply step and the Build Status table for the component to be updated are used as a first input to VMFBLD. The Select Data File is read from the alternate APPLY disk, and the Build Status table from the intermediate APPLY disk (since no builds have been performed yet, in our scenario, on the alternate).
4. Using the list of serviced parts from the Select Data File, and the build lists (which describe which parts an object is built from), the objects that have to be rebuilt are identified and flagged in the Build Status table as SERVICED. The name of the build list is also included. When the serviced part is itself a build list, VMFBLD compares the serviced build list with the previous version to determine whether any objects where deleted, added, or had their definition changed. These objects are flagged in the Build Status table as DELETE, SERVICED and SERVICED, respectively.
5. Based on the updated Build Status table, VMFBLD now starts to build objects. VMFBLD calls object-type-specific part handlers to do the job. Using the build list name (taken from the Build Status table) as the key, the appropriate part-handler is identified by searching the :BLD section of the PPF. Part handlers may need the following files:
   - Build list
   - Version Vector table
   - AUX files (for source updated objects, if a VVT is not available)
   - PTF-numbered parts

The actions taken by a part handler vary from simply copying and renaming a file, to the creation of a nucleus.

6. The output from a part handler is:
   - A usable form
   - Status information on the built objects
7. Based on the status information provided by the part-handlers, VMFBLD updates the Build Status table.

Figure 7 on page 35 schematically shows the data flow, and where the files reside after the Build step.

There are some objects which VMFBLD cannot build. Servicing parts contained in these objects will cause the objects to be flagged in the Build Status table as requiring rebuilding. So, when VMFBLD processing completes, there may remain (depending on the product) a few manual build steps. However, the software inventory can be updated manually and thus, still reflect the true build status of your system.

A description of the Build process can be found in “How Build Works” on page 110.

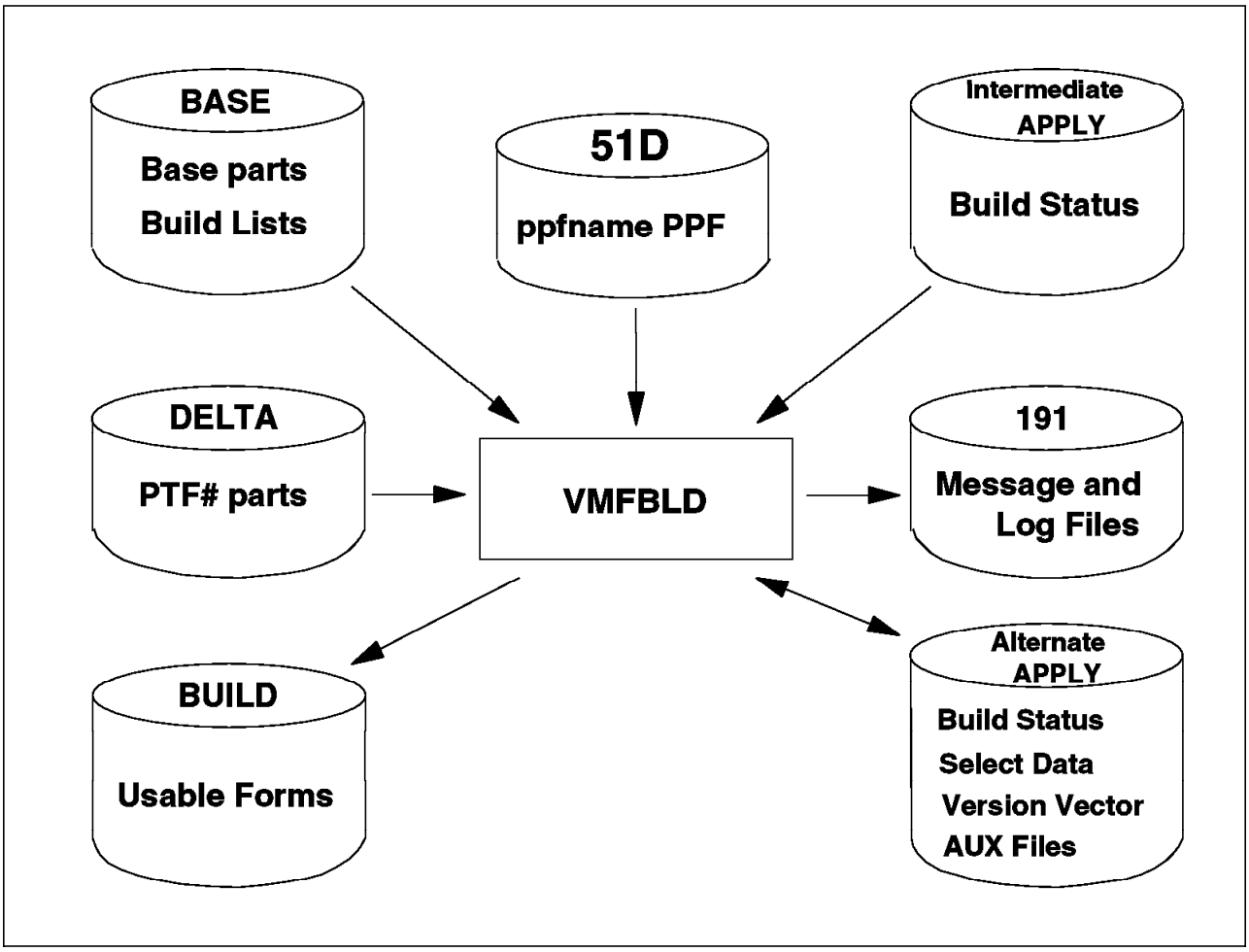



*Figure 7. Build Step - Files and Data Flow*

### Moving the Serviced Product to Production

After thoroughly testing the serviced product you will move it into production. How you do it depends on several factors:

- Build disks shared with other products
- Use of SFS directories or minidisks
- Product defined NSSs or saved segments

The VM/ESA Release 1.1 service publications described a method of moving a serviced component to production that had the following steps:

1. Using DDR, copy the production disk to the test disk.
2. Build the component and test.
3. Swap the test and production build disks on the CP directory.

This method:

- Is restricted to minidisks.

- Requires that the build disks are not shared by independent products. If the disks are shared, when you do the DDR copy you may back-level the products you are not servicing.

  Some VM/ESA components share disks, for example, CMS and REXX, CP and DV. It is very important to service these components following the rules and sequence defined in the *VM/ESA: Service Guide*, otherwise back-leveling may occur.
- However, this method allows you to quickly fall back to the old production level, if something goes wrong with the new level.

**This method should not be followed.** As a matter of fact, the VM/ESA Release 2.1 and Release 2.2 service publications recommend a very different method. However, the new method does not eliminate the back-leveling problem discussed above, and as such only works for VM/ESA or for products using separate build disks.

If you are using SFS directories you will have to be careful or you may lose the authorizations and aliases.

Saved segments cannot be tested: once generated, they immediately replace the production version. Should they cause errors, the previous versions will have to be made available, either by restoring the old system data files from tape or by re-generating them.

On “Production” on page 108 we further discuss the disadvantages of the new method and how to avoid them. We also address the problem of falling back to the previous production level.

# Chapter 3. Software Inventory

The Software Inventory concept was introduced in “Software Inventory” on page 5. Its purpose is to keep track of the status of the products and service that have been installed on the system, and to record and maintain information about saved segments. The Software Inventory implements a “relational view” of the components, parts, and objects of the system.

A VMSES/E utility command, VMFSIM, allows you to query and manipulate the Software Inventory.

VM/ESA Release 2 introduced two tools that exploit the Software Inventory: VMFQOBJ and VMFINFO. VMFQOBJ complements VMFSIM by providing information about objects defined in build lists. VMFINFO is a full-screen, panel-driven interface to VMFQOBJ and most VMFSIM query functions, which helps you manage the software on your system(s).

In “Installed Software” on page 10, we briefly discussed the Software Inventory and what it can be used for. This chapter expands on that subject.

Complete information on the Software Inventory design, and examples of the tables that constitute it, can be found in *VM/ESA: VMSES/E Introduction and Reference*.

## Introduction

You, as a systems programmer, may from time to time have many questions concerning the status of your systems. Some of those questions might be:

- Which products are installed on my system?
- What are the prerequisites for a component?
- Which PTFs were applied to a component?
- Which APARs are contained in a PTF?
- What is the status of a given APAR?
- Which PTFs depend on a given PTF?
- Which parts are serviced by a PTF?
- What service is applied to a part?
- Which parts must be rebuilt after applying service?
- What will be the impact of removing this PTF?

The questions above, and many more, may be answered using the Software Inventory. The VMFSIM command is the interface to the Software Inventory. As briefly explained above, VMFINFO is a new utility command that actively exploits VMFSIM. Examples on the use of VMFINFO can be found in “VMFINFO Command” on page 145. Now that we have seen what the Software Inventory may be used for, let us take a closer look at how it is set up.

**Note:** Throughout the remainder of this chapter the term product may also be used in the place of component.

The Software Inventory consists of two main parts:

- The system-level Software Inventory
- The service-level Software Inventory

Each of these two levels is a collection of tables. These tables can also be grouped by type. Most of these types exist in both levels, but some are unique to one level, as indicated below.

The system-level Software Inventory resides on a separate disk, by default MAINT 51D, hereafter referred to as the SIDISK (or SID-DISK, as 51D visually looks like SID, just as 5E5 looks like SES). The system-level tables are:

- Receive Status table (SYSRECS), describing when each product was received.
- Description table (SYSDESCT), describing each product.
- Requisite table (SYSREQT), describing the requisites for each product.
- Apply Status table (SYSAPPS), describing the apply status for each product.
- Build Status table (SYSBLDS), describing the build status for each product. There is a separate system-level table to track the service build status of system objects (SRVBLDS), such as saved segments.

The following system-level file resides, by default, on the SIDISK, but can be moved:

- Segment Data File (SEGDATA), contains customized parameters for building saved segments.

The following system-level table resides on the SESDISK:

- Filetype Abbreviation table (SYSABRVT), contains the list of abbreviations of CMS file types used by the IBM service process.

The service-level Software Inventory tables for each product reside on that product's DELTA and APPLY strings. The tables are:

- Receive Status table (SRVRECS), describing when each PTF was received.
- Description table (SRVDESC), describing each PTF.
- Requisite table (SRVREQT), describing the requisites of each PTF.
- Apply Status table (SRVAPPS), describing the apply status of each PTF.
- Build Status table (SRVBLDS), describing the build status of each serviced object.
- Version Vector table (VVTlvl), contains the history of all PTFs that have been applied to the parts of a product.

Also residing on the SIDISK, though not formally part of the Software Inventory:

- Product Parameter File (PPF and $PPF), which is the key file for VMSES/E to install and service a component.
- Product Parts file (PRODPART), which describes a product in terms of tailorable parts, requisites, and so on.

Also residing on the DELTA string, though not formally part of the Software Inventory:

- The PTF Parts file ($PTFPART), which describes a PTF in terms of parts, requisites, and so on.

**Notes:**

- VMFSIM also manages the PPF, PRODPART, and $PTFPART files.
- The PRODPART and PTFPART files are described in *VM/ESA: VMSES/E Introduction and Reference* as belonging to the Software Inventory. We believe this to be wrong: These files *provide input* to the Software Inventory but are part of a product.

The Software Inventory also contains a parts catalog, which consists of one file (table) for every disk used by VMSES/E. The file is always named VMSES PARTCAT, and we refer to it as the PARTCAT file. The PARTCAT file describes the status and change history of all the VMSES/E managed files that reside on *that* disk. In this way VMSES/E can keep track of all the files belonging to each of the components of the products installed, as long as the products adhere to the VMSES/E rules.

All the tables in the Software Inventory are implemented as CMS flat files. The internal structure is made up of “tags,” just as in the CMS NAMES file. Each table consists of one data structure. It has one key field, and one or more data fields subordinated to the key field.

Figure 8 shows the layout of the Software Inventory and the location of the tables.

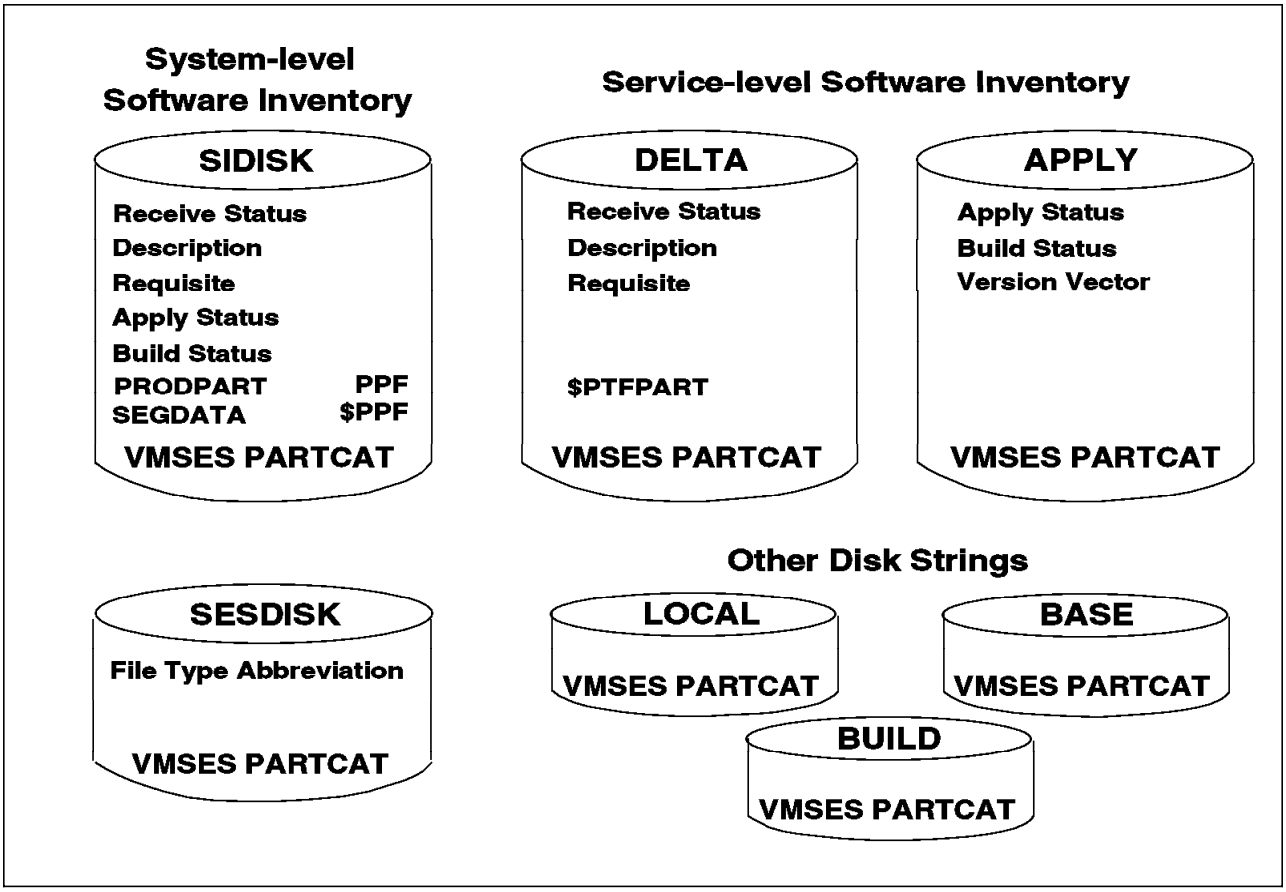



*Figure 8. Location of the Software Inventory Tables*

## System-level Software Inventory

The Software Inventory consists of several tables, containing the following types of information:

- Product information
- Build status information
- Supporting information

All tables in the system-level Software Inventory are kept on the SIDISK, together with the PRODPART files, $PPFs, PPFs, and the SEGDATA files. Because the SEGDATA files contain shared-product information on system objects they are not described here (see "System-Level View" on page 67 and "Segment Data File" on page 162).

Before we proceed, let us have a look at one of the tables. Figure 9 shows the system-level Receive Status Table.

```
:PPF.ESA       VMSES   :PRODID.6VMVMK22%VMSES    :STAT.RECEIVED.03/22/94.16:55:02.MAINT.201-0000
:PPF.ESA       CP      :PRODID.6VMVMB22%CP       :STAT.RECEIVED.03/22/94.17:28:06.MAINT.201-0000
:PPF.ESA       DV      :PRODID.6VMVMI22%DV       :STAT.RECEIVED.03/22/94.17:43:32.MAINT.201-0000
:PPF.ESA       CMS     :PRODID.6VMVMA22%CMS      :STAT.RECEIVED.03/22/94.18:38:59.MAINT.201-0000
:PPF.ESA       REXX    :PRODID.6VMVMF22%REXX     :STAT.RECEIVED.03/22/94.18:46:24.MAINT.201-0000
:PPF.ESA       GCSSFS  :PRODID.6VMVML22%GCS      :STAT.RECEIVED.03/22/94.06:38:15.MAINT.201-0000
:PPF.ESA       TSAFSFS :PRODID.6VMVMH22%TSAF     :STAT.RECEIVED.03/22/94.07:20:26.MAINT.201-0000
:PPF.ESA       AVSSFS  :PRODID.6VMVMD22%AVS      :STAT.RECEIVED.03/22/94.07:38:08.MAINT.201-0000
:PPF.CUF       CUF     :PRODID.6VMVME11%CUF      :STAT.RECEIVED.06/29/92.13:17:44.MAINT
:PPF.MYCUF     CUF     :PRODID.6VMVME11%CUF      :STAT.DELETED.07/12/92.15:28:47.MAINT
:PPF.5798DWDC  NONE    :PRODID.5798DWDC%RTM      :STAT.RECEIVED.04/22/93.12:08:11.MAINT
:PPF.ESA       TSAF    :PRODID.6VMVMH22%TSAF     :STAT.DELETED.04/22/94.11:33:22.MAINT.201-0000
```

*Figure 9. System-Level Receive Status Table - VM SYSRECS*

Notice that there are three fields in this table:

**:PPF** The PPF name used to receive the component or product, followed by the component name in the same PPF.

**:PRODID** The product identifier.

**:STAT** The receive status for the component or product. As you see, it can have four or five sub-fields:

- Status (RECEIVED, DELETED)
- Date of last status change
- Time of last status change
- User ID performing the last operation
- Service level of the component, if supplied

This example of one of the tables gives you the general concept. Hence, we will only summarize the rest of the tables below, indicating, for the most important tags, which table they belong to and their meaning.

Table 1 on page 41 summarizes the system-level tables.

Please note the "file type" row. It contains some names (actually file types) that we will use as a short name for the tables in the following text.

*Table 1. System-Level Tables Summary*

| Field Description | Tag Name | Description table | Requisite table | Receive Status table | Apply Status table | Build Status table | Filetype Abbreviation table |
|---|---|---|---|---|---|---|---|
| Disk of residence | n/a | SIDISK | SIDISK | SIDISK | SIDISK | SIDISK | SESDISK |
| File type for this table | n/a | SYSDESCT | SYSREQT | SYSRECS | SYSAPPS | SYSBLDS | SYSABRVT |
| Product identifier | :PRODID | √ | √ K | √ | √ | √ | |
| Name of PPF used | :PPF | √ K | | √ K | √ K | √ K | |
| Component name in PPF | :PPF | √ K | | √ K | √ K | √ K | |
| Product description | :DESC | √ | | | | | |
| Requisite information | :PREREQ<br>:REQ<br>:SUP<br>:IFREQ<br>:NPRE<br>:DREQ | | √<br>√<br>√<br>√<br>√<br>√ | | | | |
| Product status | :STAT | | | √ | √ | √ | |
| Date/time of last change in product status | :STAT | | | √ | √ | √ | |
| User ID last changing product | :STAT | | | √ | √ | √ | |
| Service level | :STAT | | | √ | | | |
| 3-character file type abbreviation | :ABBRFT | | | | | | √ K |
| Corresponding real CMS file type | :REALFT | | | | | | √ |
| 8-character identifier for associated base part | :BASEFT | | | | | | √ |

**Legend:**

√ A check indicates that the field so marked exists in the listed table.

**K** This character is used to identify key-fields.

## Service-level Software Inventory

There are three types of information in the service-level Software Inventory:

- PTF information
- Maintenance-level information
- Build status information

As we have seen in “Introduction” on page 37, the service-level Software Inventory tables are located on the DELTA and APPLY strings. The service-level Software Inventory is maintained only for products in VMSES/E format. Table 2 on page 42 shows the most important tags in the service-level Software Inventory tables.

Note that the tables are cumulative. For example, the APPLY string has three levels and, over time, each level will contain an Apply Status table. The Apply Status table on the alternate apply disk will, then, contain all the PTFs that were ever applied to the system, while the Apply Status table on the intermediate apply disk will contain its own PTFs, and the PTFs on the production level.

Also note there is a Version Vector table for each maintenance level defined in the control file. For a description of the control file, see “Update Control Files”

on page 117. For example, if you have local fixes and are using the IBM defined naming conventions, you will have two VVTs of file types VVTVM (for IBM service) and VVTLCL (for the local service).

*Table 2. Service-Level Tables Summary*

| Field Description | Tag Name | Description Table | Requisite Table | Receive Status Table | Apply Status Table | Build Status Table | Version Vector Table |
|---|---|---|---|---|---|---|---|
| String of residence | n/a | DELTA | DELTA | DELTA | APPLY | APPLY | APPLY |
| File type for this table | n/a | SRVDESCT | SRVREQT | SRVRECS | SRVAPPS | SRVBLDS | VVTlvl |
| PTF number | :PTF | | √ K | √ K | √ K | | √ K |
| Status of PTF/object | :STAT | | | √ | √ | √ | |
| Date/time of change in PTF/object status | :STAT | | | √ | √ | √ | |
| User ID doing last change of PTF/object status | :STAT | | | √ | √ | √ | |
| APAR number | :APARNUM | √ K | √ | | | | |
| APAR description | :ABSTRACT | √ | | | | | |
| Requisite information for PTF | :PREREQ<br>:COREQ<br>:IFREQ<br>:SUP<br>:HARDREQ | | √<br>√<br>√<br>√<br>√ | | | | |
| PTF/APAR number applied to part | :PTF | | | | | | √ |
| Part ID | :PART | | | | | | √ K |
| 8-character file type of source update file | :PTF | | | | | | √ |
| Build list name | :BLDLIST | | | | | √ K | |
| Name of object generated from build list | :OBJECT | | | | | √ | |
| Name of parts serviced by last apply | :PARTID | | | | | √ | |

**Legend:**

√ A check indicates that the field so marked exists in the listed table.

**K** This character is used to identify key-fields.

## Basic Information Sources

As described in “Parameter Files on the Product Tape” on page 12, and “Parameter Files on the Service Tape” on page 13, the input files for the Software Inventory are:

- PPF for a product or component
- PRODPART file for a product or component
- $PTFPART file for a PTF
- Tape contents for a product or component

We also observed in “Introduction” on page 37 that the PRODPART and $PTFPART files are not part of the Software Inventory.

These source data files for the system-level Software Inventory and the service-level Software Inventory, are described in more detail in the following sections. Although, formally, it is not part of the Software Inventory, the Product Parameter File is so important that we will begin our descriptions with it.

## Product Parameter File

The use of the Product Parameter File (PPF) was introduced in “Concepts and Sources of Information” on page 10. In this section, we will describe this file′s contents in more detail.

The PPF is really the focal point of VMSES/E for a product or a component. It contains all the information about how the product is defined and how it will be handled by the different stages of the installation and service processes.

### Overview

For VMSES/E-enabled products (VM/ESA included) both the source form and the usable form of the PPF are shipped in the distribution media. This PPF contains the IBM defined defaults for the product. Having the source form available allows you to override those defaults (see “Default Overriding” on page 13). The source form has a file type of $PPF and is not directly usable by the VMSES/E commands. To make it usable, you have to “compile” the source file using the VMFPPF command. This command produces a new file, with file type PPF, which is the usable form. The key points to observe are:

- VMSES/E functions use only “compiled” PPFs.
- After any local changes or service to the source PPF, the systems programmer is responsible for generating the usable form of the PPF.

Because compiling the PPF is a human decision, performance has improved over the previous implementation in VMSES: the old functions would automatically re-compile the PPF every time they were called, to make sure they were using the most recent version.

VMSES/E-enabled products have one or more PPFs. The convention in VM/ESA is one source PPF per component. The name of the file is normally the product or component ID, which may be rather cryptic. For example, the component ID (or product ID) for CMS in VM/ESA Release 2.2 is 6VMVMA22, which is not obvious. To remedy this problem, provide flexibility, and because IBM strongly recommends that you never alter the PPFs that IBM delivers, the concept of PPF override has been introduced.

A PPF override is logically similar to a CMS UPDATE file because it introduces changes to the base PPF. Just as a CMS file may have many levels of updates, so may the PPF have a chain of overrides. This gives you flexibility in tailoring the environment to your needs.

An example of a PPF override is found in “Creating a PPF Override” on page 189.

## VMFOVER and VMFPPF Commands

To extend the analogy to the CMS UPDATE command a little further, we introduce the VMFOVER command. It has the same function as the CMS UPDATE command: to apply the changes made in the PPF override to the base PPF.

The result is a file with the same file name as the override file, but with a file type of $PPFTEMP. This file must then be "compiled" using the VMFPPF command to produce the executable form of the PPF.

Note that the VMFPPF command will automatically call the VMFOVER command; therefore, in practice, you would only invoke VMFOVER if you wanted to separately validate your overrides before attempting to compile. VMFPPF can also compile, with a single invocation, all components of a source PPF.

Invoking VMFPPF is a manual operation. There is no provision in VMSES/E to automatically call it. Therefore, you completely control when to bring a newly serviced PPF into production status.

## PPF General Structure

To begin our description of the PPF, let us take a look at its general structure, shown in Figure 10.

```
:COMPNAME   - PPF COMPONENT DEFINITIONS (KEY)
  :PRODID   - PRODUCT IDENTIFIER (FIELD)

  :CNTRLOP   - CONTROL OPTIONS (BLOCK)
    ⋮
  :ECNTRLOP    - END CONTROL OPTIONS (EBLOCK)
  :DCL       - DECLARE VARIABLE SECTION (FIELD)
    ⋮
  :EDCL        - END DECLARE VARIABLE SECTION (FIELD)
  :MDA       - MINIDISK/DIRECTORY ASSIGNMENTS (FIELD)
    ⋮
  :EMDA        - END MINIDISK/DIRECTORY ASSIGNMENTS (FIELD)
  :RECINS    - INSTALL TAPE DEFINITIONS (FIELD)
    ⋮
  :ERECINS     - END INSTALL TAPE DEFINITIONS (FIELD)
  :RECSER    - SERVICE TAPE DEFINITIONS (FIELD)
    ⋮
  :ERECSER     - END SERVICE TAPE DEFINITIONS (FIELD)
  :BLD       - BUILD DEFINITIONS (FIELD)
    ⋮
  :EBLD        - END BUILD DEFINITIONS (FIELD)
  :DABBV     - PTF/REAL FILETYPE DEFINITIONS (FIELD)
    ⋮
  :EDABBV      - END PTF/REAL FILETYPE DEFINITIONS (FIELD)

  :END      - END PRODUCT/COMPONENT DEFINITIONS (EBLOCK)
```

*Figure 10. PPF Structure*

Now let us look at the PPF in more detail. We will use the name PPF interchangeably for both $PPF files and PPF files, because their general

structures and contents are the same. The differences are summarized in Table 3 on page 45.

A PPF contains several areas. Each area can be further subdivided in sections. Each area, or section, is made up of a number of tags, in uppercase, starting with a colon (:) in column 1. The tag may accept parameters, which may either be on the same line, or on the following lines. In the latter case, the parameters are terminated by an end tag (for example :MDA and :EMDA).

The sections of each file are described below, but details for all tags are not given.

*Table 3. Valid Areas and Sections for Each PPF Form*

| **Area/ Section** | **Source $PPF** | **Override $PPF** | **Temporary $PPFTEMP** | **Usable Form PPF** |
|---|---|---|---|---|
| Header | √ | √ | | |
| Component | √ (•) | | √ (•) | √ (•) |
| Control options | √ | | √ | √ |
| Variable declarations | √ | | √ | √ |
| Minidisk/directory assignments | √ | | √ | √ |
| Receive install tape definition | √ | | √ | √ |
| Receive service media definition | √ | | √ | √ |
| Build product definition | √ | | √ | √ |
| Filetype abbreviations | √ | | √ | √ |
| Override | √ (•) | √ (•) | | |
| Control options | √ | √ | | |
| Variable declarations | √ | √ | | |
| Minidisk/directory assignments | √ | √ | | |
| Receive install tape definition | √ | √ | | |
| Receive service media definition | √ | √ | | |
| Build product definition | √ | √ | | |
| Filetype abbreviations | √ | √ | | |

**Notes:**

- Can have one or more component areas
- Only one component area is allowed
- Can have zero or more override areas
- Can have one or more override areas

## Header Area

The header area is the first area in the PPF. Among other information, it contains the list of the components described in the next area, the component area. Figure 11 shows the header area for the CMS component.

```
:COMPLST.  CMS
:OVERLST.  CMSUCENG CMSPTFS
```

*Figure 11. $PPF for CMS - Header Area*

The :COMPLST tag lists those components that are described in this PPF, and is important if you want to create an override file. The override must refer to the PPF file name and the component name.

The :OVERLST tag indicates that this PPF contains several override areas.

## Component Area

The next area of the PPF is the component area, which consists of a set of data blocks, called “sections.”

The :CMS tag starts the definition of the CMS component, as shown in Figure 12.

The :PRODID tag is also important. It contains the product ID (in this case 6VMVMA22) and the component name (in this case CMS). The composite term 6VMVMA22%CMS is called the REQID, and is used only during installation by VMFINS. This tag is also found in the PRODPART file and in the Software Inventory tables, which are discussed in “System-level Software Inventory” on page 40.

```
:CMS.
:PRODID. 6VMVMA22%CMS
```

*Figure 12. $PPF for CMS - Component Area (Excerpt)*

### Control Options Section

The first block is the Control Options section. It describes general attributes of the component such as:

- File names for some of the service-level Software Inventory tables.
- Whether messages will be logged.
- Whether VMSES/E should automatically access the pertinent disks each time a VMSES/E command is issued.
- The system national language.
- The names of the control file, default apply list, and exclude list.
- The type of consistency checking to be done during the apply and build steps.
- The PTF and APAR file type prefixes for the component.

Figure 13 on page 47 shows the control options section for CMS.

**Note:** The :USEREXIT tag allows you to define a pre- and post-processing Exit EXEC for several VMSES/E functions. If defined, this EXEC is called by the VMFREC, VMFAPPLY, VMFBLD, VMFMRDSK, VMFASM, VMFHASM, VMFHLASM, GENCPBLS, VMFEXUPD, and VMFNLS commands, upon entry and exit.

Two or three parameters are passed to the user exit EXEC:

- The name of the command being executed.
- The word SET-UP or CLEAN-UP, for pre- and post-processing exits, respectively.
- On the CLEAN-UP call, if an error has occurred, an optional third parameter, the return code with which the calling function failed, is passed.

The :VERSION tag was introduced in Release 2 for compatibility purposes. It allows VMSES/E to recognize products formatted for earlier releases and act accordingly. Also, the :CKAUX tag is no longer used.

As can be seen, this section contains many items of useful information, some of which may be overridden by options on the commands that use the PPF.

```
:CNTRLOP.

* TAG      VALUE(S)
*--------- ---------
:PRODDESC. CMS for VM/ESA 2.2   * Product description
:BCOMPNAME. CMS                 * Base component name
:VERSION.  VM/ESA 1.2.2         * VMSES/E level required
:RECID.    6VMVMA22             * File name of service
                                * receive status table
:APPID.    6VMVMA22 6VMVMF22    * File name of service
                                * apply status table
:BLDID.    6VMVMA22             * File name of service
                                * build status table
:LOG.      YES                  * Log all messages
:RECVALL.  NO                   * Receive missing parts for
                                * committed PTFs
:SETUP.    NO                   * Call VMFSETUP
:SLVI.     V/DS                 * System level and version
                                * indicator
:NLS.      AMENG                * System language
:CNTRL.    DMSVM                * Control file name
:AXLIST.   DMSVM                * File name of IBM supplied
                                * APPLY list and EXCLUDE list
:EXCLIST.                       * File name of user's own
                                * EXCLUDE list
:UPDTID.   AUXVM                * File type of AUX file
:CKAUX.    YES                  * VMFBLD compare AUX file to
                                * Standard Self Documentation
                                * Information in text decks
:CKSDI.    NO                   * VMFAPPLY check Standard Self
                                * Documentation in text decks
:CKVV.     NO                   * Check AUX file against
                                * corresponding version vector
                                * during vmfbld
:CKGEN. YES                     * Check AUX file against
                                * corresponding version vector
                                * during vmfasm, vmfhasm, vmfhlasm
                                * vmfnls, and vmfexupd
:RETAIN.                        * List of file modes that
                                * VMFSETUP cannot use
:USEREXIT.                      * User exit EXEC called for
                                * setup and cleanup at the
                                * beginning and end of each
                                * service function
:PTFPFX.   UM                   * two character PTF prefix
:APARPFX.  VM                   * two character APAR prefix

:ECNTRLOP.
```

*Figure 13. $PPF for CMS - Control Options Section*

## Variable Declarations Section

The next section is the Variable Declarations section. This section was introduced in VM/ESA Release 1.1. It defines three types of symbolic variables:

**LINK** Link statement to link to a minidisk

**DIR** The name of an SFS directory

**USER** The user ID of a virtual machine needed for component execution

The variable names start with an ampersand (&). The next word is the keyword, as defined above, which declares the type of variable defined. These variables are used in the PPF and in the PRODPART file.

The concept of variables has several advantages over previous VMSES implementations. Variables allow you to:

- Define in one place all the minidisks or directories to be used for a product. The symbols thus defined can be used in other files; for example, the PRODPART file.
- Automatically link to minidisks or access directories not owned by you.
- Define user IDs to be used by VMFINS when performing installation tasks.

Using variables has several consequences:

- It allows users other than MAINT to perform maintenance and installation tasks, given appropriate access to the disks involved. If you are performing an installation task, or servicing with the PSU process, you will need write access to the SIDISK (MAINT 51D, by default).
- It allows a full description of the disks used, not only the virtual address, but also the owning user ID.
- When servicing program products, it allows the servicing user ID to automatically link and access the product disks needed.

The VMFSETUP command accesses the disks for a component, and it reads the DCL section to get the LINK information.

Figure 14 shows an excerpt from the variable declaration section from the $PPF for CMS.

```
:DCL.
&LMODZ LINK MAINT 3C4 3C4 MR  * Disk for local mods
&SAMPZ LINK MAINT 3C2 3C2 MR  * Sample files
&DELTZ LINK MAINT 3D2 3D2 MR  * CMS service
&APPLX LINK MAINT 3A6 3A6 MR  * Aux & software inventory files
&APPLY LINK MAINT 3A4 3A4 MR  * Aux & software inventory files
&APPLZ LINK MAINT 3A2 3A2 MR  * Aux & software inventory files
&BAS2Z LINK MAINT 3B2 3B2 MR  * CMS object code & macros
       ⋮
&SRVSU USER VMSERVS           * VMSERVS userid
&SRVUU USER VMSERVU           * VMSERVU userid
&SRVRU USER VMSERVR           * VMSERVR userid
&CMSBU USER CMSBATCH          * CMSBATCH userid
       ⋮
:EDCL.
```

*Figure 14. $PFF for CMS - DCL Section (Excerpt)*

### Minidisk/Directory Assignment Section

This section defines the disk strings described in “Product Database Layout” on page 14, using the symbolic variables defined in the previous section. Note that a string may be empty. Starting with VM/ESA Release 2, components have one disk in the DELTA string (VM/ESA Release 1.1 has two). The APPLY string has three disks. This is reflected in the supplied PPFs. As discussed in “Recommended Logical Strings and Service Levels” on page 27, you may wish to create an override defining a small alternate DELTA area. Figure 15 shows the minidisk/directory section of the $PPF for CMS.

Note that the symbolic variables from Figure 14 on page 48 are used here. The variables are replaced by the actual (linked) minidisk address by the VMFPPF command during PPF compilation.

```
:MDA.
TASK                            * Disks accessed before the
                                * product service database
LOCALMOD    &LMODZ              * Disk for local mods
LOCALSAM    &SAMPZ              * Sample files
APPLY       &APPLX &APPLY &APPLZ * Aux & software inventory
                                * files
DELTA       &DELTZ              * CMS service
BUILD7      &BLD7Z              * Test CMS system tools
BUILD6      &BLD6Z              * Test CMS system disk
BUILD5      &BLD5Z              * HELP disk
BUILD2      &BLD2Z              * CMS system tools
BASE2       &BAS2Z              * CMS object code
BASE3       &BAS3Z              * CMS source definition
SYSTEM                          * Disks accessed after the
                                * product service database
:EMDA.
```

*Figure 15. $PPF for CMS - MDA Section*

### Receive Install Tape Definitions (RECINS) Section

This section describes the layout of the product tape, and what to do with the different files on the tape. Note that although we use the term tape for simplicity, the following discussion equally applies to electronic envelopes (remember that CD-ROMs emulate a tape). The section also introduces the concept of *tape part handlers*. A tape part handler is a specialized EXEC that handles a specific type of part on the input medium (we will talk about *build* part handlers later). This concept allows VMSES/E to grow, because new part types and part handlers may be added when needed.

Figure 16 on page 50 shows the receive install tape definitions section of the $PPF for CMS.

```
:RECINS.
* TAPEFILE  PARTHAND TARGET  DESCRIPTION
* --------- -------- ------- ------------
LOCALMOD   VMFRCALL LOCALMOD * Local fixes
AXLIST     VMFRCAXL DELTA    * Apply and Exclude lists
PARTLST    VMFRCPTF DELTA    * $PTFPART files
DELTA      VMFRCCOM DELTA    * Service
APPLY      VMFRCALL APPLY    * Service
TOOLS      VMFRCALL BUILD7   * CMS tools
SYSTEM     VMFRCALL BUILD6   * CMS system disk
NCHELP     VMFRCALL BUILD5   * New/changed HELP files
MACRO      VMFRCALL BASE2    * Macros
SOURCE     VMFRCALL BASE3    * Source files

:ERECINS.
```

*Figure 16. $PPF for CMS - RECINS Section*

As you see, the first column lists the symbolic names of the tape files, the second column the part handlers, and the third column the disk string that is to receive the files.

**Note:** The files are always received at the highest level of the disk string, which is the leftmost address in the definition of the string.

The installation tape contains two very important files (see Appendix B, “Product Packaging and Distribution Media Formats” on page 205):

- The Tape Descriptor File (TDF), shows the position of each component’s logical tape on the physical tape. A physical tape may contain one or more logical tapes, but a logical tape contains only one component.
- The Product Contents Directory (PCD), defines the sequence of the files on the logical tape, and can be thought of as a map of the logical tape. Its file name is the component ID.

The tape file names in the RECINS section must match the corresponding names in the PCD.

### Receive Service Media Definition (RECSER) Section

This section deals with service tapes and electronic envelopes, instead of installation tapes. Both media have a layout that differs from that of the installation tape, but the layout of the section in the PPF file is identical.

Figure 17 shows the receive service media definitions section of the $PPF for CMS.

```
:RECSER.
* TAPEFILE  PARTHAND TARGET  DESCRIPTION
* --------- -------- ------- ------------
AXLIST     VMFRCAXL DELTA    * Apply and Exclude lists
PARTLST    VMFRCPTF DELTA    * $PTFPART files
DELTA      VMFRCCOM DELTA    * Service

:ERECSER.
```

*Figure 17. $PPF for CMS - RECSER Section*

Although the figure does not show it, it is possible to skip files on the tape. When a tape-file is not listed in the PPF file, it is not loaded. Thus, those files will **not** be received.

Hence, neither AUX files nor MACLIBs will be received from the tape anymore. The AUX files are built from VVTs during the apply process, while the MACLIBs are built, like any other object, during the build process. This effectively eliminates the risk of accidentally back-leveling the product.

We will observe at this point one basic difference between installation and service. During installation we receive usable forms; during corrective service we do not.

**Note:** All the files received during service are placed on the DELTA string.

We also note that in during install (Figure 16 on page 50), the part handler VMFRCALL, which unconditionally loads all the files on the tape file, is used most often. The part handler VMFRCCOM, in the service case, does not load all the files it finds. VMFRCCOM does not load files already on the target disk string, and it does not load files belonging to a PTF that has a status of "committed," even if they do not exist on the target string.

The COMMITTED status was introduced by VMSES/E in VM/ESA Release 1.0. If you want to give a PTF the status of committed, you must do so with the VMFSIM command.

If a PTF is committed on the system, it will not be received at a later time. Hence, for committed PTFs, you may be able to erase any replacement parts shipped with them for which there are more recent replacement parts. Committing PTFs allows some disk space to be saved.

For example:

| **PTF** | **Services parts** |
|---|---|
| UM11111 | DMSAAA DMSBBB DMSCCC |
| UM22222 | DMSAAA DMSDDD DMSEEE |
| UM33333 | DMSAAA DMSCCC DMSDDD |

If you commit PTFs UM11111 and UM22222, you can erase (using the VMFERASE command):

```
DMSAAA TXT11111   DMSAAA TXT22222
DMSCCC TXT11111   DMSDDD TXT22222
```

but you **cannot** erase:

```
DMSBBB TXT11111   DMSEEE TXT22222   DMSAAA TXT33333
                                    DMSCCC TXT33333
                                    DMSDDD TXT33333
```

You must always keep the $PTFPART file and source updates. Committing PTF UM33333, though valid, would be pointless, as you cannot delete any of the parts shipped with it.

VMSES/E helps you to find which parts can be erased. You can query this information from the $PTFPART files in the Software Inventory. Also, Appendix D, “VMFSIM Exploitation Code Examples” on page 225 describes a sample procedure, PTFCOMIT EXEC, that helps you identify which parts can be safely erased (see “Erasable Parts for Committed PTFs” on page 228).

**Note:** For readers familiar with VMSES prior to VM/ESA Release 1.1, the APPLY section has been removed. This is simply because the apply step no longer uses individual part handlers. VMSES/E completely separates the receive, apply, and build steps, so that receive only receives files, apply only applies service, and build only builds usable forms.

### Build Product Definition Section

This section contains information that allows VMSES/E to build the product’s objects. This is done by providing pointers to files, called *build lists* that actually describe how to build each object, and linking those pointers to *build part handlers*. Each build part handler is specialized in a particular type of object; for example, there is a part handler to build nuclei, another to build modules, another to build MACLIBs, and so on. The build process, and build part handlers, are covered in more detail in “Build” on page 107.

The term “build,”as used by VMSES/E, does not necessarily mean that a new executable part is really built. For example, in the case of help panels, it means that the serviced panel files needed to actually generate the executable form are copied from the DELTA to the BUILD disk, and renamed.

Another example is the REXX component. Since most of REXX is part of the CMS nucleus, there is no REXX nucleus to build. So, VMFBLD just copies the new TEXT files to the appropriate BUILD disk (which is the shared REXX and CMS BUILD disk), and makes a note that the CMS nucleus must be rebuilt. Figure 18 shows an excerpt from the build log, which illustrates this point.

```
ST:VMFBLD1851I Processing build list 1 of 1, IXXBLNUC EXEC, with part
ST:            handler HCPBDUTL
BD:VMFBDU2180I Build Requirements:
BD:            Bldlist      Object       Status
BD:VMFBDU2180I IXXBLNUC     -            REBUILD CMS & GCS NUCLEI
ST:VMFBDU2178I Processing of build list IXXBLNUC complete:
ST:            1 object(s) were selected to be built
ST:            1 object(s) were successfully built
ST:            0 object(s) failed
ST:VMFBLD2760I VMFBLD processing completed successfully
```

*Figure 18. Build Log for the REXX Component (Excerpt)*

Figure 19 on page 53 shows an excerpt from the build product/definition section of the $PPF for CMS.

```
:BLD.
* BUILDLIST EXEC     TARGET   DESCRIPTION
* --------- -------- -------  ------------
DMSBL490    VMFBDCOM BUILD6   * Build files for system disk
DMSBLSRC    VMFBDCOM BUILD7   * Build source files
ASDREUSP   -VMFBDMLB BUILD7   * Build ASDREUSP MACLIB
DMSOM       VMFBDMLB BUILD6   * Build DMSOM MACLIB
DMSGPI      VMFBDMLB BUILD6   * Build DMSGPI MACLIB
DMSRP      -VMFBDMLB BUILD7   * Build DMSRP MACLIB
OSMACRO     VMFBDMLB BUILD6   * Build OSMACRO MACLIB
      ⋮
DMSBLPRP    VMFBDLLB BUILD6   * Build PROPLIB LOADLIB
CMSSAA      VMFBDTLB BUILD6   * Build CMSSAA TXTLIB
DMSBLBAS    VMFBDTLB BUILD6   * Build DMSBASE TXTLIB
DMSBLAEN    VMFBDTLB BUILD6   * Build DMSAENV TXTLIB
      ⋮
DMSBLVML    VMFBDCLB BUILD6   * Build VMLIB CSLLIB and CSLSEG
DMSBLVMT    VMFBDCLB BUILD6 (LIBTYPE SEG * Build VMMTLIB CSLSEG
DMSBLASM    VMFBDCOM BUILD7   * Build text for F-assembler
DMSBLDOS    VMFBDCOM BUILD7   * Build text for DOS segments
      ⋮
CMSMLOAD    VMFBDMOD BUILD6   * Build modules that reside on
                              * system disk
IOCPLOAD    VMFBDMOD BUILD6   * Build IOCP modules
CMSLOAD     VMFBDNUC BUILD6 TXT TXC ( BLDREQ DMSBLVMT
                              * Build CMS nucleus
      ⋮
DMSSBSFS    VMFBDSBR BUILD7   * CMSFILES Segment
DMSSBVML    VMFBDSBR BUILD7   * VMLIB Lseg in CMSVMLIB Seg
DMSSBVMT    VMFBDSBR BUILD7   * VMMTLIB Lseg in CMSVMLIB Seg
DMSSBPIP    VMFBDSBR BUILD7   * CMSPIPES Segment
DMSSBINS    VMFBDSBR BUILD7   * CMSINST  Segment
      ⋮
:EBLD.
```

*Figure 19. $PPF for CMS - BLD Section (Excerpt)*

Though VMSES/E has been continuously improved to automate object building, there are still a few objects it cannot automatically build. VMSES/E will warn you whenever you have to rebuild those objects. These are built by manually entering the appropriate commands. For more information, refer to “Build” on page 107.

### Filetype Abbreviations Extensions Section

This section provides information similar to, and that may override or supplement, the information found in the Filetype Abbreviation table (see Table 1 on page 41).

## Override Area

The final area of the PPF is the override area. It has the same structure as the component area. A source PPF supplied by IBM may have override areas, when it is anticipated that many users want or need certain changes to the defaults for the product. For example installing on SFS directories instead of on minidisks. By “compiling” the base component or the override you may elect to use the product’s defaults or its IBM tailoring.

You can also change the supplied defaults and tailoring by supplying your own overrides.

## Build Lists

In “Build Product Definition Section” on page 52, we saw some examples of build list names. Though build lists are not, formally, part of the Software Inventory, they are so important we describe them here. The concept of build lists is not new in VM. The build list is an extension of the load list, which has been available since VM/370.

VM/ESA Release 1.0 introduced a new format for the build list, called Format 2 build list, as opposed to the old load list format, which is now called Format 1. VM/ESA Release 2 introduced yet a new format, called Format 3.

The new formats provide more flexibility than the old, which was basically a list of text decks to be loaded, interspersed with some loader control statements. Figure 20 shows part of the CMSLOAD EXEC, the Load List (Format 1 build list) for CMS.

```
&TRACE OFF
&1 &2 &3 HCPLDR LOADER
&1 &2 &3 DMSNUC
&1 &2 &3 DMSZNR
&1 &2 &3 SLC L00E000
&1 &2 &3 DMSZAT
&1 &2 &3 SLC L020000
&1 &2 &3 DMSZUS
&1 &2 &3 DMSINS
&1 &2 &3 DMSIND
******** DMSINN must be after DMSIND
&1 &2 &3 DMSINN
&1 &2 &3 DMSINV
&1 &2 &3 DMSINR
******** DMSZIN MUST BE THE LAST MODULE BEFORE DMSINI
&1 &2 &3 DMSZIN
      ⋮
```

*Figure 20. Load List for CMS - CMSLOAD EXEC (Excerpt)*

Format 2 build lists allow many different objects to be defined in a single place. Those objects do not have to be related to each other except in two aspects:

- They must be processed by the same build part handler.
- They must reside on the same target disk.

The Format 2 build lists use a set of tags that allow you to specify the name of the objects being built, and to supply appropriate options to be passed to the CMS commands, such as LOAD, called during the build process. VM/ESA Release 2 also introduced the capability to specify the object’s file type, as well as build requisites and libraries to be made global during the build. Figure 21 on page 55 is an example of a Format 2 build list.

```
:FORMAT. 2
*
      ⋮
:OBJNAME. VMFHASHM.MODULE NOMAP ALL
:OPTIONS. NOMAP CLEAR RLDSAVE NCHIST
:PARTID. VMFHASHM TXT
:EOBJNAME.
      ⋮
*
:OBJNAME. VMFRDTBL.MODULE NOMAP ALL SYSTEM
:OPTIONS. NOMAP CLEAR RLDSAVE NCHIST NOUNDEF RESET VMFRDTBL
               AMODE 31  RMODE ANY
:PARTID. VMFRDTBL TXT
:OPTIONS. SAME UNDEF
:PARTID. VMFRDTBT TXT
:EOBJNAME.
      ⋮
```

*Figure 21. VMSES/E Format 2 Build List (Excerpt from VMFMLOAD EXEC)*

We see that two modules (VMFHASHM and VMFRDTBL) will be built. Let us look at the CMS commands that will be issued to build the VMFRDTBL.MODULE object:

```
load VMFRDTBL (nomap clear rldsave nchist noundef reset vmfrdtbl amode 31 rmode any
include VMFRDTBT (same undef
genmod VMFRDTBL (nomap all system
```

The command options specified on a :OPTIONS statement remain in effect until another :OPTIONS statement is found or the end of the object is reached.

Format 3 build lists were introduced in VM/ESA Release 2 and support the building of libraries. Each library is defined in a separate build list, and each member is described as an object in the build list. They also allow specifying libraries to be made GLOBAL, and support object requisite specification.

With Format 3 build lists you can use the :LIBNAME tag to explicitly name the built library (the default is the file name of the build list file). Figure 22 shows an excerpt of the Format 3 build list for the CMSSAA TXTLIB library.

```
:FORMAT. 3
*
:LIBNAME. CMSSAA
*
     ⋮
:OBJNAME. DMSSAA
:PARTID. DMSSAA TXT
:EOBJNAME.
:OBJNAME. DMSSRR
:PARTID. DMSSRR TXT
:EOBJNAME.
     ⋮
```

*Figure 22. VMSES/E Format 3 Build List (Excerpt from CMSSAA EXEC)*

## PRODPART File

The PRODPART file is the basic descriptor of any product or component that is loaded into the VM system. It contains the name of the PPF, and also lists resource requirements for the product.

During product installation, VMFINS extracts information from the PRODPART file and updates the system-level Software Inventory. The segment management tool, VMFSGMAP, also uses the PRODPART file to dynamically obtain default segment definition information.

The PRODPART file resides on the SIDISK. However, the user can override this file by placing a PRODPART file on the A-disk.

## Overview

The PRODPART file contains the following sections:

- Header Section
- Loadable Units Section
- Parts Section
- Saved Segments Definition Section
- Product Parameters Section

Figure 23 shows the general structure of the PRODPART file.

```
:RECID       - PRODUCT RECEIVE IDENTIFIER (KEY)

   :PRODDESC - PRODUCT DESCRIPTION (FIELD)
      ⋮
   :LU         - LOADABLE UNIT(S) DEFINITIONS (BLOCK)

   :PPF          - PPF AND COMPONENT IDENTIFIER (SUBKEY)
      ⋮
   :ELU          - END LOADABLE UNIT(S) DEFINITIONS (EBLOCK)

   :PARTS      - PARTS DEFINITIONS (BLOCK)
   :PARTID       - PART IDENTIFIER (SUBKEY)
      ⋮
   :EPARTS       - END PARTS DEFINITIONS (EBLOCK)

   :SEGDEF     - SEGMENT DEFAULT DEFINITIONS (SUBKEY)
      ⋮
   :ESEGDEF      - END SEGMENT DEFINITIONS (EBLOCK)

   :PARMS      - PRODUCT INSTALL PARAMETERS (SUBKEY)
      ⋮
   :EPARMS       - END PRODUCT INSTALL PARAMETERS (FIELD)
```

*Figure 23. PRODPART File General Structure*

## Header Section

The header section of the PRODPART file defines the product in terms of program name, program number, and service level, and provides other information to VMSES/E. Figure 24 shows the header section of the PRODPART file for CMS.

```
:RECID.6VMVMA22
:PONUM.5684112
:PRODDESC.CMS component for VM/ESA 2.2
:SERVLEV.200-0000
:PROCTYPE.VMSES
```

*Figure 24. PRODPART File for CMS - Header Section*

Besides the component ID, the product number, and the service level, the component is described in words, and the :PROCTYPE tag indicates that the component is fully utilizing VMSES/E.

## Loadable Units Section

From the Software Inventory perspective, the loadable unit section is the most interesting. A component may have more than one loadable unit. This allows subsets of the component to be defined. For example, the product might be loaded with or without the source code, or support for different national languages might come in separate loadable units.

The loadable unit section contains the PPF name for the component, the component's description, and requisite information. The loadable unit section for CMS is shown in Figure 25.

```
:LU.
:PPF.ESA CMS
:PRODID.6VMVMA22%CMS
:DESC.CMS component for VM/ESA 2.2
:PREREQ.6VMVMK22
:DREQ.6VMVMF22
:SUP.6VMVMA11 6VMVMA20 6VMVMA21
:ELU.
```

*Figure 25. PRODPART File for CMS - Loadable Unit Section*

The :PRODID information goes into all the system-level tables, the :DESC information ends up in the SYSDESCT table, and the requisite information is transferred to the SYSREQT table.

Much of the same information is found in the PPF, described in "Product Parameter File" on page 43.

## Parts Section

The parts section, if it exists, contains the list of tailorable parts for the product. Any part listed here may be changed by the user, to meet the installation's specific requirements. Figure 26 on page 58 shows an entry on the parts section of the PRODPART file for CMS in Release 1.1. In Release 2 and later, and Release 1.5, the parts section is empty.

```
:PARTS.
      ⋮
:PARTID.DMSNGP SAMPLE
:PROCOPTS.RGROUP SYSSAMP TPART  09315793419640
      ⋮
:EPARTS.
```

*Figure 26. PRODPART File for CMS - Part Section (VM/ESA Release 1.1)*

The parameter on the TPART statement is a hash value (in this context, a check-sum) of the file contents at the time of installation. By performing the same hash procedure on the current copy, at a later time, VMSES/E is able to determine whether the part was altered by the installation. If the part was altered, it will not be replaced by refresh service. Also, the changed part is preserved by the migration process (VMFINS MIGRATE command).

When parts are received, using the VMFINS INSTALL command, they are identified and their names are entered in the VMSES PARTCAT file on the target disk, together with their full product ID and status information.

Whenever the part is altered, the PARTCAT is updated to reflect the current status of the part.

## Saved Segment Definitions Section

The Saved Segments definition section was introduced in VM/ESA Release 2, and contains default values for generating saved segments.

The VMFSGMAP EXEC, the mapping and planning tool for segment support, is able to extract this information and allows the user to change it. The changed (customized) information is used to update the SEGDATA file. Thus, the original default information is preserved in the PRODPART file. See Chapter 4, “Saved Segments” on page 65.

Figure 27 on page 59 shows an excerpt from the :SEGDEF section of PRODPART file for CMS.

```
:SEGDEF.6VMVMA22 CMS
*************************************************************
:OBJNAME. HELPINST
:DEFPARMS. C00-CFF SR
:TYPE. PSEG
:OBJDESC. CMSINST,CMSQRYH, CMSQRYL, AND HELP LSEGS
:OBJINFO. CMSQRYH CAN BE MOVED ABOVE 16M
:GT_16MB. NO
:BLDPARMS. PPF(ESA CMS DMSSBINS) PPF(ESA CMS DMSSBQYH)
          PPF(ESA CMS DMSSBQYL) PPF(ESA CMS DMSSBHLP)
:OBJNAME. CMSPIPES
:DEFPARMS. 700-77F SR
:TYPE. PSEG
:OBJDESC. CMS PIPES SEGMENT
:GT_16MB. YES
:BLDPARMS. PPF(ESA CMS DMSSBPIP)
       ⋮
:ESEGDEF.
```

*Figure 27. PRODPART File for CMS - SEGDEF Section (Excerpt)*

The HELPINST segment is a physical segment (PSEG) containing the CMSINST, CMSQRYH, CMSQRYL, and HELP logical segments (LSEGs). The CMSPIPES segment is another PSEG, containing only one logical segment.

The :BLDPARMS tag describes how the segments are built. The CMSINST logical segment is defined in the DMSSBINS build list, and the HELP logical segment is defined in the DMSSBHLP build list. The CMSPIPES segment is defined in the DMSSBPIP build list. These build lists are listed in the :BLD section of the CMS component area of the ESA PPF.

## Product Parameters Section

The PARMS section contains information on resources, such as minidisks and user IDs, that have to be present so that the product can be installed (or added, or migrated) and run properly. The minidisk entries include size information. Figure 28 on page 60 shows both a user entry and minidisk entry.

```
:PARMS.6VMVMA22 CMS
*****************************************
:RMT.
      ⋮
:USERDEF.
USER &SRVSU NOLOG 32M 32M BG
 ACCOUNT 1 VMSERVS
 MACH XC
 OPTION MAXCONN 2000 NOMDCFS APPLMON ACCT QUICKDSP SVMSTAT
 SHARE REL 1500
 XCONFIG ADDRSPACE MAXNUMBER 100 TOTSIZE 8192G SHARE
 XCONFIG ACCESSLIST ALSIZE 1022
 IUCV ALLOW
 IUCV *IDENT RESANY GLOBAL
  IPL 190
 CONSOLE 009 3215 T MAINT
 SPOOL 00C 2540 READER *
 SPOOL 00D 2540 PUNCH A
 SPOOL 00E 1403 A
 LINK &BLD3Z 190 RR
 LINK &BLD2Z 193 RR
 LINK &BLD5Z 19D RR
 LINK &PRODZ 19E RR
:EUSERDEF.
      ⋮
:TARGET.
  :TARGID.&DELTZ
  :SIZE.12000
  :BLKSIZE.4K
  :FORMAT.CMS
  :MODE.MR
:ETARGET.
      ⋮
:ERMT.

:EPARMS.
```

*Figure 28. PRODPART File for CMS - Parms Section (Excerpt)*

As can be seen in the disk entry, the space requirement is 12000 4-KB blocks on a CMS formatted minidisk, which will be linked MR, if requested on the VMFINS command.

The entry also shows the use of variables, introduced in VM/ESA Release 1.1. The variables, such as &SRVSU and &DELTZ, are defined in the PPF in the variable declaration (DCL) section (see Figure 14 on page 48). Figure 29 shows an excerpt from the :DCL section of the usable form PPF file for CMS. For more information on the PPF file, please refer to “Product Parameter File” on page 43.

```
:DCL.
       ⋮
&DELTZ LINK MAINT 3D2 3D2 MR  * CMS service
&APPLX LINK MAINT 3A6 3A6 MR  * Aux & software inventory files
&APPLY LINK MAINT 3A4 3A4 MR  * Aux & software inventory files
       ⋮
&SRVSU USER VMSERVS           * VMSERVS userid
&SRVUU USER VMSERVU           * VMSERVU userid
       ⋮
:EDCL.
```

*Figure 29. PPF for CMS - DCL Section (Excerpt)*

For a more detailed discussion of variables, please refer to “Variable Declarations Section” on page 48.

The installation tool, VMFINS, when the RESOURCE option is specified, uses the PARMS section in the PRODPART file, and the corresponding information in the PPF, to alter the source CP directory. Thus, it is able to define the resources (user IDs and minidisks or SFS directories) necessary to install the product.

**Note:** Use the RESOURCE option with care. See “RESOURCE Option” on page 87.

This information can also be used to allow installation of another copy of a product. Again, using the NOPLAN option of VMFINS, you are prompted to change the virtual addresses for the required minidisks and the virtual addresses at which they are linked before use. Your input is saved in a PPF override, which is compiled immediately. Its usable form is subsequently used in the installation process.

During the installation, VMSES/E updates the Software Inventory with information on the new copy of the product, using the new PPF file name as the main key. This is the only way VMSES/E can distinguish between the products and their copies.

## PTFPART File

There is one PTFPART file for each PTF received on your system. The file name is ptfnum, the PTF number. The file type is always $PTFPART.

The PTFPART file comes on the service media, and describes the PTF, its parts, and its dependencies. It is placed on the first (or only) disk of the DELTA string when the service is received. During the service process, VMFREC uses information in this file to update the service-level Software Inventory.

## Overview

The PTFPART file has three major sections:

- A header section, describing the received service
- A requisite section, describing the requisite relationships
- A parts section, defining the parts being serviced

Figure 30 on page 62 shows the general structure of the PTFPART file.

```
:PTF         - PTF TOOL CONTROL STATEMENTS (KEY)
   :PRODID     - PRODUCT IDENTIFIER (FIELD)
   :APARDESC   - APAR DESCRIPTION BLOCK (BLOCK)
       ⋮
   :EAPARDESC    - END APAR DESCRIPTION BLOCK (EBLOCK)
   :UMEMO      - USER MEMO (FIELD)
       ⋮
   :EUMEMO       - END USER MEMO (FIELD)
* Requisite information
   :PREREQ     - PRE-REQUISITE PTF(S) (FIELD)
       ⋮
* Parts information
   :PARTS      - PARTS DEFINITIONS (BLOCK)
       ⋮
   :EPARTS       - END PARTS DEFINITIONS (EBLOCK)
```

*Figure 30. PTFPART File General Structure*

## Header Section

The header section describes the PTF, both in words and by listing which APARs constitute the PTF. Figure 31 shows an example of the header section in a PTFPART file.

```
** PTF UM12345 Release = 1.2.2, ProdId = 6VMVMA22
:PTF.UM12345
:PRODID.6VMVMA22
:APARDESC.This is a sample PTF for VM/ESA 2.2
:APARNUM.VM54321
:ABSTRACT.This APAR corrects ...
:EAPARDESC.
:UMEMO.
Here comes a verbal description of the PTF
:EUMEMO.
```

*Figure 31. PTFPART File Header Section Example*

The tags should be self-explanatory. Note that, as a PTF may provide fixes for more than one APAR, you may see many :APARDESC :EAPARDESC pairs for one PTF. Much of this information is inserted in the service-level Description table. If a Memo to Users is present, the Receive Log file will contain a message reminding you to read the memo.

## Requisite Section

The requisite section of the PTFPART file describes the requisite relationships for the PTF. Figure 32 on page 63 shows an example of a PTFPART file requisite section.

```
:PREREQ.UM23456 UM45678
:COREQ.UM11111.6VMVMB22 UM22222
:IFREQ.UM33333.6VMVMA22
:SUP.UM99999
:HARDREQ.VM80001 VM80000.6VMVMB22.UM70002
```

*Figure 32. PTFPART File Requisite Section Example*

As you see, there are five types of dependencies listed. They were defined in "Definitions and Terms" on page 10.

Please note also that the requisite information may be presented in several ways. Whenever you see a component name, for example 6VMVMB22, it means that the requisite is in another component. This information is used to update the service-level Requisite table.

A requisite may be either a PTF or an APAR. In the latter case, if it is an out-of-component requisite, the PTF number may be listed as the third specifier (as seen on the :HARDREQ tag).

The difference between hard requisites (HARDREQs) and soft requisites (PREREQs) is important. If you have to apply a specific PTF in an emergency, you may not want to apply all prerequisite PTFs at that time. You may then use the VMFSIM command to list the hard-requisites and apply only these. However, we strongly advise against doing it, because this is a manual procedure, requires a significant effort and a very good knowledge of VMSES/E, and may compromise the integrity of the Software Inventory.

## Parts Section

The parts section lists all the parts shipped with the PTF and the form in which they are delivered. It also indicates which part they replace or update. Figure 33 shows an example of a parts section of a PTFPART file.

```
:PARTS.
 :PARTDEF.DMSABC ASSEMBLE
 :PROCOPTS.AUX SDI
 :REPPART.DMSABC TXT12345
 :PROCOPTS.NOAUX NOSDI
 :REPPART.DMSABC MOD12345
 :UPDATES.V23456DS
 :NRSOURCE.DMSABC ASM12345
 :APARS.VM23456 VM34567.M
       ⋮
:EPARTS.
```

*Figure 33. PTFPART File Parts Section Example*

This example is full of information. We are to service the CMS module DMSABC, which is identified here by its source component in the :PARTDEF tag. We will replace two parts, identified in the :REPPART tags. The replacement operation uses the options defined in the :PROCOPTS tags. The information in a :PROCOPTS tag remains valid until replaced by another :PROCOPTS tag.

The following processes take place:

- DMSABC TXT12345 - The Self-Documenting Information (SDI) in the TEXT deck prologue is checked; the VVT is updated with this PTF's information and used to generate an AUX file that includes the new APARs. VMSES/E no longer depends on the SDI, but will still check that the SDI is consistent with the information in the Software Inventory.
- DMSABC MOD12345 - No additional processing is done.

The next tags identify the supplied update file, corresponding to APAR VM23456, and an updated source file DMSABC ASM12345. The APAR VM34567 has been merged into the source file, as indicated by the .M suffix on the :APARS tag.

# Chapter 4. Saved Segments

Support for managing and building saved segments was introduced in VM/ESA Release 2. This support does not include building Named Saved Systems.

This chapter describes, from a conceptual point of view, the VMSES/E functions and structure that allow mapping, tailoring, servicing, and building saved segments, including those of products not otherwise supported by VMSES/E.

**Note:** VM/ESA Release 1.5 370 Feature supports only notification of service to saved segments. Mapping, tailoring, and building are **not** supported.

A more detailed discussion, based upon actual examples, can be found in Chapter 8, “Saved Segment Experiences” on page 157.

The following publications contain further information regarding saved segments:

- *VM/ESA: Planning and Administration*
- *VM/ESA: VMSES/E Introduction and Reference*
- *VM/ESA: Service Guide*
- *VM/ESA: Installation Guide*
- *VM/ESA: Service Guide for 370*

## Overview

Saved segments are the first objects supported by VMSES/E with cross-product, system-wide characteristics. At the same time, saved segments are also product objects, requiring handling just as other objects do. VMSES/E must provide support for these seemingly separate, even conflicting, situations. Indeed, saved segments must be seen from two perspectives:

- At the system level, where they interact with other segments belonging to the same or different products. Also, it is possible that a segment is built from objects belonging to different products.
- At the product level, because the objects a segment is built from are themselves built from parts serviced through normal product service.

Saved segments support requirements include:

- Capability to study segment interactions and play “what if” games; that is, plan segment characteristics without changing the live system.
- Detect any product changes to objects included in segments and automatically flag the need to rebuild the affected segments.

The above functions must also be provided for:

- Segment spaces and their members, including members belonging to several spaces.
- CMS logical segments and associated physical segments.

The next sections further detail these points, as well as the limited support provided for Named Saved Systems (NSSs).

The objects we will be discussing are:

**Segment** In the System/390* architecture, it refers to a 1 MB area of main storage that starts on a MB boundary. Also referred to as an *architected segment*.

**Saved Segment** This is a general term for DCSSs, Member Saved Segments, and Logical Saved Segments. Each saved segment has a unique name.

**DCSS** A Discontiguous Saved Segment is a set of one or more areas of main storage. The areas in the set do not have to be adjacent. When in use, each area is adjusted, if necessary, so that it begins on a megabyte boundary, is a multiple of one megabyte, and is contiguous. In other words, each area is adjusted to occupy one or more segments.

A DCSS is defined by issuing a CP DEFSEG command, created by loading data or programs in the defined areas and issuing a CP SAVESEG command, and used by asking CP to attach the segment to the machine. The attach request is either a program-issued DIAGNOSE X′64′ or the CMS SEGMENT LOAD command.

Several virtual machines may attach the same DCSS and concurrently share the storage areas. This is why DCSSs are often referred to as *shared segments*.

**NSS** Named Saved Systems are similar to DCSSs, in the fact that they can be shared by several virtual machines, but they are also different:

- DCSSs contain application code and data, and can be dynamically attached to, and released by, a virtual machine without resetting the machine′s environment.
- NSSs are defined by issuing a CP DEFSYS command, created by loading an operating system in the defined areas and issuing a CP SAVESYS command. The operating system is “frozen” in time until a CP IPL restarts it.

**Member Saved Segment**
When in use, DCSSs areas are always one or more megabytes long. If the object in a DCSS does not fully occupy this area, storage addressing capability is wasted. A Segment Space is a CP concept, which allows subdividing a DCSS into areas that begin and end on a page boundary. Each of these areas is a Member Saved Segment, has its own name, and is defined using the CP DEFSEG command, with the SPACE parameter. It is then created and used like a DCSS. When you attach a member of a space to a virtual machine the whole space becomes available. A member may belong to several spaces.

**Segment Spaces** Segment Spaces are implicitly created and maintained by CP whenever a Member Saved Segment is defined. A segment space may contain up to 64 members.

**PSEG** Physical Saved Segments are a CMS concept. PSEGs can be implemented either as a DCSS or as a member saved segment. PSEGs are defined using the CP DEFSEG command and created by the CMS SEGGEN command. PSEGs may contain one or more LSEGs, and so have an internal structure, defined by CMS, which also includes a directory of the component LSEGs.

**LSEG** Logical Saved Segments are another CMS concept. LSEGs are to a PSEG what member saved segments are to a segment space. There are also some differences:

- CP is unaware of PSEGs and LSEGs. CMS manages those.
- Member Saved Segments can be defined and created one at a time. A PSEG and the associated LSEGs must be defined and created together.
- Segment Spaces first appeared with VM/XA, and are supported only by VM/XA and VM/ESA running in ESA mode, thus on processors with the 370-XA or System/390 architectures. PSEGs and LSEGs are managed by CMS and have been available since VM/SP Release 6. Thus they can be exploited in the System/370*, 370-XA, ESA/370*, ESA/390*, or ESA/XC architectures.
- Using LSEGs is easier than using DCSSs. The CMS SEGMENT LOAD command automatically processes the LSEG definitions, making the objects in the LSEG available to the CMS environment.

## System-Level View

For VMSES/E enabled products, default segment information is provided with the product in the :SEGDEF :ESEGDEF block of the PRODPART file. For non-VMSES/E products, equivalent information must be entered by the user.

It is quite common to alter the segments′ characteristics, notably the load addresses, but the default information must not be altered because it must be available at all times. The solution is to provide a file to hold the customized segment information, either extracted from the PRODPART file or entered by the user. This file resides in the system-level Software Inventory and is called the Segment Data file, or SEGDATA file. It describes a particular segment arrangement, or storage layout. It is possible to create and maintain several of these files (for example, to do segment planning without interfering with the production system, or to maintain several systems from a single site).

Because saved segments are system objects, they are built at a system level, which requires a system-level segment build list (VMSES/E requires that all objects it can build be described in build lists). However, this build list does not contain detailed build instructions, which exist in product-level build lists. The system segment build list, then, only points to the product build lists. Each SEGDATA file has a corresponding system segment build list.

Building objects, in VMSES/E, is exclusively a function of the VMFBLD command and this command requires information from a PPF. Therefore, a system-level PPF for system objects is introduced. This PPF has only one component, and its main function is to list the build lists for system objects. This PPF also supplies system *appids* and a system *bldid*.

There is also a system-level Service Build Status table. It contains the status of any serviced saved segments.

## Saved Segment Planning

Segment planning must be done at the system level. Indeed, segments from different products interact, and all segments, as well as named saved systems, must be considered as a whole in any meaningful planning. Planning is mostly prompted by adding new segments or changing existing ones; for example, changing the storage location of a segment.

VMFSGMAP is the VMSES/E tool that provides these system-level management support functions. Figure 34 on page 69 shows a logical view of the data flow in VMFSGMAP.

The VMFSGMAP command:

- Obtains saved segment information from:
  - The PRODPART files
  - The SEGDATA file
  - User input
  - Existing saved segments available on the system
- Builds a storage map of the segments defined on the system and on the SEGDATA file, and shows this map through a full-screen interface (XEDIT based).
- Allows you to change the information (you can discard any changes).
- Saves the information back in the Software Inventory files.

VMFSGMAP is most useful as a planning tool because any newly defined or changed segments are immediately reflected on the segment map. Also, as VMFSGMAP does not actually build any segments, your production system is safe and you can study any segment arrangements (or layouts) and solve all conflicts, such as overlaps, before committing the changes.

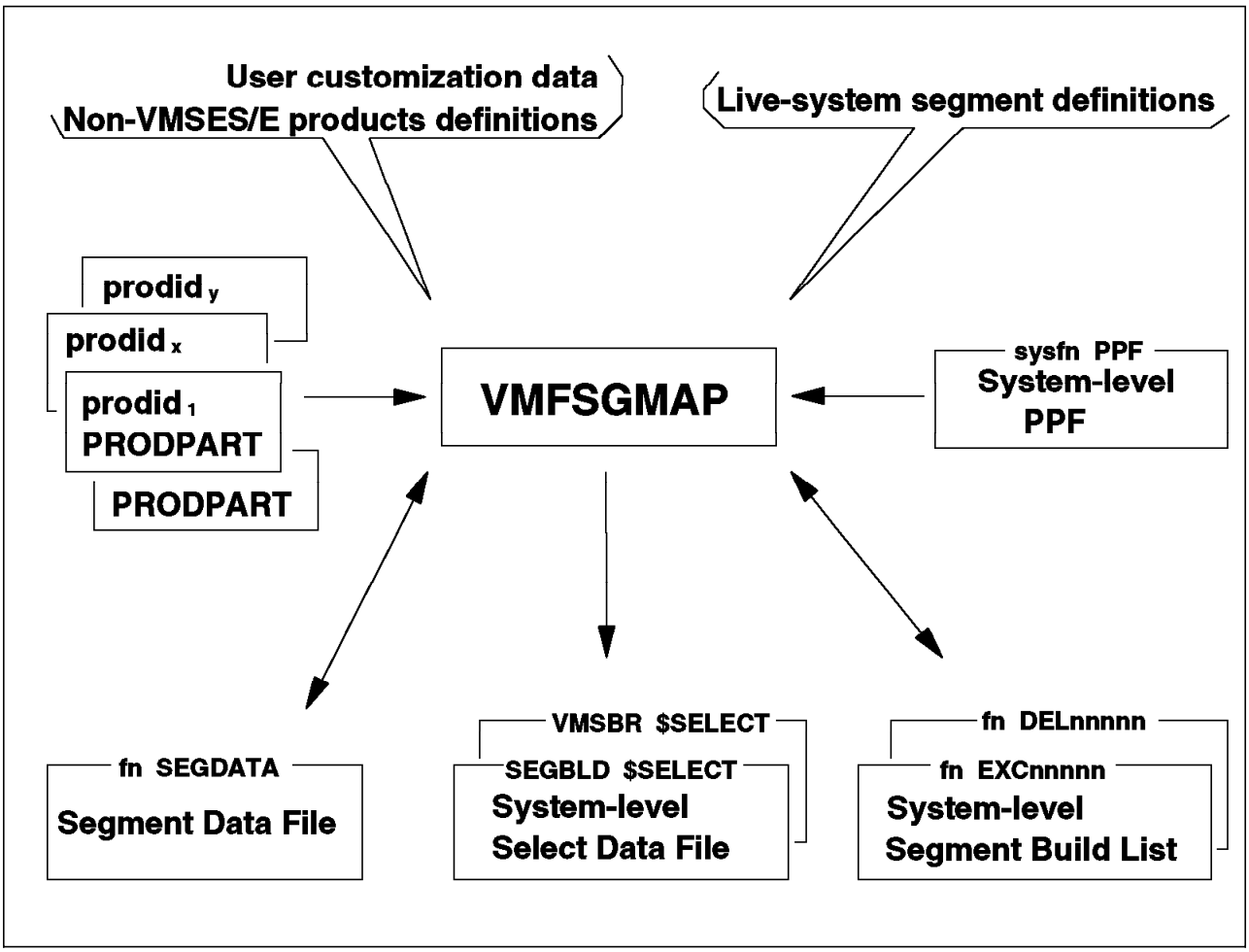



*Figure 34. Logical Data Flow in VMFSGMAP*

## VMFSGMAP Command

VMFSGMAP's main function is to allow segment planning based on a full-screen map of the defined segments. This map shows:

- Discontiguous saved segments (DCSS)
- Segment spaces
- Member saved segments
- NSSs
- Segment location (address ranges and page descriptors)
- Overlapping members
- Gaps in segment spaces

This map can be saved in a file. Notice that information on NSSs is extracted from the system and presented on the map, but that is the only support provided by VMFSGMAP. You cannot add NSS definitions to the SEGDATA file or customize them.

By comparing the system definitions with those in the SEGDATA file, VMFSGMAP flags any segment that is defined only to the system, defined only in the SEGDATA file, or defined on both but with definition mismatches. VMFSGMAP also detects any invalid ranges in segment spaces.

Any changes can be saved and reflected in the Software Inventory files. VMFSGMAP also automatically changes the system segment build list, and adds entries to the system Select Data file, to signal the need to rebuild the changed segments. The system Select Data file has the same role, for system objects, as a product-level Select Data file. In Release 2.2, the :APPID tag of the system-level segments PPF lists two names. The first, or primary, by default SEGBLD, is used by VMFSGMAP as the file name of the Select Data file. The second, or alternate must be VMSBR, as explained in "VMFBDSBR Part Handler" on page 76. Prior to Release 2.2, there was only VMSBR in the :APPID tag.

Note that VMFSGMAP does not build or delete the segments, nor does it call the build function.

## Segment Map Screen

Figure 35 on page 71 shows a part of the segment map built by VMFSGMAP. Notice that VMFSGMAP divides the address space into 4-MB blocks. Each map column corresponds to 64 KB (16 pages) and has a special code called a "page descriptor":

**R** Pages defined as exclusive (ER) or shared read (SR).
**W** Pages defined as exclusive (EW) or shared write (SW).
**N** Pages defined as shared, no data saved (SN).
**X** Overlapping range (for member saved segments).
**C** CP writable pages.
**.** Pages empty and free.
**-** Pages empty but not free, such as, in a DCSS, pages between the end of the range and the next MB boundary. For example, see the CMSVMLIB segment.
**=** Pages of a segment space that are occupied by its members. For example, see the DOSBAM space.
**>** Continued in another 4 MB range. For example, see the CMS NSS.

The map displays the following segment types:

**CPD** CP system service saved segment
**DCS** Discontiguous saved segment (DCSS)
**SYS** Named saved system
**SPA** Segment space
**MEM** Member saved segment

Because VMFSGMAP extracts data from the reply to the CP QUERY NSS command, it can also map any named saved systems or segments not defined in the SEGDATA file. The first column of the map contains a status indicator resulting from comparing the system data with the definitions in the SEGDATA file.

The possible statuses are:

**blank** The definition in the system matches that in the SEGDATA file.
**D** The system and SEGDATA file definitions are different. When this happens the SEGDATA definition is shown.
**E** Either the system definition or the SEGDATA file definition is in error.
**M** The segment is only mapped: It is defined to the system but is not in the SEGDATA file.

**P** The segment is only planned: It is defined in the SEGDATA file but not to the system.

When a segment is defined in both places, the SEGDATA definitions are shown. Also, if both a skeleton and an active segment that is defined only to the system exist, the definition of the active segment is shown. In Figure 35, the only flagged lines with a non-blank status are the NSSs (flagged as “M”) because they are defined only to the system.

In Release 2.2, the first column of the map may show, instead of the status, the spool file class of the segments that exist on the system. Pressing PF12 will toggle the screen to spool class mode. Possible spool class codes, which mimic standard CP NSS class codes, are:

**A** Active: the saved segment or saved system has been defined and code has been loaded into the saved segment or system.

**R** Restricted: code has been loaded into this saved segment or system. However, this code is only available to users with the NAMESAVE option in their directory.

**S** Skeleton: a skeleton has been defined, but no code has been loaded into the saved segment or system.

The segment displayed will either be a class A or R, or if neither exist, a class S file (if there is one) will be shown. To work with one of the segments, point at it with the cursor and press the appropriate PF key.

```
                          VMFSGMAP - Segment Map                  More: +
                                                               Lines 1 to 26 of 36

Meg            000-MB          001-MB          002-MB          003-MB
St  Name   Typ 0123456789ABCDEF0123456789ABCDEF0123456789ABCDEF0123456789ABCDEF
M CMS      SYS W-W-------------1...............2...............3...............
M GCS      SYS W---------------1...............2...............3...............

               004-MB          005-MB          006-MB          007-MB
  Name     Typ 0123456789ABCDEF0123456789ABCDEF0123456789ABCDEF0123456789ABCDEF
  CMSPIPES DCS 4...............5...............6...............RRRR------------
M GCS      SYS RRRRRRNNNNNNNNNNNNNNNNNNNNNNNNNN6...............7...............

               008-MB          009-MB          00A-MB          00B-MB
  Name     Typ 0123456789ABCDEF0123456789ABCDEF0123456789ABCDEF0123456789ABCDEF
  DOSBAM   SPA 8...............9...............A...............====------------
  CMSBAM   MEM 8...............9...............A...............RRRR............
  CMSDOS   MEM 8...............9...............A...............R...............
  CMSFILES DCS 8...............-----RRRRRRRRRRRRRRRRRRRRRRRRRB...............
  CMSVMLIB DCS RRRRRRRR--------9...............A...............B...............
  DOSINST  DCS 8...............R---------------A...............B...............

               00C-MB          00D-MB          00E-MB          00F-MB
  Name     Typ 0123456789ABCDEF0123456789ABCDEF0123456789ABCDEF0123456789ABCDEF
  HELPINST DCS RRRRRRRRRRRRRRRRD...............E...............F...............
M CMS      SYS C...............D...............RRRRRRRRRRRRRRRRRRRRRRRRRRRRRRR>

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

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

*Figure 35. Segment Map*

## VIEW Subcommand

The VIEW subcommand has been introduced in this release of VMSES/E. This command can be specified with one of the following options:

**ALL** Displays information for all the segments defined on the system.

**SEGDATA** Displays information about the segments defined in the SEGDATA file. Segments that were defined and saved without using VMSES/E are not displayed.

**ERROR** Displays information about all segments the have a class of E.

A sample of the VMFSGMAP screen displaying information about all the segments is shown in Figure 36.

```
                          VMFSGMAP - Segment Map                  More: +
                                                                Lines 1 to 18 of 93

Meg             000-MB          001-MB          002-MB          003-MB
St Name    Typ 0123456789ABCDEF0123456789ABCDEF0123456789ABCDEF0123456789ABCDEF
M CMS      SYS W-W-------------1.............2.............3.............
M GCS      SYS W---------------1.............2.............3.............
M PSEG1    DCS 0.............1.............2.............WWWWWWWWWWWWWWWW
M MONDCSS  DCS 0.............1.............2.............CCCCCCCCCCCCCCC>

Meg             004-MB          005-MB          006-MB          007-MB
St Name    Typ 0123456789ABCDEF0123456789ABCDEF0123456789ABCDEF0123456789ABCDEF
M SNASEG   SPA 4.............================================7.............
M VTAM     MEM 4.............RRRRRRRRRRRRRRRRRRRRRRRR.......7.............
M NPM      MEM 4.............5.............6........RRRR...7.............
M NETVSG00 MEM 4.............5.............6............RRR7.............
  CMSPIPES DCS 4.............5.............6.............RRRRRRRR--------
M GCS      SYS 4.............5.............6.............RRRRRRNNNNNNNNN>
M MONDCSS  DCS >CCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCC

Meg             008-MB          009-MB          00A-MB          00B-MB
 F1=Help     F2=Chk Obj   F3=Exit      F4=Chg Obj   F5=File       F6=Save
 F7=Bkwd     F8=Fwd       F9=Retrieve  F10=Add Obj  F11=Del Obj   F12=Class
====>
```

*Figure 36. VMFSGMAP Segment Map Using VIEW ALL*

## Check Object Enhancement

Check Object (PF2 CHK OBJ on the VMFSGMAP screen) has been enhanced to provide a display of the CP QUERY NSS command. The highlighted areas in Figure 37 show the changes to this interface.

```
                              VMFSGMAP - Segment Map                  More: +
                                                                  Lines 1 to 18 of 93
 ------------------------------------------------------------------------------
                           Query NSS Map For CMSPIPES
                                                                  Lines 1 to 3 of 3
 FILE FILENAME FILETYPE MINSIZE  BEGPAG ENDPAG TYPE CL #USERS PARMREGS VMGROUP
 0129 CMSPIPES DCSS       N/A    00700  0077F   SR  P  00025   N/A       N/A
 0167 CMSPIPES DCSS       N/A    00700  0077F   SR  A  00002   N/A       N/A

 F1=Help F3=Exit F6=File Query F7=Bkwd F8=Fwd F9=Retrieve F12=Cancel
 ------------------------------------------------------------------------------

Meg             000-MB          001-MB          002-MB          003-MB
St Name    Typ 0123456789ABCDEF0123456789ABCDEF0123456789ABCDEF0123456789
M CMS      SYS W-W-------------1.............2.............3.........
M GCS      SYS W---------------1.............2.............3.........
M PSEG1    DCS 0..............1.............2.............WWWWWWWWWW
M MONDCSS  DCS 0..............1.............2.............CCCCCCCCCC
Meg             004-MB          005-MB          006-MB          007-MB
VMFSMD2032I System and SEGDATA definitions are the same for segment CMSPIPES
 F1=Help     F2=Chk Obj   F3=Exit      F4=Chg Obj   F5=File       F6=Save
 F7=Bkwd     F8=Fwd       F9=Retrieve  F10=Add Obj  F11=Del Obj   F12=Class
====>
```

*Figure 37. VMFSGMAP Check Object Screen*

## Adding a Segment Definition

If, while on the Segment Map panel, you invoke the Add Obj function (PF10) you are presented Add Segment Definition panel. If the cursor was placed on a map record, that segment (or space) definition is used to pre-fill the Add Segment Definition panel. If the cursor was not pointing to a map record, the panel will be mostly blank, as shown in Figure 38.

```
                              Add Segment Definition
                                                                  Lines 1 to 12 of 12
 OBJNAME....:  ????????
 DEFPARMS...:
 SPACE......:
 TYPE.......:  SEG
 OBJDESC....:
 OBJINFO....:
 GT_16MB....:  NO
 DISKS......:
 SEGREQ.....:
 PRODID.....:
 BLDPARMS...:  UNKNOWN

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

*Figure 38. VMFSGMAP Add Segment Definition Panel*

By entering the name of an object in the OBJNAME field you can:

- Obtain current segment information from the SEGDATA file and immediately customize it (PF2). This speeds up segment definition when several segments are similar.

- Obtain default segment information from the PRODPART file and immediately customize it (fill the PRODID field then press PF10).

You can also enter information relative to segments of non-VMSES/E enabled products.

Of the data fields on the panel, some are common to all products, and some are more useful, or specific to, non-VMSES/E enabled products.

The fields in this panel are:

**OBJNAME** This is the segment name. It must be unique within the system.

**DEFPARMS** Defines the location of the segment's pages by describing each page range and its type. On VM/ESA Release 2.2, if a segment is deleted, the word DELETED is inserted in front of the page range values. You can recover the segment by removing the word. On previous releases the word DELETED replaced the page range values, and there was no way to recover the tailored values.

**SPACE** Used only for segment space members. It is a list of the segment spaces of which the segment is a member. For DCSSs the field must be left blank.

**OBJDESC** Contains any text describing the segment.

**OBJINFO** Descriptive text providing special information concerning the segment.

**GT_16MB** Specifies whether the segment can run above 16 MB. It may still reside below 16 MB, even if it can run above, but must reside below 16 MB if NO is specified.

**DISKS** Lists disk addresses VMFBLD should automatically access when building the segment. If the disks are not linked, the segment will not be built.

**SEGREQ** Identifies prerequisite segments. Any prerequisite segment which does not have a status of BUILT, in the Segment Build Status table, will be built prior to building the initial segment.

**PRODID** This field identifies the PRODPART file that contains the default definition for the saved segment.

**BLDPARMS** This field contains the build directions for the segment. Products in VMSES/E format use the keyword PPF followed (within parentheses) by the PPF, component, and build list names. Non-VMSES/E products use the keyword PROD followed (within parentheses) by the name of an EXEC and, optionally, one or more parameters. This EXEC is either provided by the program product, or written by the installation.

When this field has the value UNKNOWN, VMFBLD issues only the DEFSEG command. The actual build will be done later by a user called procedure.

If you are defining a member saved segment, the Chk Mem (PF6) function is very useful. It checks for member overlaps in all the spaces the member you are defining will belong to. You can then adjust the storage ranges by using the Adj Mem (PF11) function.

When you return to the Map panel (PF5) any changes you made are preserved. However, you can still discard them if you leave the Map panel by any other means than File (PF5).

## Viewing and Changing a Saved Segment Definition

On the Segment Map panel, you can move the cursor to the map record for a particular saved segment and press the PF4 key to display the definition record for the saved segment. For example, on the segment panel shown in Figure 35 on page 71, if you move the cursor to the map record for CMSBAM and press PF4, the Change Segment Definition panel shown in Figure 39 is displayed.

```
                            Change Segment Definition
                                                          Lines 1 to 12 of 12

 OBJNAME....:  CMSBAM
 DEFPARMS...:  BOD-B37 SR
 SPACE......:  DOSBAM
 TYPE.......:  SEG
 OBJDESC....:  CMSBAM MEMBER OF THE DOSBAM SEGMENT SPACE
 OBJINFO....:
 GT_16MB....:  NO
 DISKS......:
 SEGREQ.....:  DOSINST
 PRODID.....:  6VMVMA22 CMS
 BLDPARMS...:  PPF(ESA CMS DMSSBBAM)

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

*Figure 39. VMFSGMAP Change Segment Definition Panel*

You can change any of the fields in this panel except the OBJNAME field. In other words, you cannot rename a segment. To rename a segment, create a new one with the add function, then delete the old segment.

For example, you can alter the storage defined for the segment by entering the new range in the DEFPARMS field. You should use the “Chk Mem” (PF6) function to check for any overlaps. Next, press PF5 to return to the Segment Map panel, which is refreshed to show the changes. If the updated map is correct in relation to the other saved segments, you can press PF5 to record the changes and exit VMFSGMAP.

**Notes:**

- The changed information is stored in both the SEGDATA file and in the segment build list.
- To activate the new definitions, the segment must be redefined, reloaded, and re-saved. This is done with the VMFBLD command, or through a product-defined procedure.
- See “Building Saved Segments” on page 163 for a detailed description of the segment build process.

## Product-Level View

Rebuilding segments is prompted by two situations: because the segment's definitions were changed by the user, or because normal product service has changed one or more of the objects used to build the segment.

The first case was explained in "System-Level View" on page 67; we will now concentrate on the second one. This discussion applies only to VMSES/E-enabled products. Service for non-VMSES/E products includes specific instructions on how to rebuild the respective segments.

The requirements are to:

- Detect that service has been applied to objects a segment is built from.
- Detect changes to segment-requisite objects.
- Signal the need to rebuild the affected segments.

Each segment, because it is a system-level object, is built not from product parts but from product *objects*. For example, the CMSPIPES segment is built from the DMSPIPE.MODULE object, which, in turn, is built from several parts (TEXT files). A change in one of the parts forces the building of the DMSPIPE module and of the CMSPIPES segment, in this order.

Regardless of being implemented as LSEGs, DCSSs, or member saved segments, CMS saved segments are defined at the product level in Format 2 build lists. LSEGs are later (at the system level) created as members of a PSEG; the PSEG itself is defined only at the system level, in the SEGDATA file, and can be created either as a DCSS or as a member saved segment. CMS saved segments to be implemented as DCSSs or member saved segments are defined as such in the SEGDATA file, and are created (at the system level) as DCSSs or member saved segments, respectively. No PSEGs are involved.

Each segment is defined by an :OBJNAME :EOBJNAME block in the build list. The :BLDREQ tag is used to identify the objects the segment is built from. The :OPTIONS tag defines the function used to create the segment. The values in this tag vary widely with the type of segment.

VMFBDSBR is a build part handler called during the build step of product service. Its function is to pass to the system-level information on which segments have to be rebuilt. Building segments requires control files with system-level information.

**Note:** On a VM/ESA Release 1.5 370 Feature system, there is no system-level information for segments. VMFBDSBR only logs messages in the build message log.

## VMFBDSBR Part Handler

The product-level service process that triggers the building of **any** object is as follows:

1. VMFAPPLY creates, as part of the apply process, a list of parts affected by service - the Select Data File (*appid* $SELECT). This file resides on the product's Alternate Apply disk.

2. The STATUS function of VMFBLD uses the $SELECT file to scan the product's build lists for objects containing those parts, and changes the status of those objects in the product's Build Status table (*bldid* SRVBLDS) to SERVICED.
3. The SERVICED or ALL options of VMFBLD scan the SRVBLDS file and call the appropriate part handlers to build the objects.

However, saved segments must not be built at this point. Indeed, you may need to coordinate service for several products before you can build a segment; you may have a physical segment grouping logical segments from several products, which requires that they are all built at the same time. In addition, you may have to change segments' definitions, and this must be done from the system level, by using VMFSGMAP. This ensures the changes are correct; for example, that the changed segment does not overlap existing segments or that it is large enough for new service.

So, the STATUS option of VMFBLD only identifies the *need* to build the segments. During the SERVICED (or ALL) processing, a build part handler, VMFBDSBR, issues messages (and also logs the messages) informing which segments have to be rebuilt.

On VM/ESA Release 2 and later systems (but not on VM/ESA Release 1.5 370 Feature systems) VMFBDSBR is also used to communicate to the system level any segment build requirements. This is done by adding to a predefined system-level Select Data file, VMSBR $SELECT, the build list names of the segments that have to be rebuilt. In Release 2.2, because the VMFSGMAP uses a different Select Data file, user changes to the segment's definition and service changes to a segment's contents can be kept distinct.

But VMFBDSBR has to know on which disk the VMSBR $SELECT file resides. When VMFBDSBR is executing, the product's disks are accessed. Later, when the segments are built using VMFBDSEG, there is a good chance the disks will not be accessed anymore. So there is a need for a common disk, the SIDISK. In order to allow flexibility in the actual location of the VMSBR $SELECT file, another file pointing to it was placed in the SIDISK. This latter file is named "VMSESE PROFILE."

## Segment Building

To actually build segments requires system-level information. Here we will give only an outline of the process. A detailed description, requiring information given in Part 2, "VMSES/E Usage Experiences," can be found in "Building Saved Segments" on page 163.

Segments, like any other objects, are built using the VMFBLD command. As noted above, the first step in any VMFBLD invocation is known as STATUS. Before VM/ESA Release 2.2, this step was **always** executed, even when specifying the SERVICED or ALL options. Starting with VM/ESA Release 2.2, the STATUS step will be skipped for the SERVICED and ALL options when the time stamps in the $SELECT files and in the :LASTAPP tag of the SRVBLDS table match, indicating STATUS has been performed and no new service has been applied. This step can also be executed by itself, by invoking VMFBLD with the STATUS option.

The STATUS step essentially identifies any build requirements. The System Segment PPF has control information for the overall process, such as the name of the system-level Service Build Status table. The SEGBLD $SELECT and

VMSBR $SELECT files are used to search the system segment build list and identify the changed segments. In this way, changes made by both the user and product service are brought together. Any segments that have to be built have their status in the system-level Service Build Status table changed to SERVICED. Dependent segments are also flagged as SERVICED. Any segments that the user marked for deletion are also identified at this point in time, and flagged as DELETE in the system SRVBLDS table.

Next, during the SERVICED or ALL steps of VMFBLD, the changed segments are actually built (or deleted).

## VMFBDSEG Part Handler

VMFBDSEG, another part handler introduced in VM/ESA Release 2, is responsible for the actual building.

### LINK Option

VMFBLD can be invoked to build a specific segment, all serviced segments, or all segments. When you are building segments for several products, VMFBDSEG has to access the disks for a product, build the segment, then access the disks for the next product, and so on. As products may use the same virtual addresses for some of the disks you could run into minidisk address conflicts, and VMFBDSEG could not continue to build all segments.

You should build the segments from a special purpose user ID (MAINT is not a good choice because of minidisk address conflicts). This user ID has no disks of its own (apart from the 191) but can link to the other product's disks. With VM/ESA Release 2.2, specifying the LINK option allows VMFBDSEG to link and access the required disks, build the segment, and detach the disks, freeing the address for the next segment.

### Execution Flow

VMFBDSEG is called by VMFBLD for the specified build list, and it will:

- Release all saved segments and drop all nucleus extensions before the build.
- Determine which segments have to be built or deleted.
- Copy the SYSTEM SEGID file from the S disk to the target build disk, if needed.

  **Note:** When a logical segment is added to or deleted from a physical segment the SYSTEM SEGID file has to manually copied back to the S disk at the end of the build. This, in turn, implies re-saving the CMS saved system.
- Validate the definitions in the SEGDATA file and in the product build list, for each segment that has to be built. The products disks are accessed through VMFSETUP if needed. If errors are found, no segment is built.
- Delete any segments that have to be deleted.
- Define the segments that have to be built. If errors are found, no segment is built. Defining the segments involves:
  - Purging existing segment skeletons.
  - Defining the skeleton file, using the information in the SEGDATA file.
  - Copying unchanged members of Active or Restricted segment spaces.

- Build requested segments:
  - Link the product disks, when the LINK option is used.
  - Access the products disks:
    - The disks in the DISKS field are accessed (must be already linked).
    - If the BLDPARMS field specifies a PPF, the VMFSETUP command is invoked.
  - Activate any needed GLOBAL libraries defined in the product level segment build list.
  - For CMS logical saved segments:
    - Create the PSEG and LSEG files.
    - Create any logical segment profile and epifiles, if needed, and erase them if SEGGEN is successful.
    - Call the SEGGEN command.
  - For segments that do not contain CMS logical segments, call the appropriate routine.
- Return the results to VMFBLD. VMFBLD then updates the system-level Service Build Status table.
- If the message VMFBDS2003W is issued, serviced parts may need to be moved to a selected disk. For example, the SYSTEM SEGID may require copying to the S-disk.
- If message VMFBDS2006E is issued, VMFBDSEG has determined that a segment it is trying to build might contain a part that has been serviced but not rebuilt (for example, a text deck in the DMSPIPE MODULE has been serviced but the DMSPIPE MODULE has not been rebuilt). In this case, VMFBDSEG will not build the segment. To correct this condition, invoke VMFBLD for the PPF, component, and build list names specified in the message, with the All and SETUP options. The build requisites in the build list will cause VMFBLD to build any serviced objects contained in the segment. Then invoke VMFBLD again to build the segment, specifying the SETUP option to re-access the required disks.

# Part 2. VMSES/E Usage Experiences

This part discusses experiences gained when servicing a VM/ESA system, and installing and servicing selected program products. Several examples of Software Inventory exploitation are presented and discussed.

# Chapter 5. Installation Experiences

In “Installing Products” on page 18, we presented an overview of the installation process. This chapter expands on that information by taking a closer look at the VMFINS EXEC, the VMSES/E installation tool, and giving actual installation experiences.

Any VMSES/E commands (of the format VMFxxxxx) mentioned in this chapter are fully described in *VM/ESA: VMSES/E Introduction and Reference* or *VM/ESA: VMSES/E Introduction and Reference for 370*.

## VMFINS Command

The VMFINS command was introduced by VMSES/E and is the primary command used to install licensed program products. Installing VM/ESA follows a different procedure (see “Installing VM/ESA” on page 90). VMFINS is a high-level, though flexible, interface and provides in a single command the ability to:

- Install products or additional copies of products.
- Migrate products while keeping tailored files and data.
- Build products, as the last step in the installation or migration.
- Delete products from your system.

The general syntax of the VMFINS command is shown in Figure 40.

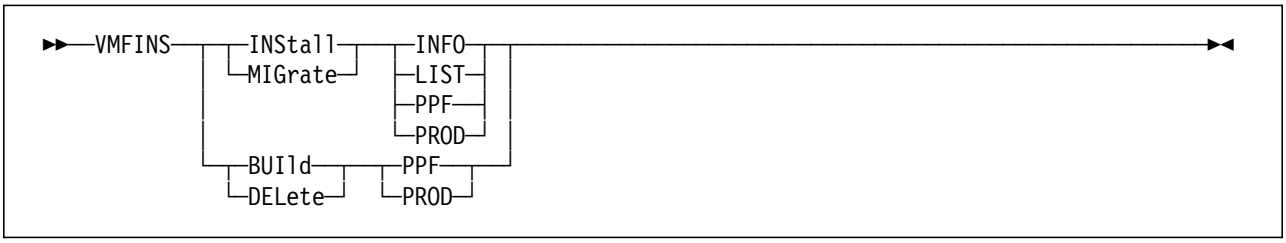



*Figure 40. VMFINS Command General Syntax*

You can use VMFINS to install products in the following formats:

- VMSES/E
- INSTFPP

The products may be distributed on the following tape formats:

- VMSES/E
- Parameter Driven Installation (PDI)

VMFINS is also capable of handling the old VM/SP System Offering format (INSTFPP).

Refer to Appendix B, “Product Packaging and Distribution Media Formats” on page 205 for a description of these tape formats.

The remainder of the chapter further details the functions of the VMFINS command.

## VMFINS DEFAULTS

As this file is enhanced in VM/ESA Release 2.2, you should become familiar with its purpose. The file resides on the SESDISK, and supplies defaults for the VMFINS command, for example:

- Whether product installation should add a copy or replace an existing product
- The file id of the source directory
- File pool used during installation or migration
- System name
- Minidisk address or SFS directory, and access mode for the SIDISK

You may edit this file, and change these values to suit your installation's defaults.

## VMFINS INSTALL and VMFINS MIGRATE Commands

The complete syntax of the VMFINS INSTALL command is shown in Figure 41 on page 85. The syntax of the VMFINS MIGRATE command is identical (with the MIGRATE keyword, instead of INSTALL). We will discuss only a few of the numerous operands and options. For a full description of all the operands and options, please see *VM/ESA: VMSES/E Introduction and Reference*.

Installation and migration are very similar. Both can:

- Place another copy of an existing product on the system (ADD option)
- Replace an existing product (REPLACE option)

The main differences between migration and installation are:

- The product to be migrated must have been previously installed by VMSES/E.
- Migration preserves all your tailored files (the files must be listed in the Parts section of the PRODPART file). Note that if you alter a file not listed in the PRODPART file, your changes will be lost.
- Since migration always uses a SFS directory for its save area, an active CMS Shared File System is required. Until VM/ESA Release 2.2, only the VMSYS: file pool could be used. Though this is the default, any file pool can now be used.

**Note:** Migration cannot be used to replace a VM/ESA system. A supported installation method must be used.

VMFINS supports all three main steps in any product installation or migration:

1. Planning

   **vmfins install info ... ( memo**

   and

   **vmfins install ppf ... ( plan nomemo**

   or

   **vmfins install prod ... ( plan nomemo**

2. Loading the materials

   **vmfins install ppf ... ( noplan nomemo**

   or

   **vmfins install prod ... ( noplan nomemo**

3. Generating any required usable forms (product build)

   **vmfins build ...**

**Note:** Though the LIST operand can be used for the planning and load steps, we recommend its use be restricted to the planning step, if used at all.

We will now take a closer look at these steps.

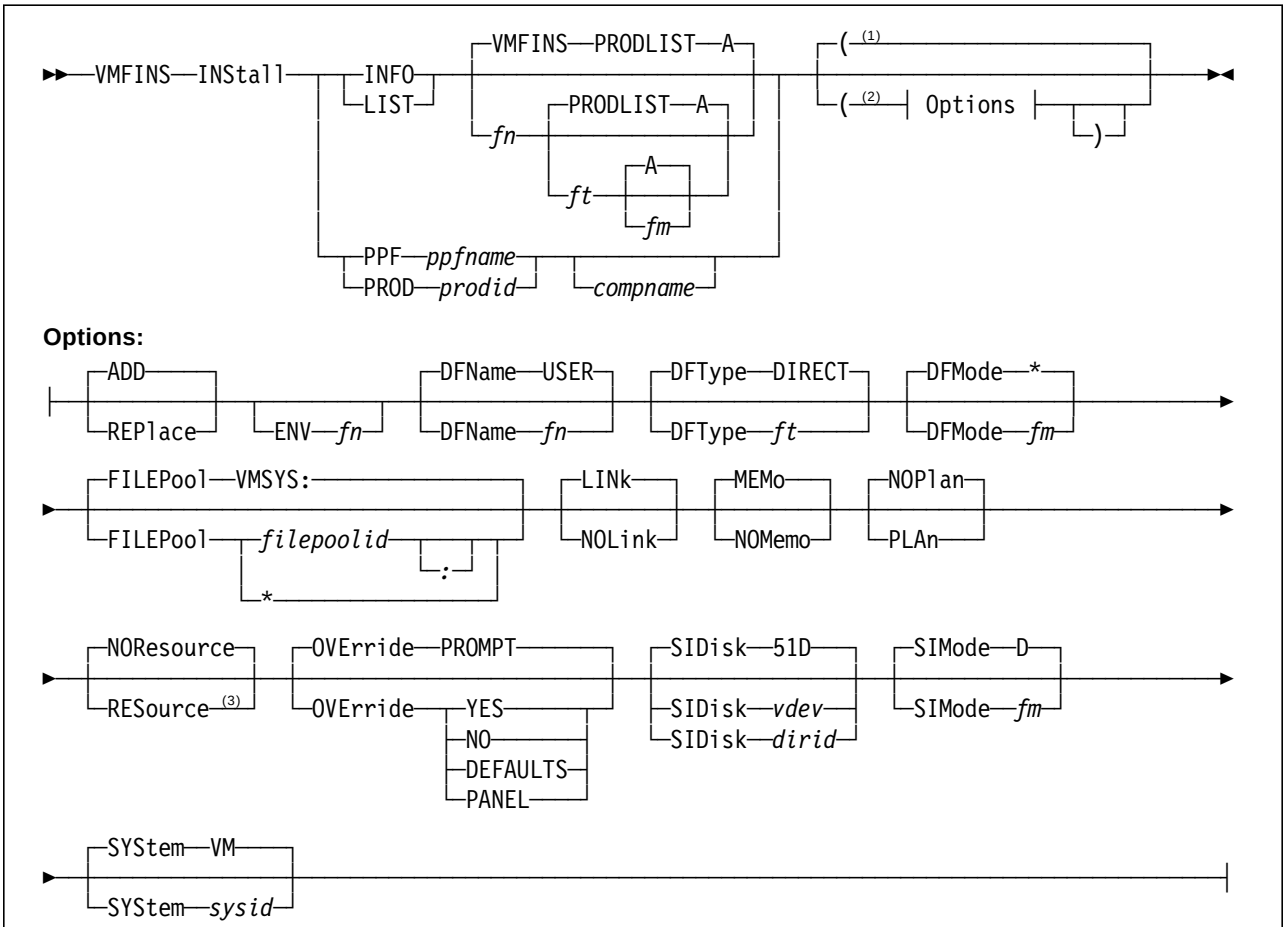



*Figure 41. VMFINS INSTALL Command Syntax*

## Planning Step

Planning for a product installation, or migration, mainly consists of:

- Verifying that all product requisites are satisfied
- Making the required resources available

**Note:** Planning for saved segments should be done later, during the build step.

Useful information is contained in each product's installation guide and in the printed Program Directory that accompanies the distributed materials. In past VM/ESA releases, information used to be distributed in the Memo to Users, included as a file on the product's tape. Products in VMSES/E format no longer include installation information in the Memo to Users. The Memo to Users files will be empty, and are retained on the product tape for compatibility. All relevant information is included in the other documents.

## SIDISK, SIMODE, and SYSTEM Options

These options provide greater flexibility for managing installations with multiple systems:

- The SIDISK options lets you select any minidisk address (or SFS directory name) for the Software Inventory disk (the default is 51D).
- The SIMODE options allows the SIDISK to be accessed using any valid disk mode (default is "D"). Take care when the SIMODE is not D, because VMFSETUP will automatically release the disk, unless you add it to the :RETAIN tag of the PPF, or use the RETAIN option of the VMFSETUP command.
- The SYSTEM option lets you name the system you want to work with (default is "VM").

## INFO and LIST Operands

With the INFO operand, you can create lists of the products that were shipped on the product tapes and may be added to the system, or that may replace existing products. A file listing those products is created on your A-disk. It is named VMFINS PRODLIST and can be subsequently used as input to the LIST parameter. However, we recommend that you limit the use of the LIST operand to the planning step. Do not use the LIST operand for the product loading step because you have better control of the installation process when you install products one by one.

Figure 42 shows an example of the VMFINS PRODLIST file.

```
VMFINS   PRODLIST A1  V 94  Trunc=94 Size=5 Col=1 Alt=1
|...+....1....+....2....+....3....+....4....+....5..

*** Top of File ***
PPF  5684096 RSCS      Remote Spooling Communications Subsystem
PPF  5684096 RSCSSMALL RSCS without opt source and text
PROD 5748XX9 NONE      DCF Version 1
PPF  5668962 ASSEMH    PRODID 5668962%ASSEMH ASSEMBLER-H
PPF  5798DWD RTM       PRODID 5798DWH%RTM Real Time Monitor VM/ESA
```

*Figure 42. VMFINS PRODLIST File*

## MEMO Option

This option prompts you for which Memo to Users you want to print on the system printer. Products in VMSES/E format no longer include installation information in the Memo to Users. The Memo to Users files are empty, and are retained on the product tape for compatibility purposes only. For previous releases, please note that you do not have to print the memos, as they are loaded to the SIDISK whenever VMFINS INSTALL is invoked, so you can consult them anytime. However, we do recommend you read the memos.

## PPF and PROD Operands

Products in non-VMSES/E format must use the PROD operand. VMSES/E-formatted products may use either operand, although PPF is the preferred operand. The VMFINS PRODLIST file contains the PPF and component names.

During the planning step these operands are used in conjunction with the PLAN option.

## PLAN Option

The PLAN option helps you check product requisites and resource requirements before actually performing any actions. It does **not** generate, allocate, or commit any system resources.

The PLAN option creates an output file with a file name equal to the product id, and a file type of PLANINFO. Within this file, you can find the information to:

- Check for any requisite products
- Identify the minidisks that will be required
- Determine the amount of space required on each minidisk
- Define any user IDs that might be required

The information provided by the PLAN option helps you to install multiple copies, or new releases, of a product, without disturbing the existing products. As an example, the PLANINFO file created when planning the installation of the CMS Utilities feature is shown in Figure 43 on page 93.

This file is especially useful if you have a CP directory management product, such as DIRMAINT, as discussed in “RESOURCE Option.”

## RESOURCE Option

The RESOURCE option of the VMFINS command uses the resource definition information from the Product Parameters section of the PRODPART file to add user IDs and minidisks to your system (or remove them if you are doing a delete). For a further description of the PRODPART file, please see “PRODPART File” on page 56.

The RESOURCE option **generates, allocates, and commits system resources**. It requires access to the source CP Directory in single file format, and a special data file, called VMFRMT EXTENTS, as explained below.

The VMFRMT EXTENTS file is supplied with VM/ESA, as a sample file of the CP component, and allows the RESOURCE option to locate free space in which new minidisks can be allocated. The default entries provided are samples only, so

you need to tailor this file (use XEDIT) to define the DASD devices you want the RESOURCE option to examine for free space.

The DASD devices whose labels you specify in this file are checked to ensure that they are available to the system. If a device you specified is not available, it will be ignored. You also have to specify the type of DASD device, a starting cylinder value, and the number of cylinders that are to be checked for free space.

If you wish to use the RESOURCE option, you must ensure that both the CP directory and the VMFRMT EXTENTS files are on minidisks that will still be accessed after all the minidisks of the component have been accessed by the VMFSETUP EXEC; for example, by using the RETAIN option to reserve file modes. VMFSETUP is the VMSES/E tool that accesses the disks of a component, listed on the :MDA section of the PPF, and you can use the RETAIN option to reserve file modes.

The RESOURCE option requires the source CP Directory file to be in single-file format, in order to be able to modify it. Therefore, if you have the DIRMAINT program product, or a similar product, on your system, we recommend you do **not** use the RESOURCE option. However, if you decide to use it, remember that, beginning with Release 4, DIRMAINT maintains the source directory as a group of files. Therefore, you will have to perform some manual steps in order to have DIRMAINT re-create the directory into a single file. You should also stop DIRMAINT operations during this period. Use the commands:

```
dirm disable
dirm user backup
dirm send user backup b
dirm shutdown
```

You can then receive the directory file, and give it the VMFINS default name: USER DIRECT. After VMFINS alters it, you will then have to re-initialize DIRMAINT, following the directions for DIRMAINT initial installation.

Even if you do have DIRMAINT, you can still use VMFINS planning capability: you can extract the relevant information from the PLANINFO file, created by using the PLAN option, and manually create or update the necessary user IDs.

Also, if you have RACF or a similar product, you will have to define the resources to the system and permit access to any resources as required. The RESOURCE option does not provide any interface to these functions, which must be manually performed.

## Product Loading Step

VMFINS inspects the product on the tape and, if it is a VMSES/E-formatted product, uses the VMFREC command to actually load the product materials. For non-VMSES/E products, INSTFPP is called.

## PPF and PROD Operands

VMSES/E-enabled products should be installed using the PPF operand. All other products must use the PROD operand, instead of PPF. The VMFINS PRODLIST file contains the PPF and component names.

To install or migrate a VMSES/E-enabled product you issue one of the following commands:

```
vmfins install ppf ... ( add noplan nomemo
vmfins install ppf ... ( replace noplan nomemo
vmfins migrate ppf ... ( add noplan nomemo
vmfins migrate ppf ... ( replace noplan nomemo
```

These commands allow you to:

- Install a product and keep it separate from the current production version of the product.
- Install a product and replace the current version of the product.
- Install another copy of a product without loosing any user tailored files.
- Install a new version of a product, which replaces a current version, without loosing any user tailored files.

**Notes:**

- You must select the NOPLAN option to load the product's materials.
- The NOMEMO option prevents printing the Memo to Users.

## PPF Overrides

The VMFINS command allows you to create a PPF override if you do not want to use the supplied default minidisk addresses for a product. You will have to create one if you are installing an additional copy of the product. In VM/ESA Release 1.1, override creation occurred only when using the PROD operand. Starting with VM/ESA Release 2, you can create a PPF override when using either the PPF or PROD operands (if the product is in VMSES/E format).

During product loading, VMFINS prompts you to override the default PPF values. In VM/ESA Release 2.2, prompts can be pre-answered using the OVERRIDE option. If you reply yes, you are placed on a panel that allows you to change the default names for user IDs required by the product, and the default minidisk addresses or shared file system directories. You then save this overriding information under a name of your choice. VMFINS will cause your override to be compiled into the usable form PPF that is used in the subsequent processing.

**Note:** The Make Override utility now allows you to define SFS directories in any file pool. On VM/ESA Release 2.1 and previous releases you were forced to use the VMSYS: file pool.

---

## Product Building Step

Since the installation process loads executable parts to the base minidisks, the build process is usually a short one. Objects require building because of local tailoring or service.

The VMFINS BUILD command calls VMFBLD to actually build the objects, and then updates the system-level Build Status table.

If the product has saved segments VMFINS BUILD will not build them. You should use the VMFSGMAP command to add the definitions to the Segment Data file, and resolve any conflicts. Next, manually invoke the VMFBLD command to build the segments. See Chapter 8, “Saved Segment Experiences” on page 157.

## VMFINS DELETE Command

The VMFINS DELETE command can be used to remove a product from your system, for example, to remove an older version of a program product after a newer version has been successfully placed into production. It can also be used to remove a product that has just been installed but has caused some sort of problem.

Please note that if you use the PLAN option with the DELETE operand, you can find out:

- Whether any other products depend on this product
- What user IDs can be deleted from the system
- How much DASD space will be released
- Which files will be erased and the minidisks they are on

The same considerations discussed in “RESOURCE Option” on page 87 apply here.

## Installing VM/ESA

It is not the purpose of this book to explain the installation process for VM/ESA. However, it is important to mention here a few key concepts, because the installation of VM/ESA has undergone a quick evolution.

## Installing VM/ESA Release 1.1

After an initial “boot-strapping” step to load the basics of the system, the installation process makes use of the same tools that are used to service and build the system. VM/ESA Release 1.1 is distributed in Refreshed Product Tapes.

The *VM/ESA: Installation Guide* describes two basic ways to perform a VM/ESA Release 1.1 installation:

- By using a starter system, either:
  - Stand-alone
  - Under VM/XA or VM/ESA
- By installing the new system under a currently running 370-mode VM system.

Additional information is available in *VM/ESA Release 1.1 Overview and Usage Experiences*.

## Installing VM/ESA Release 2

Like VM/ESA Release 1.1, VM/ESA Release 2 can be installed by using a starter system and Refreshed Product Tapes. However, the preferred method is a new process, called *flex-DDR* (see “Installing VM/ESA” on page 18).

Additional information is available in *VM/ESA Release 2 Usage and Experience*.

## Installing VM/ESA Release 1.5 370 Feature, VM/ESA Release 2.1, and Later Releases

Installing VM/ESA Release 1.5 370 Feature, VM/ESA Release 2.1 or later releases, exclusively uses the flex-DDR process; Refreshed Product tapes are **not** available.

Additional information is available in *VM/ESA Release 2.1 Usage and Experience* and *VM/ESA Release 2.2 Overview and Usage Experiences*.

## Product Identification for VM/ESA Components

For VM/ESA, the relation of the component name to the prodid (used as the file name in the base PPF) is shown in Table 4.

*Table 4. VM/ESA Component PPF File Names and Aliases*

| VM/ESA Component | Base $PPF File Name | Description |
|---|---|---|
| CMS | 6VMVMAnn | Conversational Monitor System (•) |
| CP | 6VMVMBnn | Control Program (• •) |
| CP370 | 6VMVMCnn | Control Program for 370 Feature (• •) |
| AVS | 6VMVMDnn | APPC/VM VM/VTAM Support (•) |
| CUF | 6VMVMEnn | CMS Utility Feature (optional •) |
| REXX | 6VMVMFnn | Procedures Language VM/REXX (•) |
| TSAF | 6VMVMHnn | Transparent Services Access Facility (•) |
| DV | 6VMVMInn | Dump Viewing Facility (• •) |
| IPCS | 6VMVMJnn | Interactive Problem Control System for 370 Feature (• •) |
| VMSES | 6VMVMKnn | VMSES/E (•) |
| GCS | 6VMVMLnn | Group Control System (• •) |
| GCS370 | 6VMVMMnn | Group Control System for 370 Feature (• •) |

**Notes:**

- The value of “nn” indicates the VM/ESA Release:
  - **11** Release 1.1
  - **20** Release 2
  - **21** Release 2.1
  - **22** Release 2.2
  - **15** Release 1.5 370 Feature
- For ESA mode systems only
- For 370 mode systems only

## Installing the CMS Utilities Feature

This section describes some observations we made while installing the CMS Utilities Feature. Note that a product tape as available for VM/ESA Release 1.1 was used.

## Planning

Since the CMS Utilities Feature is an optional feature, the CP directory shipped with the starter system does not include either the minidisks for the MAINT user ID, or the user IDs that the CMS Utilities Feature requires. You will find the minidisk and user ID requirements in *VM/ESA: CMS Utilities Feature*. The installation steps in the manual do not use the PLAN or RESOURCE options of VMFINS, but if you wish to do so, all the required information is contained in the product part file for the CMS Utilities Feature, 6VMVME11 PRODPART.

As described in "PLAN Option" on page 87, the prodid PLANINFO file (see Figure 43 on page 93) contains detailed information that you can use to plan the installation. Each disk is defined by the following data fields:

**OWNER** The virtual machine to which directory entry the MDISK statement is added.

**TARGID** The virtual address, or directory name. The combination of SIZE, BLKSIZE, and the device type is necessary for computing the total required space.

**SIZE** Number of CMS blocks; this number must be converted to cylinders or FBA blocks. VMSES/E supplies the VMFCNVT command to do this conversion.

**BLKSIZE** The CMS block size; the total space required, in bytes, is the product of the SIZE and BLKSIZE values.

**FORMAT** Is the disk format; CMS is the default.

**RECOMPED** This is used for disks, such as the CMS test and production system disks, to reserve an area for the system.

**PREFERRED** Used by RESOURCE option of VMFINS to allocate the minidisk on the higher-performance area, as close to the central cylinders of the physical disk as possible.

**SEPARATED** Lists minidisks that must be placed on a different physical disk. For example, two log areas, or a disk and its backup, should be on physically separated disks.

```
********************************************************************************
****        VMFINS   INSTALL               USERID: MAINT            ****
********************************************************************************
****            Date: 12/09/93            Time: 10:01:28            ****
********************************************************************************
VMFINS2195I VMFINS INSTALL PPF CUFINS CUF ( SYSTEM VM SIDISK51D SIMODE
            D PLAN NORESOURCE LINK DFNAME USER DFTYPEDIRECT DFMODE *
            NOMEMO ADD
********************************************************************************
*                     Requisite Planning Information                           *
********************************************************************************
*                PPF: CUFINS  CUF    PRODID: 6VMVME11%CUF                      *
*           DATE: 12/09/93    TIME: 10:01:28    USERID: MAINT                  *
********************************************************************************
VMFREQ2805I Product :PPF CUFINS CUF :PRODID 6VMVME11%CUF has passed
            requisite checking
********************************************************************************
*                Resource Allocation Planning Information                      *
********************************************************************************
*                PPF: CUFINS  CUF    PRODID: 6VMVME11%CUF                      *
*           DATE: 12/09/93    TIME: 10:01:28    USERID: MAINT                  *
********************************************************************************
********************************************************************************
 Resource requirements for product 6VMVME11 component CUF
********************************************************************************
OWNER: AUDITOR1
  TARGID:       191
    SIZE:       750
    BLKSIZE:    4K
    FORMAT:     CMS
    RECOMPED:   NO
    PREFERRED:  Y
    SEPARATED:  NONE

OWNER: SYSMON1
  TARGID:       191
    SIZE:       750
    BLKSIZE:    4K
    FORMAT:     CMS
    RECOMPED:   NO
    PREFERRED:  Y
    SEPARATED:  NONE

OWNER: MAINT
  TARGID:       FD6
    SIZE:       2250
    BLKSIZE:    4K
    FORMAT:     CMS
    RECOMPED:   NO
    PREFERRED:  NO
    SEPARATED:  NONE
```

*Figure 43 (Part 1 of 2). 6VMVME11 PLANINFO File*

```
  TARGID:       FD2
    SIZE:       2250
    BLKSIZE:    4K
    FORMAT:     CMS
    RECOMPED:   NO
    PREFERRED:  NO
    SEPARATED:  NONE

  TARGID:       FB2
    SIZE:       1500
    BLKSIZE:    4K
    FORMAT:     CMS
    RECOMPED:   NO
    PREFERRED:  NO
    SEPARATED:  NONE

  TARGID:       F9E
    SIZE:       1500
    BLKSIZE:    4K
    FORMAT:     CMS
    RECOMPED:   NO
    PREFERRED:  NO
    SEPARATED:  NONE

  TARGID:       F9D
    SIZE:       750
    BLKSIZE:    4K
    FORMAT:     CMS
    RECOMPED:   NO
    PREFERRED:  NO
    SEPARATED:  NONE

REPLACE USER: AUDITOR1
   USER AUDITOR1 NOLOG 2M 4M ABEG
   ACCOUNT 0000 DISTXYZ
   IPL CMS
   CONSOLE 0009 3215 T
   SPOOL 000C 2540 READER *
   SPOOL 000D 2540 PUNCH A
   SPOOL 000E 1403 A
   LINK MAINT 190 019D RR
   LINK MAINT 19D 019D RR
   LINK MAINT 19E 019E RR

REPLACE USER: SYSMON1
   USER SYSMON1 NOLOG 2M 4M DG
   ACCOUNT 0000 DISTXYZ
   IPL CMS
   CONSOLE 0009 3215 T
   SPOOL 000C 2540 READER *
   SPOOL 000D 2540 PUNCH A
   SPOOL 000E 1403 A
   LINK MAINT 190 019D RR
   LINK MAINT 19D 019D RR
   LINK MAINT 19E 019E RR
```

*Figure 43 (Part 2 of 2). 6VMVME11 PLANINFO File*

## First-Time Installation

The installation of the CMS Utilities Feature is described in *VM/ESA: CMS Utilities Feature*.

You start by loading the PPFs and other control files from tape with the command:

```
vmfins install info ( nomemo
```

If you wish to use different disks from the default addresses, you can create an override file to change those addresses. CUF already supplies its own overrides to the base PPF, 6VMVME11 $PPF:

**CUFINS** For installation. The override has two components:

- **CUF** For full product installation
- **CUFNSRC** For installation without the source

**CUFSRV** For service. The override has two components:

- **CUF** For normal service
- **UCENG** For servicing uppercase English HELP files

You should not use the VMFINS INSTALL PROD command. Instead, create an override, for instance “MYCUF $PPF,” which points to the IBM override, CUFINS $PPF.

Using the VMFPPF command you must then compile the overrides to create the usable form PPF that VMFINS will use in the rest of the installation:

```
vmfppf mycuf cuf
```

or, if you are installing without the source

```
vmfppf mycuf cufnsrc
```

You should also create the equivalent override for the CUFSRV $PPF.

The code is then loaded using the VMFINS command with the INSTALL operand.

After all the code is loaded, you have to run the build EXEC supplied by the CMS Utilities Feature. The CUFINS EXEC moves files to system disks and also, if you so choose, saves any old versions of the tailorable files.

Saving is accomplished by renaming any old files so they have a file type with a prefix of FMT. You then should check all files with this prefix, in order to tailor the new versions of these files with any local changes.

**Note:** Since CMS Utilities Feature loads files to the 19D and 19E minidisks, you will have to re-save the HELP saved segment and the CMS named saved system.

## Migrating

The CMS Utilities Feature uses a separately supplied EXEC to manage tailored files, as they are not identified in the PRODPART file. As a result, the MIGRATION option of VMFINS does not work with the CMS Utilities Feature. However, the CUFINS EXEC allows you the choice of saving any tailored files that you may have. These files are saved by simply renaming them by adding a prefix of “FMT” to the file type. A table listing these files can be found in

*VM/ESA: CMS Utilities Feature* and you must manually compare the old and new files. If you have made any changes, you must redo these changes in the newer versions of these files.

## Building

The CMS Utilities Feature uses VMFINS to load all the parts onto the system, but then uses the CUFINS EXEC, which is supplied on the tape, to complete the installation. This EXEC will copy files to several minidisks.

If you have existing IPF or CMS Utilities Feature 1.0 files, this EXEC will rename any existing tailorable files by adding a prefix of “FMT” to the file type.

You may also have to alter the PROFILE EXEC of the AUTOLOG1 user ID, if you wish to autolog the service machines AUDITOR and SYSMON.

## Deleting

As an exercise, we installed a second copy of the CMS Utilities Feature using an override (MYCUF PPF) to use separate minidisks. We then used the command:

```
vmfins delete ppf mycuf (resource
```

to delete the second copy.

This worked as expected by erasing all the files that belonged to the CMS Utilities Feature component. We encountered only one minor problem. We did not have a copy of the default directory (USER DIRECT) accessed after the VMFSETUP command completed for CMS Utilities Feature. This occurred because we did not use the RETAIN option on the VMFSETUP command. As a result, the empty minidisks for CMS Utilities Feature were not deleted from the directory.

When we executed the VMFVIEW command, we found warning messages in the log advising us which user IDs were no longer needed.

**Note:** The DELETE operand of VMFINS uses the log file $VMFINS $MSGLOG so that you have to enter:

```
vmfview install
```

to examine the log file, even though you have just deleted a product.

---

# Installing a Non-VMSES/E Product

We describe here the installation of PVM from a VM/ESA 1.0 SDO tape. Since PVM Version 2 Release 1 Modification Level 1 is now supplied in VMSES/E format, you should consider the following sections only as a description of what happens to a product in the old INSTFPP format.

## Planning

The PVM product we used was supplied on an SDO tape. To attempt the planning step for this product we issued:

```
vmfins install info
```

to map the SDO tape, produce a list of memos to be printed, and place a file called VMFINS PRODLIST on the A-disk. Then, to actually do the planning, we issued:

**vmfins install prod 5748rc1 ( plan**

But since PVM was in the old INSTFPP product format, there was no planning information available. The only option available was to print the Memo to Users. The memo contained all the information required to build the minidisks and the PVM virtual machine. As in the past, these steps had to be performed manually.

After adding the minidisks to the MAINT user ID, formatting the minidisks, and adding the PVM user ID to the system, we could proceed with the installation. See "First-Time Installation."

## First-Time Installation

We attempted to install PVM from a VM/ESA 1.0 SDO tape with the command:

**vmfins install prod 5748rc1**

We received response messages warning that the tape was INSTFPP format, and that an override file could not be created. VMFINS called INSTFPP to process the tape. If you have a multiple-volume SDO tape, INSTFPP then scans the tape looking for the product and, if additional tapes are required, you are advised to mount the next tape.

When the PVM product is found, the processing is:

1. The installation EXEC for the product (I5748RC1 EXEC) is loaded by INSTFPP to a temporary disk (accessed as "C").
2. INSTFPP calls the installation EXEC.
3. The installation EXEC then loads all the parts of PVM into the minidisk addresses which are hard coded in the installation EXEC itself.

**Note:** If the product updates the 19E and 19D minidisks, you will have to re-save CMS and the HELP saved segment.

## Migrating

The PVM program product was in the old INSTFPP format on the SDO tapes that were available to us. We did not attempt to migrate this product. If you enter the command:

**vmfins migrate prod 5748rc1**

you will receive the message:

VMFINS2604W

indicating that the product could not be migrated.

## Building

Since the PVM product was in the old INSTFPP format, executable forms were loaded directly to disk. There are no build steps required unless you wish to apply service or local modifications to PVM. Those build steps are defined by the product, and have to be manually performed.

Any tailoring of PVM files, such as PVM CONFIG, remains a manual process.

At this time, you should use the VMFSIM command to update the Software Inventory. This can easily be done with the command:

**vmfsim modify vm sysblds * tdata :ppf 5748rc1 none :prodid 5748rc1 :stat built**

This will add an entry in the VM SYSBLDS table to indicate that PVM has been built on your system. See Chapter 7, “Exploring the Software Inventory” on page 135 for more information on the VMFSIM command.

## Deleting

Unfortunately, since PVM is an INSTFPP formatted product, there is no information for VMFINS to use to delete PVM. If you attempt to run:

```
vmfins delete prod 5748rc1
```

you will receive error messages indicating that no deleting will be done.

If you added information on PVM during the build process and now wish to remove it, issue:

```
vmfsim modify vm sysblds * tdata :ppf 5748rc1 none (delete
```

However, leaving the information in place will provide a history record. VMSES/E does not delete information for deleted products. Instead, it records that products have been deleted. You may do the same by issuing:

```
vmfsim modify vm sysrecs * tdata :ppf 5748rc1 none :prodid 5748rc1 :stat deleted
vmfsim modify vm sysapps * tdata :ppf 5748rc1 none :prodid 5748rc1 :stat deleted
vmfsim modify vm sysblds * tdata :ppf 5748rc1 none :prodid 5748rc1 :stat deleted
```

Please see Chapter 7, “Exploring the Software Inventory” on page 135 for more information about the VMFSIM command.

# Chapter 6. Service Experiences

This chapter briefly describes the service process for VM/ESA Release 2.2. An introduction to the service process was provided in “Maintaining Your System” on page 23.

The intent of this chapter is not to replace the *VM/ESA: Service Guide* or the *VM/ESA: VMSES/E Introduction and Reference*, but rather to be an overview that you might be able to use as a checklist.

A brief description of the basic steps of the entire process is followed by a discussion of each of the steps, presentation of some of the errors that might occur, and possible recovery steps.

While the discussions are based on the service of VM/ESA Release 1.1 or 1.5 and up, you should bear in mind that this discussion also applies to program products and user applications distributed in VMSES/E format.

## Basic Steps

The service process has some fundamental steps that apply to all the components of VM/ESA Release 2.2.

In order to familiarize you with the service processes and the commands associated with each of them, we start with a brief list of the basic steps. Each of the commands will be discussed in the following sections. The steps are:

1. Refresh the test build disks.

   See “Refresh” on page 100 for the procedure to be followed.

2. Receive the service documents:

   ```
   vmfrec info
   ```

3. Set up the minidisks:

   ```
   vmfsetup...
   ```

4. Merge the levels of disks as desired:

   ```
   vmfmrdsk...
   ```

5. Load the files from the service tape:

   ```
   vmfrec...
   ```

6. Check that the receive process worked:

   ```
   vmfview...
   ```

7. Actually perform the servicing:

   ```
   vmfapply...
   ```

8. Check that the apply process worked:

   ```
   vmfview...
   ```

9. Rebuild the serviced components:

   ```
   vmfbld...
   ```

10. Check that the build process worked:

    ```
    vmfview...
    ```

11. Test the serviced components.

12. Move the serviced components into production. See “Production” on page 108 for the procedure to be followed. The basic steps are:

    - Re-saving NSSs and saved segments (if necessary).

      **Note:** Before building any segments, use VMFSGMAP to make sure all segment definitions are correct.

    - Moving the test build disk materials to the production build disk.
    - IPL (if necessary).

Some of the components may require additional steps. These steps are described in each product's publications, and care must be taken to ensure that none of the steps are skipped. For VM/ESA, see *VM/ESA: Service Guide*.

## VM/ESA Servicing Highlights

During the application of service, we attempted to watch for possible error situations that could arise. You may encounter some of the problems described for each step. The good news is that VMSES/E has been designed to permit easy recovery from most errors.

### Refresh

This step ensures that the test build disks have the correct contents.

The build step places the newly built objects on the test build disks, and the last step in the service process, moving to production, copies the test build disk contents to the production disk. Therefore, it is very important that the test build disks have the same contents as the production disks *before you begin the service process*.

Simply copying all files from the production to the test build disks may not be advisable. When the disks are shared by several components or products, you should copy only the files that are related to the product or component being serviced: If you copy all the files on the disk, you may accidentally back-level the products you are not servicing. For example, suppose CP service updated HCPOM1 MACLIB. This file resides on MAINT's 193 disk, so the new file is placed on MAINT's 493 for testing. If, during the testing, you receive some urgent service for CMS, you cannot copy all files from the 193 disk to the 493 disk: doing so would back-level HCPOM1 MACLIB. Only the CMS-related files should be copied.

Before copying the files, though, you should erase all the product-related files from the target disk. This is the simplest way to guarantee that both disks are identical, regarding that product.

For VMSES/E products, this is a simple operation, using the CMS Pipelines and VMFERASE. Suppose you want to copy the CMS tools disk (MAINT 193) to the

test disk (MAINT 493). Create an EXEC (COPYLIST EXEC) containing the following statements:

```
/* product files copy */

Arg source target .

'PIPE < copylist simdata a |'        ,   /* Read VMFSIM output      */
    ' strip |'                       ,   /* remove extra blanks     */
    ' find :PARTID |'                ,   /* only keep files records */
                                     ,   /* construct a command:    */
    ' spec /COPYFILE/ 1'             ,   /*  COPYFILE               */
    '      w 2-3 nextword'           ,   /*     fn ft               */
    '      /' source '= =' target    ,   /*       fmi = = fmo       */
    '      ( OLDD REPL/ nextword |'  ,   /*                   (opts */
    ' command'                           /* pass the command to CMS */
```

Now issue:

```
access 193 b
access 493 c
vmferase prod 6vmvma22%cms from c
vmfsim query vmses partcat b tdata :prodid 6vmvma22%cms ( file copylist
exec copylist b c
vmfsim modify vmses partcat c file copylist ( add
```

In sequence, what happens is:

- The disks are accessed.
- VMFERASE erases from the C-disk all files the parts catalog lists as related to the CMS product.
- VMFSIM QUERY creates a list of files residing on the B-disk and belonging to CMS.
- The pipe copies all selected files from the B to the C-disk.
- VMFSIM updates the parts catalog on the C-disk to include the copied files.

If the product is not VMSES/E-formatted, and it shares a disk, then the only safe way is to manually keep lists of the product-owned files.

## Preparation

The INFO operand of the VMFREC command allows you to obtain the Memo to Users that is shipped with each service tape. Please take the few moments needed to review this document. For the complete list and format of the documents loaded, see Appendix B, “Product Packaging and Distribution Media Formats” on page 205.

If you have not already done so, you should also contact your IBM Support Center and request any “install error buckets” that may be available for the products being installed.

It is very important that you do a backup of your production build disks. As will be discussed in “Production” on page 108, the backup is the fastest and safest way you have to fall-back to the previous production level, should a problem arise after you have moved the tested component or product into production.

## Setup

Before you can start your service process for a product you must access all of that product's disks. VMSES/E supplies a utility, the VMFSETUP command, to do this, so you do not have to remember minidisk addresses or directory names.

### VMFSETUP Command

You may wonder what might go wrong with the accessing of some disks but the VMFSETUP EXEC has had major changes, as compared to versions previous to VM/ESA Release 1.1. The most important changes are that:

- You decide whether and when other VMSES/E commands should call VMFSETUP
- VMFSETUP no longer invokes VMFOVER
- Can link and detach minidisks

There is a tag in the PPF to control the use of VMFSETUP. The tag is :SETUP, and in the default PPFs shipped with VM/ESA it has a "NO" value. This means that the other VMSES/E commands will not call VMFSETUP unless you specify SETUP as an option on these commands.

Dropping the call to VMFOVER has a subtle implication. If you create an override file, you must now use the commands VMFOVER to test your changes, and VMFPPF to compile the PPF and generate its usable form. For example, if you wish to create an override (named YOURPPF) to the CMS component, you would use the command:

```
vmfppf yourppf cms
```

to create the compiled PPF file.

VMFSETUP will use only compiled PPFs. If you forget to compile the PPF, your override will not be invoked and the results may not be what you wanted.

You will find that the ESA PPF file is an override file itself and is a good example of the functions available in an override. Also, see the example in "Creating a PPF Override" on page 189.

Another new PPF capability allows you to specify that the LINK command is to be issued for minidisks. If you are trying to use VMSES/E from a user ID other than MAINT, you will need read/write authority for the minidisks in order for the LINK command to be successful.

Please see "Variable Declarations Section" on page 48 for a detailed discussion of this feature.

**Note:** If a disk is already linked at the correct address, with the correct link mode, VMFSETUP will not link it again.

### VMFQMDA Command

A related VMSES/E command, VMFQMDA, searches the :MDA area of a PPF and displays a list of all disks defined there, noting which ones are accessed. The format of the output is similar to that of the CMS QUERY ACCESSED command.

The great advantage of this command is that it allows you to check whether you have accessed the correct disks without destroying the current access order. It can also tell you which disks two products have in common. If you use

VMFSETUP to access the disks for one product, then invoke VMFQMDA for the other product, any disks listed with an access mode are common.

For example, after having accessed the disks for CP we issued:

**vmfqmda esa cms**

which produced the reply

```
VMFUTL2205I Minidisk|Directory Assignments:
            String    Mode  Stat  Vdev  Label/Directory
VMFUTL2205I LOCALMOD  ---     ---  3C4   ------
VMFUTL2205I LOCALSAM  ---     ---  3C2   ------
VMFUTL2205I APPLY     ---     ---  3A6   ------
VMFUTL2205I           ---     ---  3A4   ------
VMFUTL2205I           ---     ---  3A2   ------
VMFUTL2205I DELTA     ---     ---  3D2   ------
VMFUTL2205I BUILD7    K       R/W  493   MNT493
VMFUTL2205I BUILD6    L       R/W  490   MNT490
VMFUTL2205I BUILD5    M       R/W  19D   MNT19D
VMFUTL2205I BUILD2    N       R/W  193   MNT193
VMFUTL2205I BASE2     ---     ---  3B2   ------
VMFUTL2205I --------  A       R/W  191   MNT191
VMFUTL2205I --------  B       R/W  5E5   MNT5E5
VMFUTL2205I --------  D       R/W  51D   MNT51D
VMFUTL2205I --------  E       R/W  2C4   MNT2C4
VMFUTL2205I --------  F       R/W  2C2   MNT2C2
VMFUTL2205I --------  G       R/W  2A6   MNT2A6
VMFUTL2205I --------  H       R/W  2A4   MNT2A4
VMFUTL2205I --------  I       R/W  2A2   MNT2A2
VMFUTL2205I --------  J       R/W  2D2   MNT2D2
VMFUTL2205I --------  O       R/W  194   MNT194
VMFUTL2205I --------  S       R/O  190   MNT190
VMFUTL2205I --------  Y/S     R/O  19E   MNT19E
```

The reply includes:

- Disks common to CP and CMS: the ones accessed as modes K, L, M, and N.
- CMS-only disks: all unaccessed disks having a string name (all appearing before the A-disk).
- CP-only disks: all appearing after the D-disk.

## Merge

Merging the levels of disks provides a method for separating the previous version of the system from the new parts that are about to be loaded. This separation is accomplished by having different logical levels of disks for each version of the system. This level structure has been discussed in “Multiple System Levels” on page 26. VMSES/E supplies a tool, VMFMRDSK, to merge a disk level with a lower level.

### VMFMRDSK Command

The concept of logical disk strings has been introduced in “Product Database Layout” on page 14 (see also “Recommended Logical Strings and Service Levels” on page 27). In a string, each minidisk or directory holds a product’s service level. The basic idea is to let you keep several levels of your products. Depending on the string, up to three levels are recommended by IBM:

1. Alternate (work-disk level)
2. Intermediate (test-level of the product)
3. Production (“floor-level” of the product)

Though you may add new service to the service already existing on the alternate level, if you do so, later on you will not be able to separate the new service from the existing service. New service should be received into an empty level, except when:

- Adding missing prerequisites
- Receiving several COR tapes you want to apply together

The following discussion assumes the recommended three-level structure. To isolate the new service from the previous level, you may want to move the existing alternate level down one level. This can be done in either of the following ways:

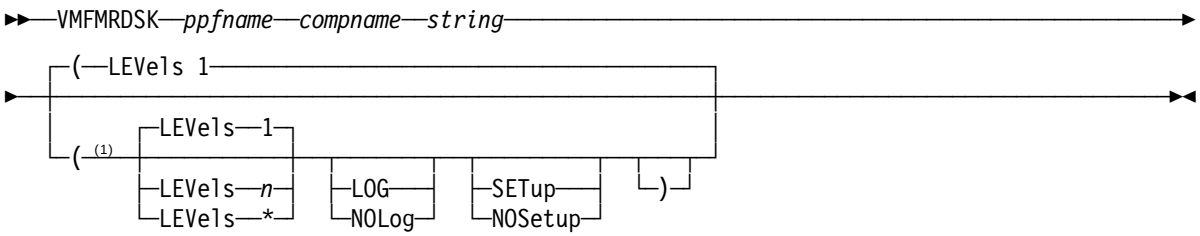
- Merge the intermediate level with the production level, clear the intermediate level, and then move the contents of the current alternate level to the empty intermediate level.

  **Note:** You should not merge the production and intermediate levels until you have thoroughly tested the service in the intermediate level. To be really cautious, you may even want to keep the intermediate level isolated for some time after putting it into production.

- Merge the current alternate service with the service on the intermediate level.

To help you in this process, VMSES/E provides the VMFMRDSK command. This command merges *disks*, so if the disks are shared by several products or components, all those products or components are merged in the same operation. You must be careful and synchronize the service process for those products or components, which is the case for VM/ESA. Otherwise you may easily create a situation from where the only way out is product re-install.

Figure 44 shows the syntax of the VMFMRDSK command.




**Note:**
[1] You can enter options in any order between the parentheses.

*Figure 44. VMFMRDSK Command Syntax*

If you wish, for example, to move the alternate level of the CMS APPLY string to the intermediate level, the command is:

```
vmfmrdsk esa cms apply ( level 1
```

You must specify the PPF name, the component name, and the string name. You may also specify the number of levels to merge. The default is one level, from the alternate to the intermediate.

If you want to merge all levels, the command is:

```
vmfmrdsk esa cms apply ( level *
```

VMFMRDSK is an enhanced copy function that checks for disk space, and updates the VMSES PARTCAT file. Figure 45 shows the two possible examples of merging a three-level APPLY string, as follows:

- Two-level merge
- One-level merge

**Note:** The merge operation always involves the first two disks for the string. It is impossible to merge only levels other than the first level; for example, you cannot merge only the intermediate and production disks; you must also merge the alternate and intermediate disks.

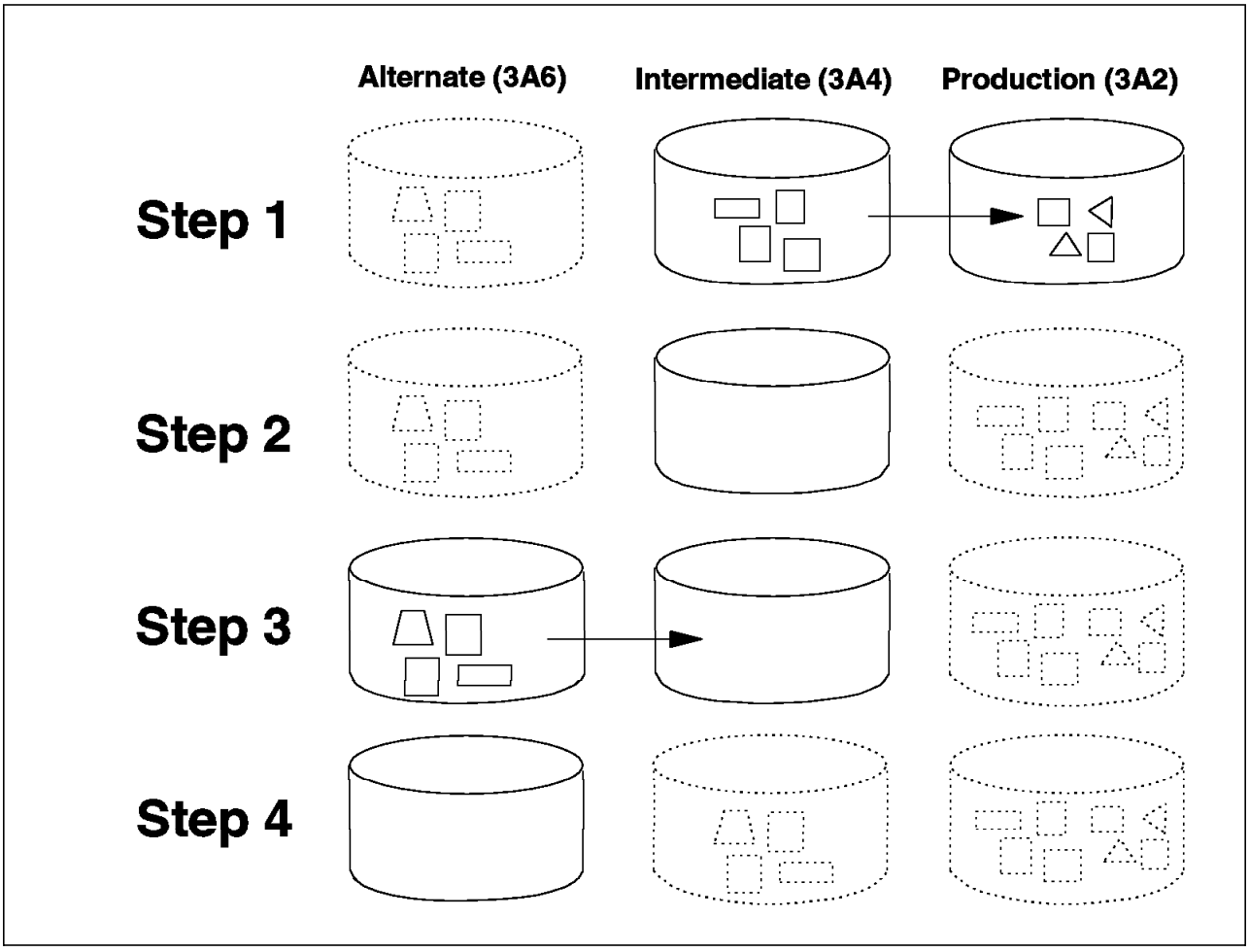



*Figure 45. Example of Merging the CMS APPLY String*

For the two-level merge, the following steps take place:

Step 1. Lower-level merge:

- Check that the production disk has enough free space to contain the files on the intermediate disk.
- Copy files from the intermediate disk to the production disk.
- Update VMSES PARTCAT on the production disk.

Step 2. Cleanup:

- Erase all files on the intermediate disk.

Step 3. Higher-level merge:

- Check that the intermediate disk has enough free space to contain the files on the alternate disk.
- Copy files from the alternate disk to the intermediate disk.
- Update VMSES PARTCAT on the intermediate disk.

Step 4. Cleanup:

- Erase all files on the alternate disk.

For the one-level merge, the last two steps of the two-level process take place:

Step 1. Higher-level merge (Step 3 of two-level):

- Check that the intermediate disk has enough free space to contain the files on the alternate disk.
- Copy files from the alternate disk to the intermediate disk.
- Update VMSES PARTCAT on the intermediate disk.

Step 2. Cleanup (Step 4 of two-level):

- Erase all files on the alternate disk.

Even though VMSES/E does not use the file mode number to indicate when a specific service fix has been applied to the system, you should not use the COPYFILE command to manually merge the disks. VMSES/E maintains a file (VMSES PARTCAT) on each disk that keeps track of all the files residing on the disk. If you wish to merge a disk manually, you should use the VMFCOPY command instead of COPYFILE.

Before beginning the copy, VMFMRDSK also checks to see whether there is enough space on the target disk. This check is very thorough. It accounts for existing files, the size of the largest file, differences in block sizes for the source and target disks, and even attempts to leave some empty space on the target disk. If the target disk does not have enough space, the copy is not initiated, and VMFMRDSK terminates. If this should happen, the recovery is very simple. You just have to make the target disk larger and re-run the VMFMRDSK function.

## Receive

The receive function has been streamlined since the previous version of VMSES. Part of its previous functions, such as loading parts in executable form into the system disks, were either changed or deleted.

**Note:** The VMFREC command only loads parts in a non-executable format, and always to the alternate DELTA disk. only to the alternate DELTA disk in a **non-executable** format.

This effectively eliminates the problem of accidentally “back-leveling” a part of your system. Since the receive process has been greatly simplified, if an error occurs (for example, a disk fills up), the receive process can easily be restarted without having to restore any disks. In the disk-full case, you only have to:

- Make the alternate DELTA minidisk larger or, if you are using an SFS directory, issue a MODIFY USER command or add more space to the SFS server.
- Re-run the VMFREC EXEC.

Also, if a PTF has been received it will not be re-received. The same is true for missing parts of committed PTFs, though this may be controlled by the value of the PPF :RECVALL tag and the RECVALL option of the VMFREC command.

## Check

The VMFVIEW tool allows you to check the results of all of the major steps of the service process. VMFVIEW should always be used after each step, and any errors that are reported by VMFVIEW must be corrected before continuing.

If you encounter an error message in the log file, you can use the online HELP facility to obtain further information about the message.

## Apply

The apply process has also been greatly simplified, and the performance of this function has been improved, compared to the previous version of VMSES.

The use of the service-level Software Inventory is the main cause for the improved performance of the VMFAPPLY command, and it has also greatly reduced the number of files on the minidisks. VMFAPPLY creates or updates Version Vector tables, using a new algorithm, at the same time it verifies PTF requisites. Since a single VVT holds all the information on the applied PTFs (each part that was ever serviced has a record in a VVT that lists all the PTFs applied to that part) the process is very fast. If any requisites are missing, VMFAPPLY does not update the VVT. Instead it creates two files: one lists all PTFs that passed requisite checking; the other all PTFs that have missing requisites.

The only remaining problem might be with the packaging of a PTF (for example, some of its parts not being properly included on the COR tape). Use the VMFVIEW command to check the apply log for missing parts. A file listing the missing parts (file type of $MISSING) is written during the apply process. In order to resolve this problem, another new feature of VMSES/E makes it possible for a PTF to completely supersede another PTF. This enables you to install a PTF that completely replaces an incorrectly packaged PTF.

Any parts for which you have created local service, and that are being serviced, are identified in the apply log. This is one more reason for you to check the log.

**Note:** Before proceeding with the service process, you should make sure that all local service affecting the parts now serviced is still valid.

Note that the apply process also generates, or appends to, a file the list of all serviced parts. This file is called the “Select Data File” and is used by the build process.

## Build

The build process has been greatly enhanced. In order to provide a coherent view of the process, without disrupting the present discussion, we only give here some basic information. For the full discussion, see “How Build Works” on page 110.

As mentioned, the VMFBLD command can now handle almost all types of objects. The following list indicates the only objects that still have to be built by hand:

- PPFs.
- Named saved systems.
- Saved segments, in VM/ESA Release 1.5 370 Feature systems.
- Stand-alone dump. You have to run the HCPSADMP EXEC if the SA object was serviced.
- ASSEMBLE command. You have to run the ASMGEND EXEC if any of the objects in the DMSBLASM build list were serviced.
- DOSLIBs, in VM/ESA Release 1.5 370 Feature systems.

**Notes:**

- If you do not specify a build list on the VMFBLD command, the VMFBLD process will re-build either all serviced objects or all objects, depending on the option specified on the VMFBLD command. While this will not do any damage, since you are working with the alternate build disks, it might take a considerable amount of time.
- If the PRIVATE option is not specified, VMFBLD deletes all objects that require deleting.
- You can build a private copy of an object, for test purposes, without disturbing the product.
- VMFBLD detects any serviced source PPFs and prompts you to stop the build step. This allows you to re-compile the PPFs and then return to the build process.
- You can detect the need to rebuild objects by using VMFVIEW to see the build message log.
- In addition, at the end of the service process, you must manually intervene after building a nucleus in order to place it on the correct disk.

## Test

You should perform a thorough test of the product before moving it into production. If such test is not possible, you should at least follow the recommendations in “Production” to establish a fall back capability.

**Never** merge the intermediate and production levels of the APPLY string before you are sure the new test system is stable. Once they are merged, it is very difficult to separate them.

## Production

Placing the tested components or products into production requires:

1. Re-generating any affected shared segments, and named saved systems.
2. Copying the newly built usable forms from the test to the production build disks.
3. Erasing from the production build disks any usable forms deleted service.
4. Re-IPLing the entire system to activate a new CP nucleus (if required).

The *VM/ESA: Service Guide* recommends, with the exception of the CMS system disk (MAINT 190), the following method to move a VM/ESA component to production:

1. Back up the production disk (optional, but highly recommended).

2. Erase the production build disk.
3. Copy all files from the test disk to the production disk.

For the CMS system disk the steps are:

1. Back up the production disk (optional, but highly recommended).
2. Do a DDR copy from the test to the production disk.
3. Use the CMS FORMAT command with the LABEL option to recover the original disk label.

This method should not be generalized to other products, because it implicitly assumes that:

- Before service, the test build disks are identical to the production build disks. You should make sure this is true; otherwise recovering from an error may require a re-install of the product.

  It is *critical* that you backup the production disks. Again, if you do not have a backup, recovering from an error may require a re-install of the product.

- The serviced product does not share the build disks with other products. Sharing the disks by several components of the same product should not cause any problems, if service for the components is coordinated.

If the build disks are shared, the only safe way of doing the copy is:

1. Back up the production build disks.
2. For each test build disk determine which files have been added, changed, or deleted.
3. Using the VMFERASE utility, erase from the production disk all files that service has deleted.
4. Using the VMFCOPY utility with the COPYFILE REPLACE options, copy from the test to the production disk all new and changed files, or simply copy all the product's files.

To determine which files have been added, changed, or deleted, follow this procedure:

1. Find the value of the :BLDID tag in the product's PPF.
2. Look for objects with a build status of ERROR or MANUAL. Enter the command (with the appropriate value for *status*):

   **vmfsim query** bldid **srvblds * tdata :status** status **:object**

   If you find any such objects, correct the errors before proceeding.

3. Now issue the commands:

   **vmfsim query** bldid **srvblds * tdata :status built :object ( file changed**
   **vmfsim query** bldid **srvblds * tdata :status deleted :object ( file deleted**

   The value of the :STATUS tag includes a date that enables you to select the correct objects.

Though we do not recommend it, we describe below an alternate procedure. This procedure is more subject to error, and this is why we prefer the other:

1. For each build disk, obtain the entries in the parts catalog of the test and production disks. Use the command:

**vmfsim query vm partcat** fm **tdata :prodid** prodid **( file** fm**-disk**

to create a file, fm-DISK SIMDATA, that lists all the files for the product "prodid" that reside on the disk accessed as "fm."

2. Compare each pair of files, for example by using XEDIT, to view them side by side.

## Service Back-Out

Backing out the service to recover from bad PTFs uses one of two techniques:

- Restoring the production disk backup.
- Regenerating all serviced objects, but at the previous service level; in other words, removing the most recent level of service. This is explained in detail in Appendix C, "Removing Service" on page 217.

---

# How Build Works

The build process was briefly outlined in "Build the Product" on page 33. In this section we will describe it in further detail.

## Overview

Each product is a collection of many objects. One can group these objects by types, such as libraries, modules, nuclei, and so on. Each object type is built using specific tools and rules, so the VMSES/E strategy to build objects is based on a two-step process and object-specific part handlers.

The two-step process makes sure that tasks common to all object types are performed in the first step, leaving the actual building, with its object-specific operations, for the second step. To execute only the first step, invoke VMFBLD using the STATUS option; to execute both steps in sequence, invoke VMFBLD with the SERVICED or ALL options.

During the STATUS step VMFBLD scans the $SELECT files listed on the :APPID tag of the product's PPF. Using the parts listed there, VMFBLD scans the product's build lists, identifying all objects that include those parts and, consequently, have to be re-built. VMFBLD flags those objects as SERVICED in the service-level Build Status table for the product.

## Object Definition Change Detection

Suppose an object is added to a build list (for example, a module is split) or is removed from the build list, or is in some way changed in the build list. Since Release 2, these occurrences are detected by VMSES/E and build requirements are automatically generated.

When service is applied to a build list, VMFAPPLY saves the file type of the previous version by inserting it in the Select Data File. VMFBLD, when executing the STATUS option, is then able to compare the old and new versions of the build list and detect any changes. Changed objects are then flagged as SERVICED in the Build Status table. Objects no longer present in the new version of the build list are flagged as DELETE. Later the flagged objects are rebuilt (or deleted) by the appropriate part handlers.

## Object Requisites

With the automation of library build, it became necessary to introduce object build requisites. For instance, an object built from members of a TXTLIB must be rebuilt when one of those members is serviced; but not only do the TXTLIB members have to be built, they have to be built *before* the object.

Support for object build requisites allows defining these kinds of relations between objects. Relations can be established at the build list level (when all objects in a build list are required) and also at the object level.

An object that requires another object or objects in order to be correctly built is a *dependent* object. The required objects are called *requisites* of the dependent object. As stated, the requisite objects are built before the dependents.

As an example, suppose object A requires object B. If you are building A, and B has been serviced, B is automatically built, then A is built. The reverse is not necessarily true, because A may have several requisites other than B. Therefore, if you are building B, A may not be built, because all A requisites have to be satisfied before A can be built. So, the status of A is changed (to SERVICED or BUILDALL).

Object A is also built if it belongs to the same product, or component, as B and one of the following conditions is true:

- VMFBLD was invoked with the ALL option, and no build list name was given.
- VMFBLD was invoked with a build list name, no object name, both A and B belong to the same build list, and A has no requisites in other build lists.

In all other cases object A is not built. VMFBLD will issue a message indicating the number of objects remaining to be built. For another example, see “Segment Servicing” on page 164.

This support consists of:

- Enhancements to VMFBLD and the build part handlers.
- New tags in Format 2 and Format 3 build lists.

Because Format 1 build lists do not support tags, the VMFBDNUC part handler was enhanced (in Release 2) to support a new option, BLDREQ, that allows specifying any object or build list required by the nucleus.

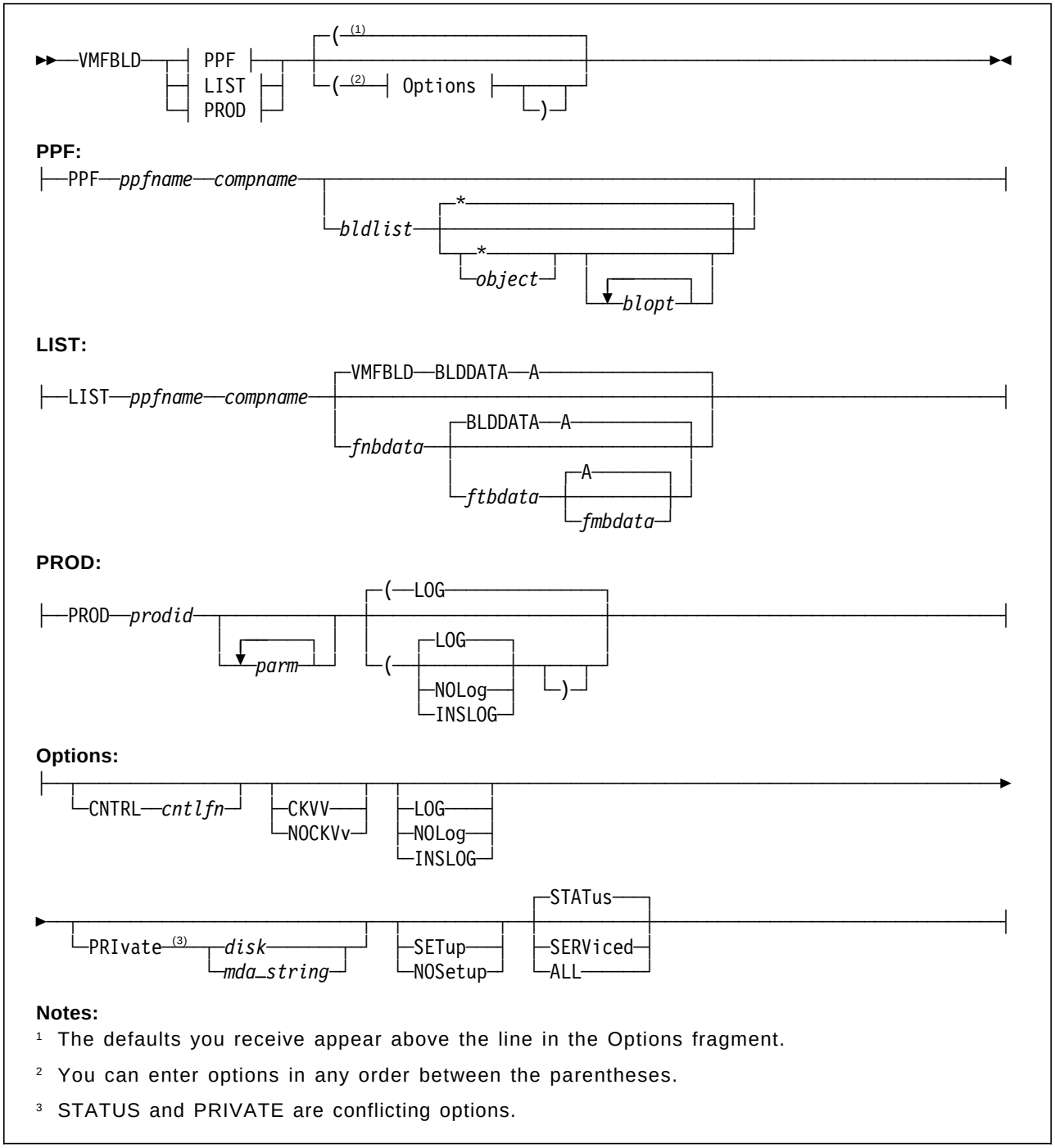
**Note:** The other part handler using Format 1 build lists, VMFBDCPY, was similarly enhanced. However, VMFBDCPY is supported only for downward compatibility. A product supported only on VM/ESA Release 2 and above should use VMFBDCOM instead of VMFBDCPY.

As described above, requisites drive the build sequence, which means the build list sequence in the :BLD section of the PPF may no longer be followed.

Build requirements for objects defined in Format 2 and Format 3 build lists are described by new tags in those lists. The :GBLDREQ tag allows specification of requirements globally. Any objects having individual requirements must specify them using the :BLDREQ tag in the object definition block. This specification overrides the global one.

## VMFBLD Command

Figure 46 shows the syntax of the VMFBLD command, the VMSES/E object build tool.




**Notes:**

1. The defaults you receive appear above the line in the Options fragment.
2. You can enter options in any order between the parentheses.
3. STATUS and PRIVATE are conflicting options.

*Figure 46. VMFBLD Command Syntax*

## STATUS Option

The STATUS option causes VMFBLD to identify all objects that have to be rebuilt, without actually building any objects. If there is no new service for the component, this step is bypassed.

Using the Select Data File created during the apply process, VMFBLD scans the build lists and identifies all objects that have had their parts serviced. These objects are flagged in the product's service-level Build Status table as SERVICED. In addition, any objects that require the serviced objects are also flagged as SERVICED. If the build list itself has changed, any objects added to the build list, or whose definition has changed are identified as SERVICED; any deleted objects are also identified, as DELETE.

## SERVICED Option

The SERVICED option first executes the functions of the STATUS option, if required. It then calls the appropriate part handlers to build and delete objects

- whose status in the Build Status table is SERVICED, BUILDALL, or DELETE
- and which are specified in the VMFBLD command, are in the specified build list (if no object is specified), or are in the specified component (if no build list is specified).

Build requisites of these objects are also built if their status is SERVICED, BUILDALL, or DELETE.

**Note:** In VM/ESA Release 2.2, if the STATUS option was already run, that part of this option is not run again.

The part handlers are responsible for actually building or deleting the objects.

## ALL Option

The ALL option first executes the functions of the STATUS option, if required. It then calls the appropriate part handlers to build and delete all objects which are specified in the VMFBLD command, are in the specified build list (if no object is specified), or are in the specified component (if no build list is specified), even if there has not been any service for these objects. Build requisites of these objects are also built if their status is SERVICED, BUILDALL, or DELETE. The ALL option is required if you wish to re-build an object (for example, the CP nucleus) but you have not applied any service to any of its parts.

**Note:** In VM/ESA Release 2.2, if the STATUS option was already run, that part of this option is not run again.

For example, if you added a new device to the HCPRIO ASSEMBLE file, and followed the common practice of directly editing this file, you must use the ALL option, since no service has been applied. In this case, you can also take advantage of the FASTPATH option of VMFBLD. In VM/ESA Release 2.2, if the STATUS option was already run, that part of this option is not run again.

**Note:** If you explicitly indicate an object on the VMFBLD command line, that object and its dependents, if any, are flagged as BUILDALL.

## PRIVATE Option

You should use this option when you want to quickly create a test copy of an object. You specify where VMFBLD should place the copy. VMFBLD does not update the Software Inventory. Note that if the object has requisites that were serviced, those requisites are built as well. Deleting objects, however, is not done.

This option should not be used to build saved segments: Saved segments are always built on the real system.

## Build Part Handlers

Build part handlers act as "snap-on" pieces to the VMFBLD base, allowing extension of the build function to any type of object. VMSES/E has defined several part handlers, as shown in Table 5. Unless indicated otherwise, these part handlers are also supported on VM/ESA Release 1.5 370 Feature.

*Table 5. Build Part Handlers*

| Name | Build List Type | Object Type | Comments |
|---|---|---|---|
| VMFBDNUC | Format 1 | Nuclei | Several user options |
| VMFBDCPY | Format 1 | Text replacement only | Supported for compatibility |
| VMFBDCOM | Format 2 | Replacement objects | Example: EXEC, XEDIT, HELP panels |
| VMFBDMOD | Format 2 | MODULE | |
| VMFBDGEN | Format 2 | Varies | Release 2.1 - generated objects |
| VMFBDMLB | Format 3 | MACLIB | Release 2 |
| VMFBDTLB | Format 3 | TXTLIB | Release 2 |
| VMFBDLLB | Format 3 | LOADLIB | Release 2 |
| VMFBDCLB | Format 3 | CSLLIB | Release 2.1 |
| VMFBDDLB | Format 3 | DOSLIB | Release 2.1 (•) |
| VMFBDSBR | Format 2 | Saved segments | Release 2 - identifies requirements |
| VMFBDSEG | Format 2 | Saved segments | Release 2 - builds segments (•) |

**Note:**

- Not available on VM/ESA Release 1.5 370 Feature

## VMFBDNUC Options

You should become familiar with the following options of the VMFBDNUC part handler: These options are:

- FASTPATH
- NUCTARG
- RLDSAVE
- MODNAME

**Note:** For VM/ESA Release 1.5 370 Feature, RLDSAVE and MODNAME are not supported options.

### FASTPATH Option

The FASTPATH option reuses the temporary load list that was created by the last invocation of VMFBLD.

If you serviced the component then you **cannot** use the FASTPATH option. The reason is that the load list contains the list of text files at the service level they were *before* applying the service. Not only the file types of the serviced text files have changed, text files may also have been added or deleted from the load list.

Combining the ALL and FASTPATH options greatly reduces the time required to build a new nucleus. As an example of the use of this option, the command to rebuild the CP nucleus is:

```
vmfbld ppf esa cp cpload * fastpath ( all
```

VMFBLD will reuse the temporary load list in file CPLOAD $NUCEXEC A, in the case of CP.

### NUCTARG Option

The NUCTARG option of VMFBDNUC lets you specify where the nucleus loader should leave the nucleus:

**PUNCH** In the virtual punch (possibly spooled to your reader)

**TAPE** Copy to a tape

**DISK** In the CP NUC area (valid only for CP)

**MODULE** As a CMS MODULE (valid only for CP)

NUCTARG PUNCH is the default value. However, see the :BLD section of the PPF for product defined defaults. This means that you now have to manually load the new nucleus to disk when you are ready.

**Note:** For VM/ESA Release 1.5 370 Feature, DISK and MODULE are not supported options.

## CP Configurability Support

Since VM/ESA Release 2 VMSES/E supports the CP configurability functions. Note that CP configurability is not available on VM/ESA Release 1.5 370 Feature. This support is specified through new options of the VMFBDNUC part handler:

- NUCTARG MODULE

  Allows saving the nucleus as a CMS MODULE file. The target disk is **not**, by default, a CP PARM disk. You will have to manually copy the MODULE file to a PARM disk. This is to prevent you from overwriting your existing CPLOAD MODULE during an automatic process. Follow the guidelines in *VM/ESA: Service Guide*.

- MODNAME fn

  Allows naming the nucleus file (the file type is MODULE).

- RLDSAVE

  Saves relocation information for the nucleus. In this way, the nucleus can be relocated at IPL time. This capability permits changing the size of the V=R area without re-generating the nucleus, and also allows system recovery in the case of a storage failure inside the nucleus area.

The following command is an example of a CP nucleus build, using the configurability support:

```
vmfbld ppf esa cp cpload * fastpath rldsave nuctarg module modname mynuc ( all
```

## More on Build Lists

As discussed in “Build Lists” on page 54, there are three types of build lists:

- Format 1 - called load lists in previous releases
- Format 2 - new with VM/ESA Release 1.0
- Format 3 - new with VM/ESA Release 2

You should be aware that the build lists are now serviced by replacement. The new part is shipped as file of file type EXCnnnnn, where “nnnnn” is the PTF number. VMFBLD locates the highest service level for the build list and uses it.

For example, if the CPLOAD EXEC is serviced by a replacement file called CPLOAD EXC12345, VMFBLD will use this file as the load list, instead of the CPLOAD EXEC. If for any reason (for instance, a product requirement) you have modified the build list, you will have to handle the change using the local modifications procedure. For detailed information, see *VM/ESA: Service Guide*.

This means you should carefully scan the apply log and if you detect any message concerning build lists (in reality any message related to local service) you should investigate whether your changes are still valid, and perform the necessary modifications.

Remember: for the apply process to detect and log these occurrences, you **must** have logged your local modifications in the Software Inventory.

## Format 3 Build Lists

Format 3 build lists were introduced in Release 2 and support the building of libraries. Each library is defined in a separate build list, and each member is described as an object in the build list. As we have seen, libraries to be declared as GLOBAL can be specified, and object-requisite support is also provided.

A new tag, specific for Format 3 build lists, was introduced. The :LIBNAME tag allows specifying a name for the library (the default is the file name of the build list file).

## Support of Global

Format 2 and Format 3 build lists were further enhanced to allow specification of libraries needed during the build process. This support is used, for example, by the VMFBDMOD part handler to issue a GLOBAL TXTLIB command for the required libraries. The VMFBDSEG part handler supports all types of libraries.

GLOBAL commands are issued for these libraries. Existing GLOBAL definitions are saved before issuing new GLOBAL commands, in preparation for building an object, and restored after building the object.

Using the :GGLOBAL tag you can define libraries required by all objects in a build list. The libraries are added to the end of the existing globals. Any objects

having individual requirements must specify them using the :GLOBAL tag in the object definition block. This specification overrides the one made in with the :GGLOBAL tag.

You can specify the following library types: CSLLIB, DOSLIB, LOADLIB, MACLIB, and TXTLIB.

### Other Build List Enhancements

For Format 2 build lists, the :OBJNAME tag requires both the file name and file type of an object to be specified. Objects with the same file name but with different file types can now be specified in the same build list. This is also required by object deletion support.

The :GOBJPARM tag allows the definition of parameters applying to all objects in the build list. Any object parameters specified on the :OBJNAME tag override the global parameters for that object.

Many new values for the :OPTIONS tag, mainly to support library build, have been introduced.

## Update Control Files

Several parts, such as ASSEMBLER files, EXECs, and XEDIT macros are *source maintained*. This means that, whenever the part is changed, only a small update file with the changed lines has to be included in the service sent, along with control information. It is then necessary to update the original (or base) source file with the changed file, according to the directives in the control information. This control information is organized in a structure with several types of files. This section describes this structure and the associated files, and though we use VM/ESA as example, it applies equally to program products. For more detailed information see *VM/ESA: VMSES/E Introduction and Reference*.

**Note:** IBM supplied service includes already assembled text files, so you do not have to update and assemble the source, unless you have local modifications to insert.

The control structure is based on a series of files “pointing” to other files. There is a primary file, which has a file type of CNTRL. This main control file contains:

- Assorted control information.
- Pointers to auxiliary control files (AUX files) and VVTs. These files, in turn, point to the actual updates.

The use of a major control file provides a place to define information that is common to all parts of a component, or product, such as the different service-levels (IBM, local, relief, patches, and so on), and to leave the part-specific information to individual AUX files (and VVTs). The main control file is used:

- By the XEDIT command, when creating a source update.
- By the UPDATE command, to locate and apply the source updates.

  **Note:** The UPDATE command is implicitly called by the VMFASM, VMFHASM, VMFHLASM, GENCPBLS, VMFNLS, and VMFEXUPD commands.
- By the VMFBLD command, to select the most recent level of a part.

The updates are changes to the source files that are shipped as part of VM/ESA Release 2.2. The updates are applied by creating a temporary copy of the source file with the changes included. This temporary source file is then compiled into a usable form that will become part of the system.

This structure anticipates, and avoids, some of the inherent problems that were described in “Maintaining Your System” on page 23. These problems are avoided simply by applying the updates in the right order. As briefly described above, the control file allows you to have multiple AUX files per part, so you can handle different types of service, and you can apply this service in a specific order, for example:

1. IBM service (RSU, or PUT, and COR)
2. Local service
3. Patches

The control and AUX files provide input to the CMS UPDATE command and the XEDIT update facility, and are read from bottom to top. The *bottom* entry in the file is read *first*.

With the exception of local service and patches, these AUX files are automatically generated for you by VMSES/E.

An example of how to make and apply local service, using this type of structure, can be found in “Local Service” on page 122.

Figure 47 shows the relationships between control files, AUX files, and updates.

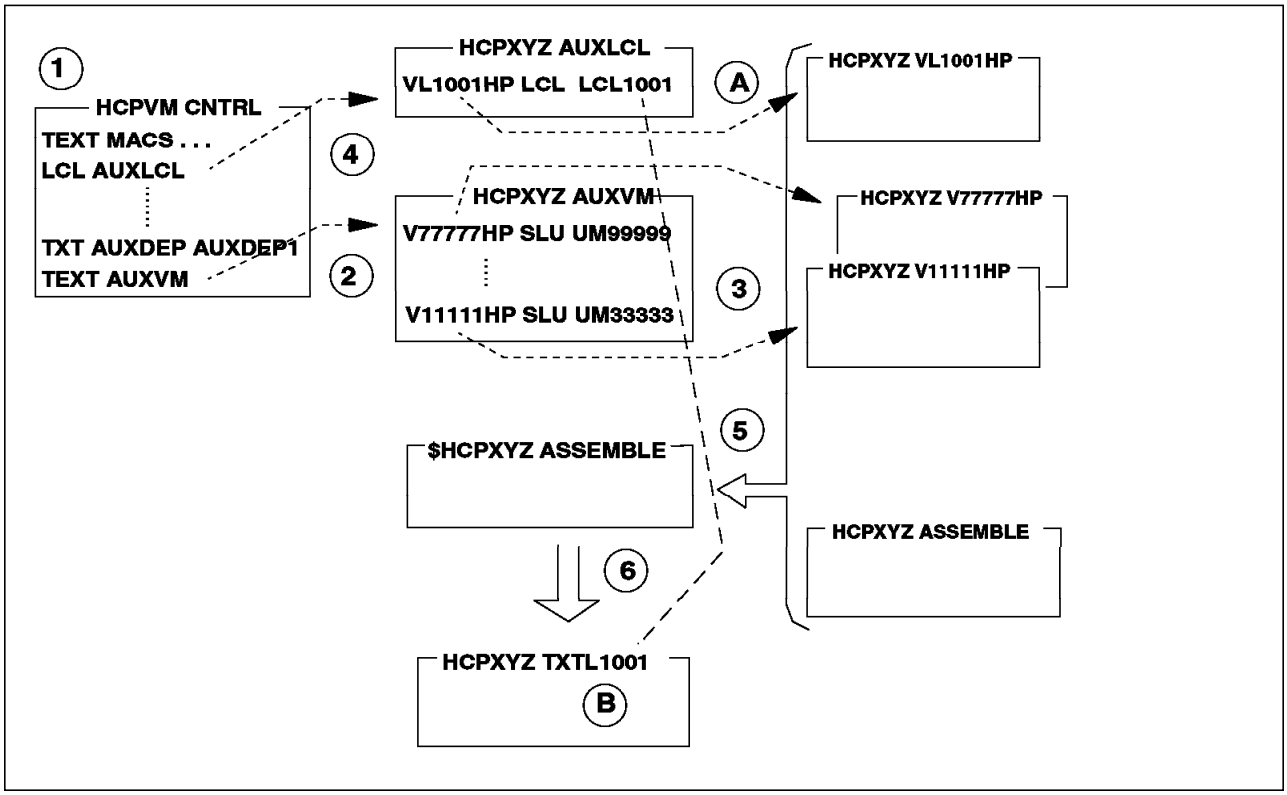



*Figure 47. Service Control File Structure*

In this example, the CP part HCPXYZ ASSEMBLE is to be updated. The sequence of events might be (points A and B are described later):

1. The main control file for CP, HCPVM CNTRL, is located. The control file points to both AUX files and VVTs, and is used by all parts of a product. The AUX files are individualized by part. Their file names, as well as the file names of the source updates, coincide with the part's file name. This file is processed from the bottom up, so entry AUXVM is processed first.
2. The file HCPXYZ AUXVM is located, and as there are no preferred AUX files, this file is used.
3. The individual updates in the AUX file are applied, from the bottom up, to a temporary copy of the source file. So, HCPXYZ V11111HP is applied first.
4. If other AUX files listed in the CNTRL file exist, steps 2 and 3 are repeated for each one. Notice that the second AUX file (AUXDEP) has a preferred file (AUXDEP1). If AUXDEP1 exists then this update level (AUXDEP) is ignored. In the example, none exists for the part HCPXYZ.
5. The temporary source file with the updates applied, $HCPXYZ ASSEMBLE, is now ready to be assembled.
6. VMFHASM generates the part from the temporary file.

VMSES/E includes three commands, VMFASM, VMFHASM, and VMFHLASM, that apply the updates (by invoking the CMS UPDATE command) and assemble the resulting updated source file, by invoking the ASSEMBLE, HASM, or HLASM commands. If necessary, they also unpack the original source file. The temporary updated file is always discarded after the assembly completes.

Which file type should be given to the newly generated TEXT file? Before VMSES/E, the VMFASM and VMFHASM commands named it by using the *level identifiers* in the control file. The first token (levelid) from the line of the highest applied AUX file is used:

- If the token is TEXT, the resulting file type is TEXT.
- If, as in the example, it is not TEXT, it is appended to the keyword TXT.

So, our example would result in a file type of TXTLCL.

But in VMSES/E all parts **must** be PTF-versioned. This allows immediate recognition of the service level for the part. VMSES/E defines a modified file type for the part. It is composed of a level identifier and a PTF (or local modification) number. On Figure 47 on page 118 you can see that:

- A. characters 3-7 (the local tracking number) from the file type of the last source update applied are appended to
- B. the default level identifier (TXT)

In the example, the resulting file type is TXTL1001. These modified rules are used by VMFASM, VMFHASM, VMFHLASM, VMFNLS, and VMFEXUPD.

**Note:** This is a subset of the rules. See *VM/ESA: VMSES/E Introduction and Reference* for the complete rules.

When VMFBLD selects parts to build objects, it calls VMFSIM GETLVL to obtain the file type of the highest version for the part. VMFSIM GETLVL uses the main control file and searches the VVTs for each defined level. There is, at most, one VVT per level. Each VVT can have many lines, each line corresponding to a

part. Each line is the equivalent of an entire AUX file. It contains the numbers of all PTFs applied to the part.

## AUX Files and VVTs

AUX files can support only source updated parts. This contrasts with Version Vector tables, which provide service control information for all types of parts. However, several CMS commands, such as UPDATE, EXECUPDT, and XEDIT, cannot use VVTs, so AUX files must still be provided. The VMSES/E functions VMFASM, VMFHASM, VMFHLASM, GENCPBLS, VMFNLS, and VMFEXUPD use the update facility to generate serviceable parts that are subsequently used for object building.

When building an object, selecting the correct level of parts can be based exclusively on VVTs. This is a uniform method for all types of parts. Doing it, however, implies that the VVTs are kept current, which is done automatically by VMSES/E for any service it applies, but remains a manual task for any local service.

For TEXT deck selection only, in the event VVTs are not available, VMFBLD uses AUX files.

## Version Support for Parts

Version support for parts is required when several versions of a product can be generated, and the same file name for a part with different contents is used on all the versions. Examples of this situation are:

- Uniprocessor and multiprocessor CP modules in VM/ESA Release 1.5 370 Feature.
- A product (called dependent) modifies code of another product (called base). This is the case of modifying CP with an External Security Manager (ESM).
- National Language Support (NLS) requires concurrent existence of several parts with the same file name.
- Several systems (or copies of a product) are maintained using the same base code, but some parts are customized for each system (or product's copy).

Version support for parts allows distinguishing between the different environments by assigning representative file types to each version of a part (and keeping a common file name).

This support must be provided for all types of parts. As we have seen in the preceding discussion, the existing control file structure, even with preferred AUX files support, only works for TEXT files.

A mechanism that allows the build function to correctly select the desired part for *all* types of parts is required. This mechanism is shown in Figure 48 on page 121.

Without version support, the file type for the part is based on the 3-character identifier for the part and the highest update identifier, as determined through the main control file and VVTs.

Version support allows modification of the file type by defining a conversion table, known as the control file extension. The table's file name is the same as the main control file, and the file type must be CNTRLEXT.

The conversion mechanism is as follows:

If a VVT for the highest applied level exists, and the VVT has an entry for the part, then:

1. The control file extension is searched for a line beginning with two tokens: the level identifier and the part's file type abbreviation.
2. If such a line exists, the new file type abbreviation is the third token on the line. It is used along with the update identifier (PTF number or local modification number) to create the part's file type.

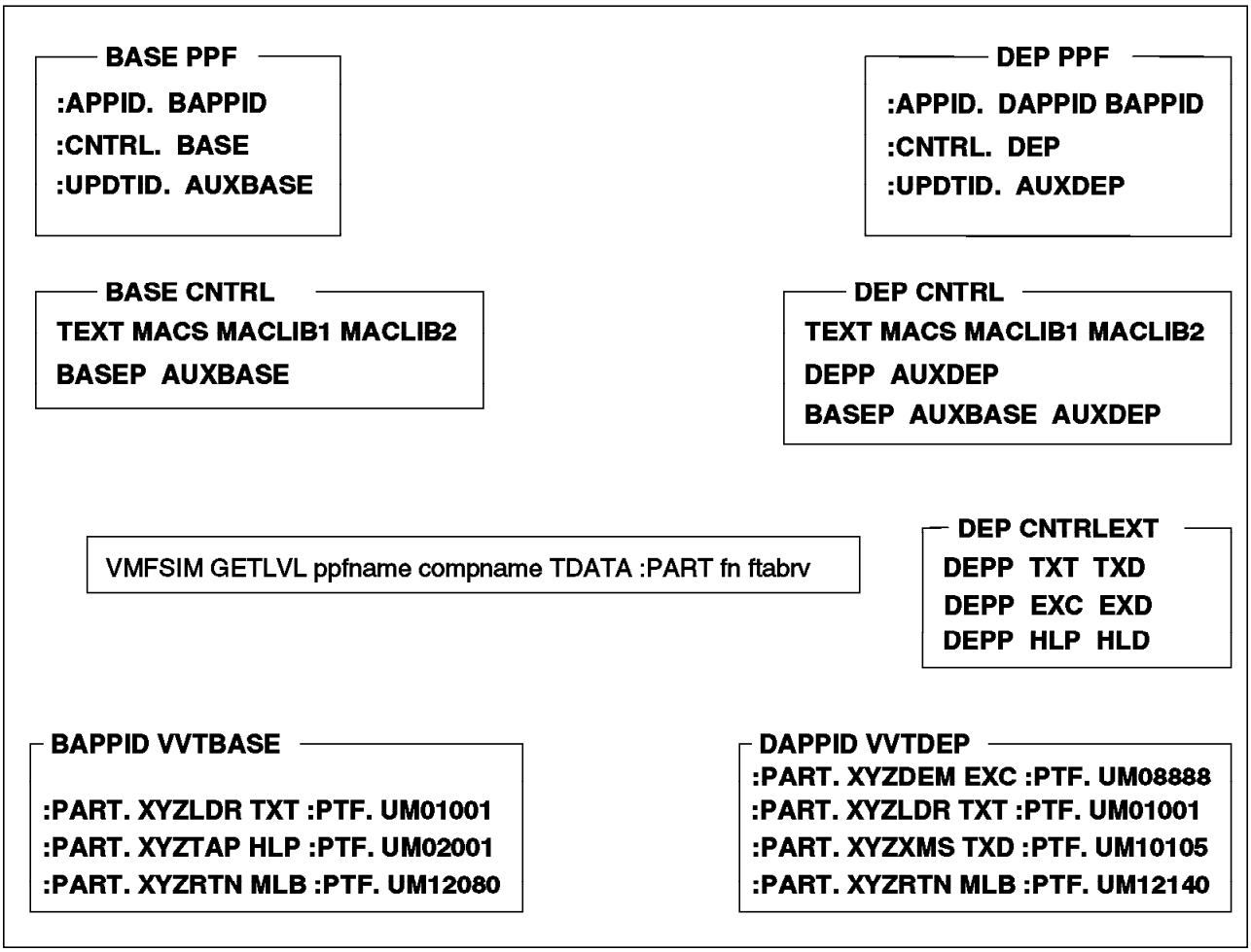



*Figure 48. Version Support for Parts*

Using Figure 48 as example, issuing the following command generates the return values in Table 6 on page 122:

```
vmfsim getlvl ppfname compname tdata :part fn ftabrev
```

| Table 6. Control File Extension Usage | | | |
|---|---|---|---|
| **fn/ftabrev** | **ppfname** | **Returned Value** | **Comments** |
| XYZLDR TXT | BASE | XYZLDR TXT01001 | |
| XYZLDR TXT | DEP | XYZLDR TXD01001 | used DEP CNTRLEXT |
| XYZTAP HLP | BASE | XYZTAP HLP02001 | |
| XYZTAP HLP | DEP | XYZTAP HLP02001 | no entry in VVTDEP |
| XYZXMS TXD | BASE | XYZXMS TEXT BASE-FILETYPE | no entry in VVTBASE |
| XYZXMS TXD | DEP | XYZXMS TXD10105 | no entry in DEP CNTRLEXT |
| XYZRTN MLB | BASE | XYZRTN MLB12080 | |
| XYZRTN MLB | DEP | XYZRTN MLB12140 | no entry in DEP CNTRLEXT |

## Local Service

Local service was briefly discussed in “Local Service” on page 26. The example in this section shows you how to create a local modification (localmod) to a source maintained part, and how you can update the Software Inventory to reflect your modification.

We also discuss converting the localmod into a “real” PTF, and we point out some of the differences between local and COR service.

Additional examples, covering local modifications to the CP nucleus build list and the CMSINST segment, can be found in *VM/ESA: Service Guide*.

## Creating a Local Update

As an exercise, let us presume it has been decided by the installation that the CP LINK command should not be generally available to the users. It is most likely that you would chose the CP OVERRIDE command to perform this task, but for this demonstration we will do this within the CP nucleus as a local modification.

This kind of change would typically be implemented as a local update.

### Creating the AUX and Update Files

In this example, the part to be modified is HCPCOM ASSEMBLE, the CP command list.

To set up the correct disk access order for CP issue the following commands (accessing the SESDISK and SIDISK should be in the PROFILE EXEC):

```
access 5e5 b/b
access 51d d/d
vmfsetup esa cp
```

This ensures that we have access to all the files needed (including the VMSES/E code). We are using the ESA PPF. You should use your own override, if you have one.

You will have to assign a local tracking number to this modification. Let us assume the number is L0001.

First, create the AUX file for the local modification, or add an entry to the local AUX file, HCPCOM AUXLCL, if it already exists. The AUX file name is the same as the file we will modify, HCPCOM. The AUX file type is defined by the LCL entry in the main control file. The file name of the main control file for the CP component, HCPVM CNTRL, is defined in the PPF for CP.

You have to edit the HCPCOM AUXLCL file and insert the single data line, as shown in Figure 49. There is just one record, because we have only one localmod.

```
VL0001HP LCL LCL0001  *CHANGE CP LINK FROM CLASS=G TO CLASS=X
```

*Figure 49. AUXLCL File for HCPCOM Update*

The record contains several one-word fields, followed by a comment. The fields, from left to right, are:

**Updft** File type of the update file. With standard service, this is a composite of the local tracking number and the service level indicators (defined by the :SLVI. tag of the PPF). This allows you to quickly associate the modification with a component and its service level, and its the method we recommend. However, some installations prefer to use more of an English file type to better distinguish local modifications.

**Lvl** Service level indicator, LCL for our case.

**Lclmodid** Local modification identifier; the assembled text deck is assigned a file type of TXTL0001

**Note:** Naming conventions for the generated file type of the resulting text file are different from the previous VMSES. See *VM/ESA: VMSES/E Introduction and Reference* for the complete description of these conventions.

**Comment** Verbal description of the update.

Next, the actual update must be created. Use XEDIT, with the control option, on the original source file. Figure 50 shows the line in the HCPCOM ASSEMBLE file that has to be modified and a few lines around it.

```
*************************************************************
*                                                           *
*  LINK COMMAND                                             *
*                                                           *
*************************************************************
                                                    SPACE ,
         COMMD COMMAND=(LINK,4),CLASS=G,EP=HCPLNKIN,
               IBMCLASS=G,FL=CMDONLY,FL1=CMDVPROT+CMDPROC+CMDVMAC,     *
               SECFLAG=CMDPROT
                                                    SPACE ,
*************************************************************
```

*Figure 50. HCPCOM ASSEMBLE Extract*

The source files are on the BASE string (BASE3, MAINT 394 accessed as Q) in packed format. The XEDIT command unpacks the source file as it loads it into storage. The command to use is:

**xedit hcpcom assemble (ctl hcpvm**

You will get a warning message saying that PTF file HCPCOM VL0001HP is missing. Ignore that message, as you are in the process of creating this file.

Locate the LINK section in HCPCOM ASSEMBLE (see Figure 50), change CLASS=G to CLASS=X, and issue the FILE command (without any parameters). The result will be a file named HCPCOM VL0001HP, on your A-disk, as shown in Figure 51.

```
./ R 00687160          $ 687950 790                  09/26/91 04:24:14
         COMMD COMMAND=(LINK,4),CLASS=X,EP=HCPLNKIN,                    *00687160
```

*Figure 51. Update File HCPCOM VL0001HP*

We now have to move the files we created to the correct minidisks, in this case the LOCALMOD disk (MAINT 2C4, accessed as E):

```
copyfile hcpcom auxlcl a = = e (oldd rep
copyfile hcpcom vl0001hp a = = e (oldd rep
```

Erase these files from your A-disk after they have been copied.

## Assembling the Updated Source File

The VMFASM, VMFHASM, and VMFNLS commands (in Release 1), VMFHLASM command (in Release 2) and the VMFEXUPD command (in Release 2.2) were enhanced with two options:

**CKGen** Validates the AUX files against the VVTs

**NOCKGen** Does not perform the validation

The following options were added in Release 2.1:

**LOGMOD** Validates the AUX files against the VVTs, and, if a mismatch is detected, automatically updates the local VVTs.

**FILETYPE** Provides a file type different from the default one for the output file that is created.

And the options below were added in Release 2.2:

**OUTMODE** Indicates where the generated output file should be placed.

**$SELECT** Indicates whether to update the $SELECT file.

Note that the combined result of the OUTMODE, LOGMOD, and $SELECT options is equivalent to the Apply step. VMFBLD will have the information that is required to detect that an object was changed by a local modification.

On VM/ESA Release 2.2, issue:

```
vmfhasm hcpcom esa cp ( logmod $select outmode localmod
```

On previous releases you would follow the method discussed below. It is still instructive to read it, because it shows what happens “under the covers.”

On VM/ESA Release 2 and previous systems, to produce an updated text deck issue:

```
vmfhasm hcpcom esa cp
copyfile hcpcom txtl0001 a = = e (oldd rep
erase hcpcom txtl0001 a
```

The VMFHASM command automatically unpacks the source file, if necessary.

So far we have produced an update, an AUX file, and an updated text deck. Furthermore, these files have been moved to the correct minidisks, as defined in the PPF.

But the Software Inventory has no record of the localmod. Until VMSES/E Release 2.1 you had to manually update the Software Inventory, starting with the Version Vector table. The command was:

```
vmfsim logmod 6vmvmb22 vvtlcl e tdata :part hcpcom txt :mod lcl0001.vl0001hp
```

Note that the local service level VVT (VVTLCL) is used, not the system level (VVTVM).

On a VM/ESA Release 2.1 system, issue:

```
vmfhasm hcpcom esa cp ( logmod
copyfile hcpcom txtl0001 a = = e (oldd rep
erase hcpcom txtl0001 a
```

When the CP nucleus is re-built, the HCPCOM TXTL0001 text deck will be used and, after an IPL, the LINK CP command will be class X.

Do not forget to include the Class X in the directory entry of authorized users.

## Comparing Local and Corrective Service

As you noticed in the example, before VM/ESA Release 2.1 everything had to be done manually. We had no help from VMSES/E whatsoever. By now we know why. The $PTFPART parameter file was missing, and without it VMSES/E cannot do any work.

So let us try to create a $PTFPART file. For a moment, imagine that you are working in the VM/ESA Release 2.2 change team. You have been assigned the task of correcting an error: the CP LINK command is class G and it should have been class X, by default.

The incident has been reported, and assigned APAR number VM12345. We now create the fix to this problem, which is what we actually did in the previous section. Let us give the fix the PTF number UM98765.

Shown below is a list of the files that must be on the COR tape:

| File name | Description |
|---|---|
| HCPCOM TXT98765 | Updated text deck - identical to HCPCOM TXTL0001 |
| HCPCOM V12345HP | Update file - identical to HCPCOM VL0001HP |
| Apply/Exclude lists | List of PTFs to be applied or excluded |
| UM98765 $PTFPART | Definition and control information for VMSES/E |

We have to create the PTFPART file manually. The resulting file, UM98765 $PTFPART, is shown in Figure 52 on page 126.

```
** COMPID FOR PTF UM98765 = 568411202, RELEASE = 122, PRODID = 6VMVMB22
:PTF.UM98765
**************************PRODUCT IDENTIFIER**************************
:PRODID.6VMVMB22
**************************APAR DESCRIPTIONS***************************
:APARDESC.
:APARNUM.VM12345
:ABSTRACT.WRONG CLASS FOR CP LINK COMMAND
:EAPARDESC.
* THIS PTF HAS NO REQUISITES
*******************************PARTS LIST*******************************
:PARTS.
**  *  *  *  *  *  *  *  *  *  *  *  *  *  *  *  *  *  *  *  *  *  **
:PARTDEF.HCPCOM   ASSEMBLE
:PROCOPTS.AUX
:REPPART.HCPCOM TXT98765
:UPDATES.V12345HP
:APARS.VM12345
**  *  *  *  *  *  *  *  *  *  *  *  *  *  *  *  *  *  *  *  *  *  **
:EPARTS.
```

*Figure 52. PTFPART File for PTF UM98765*

We now have to package the PTF in COR tape format, and ship it to the customers.

In our example, we did not bother to produce the tape. So, to “receive” the PTF, we just copied the files to the alternate DELTA disk (remember to use the VMFCOPY command, to get the VMSES PARTCAT updated).

## Receiving Manually

“Manually” means that the service-level Software Inventory for the receive step is not updated automatically. We must enter the following command to update the Software Inventory:

**vmfsim init 6vmvmb22 *** fm **tdata :ptf um98765**

where fm is the access mode of the Alternate (or only) DELTA disk. The command above is equivalent to the following command sequence:

1. Record that UM98765 has been received:

   **vmfsim modify 6vmvmb22 srvrecs tdata :ptf um98765 :stat received (add**

2. Update both the Description and Requisite tables. We use a query to map the PTF to a file, TEMP SIMDATA, that serves as input for the tables update:

   **vmfsim query um98765 $ptfpart tdata :ptf um98765 (file temp**

3. Update the Description table:

   **vmfsim modify 6vmvmb22 srvdesct file temp**

4. Update the Requisite table:

   **vmfsim modify 6vmvmb22 srvreqt file temp**

The receive step for a COR tape has now been simulated, and we are ready to apply the service, by using the VMFAPPLY command. VMSES/E will generate AUX files, and update all the service-level tables.

Note that the UPDTID option of VMFAPPLY allows you to specify the file type of the AUX file that VMFAPPLY will create. This option overrides the value specified in the PPF, which has the default of AUXVM.

If you wish to use VMFAPPLY for your local changes, you may consider using this option to keep your changes in separate AUX files.

**Note:** If you do not update the Receive Status table, you will not be able to perform the Apply step. VMFAPPLY will tell you that UM98765 cannot be applied because it has never been received, and terminate.

### Build IBM Service before Reworking your Local Service

Sometimes you might need to invoke VMFBLD to build IBM service before reworking your own local service affected by the IBM service. Several examples are:

1. If IBM services a macro, HCPXXX in HCPOM1 MACLIB, which is used in an ASSEMBLE file for which you have a local modification, you will need to rebuild the MACLIB before you reassemble your files.

   ```
   vmfbld ppf esa cp hcpom1 hcpxxx (serviced
   ```

2. If IBM ships new source code for a part, for example HCPABC ASSEMBLE, for which you have a local modification, you will need to rebuild the source part before you rework you local modification.

   ```
   vmfbld ppf esa cp hcpblsrc (serviced
   ```

3. If you have local modifications to HCPOM1 MACLIB and IBM ships service to the MACLIB which includes a new macro (for example, HCPVUBLK) for the MACLIB, you are instructed to re-build the MACLIB as part of re-working your local service. This might fail since VMFBLD tries to add the new macro to the MACLIB and cannot find the source file for the macro. In this case, you need to first invoke VMFBLD just to create the source macro.

   ```
   vmfbld ppf esa cp hcpblsrc (serviced
   ```

## Updating the CP Nucleus Build List

The procedure described here cannot be followed to change the CP nucleus on a VM/ESA Release 1.5 370 Feature system since the CPLOAD is serviced by replacement only.

The CP nucleus build list for VM/ESA, CPLOAD EXEC, is an ordered listing of the text decks in the CP nucleus. If an optional program product requires that you add a text deck to the CP nucleus you will have to manually update the CP build list.

The *VM/ESA: Service Guide* contains an appendix describing the update procedure. Therefore, here we will only comment on the process and add any information we feel is necessary to clarify the process. As the CP load list is often changed by service, it is *very important* that you follow the documented procedure, in order to make VMSES/E aware of your changes.

**Note:** In this section, the term “CP module” is used to refer to a text deck that is included in the CP nucleus. It should not be confused with the CMS module file that holds the complete generated CP nucleus.

## Overview

The CP module order is determined by the HCPMDLAT MACRO (MoDuLe ATtribute), which is invoked by the HCPLDL ASSEMBLE part. When the part HCPLDL ASSEMBLE is assembled, a text deck with a PTF-numbered file type is created. This text deck is then converted into the CP nucleus build list (also called CP load list).

The HCPMDLAT macro lists the CP modules in the order in which they will be loaded by the build process. The CP modules are also grouped according to their attributes (categories). If you are adding a new module to the CP nucleus build list, be sure to add it to the HCPMDLAT macro in the appropriate category, and in the appropriate order within that category.

Figure 53 illustrates the categories within HCPMDLAT MACRO file. The HCPMMx markers identify the different sections within the macro.

```
HCPLDR        (* loader *)
   Resident Non-executable modules
HCPMM0
   Resident Non-executable modules
   Resident Executable MP modules
HCPMM1
   Resident MP modules
HCPMM5
   Resident Non-MP modules
HCPCPE
HCPMM4
   Pseudo-pageable Initialization Modules
   Fully-pageable Initialization Modules
HCPMM7
   Fully-pageable Initialization MP Modules
HCPMM2
   Pageable MP modules
HCPMM6
   Pageable Non-MP modules
SYS CPFORMAT
   CPFORMAT Modules
LDT Card
HCPGEN
HCPMM3        (* End of CP Modules *)
LDT HCPGENUC Card
```

*Figure 53. HCPMDLAT MACRO File Structure*

The module attributes are:

- Resident. These modules are non-pageable.
- Pseudo-pageable, loaded at initialization. These modules may not be paged out until system initialization has finished.
- Fully-pageable. These modules can be paged in and out of real storage.
- Multiprocessor. These modules are capable of executing simultaneously on two or more processors.

- Non-multiprocessor. These modules are only capable of executing on a single processor, the master processor, which is usually the IPL processor.
- Save area is either STATIC or DYNAMIC.
- Data only, as opposed to executable.

In addition to being used to produce the CP nucleus build list, the HCPMDLAT macro supplies necessary attribute information when one CP module calls another.

**Notes:**

- After changing the attributes of a module, reassemble all the modules that call the changed module so they can pick up the new attributes. According to the developers, the relationship between caller and called is not obvious and not documented anywhere in the VM/ESA publications.
- When changing a module from pageable to resident, keep it within the same category, MP or non-MP, and add it to the end of the respective list.
- If you are installing a product that modifies the CP nucleus build list, it is the responsibility of that product's installation instructions to describe what the attributes of the new modules are.

## Update Procedure for VM/ESA Release 2.2

To update the CP nucleus for VM/ESA Release 2.1 and previous releases refer to "Update Procedure for VM/ESA Release 2.1." To update the CP nucleus build list for VM/ESA Release 2.2, follow the procedure in *VM/ESA: Service Guide* and *VM/ESA: Service Guide, Appendix F*, then you can rebuild the CP nucleus.

## Update Procedure for VM/ESA Release 2.1

To update the CP nucleus build list for VM/ESA Release 2.1 and previous releases, follow the procedure in *VM/ESA: Service Guide*, using the steps below as a guideline:

**1** Modify the HCPMDLAT MACRO using the local modification procedure:

**a** Access the CP disks. Enter:

```
vmfsetup esa cp
```

**b** Apply local service to HCPMDLAT MACRO:

- Define a local tracking number for this modification. It has the form Lnnnn, where "nnnn" is a number, and we will refer to it as "modid."
- Create or update, on the LOCALMOD disk, an auxiliary control file, HCPMDLAT AUXLCL, that points to the local source update file. The single data line to insert is shown below:

```
SmodidHP LCL LCmodid  * Change CPLOAD list for ...
```

- Create the local source update file. Enter:

```
xedit hcpmdlat macro ( ctl hcpvm
```

  Be sure to place the new module in the appropriate order and category. File the changes.

- Copy the source update file just created to the LOCALMOD disk. Enter the commands:

```
copyfile hcpmdlat macmodid a = = fm-local
erase hcpmdlat macmodid a
```

where:

**fm-local** is the file mode of the LOCALMOD disk

- Record the change in the Software Inventory by entering:

```
vmfsim logmod 6vmvmb21 vvtlcl fm-local tdata :part hcpmdlat macro
       :mod lcmodid.smodidhp
```

- Create or update, on the LOCALMOD disk, an auxiliary control file, HCPLDL AUXLCL, that points to the local source update file. The single data line to insert is shown below:

```
SmodidHP LCL LCmodid  * Change CPLOAD list for ...
```

- Create the local dummy source update file. Edit the HCPLDL SmodidHP file and enter the single data line shown below:

```
./ * Force re-assemble for CPLOAD change ...
```

**c** Add the following records to 6VMVMB21 $SELECT:

```
:APPLYID. mm/dd/yy hh:mm:ss
HCPMDLAT MACRO
```

**Note:** HCPLDL TXT is not added.

**d** As documented. Use the commands (you may use VMFHLASM instead of VMFHASM):

```
vmfbld ppf esa cp hcpgpi ( serviced

vmfhasm hcpldl esa cp ( logmod

copyfile hcpldl txtmodid a = = fm-local
erase hcpldl txtmodid a
```

**2** Create the modified version of the CP nucleus build list, as documented.

**3** Modify CPLOAD EXEC. Read the whole step before doing.

**a** To record the change in the Software Inventory enter:

```
vmfsim logmod 6vmvmb21 vvtlcl fm-local tdata :part cpload exc :mod lcmodid
```

**b** Add the following records to 6VMVMB21 $SELECT:

```
:APPLYID. mm/dd/yy hh:mm:ss
CPLOAD EXC
```

**c** As documented in *VM/ESA: Service Guide*.

You can now rebuild the CP nucleus.

As all local modifications are logged in the Software Inventory, every time service is applied to the CP load list you will receive a warning from VMSES/E. You should then verify whether your changes are still valid and repeat the above procedure.

## Changing GCS

There are three common changes that you may wish to make to your GCS system:

- Alter the load address of the GCS system
- Have several GCS systems
- Change the name of the GCS Saved System

You will find that *VM/ESA: Installation Guide* has an appendix that describes these changes. The following section may help clarify the process. VM/ESA Release 1.5 370 Feature users may be able to adapt the process to their environment.

## Changing the Load Address

As in the past, you have to copy the IBM supplied load list for GCS and edit it, in order to change the values for the Set Loader Counter (SLC) entries.

We recommend that you copy the most recent IBM supplied version of the file to your LOCALMOD minidisk and make all changes to your own copy of the file. You can find the file id of the most recent version by looking in the Version Vector table (6VMVML22 VVTVM).

The SLC entries determine the virtual address range that GCS resides in. If you change the entries in the load list, you have to create SLC files containing the addresses that you wish.

The problem with this method of copying and editing is that if the GCTLOAD EXEC is updated by service, you have to be aware of the change and repeat the copying and editing process.

Previously, there was no mechanism to advise you that the load list had been serviced. This made it difficult to know when to reapply the local changes.

With VMSES/E, you can log your modifications to the load list in the Software Inventory. The advantage to this is that if there are any service changes to the load list, VMSES/E detects them during the apply process, and reminds you to check for any impact the service may have had on your local modifications. Use VMFVIEW APPLY to look for messages VMFxxx2120W and VMFxxx2121I in the message log.

The steps you will have to perform to change the GCS load list, and log the changes in the Software Inventory are:

1. Access the GCS disks. Enter:

   **vmfsetup esa gcs**

2. Find the most recent IBM supplied version of the load list. Enter the following command (use gcs or gcssfs for *compname*):

   **vmfsim getlvl esa** compname **tdata :part gctload ( history**

   Any existing local modifications are listed first, after the tag :MOD, and IBM service is listed after the tags :VVTVM and :PTF. The most recent service will be in the replacement part “GCTLOAD EXCnnnnn,” where “nnnnn” is the PTF number of the *first* PTF listed after these tags. If there is no entry,

then you have no service applied and the file is the original "GCTLOAD EXEC."

3. Assign a local tracking number to this modification. This number must be of the form Lnnnn. We refer to it below as "modid."

4. Copy this file to the GCS LOCALMOD disk and perform any modifications. This file should be named GCTLOAD EXCmodid.

5. Create the SLC files, also on your local disk. Do not forget the X'02' characters in column one.

6. Add the local modification to the Software Inventory. Issue the command:

   **vmfsim logmod 6vmvml22 vvtlgct** fm-local **tdata :part gctload exc :mod lc**modid

   where:

   **fm-local** is the file mode of the LOCALMOD disk

**Note:** The file type of VVTLGCT is determined by the GCTVM CNTRL file.

You can now rebuild the GCS nucleus.

## Changing the Load List Name

You may want to have several load lists, or you may want to avoid using the IBM name of GCTLOAD EXEC. For every load list you create, you should also have an override file using that load list name, instead of modifying the ESA PPF. An example of such an override is shown in Figure 54.

```
*==============================================================
:OVERLST. GCS
:OVERLST. GCSSFS
*==============================================================
* End of Product Header
*==============================================================
:GCS. GCS 6VMVML22
:BLD. UPDATE
./INSERT GCTLOAD AFTER
TESTLOAD  VMFBDNUC BUILD7 TXT TXS * BUILD TEST GCS NUCLEUS
./END
:END.
*==============================================================
:GCSSFS. GCSSFS 6VMVML22
:BLD. UPDATE
./INSERT GCTLOAD AFTER
TESTLOAD  VMFBDNUC BUILD7 TXT TXS * BUILD TEST GCS NUCLEUS
./END
:END.
```

*Figure 54. PPF Override - TESTGCS $PPF*

This override file changes the load list name, used when the GCS nucleus is built, from GCTLOAD to TESTLOAD. There are two override areas because GCS can be installed either on minidisks (GCS) or in Shared File System directories (GCSSFS). Previously you would call VMFPPF twice, once for each override area, but starting with VM/ESA Release 2.2, you need to call VMFPPF only once:

**vmfppf testgcs gcs gcssfs**

to obtain the single, executable form, TESTCGS PPF file for both GCS variants.

You may want to log a dummy local modification, in effect just log a modification against the IBM-supplied load list in the Software Inventory. In this way, VMSES/E will warn you every time the load list is changed, so you do not forget to change your load list accordingly. Enter:

**vmfsim logmod 6vmvm122 vvtlgct** fm-local **tdata :part gctload exc :mod lc**modid

Once you log this modification, this dummy local modification ID will become the highest level of GCTLOAD instead of the IBM-serviced level. To make the levels the same, copy the highest IBM-serviced level of GCTLOAD to the LOCALMOD disk with a filetype of EXCmodid. This must also be done each time GCTLOAD is serviced.

## Changing the Saved System Name

To generate GCS you use the GROUP EXEC. Among other things, the GROUP EXEC allows you to define the name of the GCS saved system; the output of the EXEC is a GCS system configuration file in source format. You must than ASSEMBLE this file so that the resulting text deck can be included in the GCS nucleus. The GCS load list has an entry with the default name of this file.

If you wish to change the name of the GCS saved system, use the GROUP EXEC to define the requirements specific of your system, as well as a different name for the saved system. The GROUP EXEC creates a file with the file type GROUP and the file name equal to the name of the saved system (MYGCS, for example).

To include this file in the GCS nucleus you must first change its file type and then assemble it, to generate the TEXT deck. But the list of TEXT decks, which is the load list, has also to be changed, to point to your version. There are a number of manual steps to be performed, namely:

1. Access the GCS disks. Enter:

   **vmfsetup esa gcs**

2. Assign a “local tracking number” to this modification. This number must be of the form Lnnnn. We refer to it below as “modid.”

3. Copy the MYGCS GROUP file to the LOCALMOD disk, renaming it to MYGCS ASSEMBLE.

4. Create the corresponding AUX structure, the AUX file, shown in Figure 55, and the dummy source update file shown in Figure 56. Both files should reside on the LOCALMOD disk.

   **Note:** The file type AUXLGCT is determined by the GCTVM CNTRL file.

```
Vmodid GT LCL LCmodid - Local MOD to Force Text deck name to TXTLOCAL
```

*Figure 55. AUX File Example - MYGCS AUXLGCT*

```
./ * - Local MOD to get a new text deck name
```

*Figure 56. Update File Example - MYGCS VmodidGT*

5 Assemble the file using the correct PPF and component names, and log the modification on the Software Inventory. Enter:

**vmfhasm mygcs** ppfname compname **( logmod**

**Note:** If you are using Release 2.2, you could use the OUTMODE LOCALMOD option to VMFHASM, then skip the next step.

6 Copy the resulting text deck to your local disk, using the local tracking number to rename it:

**copyfile mygcs txt**modid **a = =** fm-local
**erase mygcs txt**modid **a**

7 Create your version, named TESTLOAD EXCmodid, of the GCS load list, as explained in "Changing the Load List Name" on page 132. Use the procedure explained in "Changing the Load Address" on page 131 to change the line containing:

```
&1 &2 &3 GCS
```

to:

```
&1 &2 &3 MYGCS
```

8 Also, record in the Software Inventory that you have a local version of the load list, called TESTLOAD. The command format would be:

**vmfsim logmod 6vmvml22 vvtlgct** fm-local **tdata :part testload exc :mod lc**modid

At this point, you can use the override file shown in Figure 54 on page 132 to build a second copy of GCS called MYGCS. The command format is (if you installed GCS in the Shared File System):

**vmfbld ppf testgcs gcssfs testload ( all**

Even though the files for your test version of GCS are not part of VM/ESA Release 2.2, you should log the changes in the Software Inventory, in order to maintain complete records of the status of your system.

# Chapter 7. Exploring the Software Inventory

The Software Inventory was introduced in Chapter 3, “Software Inventory” on page 37. It is heavily used by VMSES/E during the install and service processes. IBM provides you with a general and very powerful command to interrogate and manipulate the Software Inventory: VMFSIM.

Though not formally a part of the Software Inventory, build lists are very important, and Release 2 introduced the VMFQOBJ command to enhance their exploitation. VMFINFO, a full-screen, panel-driven interface that provides a more friendly interface to both VMFSIM and VMFQOBJ, was also introduced in Release 2. This chapter describes the use of these commands.

Many examples use PTFs and service-level tables from VM/ESA Release 1.1 and VM/ESA Release 2 because at the time of writing there were no PTFs for VM/ESA Release 2.2. However, since the PTFs are used solely to illustrate the use of VMSES/E functions and capabilities, and not as actual service information, we felt this should pose no problem.

The diskette that accompanies this document contains sample source code for some of the examples referred to in this chapter and described in Appendix D, “VMFSIM Exploitation Code Examples” on page 225. Refer to Appendix E, “Diskette Installation Instructions” on page 233 for information on installing the files contained on this diskette.

## VMFSIM Subcommands

Through its many subcommands, VMFSIM allows exploitation of all the table types and files shown in Table 7. The VMFSIM command is used internally by VMSES/E, but some subcommands, QUERY in particular, are also very useful to you, the systems programmer, in your daily work. VMFSIM subcommands are listed in Table 8 on page 136, with a short description of their function. For the complete command syntax, please refer to *VM/ESA: VMSES/E Introduction and Reference*.

*Table 7. Table Types and Files Used with VMFSIM*

| System-level Tables | Service-level Tables | Miscellaneous Files |
|---|---|---|
| SYSREQT | SRVREQT | $PPFTEMP |
| SYSDESCT | SRVDESCT | PPF |
| SYSRECS | SRVRECS | $PTFPART |
| SYSAPPS | SRVAPPS | PRODPART |
| SYSBLDS | SRVBLDS | PARTCAT |
| SYSABRVT | VVTxxx | SEGDATA |

| *Table 8. VMFSIM Subcommands* | |
|---|---|
| **Subcommand** | **Description** |
| INIT | Updates a table with initial information for a product or PTF |
| COMPTBL | Compares two tables of the same type and lists differences |
| CHKLVL | Checks control/AUX structure for a module versus the Version Vector table |
| GETLVL | Returns the highest PTF level of a part |
| LOGMOD | Adds, deletes, or updates local modification data in a table |
| MODIFY | Adds, deletes, or updates specific data from the Software Inventory |
| QUERY | Searches a table for specific tags and corresponding data |
| SYSDEP | Searches a requisite table for all dependencies of a given product |
| SRVDEP | Searches a requisite table for all dependencies of a given PTF |
| SYSREQ | Searches a requisite table for all requisites for a given product |
| SRVREQ | Searches a requisite table for all requisites for a given PTF |

## VMFSIM Queries

Before we show you some examples, let us look at the syntax of the VMFSIM QUERY command, shown in Figure 57 on page 137, which we will explore here. First you have to specify the file ID of the table you wish to query. Then you specify “querydata,” that is, your question. This may be in several forms:

- The TDATA operand followed by a sequence of tags, with or without values. You may specify many sets of TDATA and tags, each comprising a query.
- The keyword FILE and a file name (or file ID). This file contains the TDATA tags with the same syntax as described above. You may also combine the FILE parameter with TDATA tags information supplied in the command line.
- The keyword STEM and a stem name. Again, the stem variables contain TDATA tag syntax.
- The keyword ASTEM and an associative stem name.

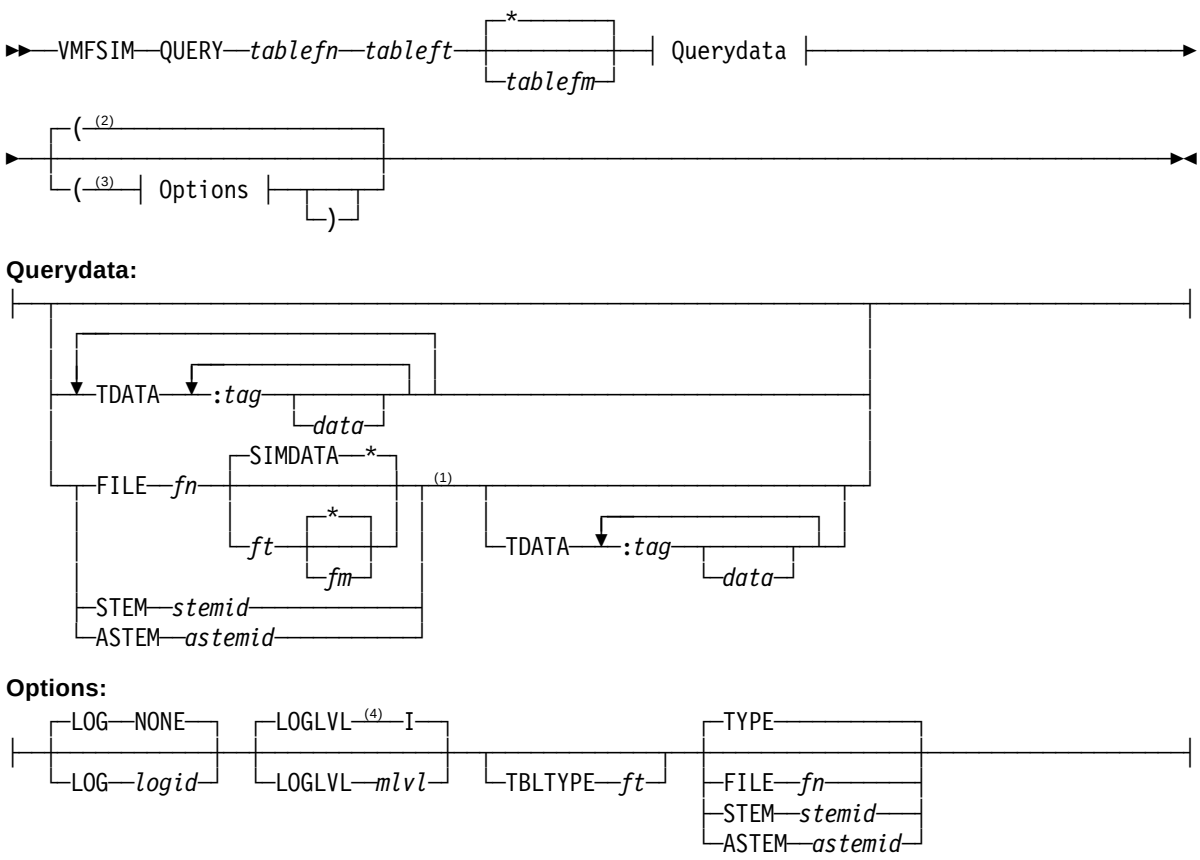
The output can be presented in the same four forms as the input:

- On the terminal
- In a file
- In a stem
- In an associative stem (ASTEM)

When using the FILE and STEM output options, if the output exceeds a screen line it is split into several lines, as it would be when displayed on the terminal.

With the ASTEM output option, this does not happen. The ASTEM option presents the output in a type of tree structure. In this case, you will have to code some additional statements to get to the data you need, but then all the data for one tag is contained in one single variable. The ASTEM structure is described in *VM/ESA: VMSES/E Introduction and Reference*.

You may explore any of the tables listed in Table 7 on page 135, using VMFSIM. Let us start out with SYSREQT, the system-level requisite table.




**Notes:**

1. One extra TDATA statement is allowed following the input file or stem specification.
2. The defaults you receive appear above the line in the Options fragment.
3. You can enter options in any order between the parentheses.
4. The LOGLVL option is ignored if you specify LOG NONE or allow LOG to default to NONE.

*Figure 57. VMFSIM QUERY Command Syntax*

## Displaying Table Fields (Tags)

To see the valid (defined) tags for a table, specify only the file ID for the table:

**vmfsim query vm sysreqt**

The reply will look like:

```
:PRODID - PRODUCT REQUISITE TABLE (KEY)
   :PREREQ - PRE-REQUISITE PRODUCT(S) (FIELD)
   :REQ - CO-REQUISITE PRODUCT(S) (FIELD)
   :DREQ - CO-REQUISITE PRODUCT(S) (FIELD)
   :IFREQ - IF-REQUISITE PRODUCT(S) (FIELD)
   :SUP - SUPERSEDED PRODUCT(S) (FIELD)
   :NPRE - NON-COMPATIBLE PRODUCT(S) (FIELD)
```

and describes the meaning of each tag field in the table.

As you may remember from Chapter 3, “Software Inventory” on page 37, each table has one type of data structure, with one key field, and several data fields. The above example illustrates this. Look at the field classifications at the end of each line.

But this structure also has another implication. When you query the Software Inventory tables with the VMFSIM command, you will always get the key field and at least one sub-field. In fact, if you specify only the key tag, you will get the entire structure. If you specify one or more field tags, you will get the key data and the data for the listed fields.

There is no way you can get only one type of tag field as a reply.

## Displaying Field Values

Now we can go one step further, and look at all existing values of one tag. This is done by just listing the tag without a value, as shown below:

```
vmfsim query vm sysreqt tdata :prodid :req
```

The reply lists all occurrences of the :PRODID tag, and the underlaying tag :REQ. This is done here to limit the amount of output, as described above.

```
VMFSIP2480I Results for
            TDATA :PRODID :REQ
:PRODID 6VMVMK22%VMSES
   :REQ
:PRODID 6VMVMB22%CP
   :REQ
:PRODID 6VMVMI22%DV
   :REQ
:PRODID 6VMVMA22%CMS
   :REQ
:PRODID 6VMVMF22%REXX
   :REQ
:PRODID 6VMVML22%GCS
   :REQ
:PRODID 6VMVMH22%TSAF
   :REQ
:PRODID 6VMVMD22%AVS
   :REQ
```

## Displaying Component Information

To look up all the information for one component (CMS, in the example) enter:

```
vmfsim query vm sysreqt tdata :prodid cms
```

The results are as follows:

```
:PRODID 6VMVMA22%CMS
   :PREREQ 6VMVMK22
   :REQ
   :DREQ 6VMVMF22
   :IFREQ
   :SUP 6VMVMA11 6VMVMA20 6VMVMA21
   :NPRE
```

The meaning of the tags is described in “Definitions and Terms” on page 10.

## Displaying Selected Fields

The next example shows usage of the service-level Software Inventory. We want to know which PTFs have been applied to the CP component. We issue the command:

**vmfsim query 6vmvmb11 srvapps * tdata :ptf :stat applied**

The output from the query is as follows:

```
VMFSIP2480I Results for
            TDATA :PTF :STAT APPLIED
:PTF UM18657
   :STAT APPLIED.12/18/92.13:36:11.MAINT
:PTF UM18667
   :STAT APPLIED.12/18/92.13:36:11.MAINT
:PTF UM18681
   :STAT APPLIED.12/18/92.13:36:11.MAINT
:PTF UM18690
   :STAT APPLIED.12/18/92.13:36:11.MAINT
```

This example, as well as the previous one, illustrates how flexible the search for matching tag values is. Note that the value of :STATUS is composed of tokens separated by the "." character, and compare the value of the :PRODID tag, in the previous example, to the search value given. The tokens can also be separated by blanks, so if you issue the command:

**vmfsim query 6vmvma22 srvdesct tdata :abstract program exception**

you will get a list all CMS APARs in whose description both the words "program" and "exception" appear. This is quite powerful.

## Combining Table Information

Assume we want to know the apply status of the received APARs. This is a more complex query, which shows an advanced function of the VMFSIM command, namely the ability to process a query in two steps.

The information is not directly available. The service-level Apply Status table (SRVAPPS table) contains status information about PTFs, not APARs. First, we have to find the connection between the APAR number and the PTF number. This information is found in the service-level requisite table (SRVREQT table). We start by asking for the APAR numbers, with accompanying PTF numbers:

**vmfsim query 6vmvmb11 srvreqt tdata :aparnum ( file temp**

This creates a file called TEMP SIMDATA. This file is in a format that can be used directly for input to the VMFSIM QUERY command. Figure 58 shows an excerpt from the TEMP SIMDATA file.

```
TDATA
:PTF UM18650
   :APARNUM VM48164
TDATA
:PTF UM18651
   :APARNUM VM48165
```

*Figure 58. TEMP SIMDATA File (Excerpt)*

Here, all lines starting with TDATA, up to the next TDATA (or end-of-file) are concatenated and used as TDATA input for VMFSIM. So, next we issue the command:

**vmfsim query 6vmvmb11 srvapps file temp**

This command provides the following (abbreviated) output:

```
VMFSIP2480I Results for
            TDATA :PTF UM18650 :APARNUM VM48164
:PTF UM18650
   :STAT SUPED.12/18/92.13:36:11.MAINT

VMFSIP2480I Results for
            TDATA :PTF UM18651 :APARNUM VM48165
:PTF UM18651
   :STAT SUPED.12/18/92.13:36:11.MAINT
```

The real information is in message VMFSIP2480I, which lists the APAR number, followed by the corresponding PTF number and its status.

The last example shows that you should take some care when combining tables. The VMFSIP2446I message tells you that not all tags on the TDATA statement are defined in the SRVAPPS table. To combine tables, they must have at least one field in common. Table 1 on page 41 and Table 2 on page 42 show the field contents in the system-level Software Inventory and service-level Software Inventory tables respectively, and can be used as guidance when you want to combine tables.

If you do not want to see the VMF messages, you may issue the command:

**cp set emsg off**

## Other Queries

Not all the queries you may want to make can be made using the basic VMFSIM command syntax. The main reason is that, even if you ask for output to a file, or a REXX variable stem, the output will not always be in a format that can serve as input to VMFSIM. This occurs, for instance, when a field has a composite value (contains more than one token). Then VMFSIM would not, in most cases, find any match. Let us look at an example.

We want to find the description of all the APARs in a given PTF. The APAR descriptions are in the SRVDESCT table, and the information linking the PTF and APAR numbers is found in the SRVREQT table.

So, first we have to look in the SRVREQT table to get the APAR numbers. We can issue the command:

**vmfsim query 6vmvmb11 srvreqt * tdata :ptf** ptfnum **:aparnum ( file temp**

This produces the output file TEMP SIMDATA, which we will feed into the next query against the SRVDESCT table, to get the APAR descriptions. We can issue the command:

**vmfsim query 6vmvmb11 srvdesct * file temp**

Figure 59 on page 141 shows the results of the query when there is only one APAR in the PTF. After some messages about unknown fields, VMFSIM produces the description we asked for.

```
VMFSIP2480I Results for
            TDATA :PTF UM18652 :APARNUM VM48166
:APARNUM VM48166
   :ABSTRACT USER WITH CLASS D AND G AUTHORITY CANNOT PURGE
            ANOTHER USER'S TRF FILES
```

*Figure 59. Result of VMFSIM Query with One APAR for Given PTF*

Figure 60 shows the results of repeating the two queries for a PTF containing two APARs.

```
VMFSIP2481W NO entries match search arguments
            TDATA :PTF UM18777 :APARNUM VM48192 VM48191
            in table 6VMVMB11 SRVDESCT J1
```

*Figure 60. Result of VMFSIM Query with Two APARs for Given PTF*

Here, there are the same messages about unknown fields, but no match at the end, because there is no entry that contains two APAR numbers.

In order to solve this kind of query, we have to write some code of our own to manipulate the input and output to VMFSIM.

CMS Pipelines is a very attractive facility of CMS, hence we will use it in the example below. For more information see "CMS Pipelines Introduction" on page 225.

Let us look at the example in Figure 61.

```
/* */

Parse Arg ptfno
cp_prodid = '6VMVMB22'
        ⋮
  'VMFSIM QUERY' cp_prodid 'SRVREQT * TDATA :PTF' ptfno ':APARNUM (STEM TEMP.'

  'PIPE Stem temp.'           , /* pick up output from VMFSIM             */
    '| Strip'                 , /* remove leading and trailing blanks     */
    '| Find :APARNUM'         , /* find all lines starting with :APARNUM  */
    '| Spec Word 2-* 1'       , /* remove first word (:APARNUM)           */
    '| Split'                 , /* split so that one APAR number per line */
    '| Spec ,TDATA :APARNUM ,', /* insert 'TDATA :APARNUM ' at            */
          '1 1-* Next'        , /* at the beginning of each line          */
    '| Stem ntemp.'            /* save in stem ntemp.                    */

  'VMFSIM QUERY' cp_prodid 'SRVDESCT * STEM NTEMP.'
        ⋮
```

*Figure 61. Part of EXEC Using CMS Pipelines to Process VMFSIM Output*

This example basically splits the APAR numbers on the :APARNUM tag into separate lines containing: TDATA :APARNUM *ptfno*, which can be fed into VMFSIM QUERY.

Now let us look at the output, shown in Figure 62 on page 142.

```
VMFSIP2480I Results for
            TDATA :APARNUM VM48192
:APARNUM VM48192
   :ABSTRACT MSHCPXLF2880E ERROR CODE 5 H/W I/O ERROR HARDWARE HCPXLF
VMFSIP2480I Results for
            TDATA :APARNUM VM48191
:APARNUM VM48191
   :ABSTRACT HCPERM_BM, REG 9 ABOVE SYSTEM STORAGE. ABENDPRG005
```

*Figure 62. Output of EXEC Using CMS Pipelines to Process VMFSIM Output*

This example is simplified. If there are many APARs, the output lines from VMFSIM may spill, and there must be some special logic to handle those cases.

It may, in this case, be just as easy to do the processing using only the REXX language. But we want to introduce you to the power of CMS Pipelines.

## VMFSIM Output Processing Tool

Appendix D, "VMFSIM Exploitation Code Examples" on page 225 contains a REXX EXEC that can be used to solve the spill problem discussed above (see "VMFSIM Output Processor" on page 227). The command is called PSIMOUT and its syntax is listed in the Appendix.

The PSIMOUT EXEC allows you to reformat the output from VMFSIM (with multiple values), in a way that is understandable input to the next VMFSIM command. It also allows you to extract only the lines that you want to see from the VMFSIM output.

---

## VMFQOBJ EXEC

As we have seen, VMFSIM does not support build lists because build lists do not belong to the Software Inventory. However, information on objects is often required. To solve this problem VMSES/E introduced a new function in VM/ESA Release 2: VMFQOBJ.

## Overview

This command returns a wealth of information on build list defined objects. It can be used, for instance, to identify the parts that are included in an object, or what objects must be rebuilt if a certain part is changed.

VMFQOBJ also extracts information from the service-level build status table and the product's PPF, in order to present a complete set of information on the object. The following information can be obtained:

- Status
- Build requisites and dependencies
- Serviceable parts included in an object
- Part options
- Object parameters

- Build list options
- Part handler and target disk
- Required GLOBALs

Figure 63 shows the syntax of the VMFQOBJ command.

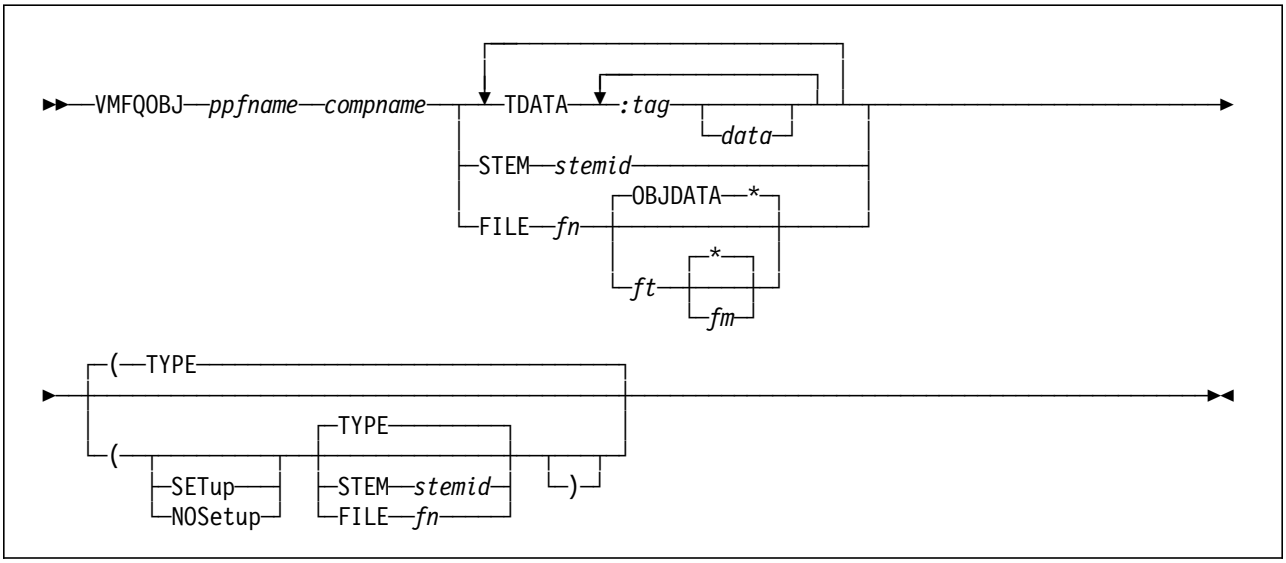



*Figure 63. VMFQOBJ Command Syntax*

## Using VMFQOBJ

When using VMFQOBJ you must know whether you are searching for a part or an object, and the part (or object) file type. Otherwise, the answer might not be what you expect.

We give below several examples of using VMFQOBJ.

## 1 - Finding the Status and Requirements of a Part

The command:

**vmfqobj esa cms tdata :part sendfile exc :stat :bldreq :blddep**

may result in:

```
VMFQOB1851I Reading build lists
VMFUTL2480I Results for
            TDATA :PART SENDFILE EXC :STAT :BLDREQ :BLDDEP
:OBJECT DMSBL490.SENDFILE.EXEC
   :STAT BUILT
      :PARTID
   :BLDREQ
   :BLDDEP
   :PART SENDFILE EXC
      :PARTOPT
```

## 2 - Finding the Objects Impacted by a Part Change

Let us suppose that we will alter the part HCPDDR with a local update. To properly plan this action, we should find out which objects have to be rebuilt after HCPDDR is altered. Entering the command:

**vmfqobj esa cp tdata :part hcpddr txt**

results in:

```
VMFQOB1851I Reading build lists
VMFUTL2480I Results for
            TDATA :PART HCPDDR TXT
:OBJECT HCPBLUTL.IPL.DDRXA
   :PART HCPDDR TXT
      :PARTOPT
   :PART HCPDNC TXT
      :PARTOPT
   :PART HCPDDC TXT
      :PARTOPT
   :PART HCPDNT TXT
      :PARTOPT
   :PART HCPDDT TXT
      :PARTOPT
:OBJECT HCPMLOAD.DDR.MODULE
   :PART HCPDDR TXT
      :PARTOPT ORIGIN 20000 NOMAP NOUNDEF CLEAR
   :PART HCPDNC TXT
      :PARTOPT SAME
   :PART HCPDDC TXT
      :PARTOPT SAME
   :PART HCPDNT TXT
      :PARTOPT SAME
   :PART HCPDDT TXT
      :PARTOPT UNDEF SAME
```

Notice that we gave the part name and type. Omitting the type would have resulted in a lengthy list of unrelated objects.

Two objects, the stand-alone IPL DDRXA and the CMS command DDR, include the HCPDDR part. In addition to the object name, the build list name is also given. This is required if you want the VMFBLD command to build just those two objects.

## 3 - Finding All the Characteristics of an Object

To find everything necessary to build object DMSSAA we issued the following command:

**vmfqobj esa cms tdata :object dmssaa**

And the response was:

```
VMFQOB1851I Reading build lists
VMFUTL2480I Results for
            TDATA :OBJECT DMSSAA
:OBJECT CMSSAA.DMSSAA
   :STAT BUILT
      :PARTID
   :LIBNAME CMSSAA
   :BLDREQ
   :BLDDEP
```

```
   :GLOBAL
   :PARTHAND VMFBDTLB
   :TARGET BUILD7
   :BLOPT
   :OBJPARM
   :PART DMSSAA TXT
      :PARTOPT
:OBJECT DMSBLVML.DMSSAA.TEXT
   :STAT BUILT
      :PARTID
   :LIBNAME
   :BLDREQ
   :BLDDEP DMSSBVML.CMSVMLIB.SEGMENT
   :GLOBAL
   :PARTHAND VMFBDCOM
   :TARGET BUILD7
   :BLOPT
   :OBJPARM
   :PART DMSSAA TXT
      :PARTOPT
```

The answer shows that there are two objects (of different types) named DMSSAA. The first object, CMSSAA TXTLIB, includes the DMSSAA TXT part. The second object, DMSSAA TEXT, is the usable form of the DMSSAA TXT part.

## VMFINFO Command

The new VMFINFO command is a panel-driven front-end to the VMFQOBJ command and to the QUERY, GETLVL, COMPTBL, SYSDEP, SRVDEP, SYSREQ, and SRVREQ subcommands of VMFSIM.

Using VMFINFO, you do not have to remember the complex command syntax and the names of the files.

VMFINFO is task oriented, has a predefined set of queries, and allows you to save the responses. The VMFINFO panels conform to the Common User Access* (CUA*) architecture. It also provides a context-sensitive help. The syntax of the VMFINFO command is shown in Figure 64.

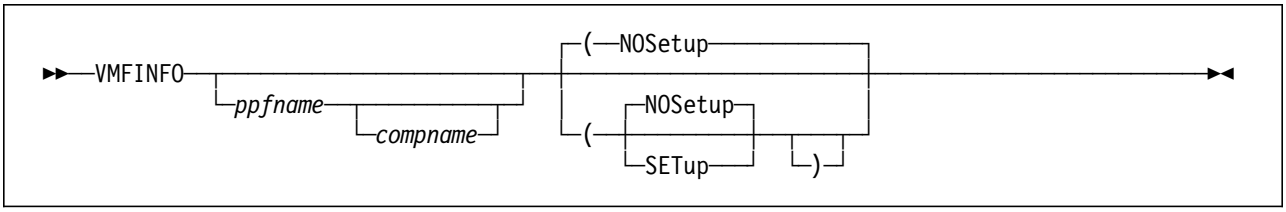



*Figure 64. VMFINFO Command Syntax*

The best way to explain this command is with an example.

## VMFINFO PPF and Component Name Selection Panels

Entering:

**vmfinfo**

on the command line will invoke the PPF Fileid Help panel shown in Figure 65. In *VM/ESA: VMSES/E Introduction and Reference* the panels are referred to as help panels, because they can be invoked by pressing the Help (PF1) key. However, they truly are selection panels, because not only do they provide explanations, they also allow you to select products or components to work with.

```
                          PPF Fileid - Help

 Product parameter files (PPFs) define the environment and key variables
 required to process the queries.  The following is a list of all PPFs
 found on all accessed disks.  Select one to continue.  The View function
 can be used to examine one or more PPFs.

 Type a "V" next to one or more PPFs to view their contents, or type an
 "S" next to one PPF to select.

      Options: S - select  V - view

 Option   PPF Fileid
          CUFINS   PPF      D1
   S      ESA      PPF      D2
          SEGBLD   PPF      D1
          UCENG    PPF      D1




Command ===>
 PF1 = Help  PF3 = Exit  PF12 = Cancel
```

*Figure 65. VMFINFO PPF Fileid Help Panel*

Because VMFINFO is product oriented, you must select a product to work with. Therefore, VMFINFO displays a list of the PPFs found on all accessed disks and waits for your selection.

Next, it accesses the PPF and displays a list of components, as shown in Figure 66 on page 147.

```
                          Component Name - Help

 Product parameter files (PPFs) can contain one or more component names,
 each specifying different environments and key variables.  The following
 is a list of component names within the PPF selected.  Select one to
 continue.

 Type an "S" next to one component name; then press enter.

 Option   Component Name/Description
          VMSES             - 6VMVMK22
          DV                - 6VMVMI22
   S      CMS               - 6VMVMA22
          REXX              - 6VMVMF22
          GCS               - 6VMVML22
          TSAF              - 6VMVMH22
          AVS               - 6VMVMD22
          GCSSFS            - 6VMVML22
          TSAFSFS           - 6VMVMH22
          AVSSFS            - 6VMVMD22
          CP                - 6VMVMB22

Command ===>
 PF1 = Help  PF3 = Exit  PF12 = Cancel
```

*Figure 66. VMFINFO Component Name Help Panel*

Again, you must make a selection.

Next, VMFINFO displays its main panel, discussed below.

**Notes:**

- While PF12 cancels the current function, returning to the previous one, PF3 ends the VMFINFO command. This behavior, consistent with the CUA rules, differs from other CMS commands.
- If only one PPF file is found, or the PPF contains only a single component, then that PPF file is automatically selected and the help panel is not displayed.
- If the number of PPFs in the list exceeds the number of lines on the screen, the screen's will include the following:
  - A "More + -" indicator.
  - On the last line, the definitions "PF7=Bkwd  PF8=Fwd" indicate that the list can be scrolled.

## VMFINFO Main Panel

If you know the PPF and component names you can bypass the two preceding panels by entering (in the example CMS is used):

**vmfinfo esa cms**

VMFINFO directly displays the Main panel, shown in Figure 67 on page 148.

If you only know the PPF name you can enter it, and VMFINFO presents the Component Name - Help panel.

```
                              VMFINFO Main Panel

 Select one of the following.  Then press enter.
  PPF fileid ...... ESA       PPF  D
  Component name .. CMS                Setup ... NO
  Product ID .....: 6VMVMA22           System .. VM

     Options: S - select
  Option   Query
           Product description
           Product status
           Product requisites
           Product dependencies
    S      PTFs/APARs
           Serviceable parts/usable forms
           Miscellaneous







Command ===>
 PF1 = Help  PF3 = Exit  PF12 = Cancel
```

*Figure 67. VMFINFO Main Panel*

You can ask VMFINFO to automatically access the product's disks when it is invoked, or from the panel: just type YES in the "Setup" field.

You can also change the product you are working with by over-typing the information on the "Component name" or "PPF fileid" fields, as appropriate.

Notice the first four options allow you to work with the system-level Software Inventory.

## VMFINFO PTF/APAR Queries Panel

In this example, we are seeking service information, so PTFs/APARs is selected.

The PTF/APAR Queries panel lists several types of information you can obtain about a specific PTF or APAR. You have to specify either the PTF or the APAR number. You can obtain a list of valid PTF or APAR numbers by placing the cursor on the appropriate field and pressing the PF1 key. The example shown in Figure 68 on page 149 selects the serviceable parts of a PTF.

```
                              PTF/APAR Queries

 Enter a PTF or APAR number and type an option code.  Then press Enter.
  PPF fileid ...... ESA       PPF  D
  Component name .. CMS               Setup ... YES
  Product ID .....: 6VMVMA22          System .. VM
  PTF number ...... UM98765
  APAR number .....

     Options: S - select
  Option   Query
           Status of PTF
           Requisites/supersedes of PTF
           Dependencies/superseding of PTF
           User memo of PTF
    S      Serviceable parts included by PTF

           Abstract of APAR(s)



Command ===>
 PF1 = Help  PF3 = Exit  PF12 = Cancel
```

*Figure 68. VMFINFO PTF/APAR Queries Panel*

Figure 69 shows the Query Output - PTF Serviceable Parts panel. Notice the PF5 = File option. It allows you to save the results of your query.

```
                  Query Output - PTF Serviceable Parts

  PPF fileid .....: ESA       PPF  D
  Component name .: CMS               Setup ..: NO
  Product ID .....: 6VMVMA22          System .: VM
 ------------------------------------------------------------
PTF: UM98765
------------
DMSABC ASSEMBLE
DMSXYZ ASSEMBLE
DMSABCDE MACRO









Command ===>
 PF1 = Help  PF3 = Exit  PF5 = File  PF12 = Cancel
```

*Figure 69. VMFINFO PTF/APAR Query Output Panel*

## How to Answer Your Top Ten Questions

In "Introduction" on page 37, there is a list of some important questions you might like to have the answer to. This section illustrates using the VMFSIM command to provide an answer to those questions. Whenever possible, an alternate method using the VMFINFO command is also presented.

First, a general remark: some VMFSIM information messages sometimes distort the output picture. Therefore, in the examples below, we disabled these messages by issuing the command:

**cp set emsg off**

If we did not, we would have received the normal information message:

```
VMFSIP2480I Results for
            TDATA :STAT :PRODID
```

When you have more complicated queries, you may wish to run with EMSG ON, and selectively disable the messages; or you can use the FILE option, which directs the output to disk, while the messages appear on the screen. This approach, however, produces a TDATA line in front of every data line.

## 1 - List Products Installed on the System

The Receive Status table of the system-level Software Inventory contains a list of all the products you have ever installed on your system, even those that were later deleted. To list all products that have been installed on your system, enter:

**vmfsim query vm sysrecs * tdata :stat :prodid**

Part of the reply looks like this:

```
:PPF ESAINS VMSES
   :PRODID 6VMVMK22
   :STAT RECEIVED.12/22/93.16:55:02.MAINT.200-0000
:PPF ESAINS CP
   :PRODID 6VMVMB22
   :STAT RECEIVED.12/22/93.17:28:06.MAINT.200-0000
:PPF CUFINS CUF
   :PRODID 6VMVME11
   :STAT RECEIVED.12/05/93.13:17:44.MAINT.910-9101
:PPF MYCUF CUF
   :PRODID 6VMVME11
   :STAT DELETED.09/16/91.15:28:47.MAINT
:PPF 5748RC1 NONE
   :PRODID 5748RC1
   :STAT RECEIVED.03/22/91.12:08:11.MAINT
:PPF ESAINS TSAF
   :PRODID 6VMVMH22
   :STAT DELETED.12/27/93.11:33:22.MAINT
```

We see some components have been deleted. To see only the active components, specify :STAT RECEIVED on the query.

Using VMFINFO, you can get a list of PPFs on the system, (PPF Help panel), hence you can find the installed products. To see the status of a component, first select a PPF, then select a component, and on the VMFINFO Main Panel (see Figure 67 on page 148) select the "Product Status" option.

## 2 - List Prerequisites for a Component

We are still using the system-level Software Inventory, this time the Requisites table. The following command will list the prerequisites for CP:

**vmfsim query vm sysreqt tdata :prodid cp**

The response is as follows:

```
:PRODID 6VMVMB22%CP
   :PREREQ 6VMVMK22
   :REQ
   :DREQ
   :IFREQ
   :SUP 6VMVMB11 6VMVMB20 6VMVMB21
   :NPRE
```

This command lists all requisite information for CP. If you want to see only the real prerequisite information, add :PREREQ to the query above.

You can also use VMFINFO, and on the Main Panel select the Product Requisites option.

## 3 - List the PTFs Applied to a Component

The apply information is found in the service-level Software Inventory, in the Apply Status table (SRVAPPS). The following command lists all PTFs applied to CP (the value of :STAT is APPLIED):

**vmfsim query 6vmvmb11 srvapps tdata :ptf :stat applied**

The answer may appear as follows:

```
:PTF UM18657
   :STAT APPLIED.09/18/91.13:36:11.MAINT
:PTF UM18667
   :STAT APPLIED.09/18/91.13:36:11.MAINT
:PTF UM18681
   :STAT APPLIED.09/18/91.13:36:11.MAINT
:PTF UM18690
   :STAT APPLIED.09/18/91.13:36:11.MAINT
:PTF UM18695
   :STAT APPLIED.09/18/91.13:36:11.MAINT
:PTF UM18696
   :STAT APPLIED.09/18/91.13:36:11.MAINT
:PTF UM18699
   :STAT APPLIED.09/18/91.13:36:11.MAINT
```

With VMFINFO you can also find about PTFs and APARs:

1. On the Main Panel select PTFs/APARs.
2. On the PTF/APARs Queries panel enter a PTF or APAR number and select Status of PTF.

You can also obtain a list of APARs (or PTFs) by placing the cursor on the APAR (or PTF) field and invoking the Help function.

## 4 - List APARS for a PTF

This type of information is found in the service-level Requisite table (SRVREQT). It contains requisite information for the PTFs and the APAR numbers that constitute that PTF. The following command will do the job:

**vmfsim query 6vmvmb11 srvreqt tdata :ptf um18699 :aparnum**

The response will look like this:

```
:PTF UM18699
   :APARNUM VM48180 VM48178 VM48175 VM48174 VM48173 VM48171 VM48170 VM48169
            VM48030 VM48028 VM48024 VM48023 VM48022
```

As we see, this is a huge PTF. Therefore, let us take a closer look at all the information about this PTF. We issue the command:

**vmfsim query 6vmvmb11 srvreqt tdata :ptf um18699**

The answer will look like this:

```
:PTF UM18699
   :APARNUM VM48180 VM48178 VM48175 VM48174 VM48173 VM48171 VM48170 VM48169
            VM48030 VM48028 VM48024 VM48023 VM48022
   :PREREQ UM18657 UM18681 UM18690 UM18696
   :COREQ
   :IFREQ
   :SUP UM18698 UM18693 UM18691 UM18686 UM18682 UM18680 UM18679 UM18674
        UM18673 UM18671 UM18669 UM18666
   :HARDREQ VM48020 VM48178 VM48176 VM48029 VM48028 VM48023
```

This PTF is a very central one. It supersedes many previous PTFs, and also lists :HARDREQ APARs. Our PTF is physically dependent on the PTFs containing those APARs, because they update the same area of the code.

Notice that APARs VM48178, VM48028, and VM48023 are listed both as included in the PTF and as a HARDREQ. This can happen because a superseding PTF has to include in its requisites all the requisites of the superseded PTFs.

Using VMFINFO follow these steps:

1. On the Main Panel select PTFs/APARs.
2. On the PTF/APAR Queries panel enter the PTF number and select Abstracts of APAR(s).

Note that using the same Queries panel you can also obtain information on requisite and superseding PTFs.

## 5 - List Status of an APAR

This is a composite query. The status information is in the Apply Status (SRVAPPS) table, which uses PTF numbers as a key. The link between PTF number and APAR number is found in the SRVREQT table, as we have seen.

Hence, we first look up the corresponding PTF number in the Requisite (SRVREQT) table, save the results in a file, and use that as input to the query to the SRVAPPS table, to get the status. The command sequence is:

**vmfsim query 6vmvmb11 srvreqt tdata :ptf :aparnum vm48180 (file temp**
**vmfsim query 6vmvmb11 srvapps file temp**

The output from the last command may look like this:

```
:PTF UM18698
   :STAT SUPED.09/18/91.13:36:11.MAINT
:PTF UM18699
   :STAT APPLIED.09/18/91.13:36:11.MAINT
```

This is an interesting example. The given APAR, VM48180, is part of two PTFs. The reason is, as we see, that the last PTF has superseded the first. So, there is only one “active” PTF with the given APAR in it.

As EMSG was turned off, the reply does not show the tag values for the second query. This is one of the cases where you would have seen several information messages telling you that :APARNUM is not a field of the SRVREQT table.

VMFINFO helps you to find the status of a given APAR and the PTF it is part of:

1. On the Main Panel select PTFs/APARs.
2. On the PTF/APAR Queries panel enter the PTF number and select Status of PTF.

Please note that VMFINFO does not have generic search capabilities; for example, using VMFSIM you can ask “Which APARs are related to this device type?,” but the same is not true for VMFINFO.

These queries tell you that the PTF exists and has been applied but is it in production, that is, have the objects affected by this PTF been rebuilt? See “Finding the Status of an APAR or PTF” on page 230 for an example of an EXEC that gives a detailed answer to that question.

## 6 - List the PTFs that Depend on a Given PTF

Here we use the VMFSIM SRVDEP subcommand. In our example, it will use the information in the SRVREQT and the SRVAPPS tables to locate all the dependent PTFs. This may be an iterative task, because dependent PTFs may themselves have dependent PTFs, and so on. The command is:

```
vmfsim srvdep 6vmvmb11 srvreqt * = srvapps * tdata :ptf um18699
```

The result looks like this:

```
:PTF UM18699
   :DEPS UM18719 UM18730 UM18755 UM18756 UM18774 UM18775 UM18777 UM18778
         UM18779 UM18780 UM18701 UM18702 UM18703 UM18717 UM18718
   :OUTREQS * NONE *
```

These PTFs are, in fact, the ones you would have to remove if you wanted to remove the original PTF.

Only PTFs with status APPLIED are considered. PTFs with status REMOVED or SUPED are ignored.

Using VMFINFO, you can find the answer by following these steps:

1. On the Main Panel select PTFs/APARs.
2. On the PTF/APAR Queries panel enter the PTF number and select Dependencies/superseding of PTF.

## 7 - List Parts Serviced by a PTF

Now we want to know which parts are affected by a PTF. This information is in the Version Vector table. The Version Vector table contains information about all parts that have been serviced, and which PTFs were involved. So, to complete this task, we will issue the following command:

**vmfsim query 6vmvmb11 vvtvm * tdata :ptf um18699**

The result may look like the following:

```
:PART HCPUDV TXT
   :PTF UM18755.VM48189.R48189HP UM18699.VM48030.R48030HP
:PART HCPMDLAT MACRO
   :PTF UM18699.VM48030.R48030HP
:PART HCPXLF TXT
   :PTF UM18777.VM48192.R48192HP UM18776.VM48191.R48191HP
        UM18699.VM48030.R48030HP UM18657.VM48020.R48020HP
```

As you see, there are three types of information for each PTF:

- The PTF number
- The APAR number
- The file type of the Update file

The file type is present only if the part is source maintained.

VMFINFO displays information about the parts serviced by a PTF:

1. On the Main Panel select PTFs/APARs.
2. On the PTF/APAR Queries panel enter the APAR or PTF number and select PTF Serviceable Parts included by PTF.

Note that VMFINFO will not generate a list of all the PTFs for a part. You have to enter each PTF by hand.

## 8 - List Service Applied to a Part

This example illustrates another usage of the Version Vector table. This time we will use it the other way around: We will search for a specific part. The command is:

**vmfsim query 6vmvmb11 vvtvm * tdata :part hcpxtc**

The output is of the same type as for the previous query:

```
:PART HCPXTC TXT
   :MOD
   :PTF UM18779.VM48194.R48194HP UM18699.VM48030.R48030HP
        UM18657.VM48020.R48020HP
```

We conclude that the part HCPXTC ASSEMBLE has been serviced three times, and the most recent PTF is listed first.

VMFINFO can show you the same information:

1. On the Main Panel select Serviceable parts/usable forms.
2. On the Serviceable Parts/Usable Forms Queries panel enter the part's name and select Service history of part(s).

## 9 - List Parts that Must be Rebuilt after Service

As explained in “How Build Works” on page 110, VMFBLD will, during the STATUS part processing, flag on the service-level Build Status table all objects that have to be built, based on the entries found in the Select Data file.

To see the information, issue the command:

**vmfsim query 6vmvmb11 srvblds * tdata :stat serviced**

The result may look like the following:

```
:BLDLIST CPLOAD
   :OBJECT BLDLIST
   :STAT SERVICED.02/21/93.15:33:11.MAINT
:BLDLIST HCPBLUTL
   :OBJECT 3CARD
   :STAT SERVICED.02/11/93.12:10:14.MAINT
:BLDLIST HCPOM1
   :OBJECT BLDLIST
   :STAT SERVICED.02/18/93.14:54:46.MAINT
```

Note that the :OBJECT tag lists the name of the object to be built. In some entries, the tag value is the keyword BLDLIST. Those entries result from two situations:

- The corresponding build list is a Format 1 build list, which defines a single object, and does not contain any tags.
- This is a global entry for the build list that signals the overall status of all the objects in the build list.

For further information about build lists, please refer to “Build Lists” on page 54.

Using VMFINFO:

1. On the Main Panel select Miscellaneous.
2. On the Miscellaneous Queries panel select Build requirements.

## 10 - List Service Impact of Backing Out a PTF

We touched on this in a previous example, see “6 - List the PTFs that Depend on a Given PTF” on page 153. There we got the PTF numbers. Now let us go one step further. We will also list the corresponding APARs and their description, so you can better evaluate the consequences of removing the PTF.

To obtain this information, we use the sample PTFREMOV EXEC listed in “Impact of Backing Out a PTF” on page 226.

We issue the command:

**ptfremov esa cp um18699**

And the output appears as:

```
Getting dependent PTFs...
Looking for corresponding APAR numbers
Looking for APAR descriptions
To back out PTF UM18699 : 16 PTFs ( 29 APARs ) must be backed out:
For details, see file UM18699  $BACKOUT A
```

The listed file, UM18699 $BACKOUT, contains the PTF and APAR numbers, and the APAR description for each APAR.

## Further Examples

Other examples, also using CMS Pipelines, of Software Inventory exploitation are given in Appendix D, “VMFSIM Exploitation Code Examples” on page 225.

Creating a list of what replacement parts can be safely erased is covered in “Erasable Parts for Committed PTFs” on page 228, and “Finding the Status of an APAR or PTF” on page 230 contains an example of how to use the Software Inventory to find out if a given APAR has been applied and is currently in production.

In the examples above, we have touched most of the tables in the service-level Software Inventory and some of the tables in the system-level Software Inventory.

For further information refer to *VM/ESA: Service Guide*.

# Chapter 8. Saved Segment Experiences

This chapter describes our experiences when applying service to saved segments, and the results of several experiments we conducted to test some of the capabilities of saved segments management.

For an overview of this VMSES/E capability, introduced in Release 2, see "Overview" on page 65.

Before we can describe our experiences we have to expand the description given in "Product-Level View" on page 76. First we will examine more closely the Software Inventory tables and files, as well as other auxiliary control files that are part of saved segments support. Next, we will better detail the build process.

**Note:** The information on this chapter does not apply to a VM/ESA Release 1.5 370 Feature system.

## Software Inventory and Other Files

As explained before, segment planning cannot be done from the product level. Actually, all segment manipulations should be done from a system perspective, therefore requiring for these objects the following system-level (and other) files:

- prodid PRODPART

  Provides default segment information.

- System-level Product Parameter File

  Allows management of system-level objects.

- System-level Segment Build List (EXC00000)

  Used in conjunction with the SEGDATA file. Points to product-level segment build lists.

- System-level Segment Delete List (DEL00000)

  Used in conjunction with the SEGDATA file and the EXC0000 file. Contains information on segments to be deleted.

- Segment Data file (SEGDATA)

  Holds customized segment information.

- System-level Service Build Status table (SRVBLDS)

  Holds build status information on system objects.

- System-level Select Data files (SEGBLD $SELECT and VMSBR $SELECT)

  Links together segment planing, product servicing, and system-level segment building. Prior to Release 2.2, this was only the VMSBR $SELECT.

- VMSESE PROFILE

  Holds a pointer to the VMSBR $SELECT file.

## Product Parts File

The PRODPART file has a section that contains the default saved segments information for the product. This was described in “Saved Segment Definitions Section” on page 58.

## System-Level PPF

This PPF, by default, resides on the SIDISK and is named SEGBLD. It is supplied with the CP component, and has only one component.

Figure 70 shows the SEGBLD $PPF file. Starting with VM/ESA Release 2.2 the file is no longer implemented as an override to the CMS component of 6VMVMAnn $PPF.

Also please notice the use of two :APPLY. IDs. This allows separating segment modifications done through VMFSGMAP from modifications done through normal product service. It also makes it easier to manage multiple versions of the SEGDATA file.

```
************************************************************
* COPYRIGHT - 5684-112 - (c) COPYRIGHT IBM CORP.- 1992 1994
*             LICENSED MATERIAL - PROGRAM PROPERTY OF IBM
*             SEE COPYRIGHT INSTRUCTIONS,  G120-2083
************************************************************
*==========================================================
* Product Parameter File for VM/ESA Release 2.2 Segment Builds
*  This PPF file used only for mapping and building segments
*==========================================================
*
* The following APAR's have been applied to this file:
*
*  APAR      Date       Description
*  =======   ========   ======================================
*
*
*==========================================================
*            NOTE: All tags must be in upper case.
*==========================================================

*==========================================================
* Start of Product Header - List of Segment Build Components
*==========================================================
:COMPLST. ESASEGS
:OVERLST.
*==========================================================
* End Product Header
*==========================================================
*
:ESASEGS.
:PRODID. SEGBLD%ESASEGS
*
```

*Figure 70 (Part 1 of 3). SEGBLD $PPF File*

```
*============================================================
* Control Parameters
*============================================================
:CNTRLOP.

* TAG      VALUE(S)
*--------- ---------
:PRODDESC. SEGMENT BUILD PPF    * Product description
:BCOMPNAME. ESASEGS              * Base component name
:VERSION.  VM/ESA 1.2.2
:RECID.    VMESASYS             * File name of service
                                * receive status table
:APPID.    SEGBLD VMSBR         * File name of service
                                * apply status table
:BLDID.    VM                   * File name of service
                                * build status table
:LOG.      YES                  * Log all messages
:RECVALL.  NO                   * Receive missing parts for
                                * committed PTFs
:SETUP.    NO                   * Call VMFSETUP
:SLVI.     V/VF                 * System level and version
                                * indicator
:NLS.      AMENG                * System language
:CNTRL.    VMFVM                * Control file name
:AXLIST.   VMFVM                * File name of IBM supplied
                                * APPLY list and EXCLUDE list
:EXCLIST.                       * File name of user's own
                                * EXCLUDE list
:UPDTID.   AUXVM                * File type of AUX file
:CKAUX.    YES                  * VMFBLD compare AUX file to
                                * Standard Self Documentation
                                * Information in text decks
:CKSDI.    NO                   * VMFAPPLY check Standard Self
                                * Documentaion in text decks
:CKVV.     NO                   * Check AUX file against
                                * corresponding version vector
                                * during vmfbld
:CKGEN.    YES                  * Check AUX file against
                                * corresponding version vector
                                * during vmfasm, vmfhasm,vmfnls
:RETAIN.                        * List of file modes that
                                * VMFSETUP cannot use
:USEREXIT.                      * User exit EXEC called for
                                * setup and cleanup at the
                                * beginning and end of each
                                * service function
:PTFPFX.   UM                   * Two character PTF prefix
:APARPFX.  VM                   * Two character APAR prefix
:ECNTRLOP.
```

*Figure 70 (Part 2 of 3). SEGBLD $PPF File*

```
*============================================================
* Variable definitions
*============================================================
:DCL.
&SID  LINK MAINT 51D 51D MR  * Disk for Segment Build files
:EDCL.

*============================================================
* Mdisk/Directory Access    -define symbolic disk names
*============================================================
:MDA.
*STRINGNAME MINIDISKS
*---------- ---------
APPLY       &SID        * Contains VMSBR $SELECT file
DELTA       &SID        * Required by VMFBLD processing
BUILD       &SID        * Contains build list and SEGDATA file
BASE        &SID        * Required by VMFBLD processing
:EMDA.
*============================================================
* RECINS section
*============================================================
:RECINS.
:ERECINS.

*============================================================
* RECSERV section -
*============================================================
:RECSER.
:ERECSER.

*============================================================
* BUILD  section
*============================================================
:BLD.
* BUILDLIST EXEC     TARGET   DESCRIPTION
* --------- -------- -------  ------------
SEGBLIST    VMFBDSEG BUILD    * Segment build
:EBLD.

:DABBV.
:EDABBV.

:END.
*
```

*Figure 70 (Part 3 of 3). SEGBLD $PPF File*

## System-Level Build Lists

The system-level segment build list is listed in the build section of the SEGBLD PPF, and it really is a list of pointers to the product-level build lists. It will have a file type of EXC0000. As stated above, it corresponds to a particular arrangement, or storage layout, of the segments. The suggested file name is SEGBLIST, but you can change it. If you do change it, you also have to add the new name to the SEGBLD PPF (removing a build list from the PPF is not

supported). This chapter assumes you are using the default name. This build list is shipped with the system. You should never change it directly.

The bldid DEL00000 build list is created dynamically by VMFSGMAP, when segments are deleted by the user (see “Deleting a Segment” on page 168).

Figure 71 shows an excerpt of the SEGBLIST EXC00000 file. As you can see:

- It is a Format 2 build list
- Each object corresponds to a segment
- Each object block includes one or more :PARTID tags. These tags do not refer to parts (as pertaining to the definition for part in “Definitions and Terms” on page 10). Instead:
  - Tags listing “parts” with a file type of EXC really specify the name of the product-level build list for the segment. This build list lists the segment build requirements, which enables the product service process to flag the segment as having to be rebuilt, whenever one of those objects is serviced.
  - One tag listing a “part” with a file type of DMY. This tag acts as a place holder entry that defines the real name of the segment on the system. You may, using the same product-level build list, create several copies of the segment, for example, with different storage ranges. This part, then provides the means to reference a particular copy; for example, when you change the segment’s storage addresses VMFSGMAP inserts the name of this part in the SEGBLD $SELECT file.

  The usage of multiple partid tags enables VMSES/E to determine if a segment requires a rebuild, whether an object within a segment has changed, or if the segment definition, such as storage, has changed.

**Notes:**

- Saved segments of non-VMSES/E products have only one part per object: the one with the DMY file type. This is because no product-level build list is available for those segments.
- Named saved systems are **not** included in this file, because building NSSs is not supported.

```
:FORMAT. 2
:OBJNAME. CMSBAM.SEGMENT
:BLDREQ. SEGBLIST.DOSINST.SEGMENT
:PARTID. DMSSBBAM EXC
:PARTID. CMSBAM DMY
:EOBJNAME.
        ⋮
:OBJNAME. CMSPIPES.SEGMENT
:PARTID. DMSSBPIP EXC
:PARTID. CMSPIPES DMY
:EOBJNAME.
        ⋮
```

*Figure 71. SEGBLIST EXC00000 File (Excerpt)*

## Segment Data File

The Segment Data file has a file type of SEGDATA. This file contains the customized information for each segment. Note that:

- Data in this file is entered using the VMFSGMAP command. It may be:
  - Extracted from the PRODPART file
  - Entered by you
  - Extracted from the live system
- The file name is the same as the system-level build list. The file type must be SEGDATA.

Figure 72 shows an excerpt of the SEGBLIST SEGDATA file.

```
:OBJNAME.CMSBAM :DEFPARMS.B0D-B37 SR :SPACE.DOSBAM :TYPE.SEG
:OBJDESC.CMSBAM MEMBER OF THE DOSBAM SEGMENT SPACE :GT_16MB.NO
:SEGREQ.DOSINST :PRODID.6VMVMA22 CMS :BLDPARMS.PPF(ESA CMS DMSSBBAM)
        ⋮
:OBJNAME.CMSVMLIB :DEFPARMS.800-8FF SR :TYPE.PSEG :OBJDESC.VMLIB CSL
SEGMENT AND VMMTLIB CSL SEGMENT :GT_16MB.YES :PRODID.6VMVMA22 CMS
:BLDPARMS.PPF(ESA CMS DMSSBVML) PPF(ESA CMS DMSSBVMT)
:OBJNAME.CMSPIPES :DEFPARMS.700-77F SR :TYPE.PSEG :OBJDESC.CMS PIPES SEGMENT
:GT_16MB.YES :PRODID.6VMVMA22 CMS :BLDPARMS.PPF(ESA CMS DMSSBPIP)
        ⋮
```

*Figure 72. SEGBLIST SEGDATA File (Excerpt)*

## System-Level Select Data Files

Starting with VM/ESA Release 2.2, two system-level Select Data files are used. The SEGBLD $SELECT file is updated by VMFSGMAP and contains a list of segments that were added or changed by the user and so have to be built (or rebuilt). VMFSGMAP updates this file as follows:

- When segments are added or changed, VMFSGMAP inserts the name of the part of type DMY, from the object's entry in the SEGBLIST EXC0000 build list. In this way, if you have several copies of a segment only the one you have changed will be flagged as having to be built.
- When segments are deleted VMFSGMAP inserts a line containing:

  **SEGBLIST EXC DEL00000**

  The deleted segments are removed from the SEGBLIST EXC00000 build list, but are retained in the SEGBLIST DEL00000 build list. During the subsequent build step VMFBLD compares the EXC00000 and DEL00000 build lists to determine which segments to delete.

Figure 73 on page 163 shows an example of the SEGBLD $SELECT file.

**Note:** The file name of this Select Data file is taken from the primary (first) name in the :APPID tag of the PPF. SEGBLD is the default.

The secondary Select Data file, VMSBR $SELECT, contains a list of segments that were serviced and have to be re-built. It is updated during the build step of the product service build process, through the VMFBDSBR build part handler. This part handler only knows the product-level build list, so this is the name it inserts in the Select Data file. In this way, even if you have several copies of a

segment, they will all be flagged as having to be built, which is correct since all of them are affected by the service.

Figure 74 shows an example of the VMSBR $SELECT file.

```
:APPLYID. 09/04/92 15:45:36
CMSPIPES DMY
   ⋮
```

*Figure 73. SEGBLD $SELECT File (Excerpt)*

```
:APPLYID. 09/11/92 10:29:19
DMSSBBAM EXC
   ⋮
```

*Figure 74. VMSBR $SELECT File (Excerpt)*

### VMSESE PROFILE

This file was introduced in "VMFBDSBR Part Handler" on page 76. Figure 75 shows a sample VMSESE PROFILE file.

```
* Default disk used for updating the VMSBR $SELECT file for saved
* segment builds.
:SHRDISK.  LINK MAINT 51D MR
```

*Figure 75. Sample VMSESE PROFILE*

## Building Saved Segments

Building saved segments is necessary:

- After you alter the segment's parameters, such as the storage ranges.
- At the end of normal product installation (if the product uses segments).
- When you have serviced a product, and the service process has flagged the need to build the segments.

Of course, you still can build any segment at any time, even if none of the above conditions apply. You should, before building any segments, re-plan the segment layout. As a minimum, you should make sure that there are no conflicts, such as storage overlaps, by using VMFSGMAP.

### Segment Planning

Using the VMFSGMAP command was discussed in "Saved Segment Planning" on page 68. We will use adding the CMSBAM segment as an example to show how the contents of the involved files change. Figure 76 on page 164 shows the interior structure of the files used by VMFSGMAP.

To invoke the command we entered:

```
vmfsgmap segbld esasegs segblist
```

We followed the directions in “Adding a Segment Definition” on page 73. On the OBJNAME field we entered CMSBAM. We then exited VMFSGMAP using the File function. VMFSGMAP placed the data that was extracted from the PRODPART file in the SEGBLIST SEGDATA and SEGBLIST EXC00000 files. Also, VMFSGMAP made an entry in SEGBLD $SELECT.

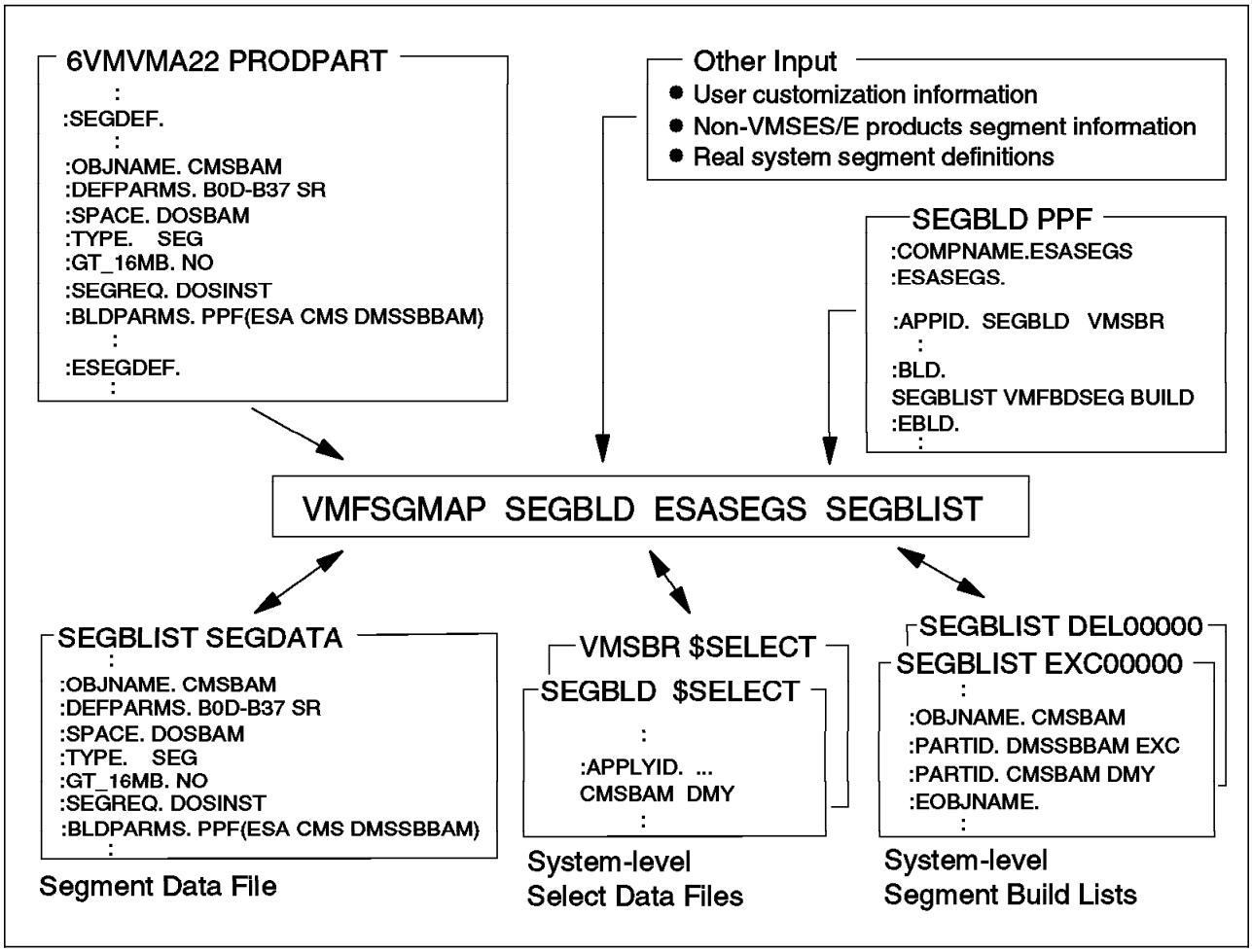



*Figure 76. Internal Data Flow in VMFSGMAP*

## Segment Servicing

To further detail the process described in “Product-Level View” on page 76, let us use an example.

### Product-Level

The first step in any VMFBLD invocation is known as STATUS. Before VM/ESA Release 2.2 this step was **always** executed, even when the given option was SERVICED or ALL. Now, VMFBLD compares the time stamps in the $SELECT files to the ones in the :LASTAPP tag of the Build Status table. If they match, the STATUS step is skipped. This step can also be executed by itself, and this is what we have done, so its effects can be analyzed. Please refer to Figure 77 on page 165 during the following discussion.

1. CMS part DMSIOS is changed by normal service. VMFAPPLY adds an entry in the Select Data file for CMS (6VMVMA22 $SELECT).
2. In the CMSMLOAD build list, the DMSPIPE.MODULE object includes DMSIOS as one of its parts. So, when the CMS component is built after service application, the STATUS option of the VMFBLD command alters the CMS Build Status table (6VMVMA22 SRVBLDS) to show as SERVICED the status of the DMSPIPE.MODULE object.
3. But, in the DMSSBPIP build list, the CMSPIPES.SEGMENT object has a build requisite for DMSPIPE.MODULE (:BLDREQ tag). So, the CMSPIPES object is also flagged as SERVICED in the CMS Build Status table.

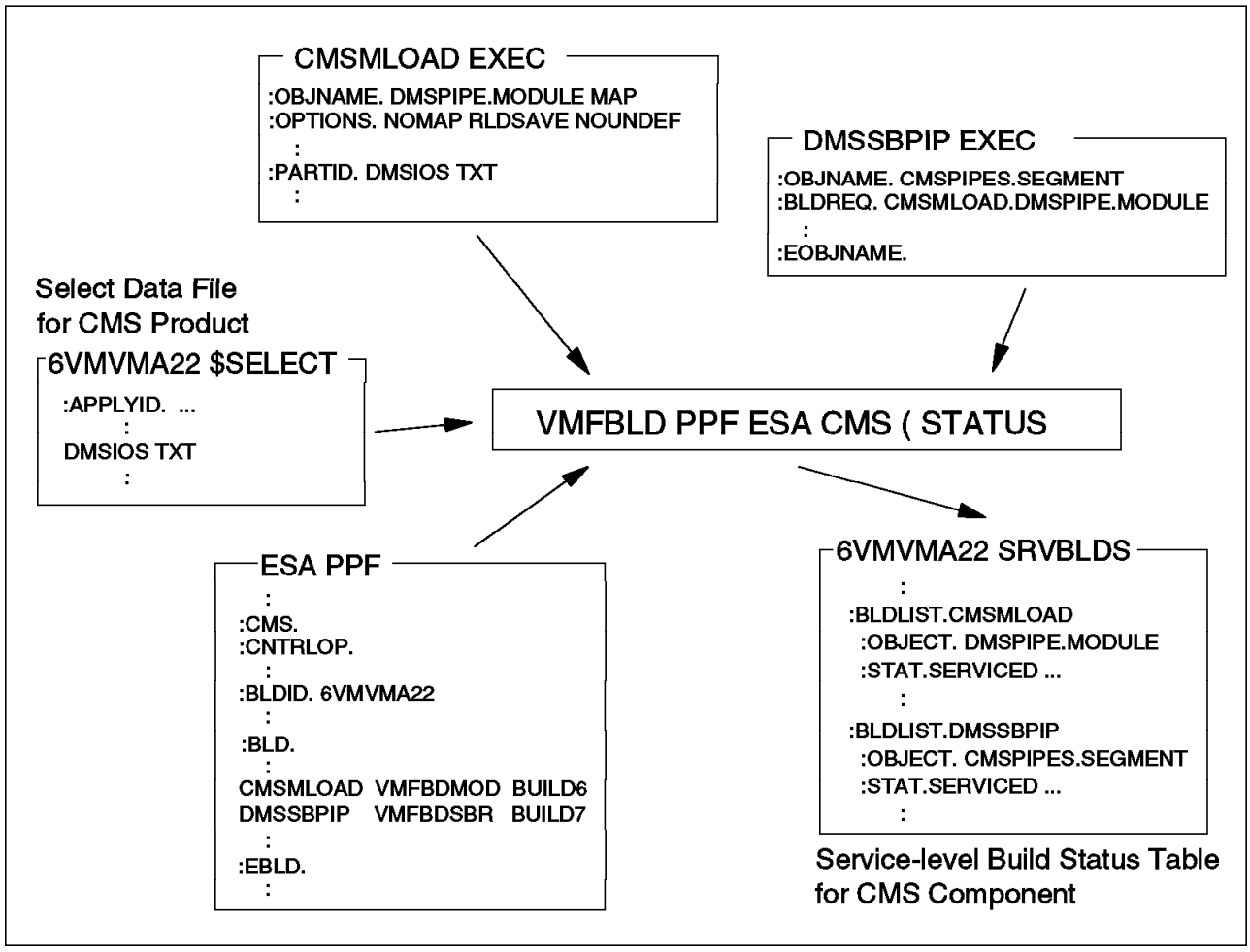



*Figure 77. Product-Level Segment Service Process - Identifying Requirements*

Please refer to Figure 78 on page 166 for the remainder of the steps.

4. When the SERVICED, or ALL, options of VMFBLD are executed, those SERVICED statuses cause the respective part handlers to be invoked. As defined in the :BLD section of the PPF, the VMFBDMOD part handler is invoked to build the DMSPIPE MODULE, and the VMFBDSBR part handler is invoked to “build” the CMSPIPES segment. Because of the build requisite, VMFBDMOD is invoked first.
5. VMFBDSBR does not build segments, in the sense the other part handlers build the other objects; instead, VMFBDSBR uses the information in the VMSESE PROFILE file to access the system level Select Data file (VMSBR

$SELECT), and, for each serviced segment, inserts a line containing the file name and abbreviated file type of the product level build list for the segment.

The service process is now complete at the product level. To build the segments we have to go to the system-level.

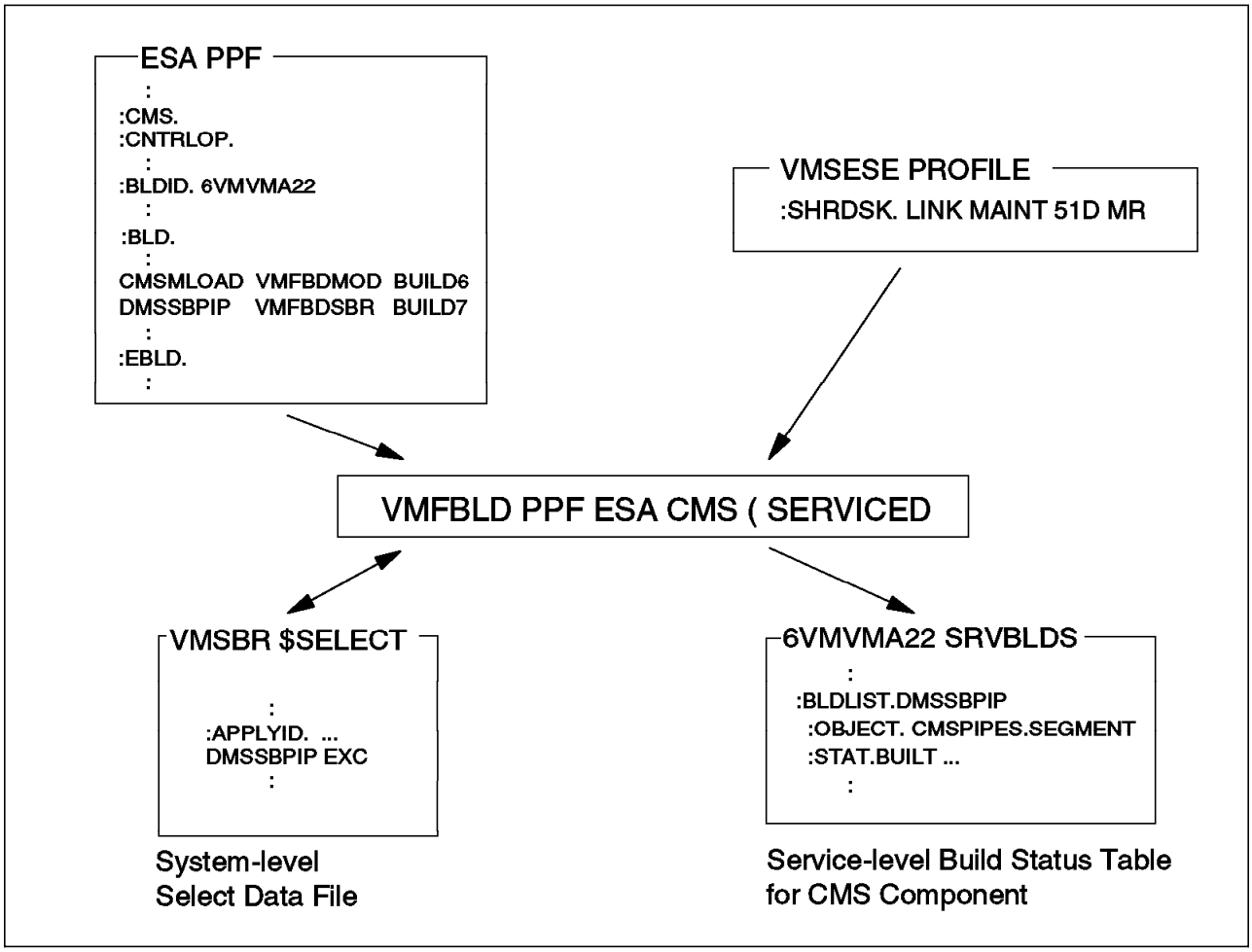



*Figure 78. Product-Level Segment Service Process - Building*

### System-Level

After doing the necessary verifications with the help of VMFSGMAP, we are ready to build the segments. As before, we will individually show the two build steps.

We invoked the VMFBLD command using:

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

The results are shown in Figure 79 on page 167. VMFBLD searches for new entries in the SEGBLD $SELECT file. In our example it finds nothing, because there was no VMFSGMAP processing. Had the user also deleted segments, the SEGBLD $SELECT file would have contained an entry which would make VMFBLD compare the EXC00000 and DEL00000 build lists. Next, VMFBLD uses the entries in VMSBR $SELECT to search the SEGBLIST EXC00000 build list, and finds the part DMSSBPIP EXC in the CMSPIPES object. It then flags this object as SERVICED in the Build Status table, VM SRVBLDS.

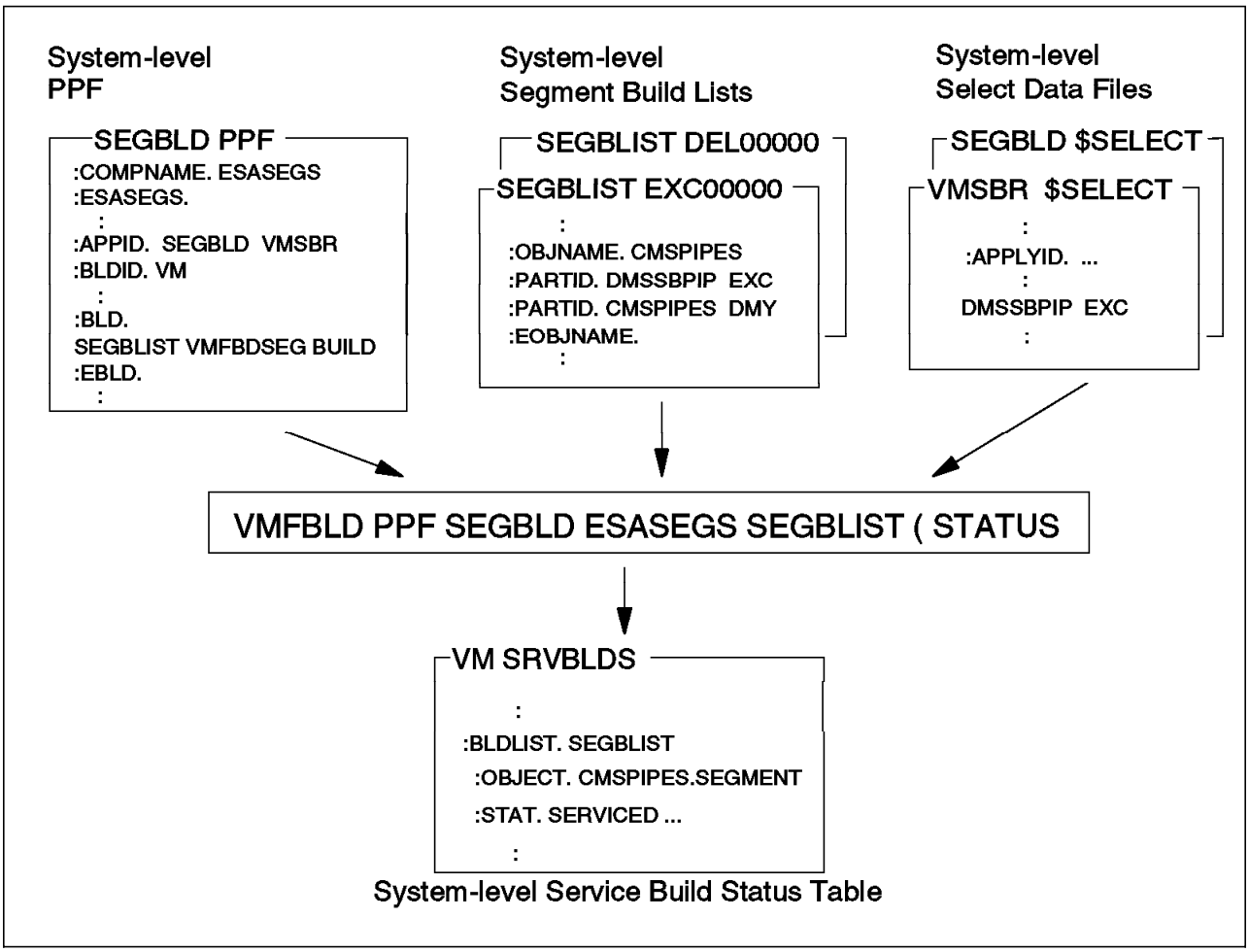



*Figure 79. System-Level Segment Building - Identifying the Requirements*

The final step is to actually do the build. We entered

```
vmfbld ppf segbld esasegs segblist ( serviced
```

The results are shown in Figure 80 on page 168. VMFBLD scans the VM SRVBLDS Build Status table, for SERVICED objects and, using the build list name (from the :BLDLIST tag) as a key, finds in the :BLD section of the PPF which part handler to call. VMFBDSEG does the building (see "VMFBDSEG Part Handler" on page 78).

In this example segments are built because of service or other changes. You might be interested in building a segment because of other reasons; for example, if the segment was accidentally purged. To force the building of a segment, even when it has not be changed, use the ALL option of VMFBLD. A sample command is:

```
vmfbld ppf segbld esasegs blname segname ( all
```

where:

**blname** is the build list name, such as DMSSBPIP

**segname** is the name of the segment, such as CMSPIPES

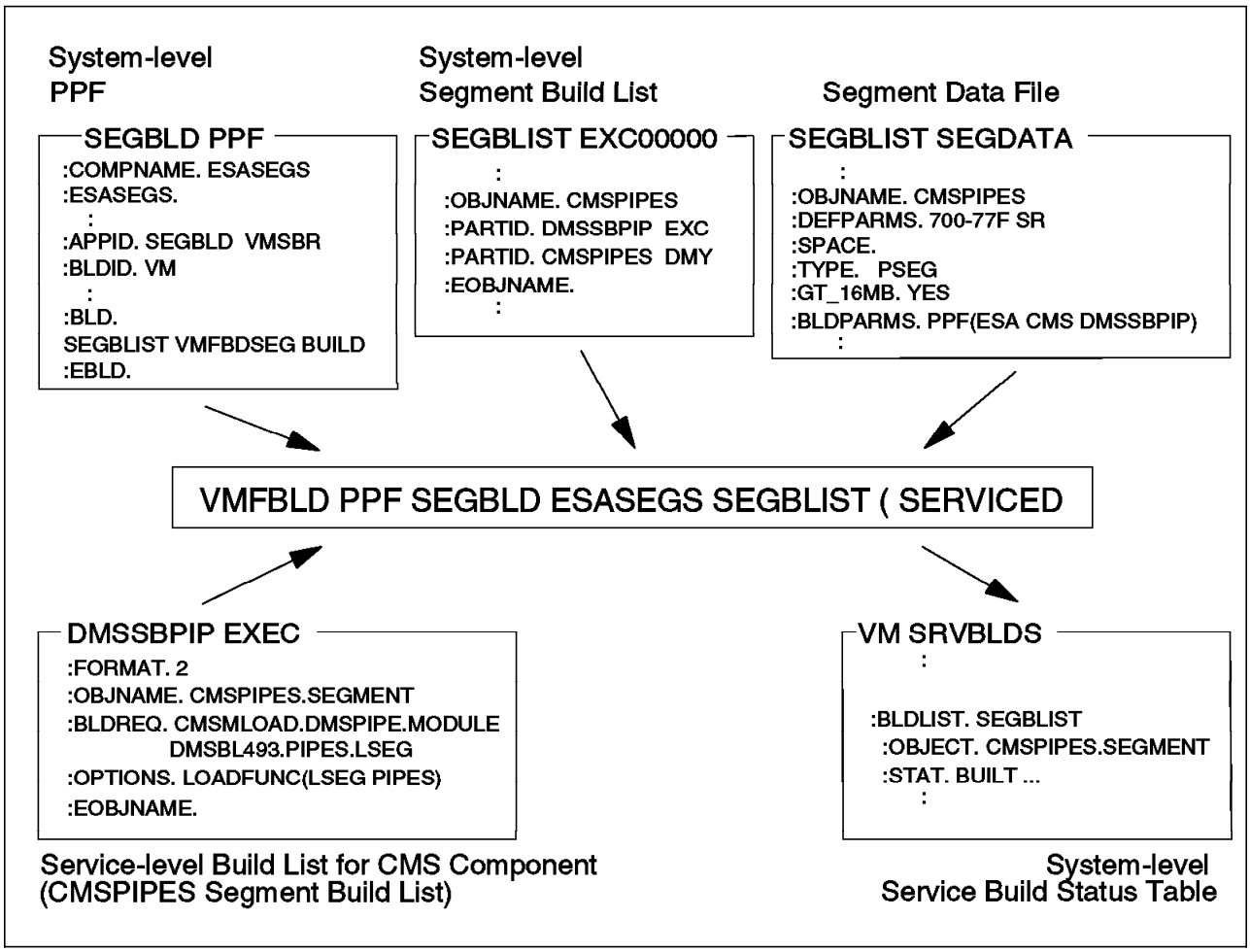



*Figure 80. System-Level Segment Building - Building*

### PRIVATE Option

Do not use the PRIVATE option of VMFBLD when building a segment, because the segment is always built on the real system.

### SEGGEN Enhancements Exploitation

On VM/ESA Release 2.2 an enhancement was made to segment processing in CMS: The logical segments' addresses are no longer obtained from the SYSTEM SEGID file. So, only if logical segments have been added to or deleted from a physical segment, is there a need to copy this file to the S-disk and re-save CMS. Message VMFBDS2003W is issued when this needs to be done.

## Deleting a Segment

From time to time it will be necessary to delete segments from your system. The segment support provides the following method for doing this:

1. Invoke VMFSGMAP
2. Select the segment you wish to delete

3. Invoke the delete function
4. File the changes

We will now detail this process:

1. Invoke VMFSGMAP. Enter:

   **vmfsgmap segbld esasegs segblist**

   The segment map, as shown in Figure 81, is displayed.

2. To select a segment, place the cursor on that line. We used the > > mark to indicate the cursor position.

3. Segment SQL320A was selected and we deleted it by pressing the PF11 Del Obj key.

```
                         VMFSGMAP - Segment Map                  More:  -
                                                              Lines 15 to 37 of 37
    CMSBAM   MEM 8.............9.............A.............RRRR............
    CMSDOS   MEM 8.............9.............A.............R.............
    CMSFILES DCS 8.............-----RRRRRRRRRRRRRRRRRRRRRRRRRRB.............
    CMSVMLIB DCS RRRRRRRRRRRRRRR9.............A.............B.............
    DOSINST  DCS 8.............R--------------A.............B.............
  M TESTPROD DCS 8.............9.............RRRRRRRRRRRRRRRRB.............

  Meg            00C-MB          00D-MB          00E-MB          00F-MB
  St  Name   Typ 0123456789ABCDEF0123456789ABCDEF0123456789ABCDEF0123456789ABCDEF
    SQLSPACE SPA ========--------D.............E.............F.............
 >> SQL320A  MEM RRRRRRRR........D.............E.............F.............
    HELPINST DCS RRRRRRRRRRRRRRRRD.............E.............F.............
  P TEST1    DCS C.............D.............E.............RRRRRRRRRRRRRRRR
  M CMS      SYS C.............D.............RRRRRRRRRRRRRRRRRRRRRRRRRRRRRR>

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

  Meg            010-MB          011-MB          012-MB          013-MB
  St  Name   Typ 0123456789ABCDEF0123456789ABCDEF0123456789ABCDEF0123456789ABCDEF
    SVM      DCS 0.............1.............2.............RRRRRRRRRRRRRRRR
  M CMS      SYS >RRRRRRRRRRRRRR1.............2.............3.............
  ============================  End Segment Map ==============================

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

*Figure 81. VMFSGMAP Display before Deleting SQL320A*

As you can see in Figure 82 on page 170, the segment SQL320A has been moved to the bottom of the display, and is only “planned for deletion.” The deletion has not yet taken place.

```
                          VMFSGMAP - Segment Map                  More:  -
                                                              Lines 22 to 36 of 36
  Meg            00C-MB          00D-MB          00E-MB          00F-MB
  St  Name   Typ 0123456789ABCDEF0123456789ABCDEF0123456789ABCDEF0123456789ABCDEF
    HELPINST DCS RRRRRRRRRRRRRRRD..............E..............F...............
  P TEST1    DCS C..............D..............E..............RRRRRRRRRRRRRRRR
  M CMS      SYS C..............D..............RRRRRRRRRRRRRRRRRRRRRRRRRRRRRR>

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

  Meg            010-MB          011-MB          012-MB          013-MB
  St  Name   Typ 0123456789ABCDEF0123456789ABCDEF0123456789ABCDEF0123456789ABCDEF
    SVM      DCS 0..............1..............2..............RRRRRRRRRRRRRRRR
  M CMS      SYS >RRRRRRRRRRRRRR1..............2..............3...............
>>P SQL320A  DCS DELETED
  ============================  End Segment Map ============================

VMFSMD2046I Segment SQL320A has been deleted
 F1= Help    F2= Chk Obj  F3= Exit      F4= Chg Obj   F5= File       F6= Save
 F7= Bkwd    F8= Fwd      F9= Retrieve F10= Add Obj  F11= Del Obj   F12= Class
====>
```

*Figure 82. VMFSGMAP Display after Deleting SQL320A*

If the cursor is placed on the SQL320A line and PF4 Chg Obj is pressed, the screen shown in Figure 83 is displayed.

```
                           Change Segment Definition
                                                          Lines 1 to 12 of 12

 OBJNAME....:  SQL320A
 DEFPARMS...:  DELETED C00-C7F SR
 SPACE......:
 TYPE.......:  SEG
 OBJDESC....:  SQL PRODUCT
 OBJINFO....:  Test version
 GT_16MB....:  YES
 DISKS......:
 SEGREQ.....:
 PRODID.....:
 BLDPARMS...:  PROD(SQLGEN)

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

*Figure 83. Change Segment Definition Panel for SQL320A*

This displays the entry in SEGBLIST SEGDATA. Notice that the storage ranges in the :DEFPARMS field have been prefixed with the keyword DELETED. Until VM/ESA Release 2.2 the keyword DELETED *replaced* the storage ranges, so you would lose those.

4. If you want to keep these changes you must exit VMFSGMAP using the PF5 File function. VMFSGMAP then updates the SEGBLD $SELECT, SEGBLIST SEGDATA, SEGBLIST EXC00000, and SEGBLIST DEL00000 files. If you exit VMFSGMAP using PF3 - Exit, all changes are discarded.

VMFSGMAP has not deleted the segment from the system. Building and deleting objects are exclusively the function of VMFBLD. To delete the segment issue the command:

**vmfbld ppf segbld esasegs segblist sql320a ( serviced**

The segment build status table, VM SRVBLDS, is also updated by this process. On VM/ESA Release 2.2 the new status for the segment is, correctly, DELETED. On previous releases, even though VMFBLD was called to delete an object, the status of SQL320A would have been BUILT because the object was not removed from the build list Consequently, issuing the command:

**vmfsim query vm srvblds * tdata :object sql320a :stat**

would show a status of BUILT, which is misleading.

This process which, in essence, replaces the CP command PURGE NSS NAME SQL320A, may seem too complicated. But this process can, as easily, execute a much more complicated set of functions: If the segment is a member of one or more segment spaces, the VMFBLD process will correctly build all the segment spaces without the systems programmer having to spend time defining and redefining segment spaces.

## Segment Requisites

On the Segment Definition panel there is a field named SEGREQ. This field allows listing segments that are prerequisites of the segment you are defining. For example, Figure 84 shows how SQL320A could be specified as a requisite of QMF310A - since SQL/DS is a PREREQ of QMF.

```
                            Change Segment Definition
                                                          Lines 1 to 12 of 12

 OBJNAME....:  QMF310A
 DEFPARMS...:  DOO-D7F SR
 SPACE......:
 TYPE.......:  SEG
 OBJDESC....:  QMF PRODUCT
 OBJINFO....:  Test version
 GT_16MB....:  YES
 DISKS......:
 SEGREQ.....:  SQL320A
 PRODID.....:
 BLDPARMS...:  PROD(SQLGEN)

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

*Figure 84. VMFSGMAP Change Segment Definition for QMF310A*

When you build QMF310A, VMFBLD will find that the SQL320A segment is a prerequisite. It checks the status of SQL320A in VM SRVBLDS. If it doesn't find the segment or the status is BUILT, then VMFBLD assumes that no further processing needs to be done on SQL320A and continues to build QMF310A. If SQL320A has a status other than BUILT, then VMFBLD will first build SQL320A, then QMF310A.

**Note:** Since VMFBLD only looks in VM SRVBLDS, and **not** on the system, for this process to work all segments defined as requisites must also be defined in the SEGDATA file.

When you invoke VMFBLD to build the prerequisite segment SQL320A, VMFBLD notes that QMF310A may be affected by this change, and alters the status of QMF310A, in VM SRVBLDS, to either SERVICED or BUILDALL, depending on how VMFBLD was invoked for SQL320A. VMFBLD does not automatically build QMF310A, since further steps may be required before the build can take place.

Before VM/ESA Release 2.2, this process had a major exposure. As explained above, when a segment was deleted it would appear in VM SRVBLDS with a status of BUILT. Thus, if SQL320A had previously been deleted, the requisite checking for QMF310A would still be satisfied, and indeed no message would be issued to inform the user of a potential problem. This will not happen on VM/ESA Release 2.2.

## Skeleton Segments on the System

VMFSGMAP compares information from the SEGBLIST SEGDATA file with the output of the QUERY NSS command. As discussed in “Overview” on page 65 there are five possible results of this comparison:

- Identical segment definitions
- Different segment definitions
- One or both definitions in error
- Segment defined only to the system
- Segment defined only in the SEGDATA file

There is only one column on the VMFSGMAP display screen to indicate this. VMFSGMAP checks the system for active segments. If it does not find an active segment, it then looks for a skeleton segment (it ignores pending purge segments). Before VM/ESA Release 2.2 there was no way of knowing whether it was an active or skeleton segment that was being compared to the SEGBLIST SEGDATA file. Therefore, in order for the systems programmer to know which segments are active on the system, it would still be necessary to issue the command:

```
query nss map all
```

On VM/ESA Release 2.2 the PF12 acts as a toggle alternatively showing on the first map column the segment status or the spool file class. In addition, the “PF2 Chk Obj” function displays the results of the QUERY NSS MAP command.

When a segment is built, VMFBLD checks for a skeleton segment, purges it when it exists, and then issues a DEFSEG command to define a new skeleton with the correct parameters. This occurs regardless of whether the segment is to be defined at a different address, different segment space, and so on.

In this way, the skeleton segment being used corresponds to the definition in the SEGDATA file.

## Disk Requirements

Disks required to build a segment can be accessed by two processes, or a combination of the two:

- If you specify a PPF value in BLDPARMS, VMFBDSEG invokes VMFSETUP to access the disks specified in the :MDA section of the PPF. Previous to Release 2.2, these disks must already be linked. In VM/ESA Release 2.2, by using the LINK option of the part handler VMFBDSEG, VMFSETUP is called with the LINK option to both link and access the specified disks.

  After building the segment, VMFBDSEG releases the disks and calls VMFSETUP to detach any disks it linked. This is very useful when building several segments for different products, since most products tend to use the same minidisk addresses. On previous releases, the disks had to be linked before invoking VMFBLD, because VMFSETUP would only access the disks.

- Using the DISKS field, on the Segment Definition panel, you can specify additional minidisks or SFS directories that are required for building a segment. This is shown in Figure 85.

```
                              Add Segment Definition
                                                          Lines 1 to 12 of 12

  OBJNAME....:  DAS1V220
  DEFPARMS...:  COO-C7F SR
  SPACE......:  VMAS
  TYPE.......:  SEG
  OBJDESC....:  THIS IS ONE OF THE VMAS SEGMENTS
  OBJINFO....:  THIS SHOULD BE BUILT WITH DAS2V220 SEGMENT
  GT_16MB....:  NO
  DISKS......:  391
  SEGREQ.....:
  PRODID.....:
  BLDPARMS...:  PROD (DASGEN)

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

*Figure 85. VMFSGMAP Add Segment Definition Panel*

When VMFBLD is invoked, the minidisks listed on the DISKS field must be already linked, and VMFBLD attempts to access them at the next available disk modes. If there is no disk linked at the address specified, VMFBLD will fail.

This capability provides a safeguard against building the segments without all the correct disks, as VMFBLD will not proceed unless it is able to access each one. If you require the disk addresses to be at specific modes (for instance, SQL/DS, which requires its 195 disk to be at file mode Q and the 193 disk to be at file mode V) then either access them manually each time you build the segment, or access them in the EXEC specified in the BLDPARMS field.

The DISKS field is primarily intended to be used for non-VMSES/E products, since VMSES/E products should specify the PPF name in the BLDPARMS field. Another possible use is to access disks from a different product; for example, to include EXECs or XEDIT macros from products other than CMS in the CMSINST segment.

When the DISKS field is not empty and the PPF is used, the specified disks are accessed and then VMFSETUP is invoked to also access the disks in the PPF.

## Changing the CMSINST Segment

For VM/ESA Release 2.2, please follow the directions in the *VM/ESA: Service Guide*. For Release 2.1, use the steps detailed in the sections below as a guideline and the *VM/ESA: Service Guide* as a detailed reference. The process described in this section cannot be used to change CMSINST on a VM/ESA Release 1.5 370 Feature system.

The CMSINST segment is defined in a CMS build list. If you want to change it by adding or deleting EXECs and XEDIT macros, you will have to change the build list. As was explained in “Local Service” on page 122, this is considered local service and you should define a local tracking number for this modification and create an entry for the build list in the 6VMVMA21 VVTLCL table (where “nn” varies with the release, see the note to Table 4 on page 91).

To change the build list follow the procedure in the *VM/ESA: Service Guide* and modify it as directed by the steps below. The following conventions are used:

**6VMVMA21** Is the value from the :APPID tag in the CMS component of the ESA PPF.

**modid** Is the local tracking number you assigned to this change, in the form “Lnnnn.”

1. As documented.
2. As documented.
3. Apply local service to CMSINST LSEG. CMSINST LSEG contains the list of EXECs in CMSINST.
   a. Determine the highest level of CMSINST LSEG.
      1. The command given in the *VM/ESA: Service Guide* only works if you have never applied local service to this part. It is preferable to use the following command:

         ```
         vmfsim getlvl esa cms tdata :part cmsinst seg ( history
         ```

         Any existing local modifications are listed first, after the tag :MOD, and IBM service is listed after the tags :VVTVM and :PTF. Use the file CMSINST SEGnnnnn, where nnnnn is the number of the first (most recent) IBM PTF.
      2. As documented.
   b. As documented.
   c. As documented.
4. If the objects you are adding to CMSINST belong to the CMS component, then you must create a local modification to the CMSINST build list, DMSSBINS EXEC. VMSES/E will then automatically flag CMSINST as having to be re-built whenever the objects are serviced. Otherwise skip this step. Since this involves updating a source file, DMSSBINS $EXEC, you need to create an update file and use VMFEXUPD.

**a** XEDIT the DMSSBINS AUXLCL file. This file must reside on the LOCALMOD disk. Add an entry for the update file:

```
VLnnnnDS LCL LCLnnnn comment
```

where:

**comment** Is any comment you want to add

**b** Create the associated update file as documented. Your changes are placed in the DMSSBINS VLnnnnDS update file.

**c** Create a replacement part for DMSSBINS EXEC, as documented.

**5** Update the 6VMVMAnn $SELECT file on the alternate APPLY disk. Add the following three lines to the **top** of the file:

```
:APPLYID.mm/dd/yy hh:mm:ss
DMSSBINS EXC EXCnnnnn
CMSINST SEG
```

where:

**mm/dd/yy** Is the current date

**hh:mm:ss** Is the current time

**nnnnn** Is the PTF number determined in 4c

**6** Rebuild the CMSINST segment.

**a** If you executed the previous step, access the CMS component disks. Enter:

```
vmfsetup esa cms
```

**b** First re-build DMSSBINS EXEC (if changed) and CMSINST LSEG. Enter:

```
vmfbld ppf esa cms dmssbins ( serviced
```

**c** Re-IPL to clear storage. Enter:

```
ipl 190 clear
```

**d** Ensure any minidisks containing the added objects are linked. Issue the appropriate CP LINK commands, if required.

**e** Rebuild the HELPINST segment, which contains CMSINST. Enter:

```
vmfbld ppf segbld esasegs segblist helpinst ( serviced
```

**f** Move the updated SYSTEM SEGID file to the S-disk and re-save CMS. If you are running VM/ESA Release 2.2, the SYSTEM SEGID will not require an update if you are only changing the address of a logical segment.

## Maintaining Segments for Multiple Systems

If you are maintaining several systems from a single site, there are a few special steps you have to do, and several important points to consider.

## PPF Considerations

You should create, for each system, one override to the system-level PPF, and keep just one component in the PPF. We think this is preferable to having one PPF with many components, where each component would correspond to a different system. The override must:

- Have the same name as the system
- Change the preferred entry in the :APPID tag to reflect the system name (and use a separate $SELECT file). You **must** keep VMSBR as the secondary name.
- Change the name in the :BLDID tag to reflect the system name.
- Change the value of the &SID variable, if you are keeping separate SIDISKS. Also update the VMSESE PROFILE file accordingly.

Also, you could:

- Define several segment layouts per system. As each layout corresponds to a build list (and a SEGDATA file), simply add one or more build lists to the :BLD section of the PPF override. You should have a minimum of two layouts: production level and test level. In this way, if something goes wrong with the test layout, you can rebuild all segments using the production layout.
- Add an entry in the :APPID tag. This allows keeping separate system-level Select Data files, as discussed in "Select Data File Considerations."

## Select Data File Considerations

VM/ESA Release 2.2 added support for multiple Select Data files and Apply Status tables, which changes the way to manage multiple systems.

### Considerations for VM/ESA Release 2.2 systems

The VMFSGMAP obtains the Select Data file name from the :APPID tag of the system-level PPF. If you are maintaining several systems from a single site we strongly suggest you override this tag.

The override should change the primary (first) name on the :APPID tag, retaining VMBSR as the secondary name. In this way, each system has its separate file for dealing with changes done through VMFSGMAP, and product service changes will be reflected in the single VMSBR $SELECT, affecting all the different systems, as expected and required.

### Considerations for VM/ESA Release 2 and VM/ESA Release 2.1 systems

Before VM/ESA Release 2.2, if you were doing some planning work with the VMFSGMAP command, and did not intend to build the resulting layout, you should save the $SELECT file under a different name before invoking VMFSGMAP. In this way, if you forgot to use the Quit function to terminate VMFSGMAP you could always restore the original file.

Overriding the PPF would allow you to have separate Select files, but as only one entry was supported in the :APPID tag, the following resulted:

- The considerations above would still apply.

- All the files would have to reside on the same disk, because the VMSESE PROFILE points to a single disk.
- The file name VMSBR is **hard-coded** in the VMFBDSBR part handler. As a net result, you **had to manually keep a copy of this file for each system**. Before building the product service for one system, you would copy that system's Select Data file to VMSBR $SELECT and save it at the end of the build step.

### Central-Site Build Considerations

If you are careful, you can even build the segments from a central site. If the same segment has different characteristics in the several systems, such as storage locations, you must name each segment differently. If the segments are exactly the same, you should have two segment definitions: one for the central system, another to build a new version of the segment. This will keep the central system safe.

Whatever technique you use, after building the segment you need to ship it to the remote system and install it there. There are two CMS Utilities Feature utilities to help you: DCSSBKUP will save a copy of the saved segments in a CMS file (of file type DCSSBKUP). Just send this file to the target systems. Then, in each target system, use the DCSSRSAV utility to install the segment on the system. As this utility also allows you to rename the segment, you can keep the names identical in all systems.

Note that, if you build a PSEG, the SYSTEM SEGID file may be changed, so you may also need to:

- Send it to each system.
- Copy it to the S-disk; you may need to tailor it at the remote system, if that system and the central-site are not identical.
- Re-save CMS on the remote system.

There are a couple of common mistakes which can be made using DCSSBKUP or DCSSRSAV to backup or restore segment spaces and physical segments.

- Never back up a segment space. Back up each member, one at a time.

  A segment space is a logical grouping of members. After it is restored, it will be considered a DCSS and not a segment space.
- Never back up a logical segment. Always back up the physical segment.

  CP is not aware of logical segments. DCSSBKUP and DCSSRSAV use CP services to access the segments.

## A few Questions and Answers Working with Segments

### Copying CMSPIPES Segment Above 16MB

Can I move the CMSPIPES segment above the 16MB line, even though I have some users who must run in 370-mode virtual machines?

Yes. You can "copy" the PIPES logical segment above the 16MB line by creating a second segment, for example, CMSPIPEH (for PIPES high), containing the same logical segment, PIPES. Then, modify your SYSPROF EXEC to check the

machine mode of the user and load the correct logical segment. Finally, SYSPROF is usually one of the EXECs in CMSINST (the installation logical segment), so you need to rebuild CMSINST to contain the new version of the SYSPROF EXEC. Follow this procedure to move the segment.

**1** Define the new segment to VMSES/E.

a. Prepare your system for building segments by accessing only the VMSES/E disks, 5E5 and 51D (see 'Rebuild the Saved Segments' in *VM/ESA: Service Guide* for a complete description of this step).

b. To get the map of your segments issue the command:

**vmfsgmap segbld esasegs segblist**

(Note, **ESASEGS** was **ESA20** in VM/ESA Release 2.0.)

- Place the cursor on CMSPIPES and press PF10 (Add Obj).
- On the Add Segment Definition panel, fill in the name for your new segment (in this example, CMSPIPEH) and change the segment range to above 16M.
- Press PF5 to keep the changes and return to the map.
- Make sure the map looks correct (for example, there are no overlaps with other required segments).
- Press PF5 to file the changes and exit the map.

**2** Build the new segment by issuing the command:

**vmfbld ppf segbld esasegs segblist cmspipeh (serviced**

Again, replace **ESASEGS** with **ESA20** for VM/ESA Release 2.0.

If you received message VMFBDS2003W, update the SYSTEM SEGID file on both the 490 disk and the 190 S-disk. If you are also going to rebuild the CMSINST logical segment (Step 5), then you can wait until then to update the SYSTEM SEGID file.

**3** Create a local modification for the SYSPROF EXEC to load the correct logical segment.

a. Access the CMS disks.

**vmfsetup esa cms**

b. Since the SYSPROF $EXEC is maintained by source updates, follow the procedure for creating a local modification with source updates in the *VM/ESA Service Guide*.

- Create or update the SYSPROF AUXLCL file on the local disk.
- Create the SYSPROF update. This update should check the virtual machine mode and load the correct segment. An example of this code is:

```
'EXECIO * CP (LOCATE /MACHINE/ STEM OUTP. STRING QUERY SET'
Parse Var outp.1 . 'MACHINE' mode ','
If mode <> '370'
  Then 'SEGMENT ASSIGN PIPES CMSPIPEH'
  Else 'SEGMENT ASSIGN PIPES CMSPIPES'
'SEGMENT LOAD PIPES (SYSTEM'
```

- In VM/ESA Release 2.2, issue VMFEXUPD to create the updated SYSPROF on the local modification disk and update the $SELECT and local VVT files (example command contains outmode, a Release 2.2 option).

  **vmfexupd sysprof exec esa cms (outmode localmod $select logmod**

  For prior VM/ESA releases, follow the local service procedures in *VM/ESA: Service Guide* for creating a replacement part from $ source files. Make sure to update the local VVT and the $select file.

**4** Build the new executable SYSPROF.

a. If you use CMSINST (the installation segment) and SYSPROF is in your version of CMSINST, then you will have to rebuild this segment in the next step (5). To inform VMSES/E of this and to build the new executable SYSPROF, issue:

   **vmfbld ppf esa cms dmssbins (serviced**

b. If you are not using CMSINST or have removed SYSPROF from it, then you need to build the SYSPROF executable and copy it to the 190 disk. To perform the build, issue:

   **vmfbld ppf esa cms dmsbl490 sysprof (serviced**

**5** Rebuild the CMSINST logical segment, if you are using CMSINST and it contains SYSPROF.

a. Prepare your system for building segments by accessing only the VMSES/E disks, 5E5 and 51D (see ′Rebuild the Saved Segments′ in *VM/ESA: Service Guide* for a complete description of this step).

b. Then issue:

   **vmfbld ppf segbld esasegs segblist helpinst (serviced**

   Again, replace **ESASEGS** with **ESA20** for VM/ESA Release 2.0.

c. If you receive message VMFBDS2003W, update the SYSTEM SEGID file on both the 490 disk and the 190 disk.

d. Copy the new SYSPROF EXEC to the 190 disk.

**6** If you updated the S-disk, rebuild the CMS Saved System.

## Copying CMSQRYH Logical Segment Above 16MB

How do I move the CMSQRYH logical segment in HELPINST above 16MB?

You can do this in one of two ways.

1. You can combine CMSQRYH logical segment with another logical segment above the 16M line. For instance, CHSQRYH could be put in CMSPIPEH when that segment is created as described in “Copying CMSPIPES Segment Above 16MB” on page 177. In this case, when using VMFSGMAP (during step 1b on page 178) the definition for the :BLDPARMS field for CMSPIPEH would need to contain entries for both PIPES and CMSQRYH and the SYSPROF EXEC should be updated to ASSIGN and load each of these logical segments.
2. Or you can create a separate segment for the CMSQRYH logical segment. In this case, just follow the same instructions for moving the logical segment

PIPES (“Copying CMSPIPES Segment Above 16MB” on page 177), using CMSQRYH as the logical segment and, say, CMSQSETH as the physical segment.

## Moving CMSQRYH Logical Segment Above 16MB

How do I move the CMSQRYH logical segment in HELPINST above 16MB, without keeping a version below 16M?

Using VMFSGMAP, you can combine the CMSQRYH logical segment with another logical segment above the 16MB line (for instance, the CMSPIPEH segment created in section “Copying CMSPIPES Segment Above 16MB” on page 177) or you can define a new segment, for example CMSQSETH, above 16MB for CMSQRYH. In either case, add PPF(ESA CMS DMSSBQYH) to the :BLDPARMS field of this segment and remove it from the :BLDPARMS field for the HELPINST segment. Since there is only one copy of CMSQRYH, you do not have to modify the SYSPROF EXEC to choose the correct copy. You only have to build both the HELPINST and the new/updated segment.

## HELP Disk is Too Large to Fit in HELPINST Segment C00-CFF

How do I fit my HELPINST segment within C00-CFF when I have such a large HELP disk?

One approach is to use VMFSGMAP to enlarge the HELPINST segment to, say, C00-D7F. Then re-build HELPINST. This might fail if CMS is using the X′D00′ MB (segment) for its own storage.

Another approach is to use VMFSGMAP to enlarge and move the segment (still under 16MB) to some available location. If no location of the correct size is available, you might consider moving the HELP logical segment into a new segment. To do this, create a new segment for HELP, for example HELPSEG, putting PPF(ESA CMS DMSSBHLP) in its :BLDPARMS field and removing it from the :BLDPARMS field for the HELPINST segment. Then build the HELPSEG and the HELPINST segments.

## Building Segments of Multiple Products from One User ID

How can I build segments for many products from one user ID?

In VM/ESA Release 2.2, the segment build process was enhanced to allow you to build segments for multiple products in VMSES/E format from one user ID.

- Create a user ID with the necessary privilege classes, storage, and an A-disk.
- Give the user ID links to the disks in the PPF of each of the products (including VM) whose segments are to be built.
- Build the segments by logging onto this user ID and issuing VMFBLD for the segments, using the LINK build list option. For example,

  **vmfbld ppf segbld esasegs segblist link * (serviced**

  will build all the serviced segments from any product by linking and detaching each product′s disks as its segments are built.

# Chapter 9. Multiple Systems and Product Versions

This chapter suggests methods for managing several systems using VMSES/E, while partially or totally sharing the software base. The focus is on how VMSES/E can help accomplish that task. Other aspects, such as backup and recovery, are not covered.

VMSES/E can also be used to manage several versions of the same program product. This allows you, for instance, to test a new version of a product before placing that version into production, or have several copies of a product, each copy tailored to the needs of a particular user group.

As lack of disk space is not unusual, you might want to consider sharing disks. Also, sharing could save you time when servicing products. There are several possibilities for disk sharing, and after a general discussion of these, the possibilities of sharing each of the disk strings in the software database are individually analyzed. The basic tool for disk sharing, the building of PPF overrides, is covered next, and finally a sharing scenario is studied in more detail.

## Managing Multiple Versions of Products

VMSES/E allows you to easily install and service several copies of the same product. For every copy, you have to create a PPF override during product installation. Product copies are distinguished by the names of the overrides.

Several copies of the same product can partially or totally share some of the logical disk strings, as detailed in “Sharing Disks” on page 185. Sharing can only be done as long as the base product code remains the same.

## Managing Multiple Systems

There are many possibilities regarding the maintenance of several VM systems by a single or multiple VM systems. They can be reduced to four basic schemes:

- All systems are independently maintained from the same VM system. No disks are shared.
- The systems are maintained from different VM systems, on different processors or LPARs, with physical DASD sharing.
- The systems are maintained from different VM systems, on different processors or LPARs, sharing only SFS directories.
- All the systems are maintained from the same VM system, and can share both minidisks and SFS directories.

The first case is simple, and is covered in “Centrally Managed Independent Systems” on page 182. Although all the disk sharing methods work, they have different degrees of complexity and problems. Physically sharing disks has several risks, as explained in “Maintaining Systems by Physically Sharing Disks” on page 182. These risks can be avoided by using CMS Shared File System directories. This scheme is described in “Maintaining Systems by Sharing SFS

Directories" on page 184. The last scheme in the list above is discussed in "Centrally Managed Disk Sharing Systems" on page 185.

These discussions assume that in each of the systems where maintenance is actually performed, only a single person at a time is involved. If this is not the case, then you have several more or less independent working groups. For the purpose of identifying the best sharing scenario, each working group can be thought of as "another system": In any real VM system these working groups will, at least, share the SESDISK and SIDISK disks.

Saved segments require special consideration. They are discussed in "Maintaining Segments for Multiple Systems" on page 175.

## Centrally Managed Independent Systems

This scheme requires little effort on your part. All maintenance and service operations are performed in one VM system, and copies of the production disks are shipped to the other systems. Even if the systems can physically share the disks, they are configured in a way that, as a minimum, prevents sharing the disks we are discussing here.

Each of the systems has its own system-level Software Inventory on a separate disk. If you also want to keep the SESDISK separate, you have to explicitly access it, in addition to the SIDISK, before doing any installation or maintenance service:

```
access ses-disk b
access sid-disk d
```

You must give a different name to each system, and use that name when invoking the VMSES/E commands. For example, to install the CMS Utilities Feature on the HQSYSVM system, whose SIDISK has the virtual address 50D, you could issue:

```
vmfins install ppf cuf cufins ( nomemo noplan sidisk 50d system hqsysvm
```

Some VMSES/E commands do not have the SIDISK and SYSTEM options. These commands use the information in the currently accessed SIDISK. Do not forget that the system name is used as the file name for the tables in the system-level Software Inventory.

The advantages of this scheme are on the operational side: systems programming skills are concentrated. The major drawbacks are that the production disks are always duplicated, and no savings in maintenance work can be realized, because all steps have to be executed for every system.

## Maintaining Systems by Physically Sharing Disks

Physically sharing minidisks between several VM images imposes some operational restrictions in order to ensure data integrity:

- While performing product installation or maintenance, all systems should turn off minidisk caching for the shared minidisks.
- At any given moment only one user, from a single VM system, can access a minidisk in R/W mode. All other users **must** access the minidisk in R/O mode.
- At all other times, all minidisks should be accessed in R/O mode by all systems.

**Warning:** In order to maintain data integrity, the CMS file system has the following limitation: minidisks to be shared must not be concurrently accessed as R/W by more than one user. If the physical disk is accessed by two or more VM systems, using the CP CSE function will provide minidisk protection across all CSE systems, similar to the protection available on a single system.

As you see, write access cannot be concurrent. This limitation requires an extra degree of coordination between the systems programmers and operations personnel of the involved systems.

See “Sharing Disks” on page 185 for detailed sharing considerations on each string. Generally, each system would have to maintain independent LOCAL and BUILD disks, as shown in Figure 86. As noted, this method has potentially severe integrity exposures, and should not be used. The only advantage is in the disk space saved. However, is it worth the risk, especially when you consider that you avoid the risk and still save space?

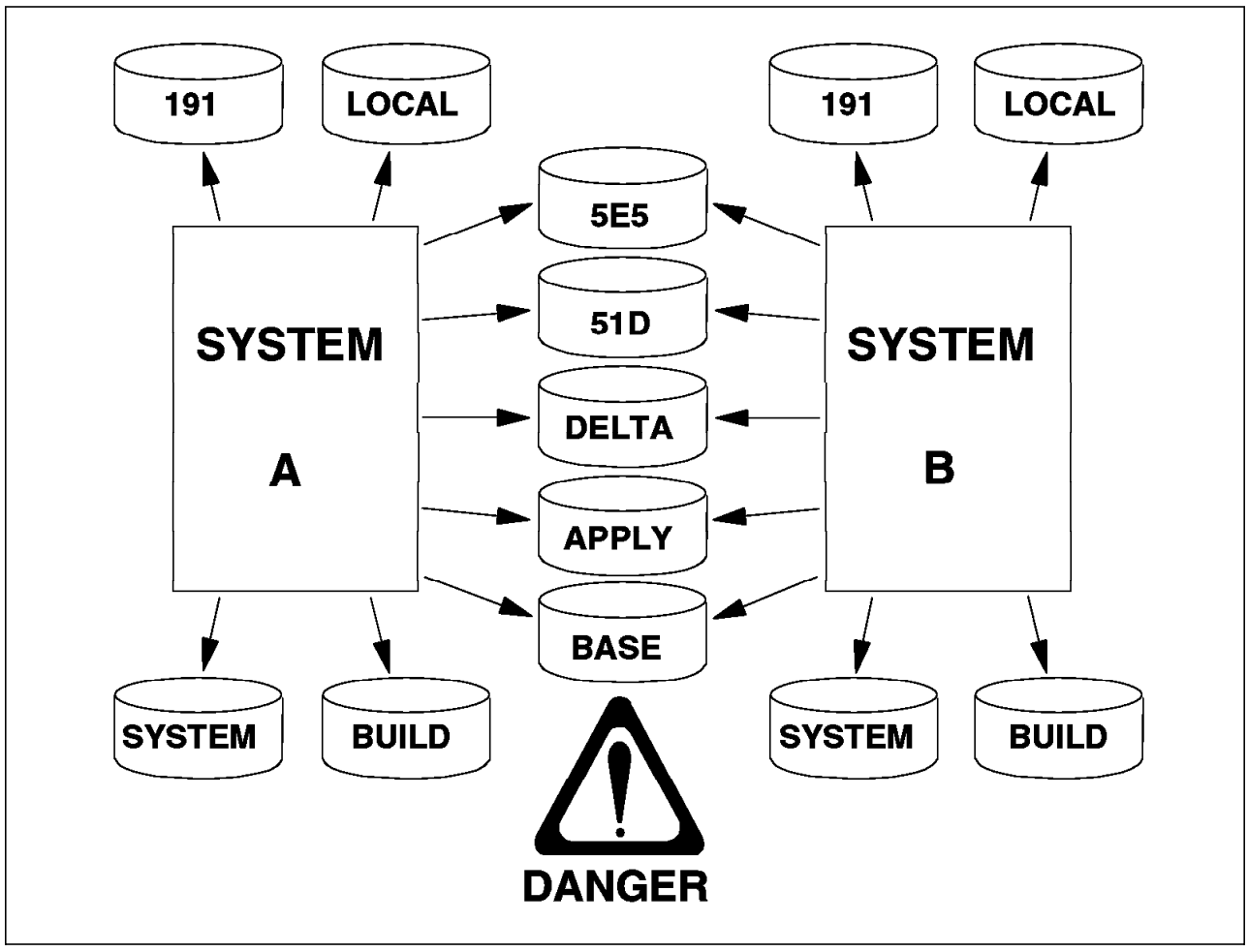



*Figure 86. Maintaining Multiple Systems with Physically Shared DASD Strings*

## Maintaining Systems by Sharing SFS Directories

To avoid the minidisk integrity exposure you can use the CMS Shared File System. Almost every minidisk involved, the SESDISK and SIDISK disks included, can be exchanged for a SFS directory. The exceptions are:

- For CMS, the system disk, system extension disk, and their alternates, by convention MAINT's 190, 193, 19E, 490, 493, and 49E minidisks.
- For GCS, the system disk, the system extension disk, by convention MAINT's 595 and 59E minidisks, as well as any disk required by a GCS application at run time.

The systems do not have to be physically close. Using TSAF, or ISFC, you can establish a VM collection in which several VM systems can share a common software base. Several VM collections can be joined through an AVS bridge. However, performance, or other reasons, may dictate that in some cases minidisks be used. Using the SFS is very attractive, in terms of space savings and the operational freedom it allows. It also requires different administrative and recovery techniques.

**Notes:**

- In VM/ESA, only the GCS, TSAF, and AVS components are supplied with alternate definitions for installation in SFS directories. The *VM/ESA: Installation Guide* contains an appendix giving detailed instructions for moving these components to SFS directories. You would have to modify the supplied materials for the other components.
- During VM/ESA installation, the CP, CMS, REXX, VMSES, and DV components are installed in minidisks before the CMS SFS is available. In order to move these components to the SFS, you would have, in essence, to re-install them. In addition, only flex-DDR tapes are available. Product tapes no longer exist. So, you would need to manually define the directories and copy the minidisks to them, as well as creating a suitable PPF override.
- Not all disks should, or can, be moved to SFS directories. If SFS has an unrecoverable error CP, CMS, VMSES, and REXX build disks should be readily available. However, some errors, such as physical damage to the CMS 190 disk, can only be quickly recovered by restoring a DDR backup.
- The *VM/ESA: VMSES/E Introduction and Reference* has a chapter that discusses how to move the system-level Software Inventory (SIDISK) to an SFS directory. This is very important if you plan to install and service products from several user IDs other than MAINT.

Two maintenance alternatives exist when using the SFS:

- Maintenance is performed from several VM systems concurrently.
- All maintenance is done from a single VM system.

Here we will discuss the first alternative, and the second is covered in "Centrally Managed Disk Sharing Systems" on page 185. As in the physically shared case, each system requires independent BUILD disks. Also, if the tailoring options or local modifications are different per system, it is more complex to share local disks than to keep them separate. However, sharing remains possible as we will see in the next sections.

In addition to the disk space saving benefits, using SFS directories allows concurrent maintenance of the several systems.

## Centrally Managed Disk Sharing Systems

A final possibility is to maintain all the systems from one central system, but share some disks. Sharing would be done from a logical point of view only.

You could use a mix of SFS directories and minidisks, or only use minidisks. All considerations about SFS made in "Maintaining Systems by Sharing SFS Directories" on page 184 also apply here. In this case, the use of SFS would be beneficial only in terms of saving disk space, because directories have no pre-allocated space, as is the case of minidisks.

As in the preceding cases, the LOCAL and BUILD strings would be separately maintained for each system. This scheme is further explained in "Case Study: Central Management" on page 191.

When compared to the previous scheme, SFS directory sharing, this scheme allows the concentration of systems programming skills.

## Sharing Disks

As the perils of physical disk sharing have been covered in "Maintaining Systems by Physically Sharing Disks" on page 182, here we discuss sharing from a logical perspective. The focus is on the individual string, not on the product or on the system as a whole.

The following discussion on disk sharing assumes that:

- The several sharing copies of a product all have the same base code.
- These copies differ operationally only in the tailoring employed, level of service applied, or both.

Both conditions are required, even if not all strings are shared, or a string is only partially shared.

VMSES/E defines the following logical disk string types:

- TASK
- LOCAL
- DELTA
- APPLY
- BUILD
- BASE
- SYSTEM

Not every minidisk or directory can or should be shared:

- Some of the disks have to reside in minidisks and physical distance between the systems might prevent sharing them.
- Even if physically possible, heavily used disks, such as the production BUILD disks, should probably not be shared for performance reasons.
- You will need different sets of tailoring information, because no two systems are *exactly* the same. Hence, you would probably want to keep separate LOCAL disks.

- The SYSTEM string includes any disks you want accessed, when maintaining a product, with a search order higher than the service database for the product. Similarly, the TASK string defines disks to be accessed with a search order lower than the service database for the product. Therefore, the following discussion does not apply to these strings.

Thus, if there is a real need to share disks between VM systems, the BASE, DELTA, and APPLY strings should be considered as the first candidates.

## Sharing LOCAL Disks

If several copies of a product have exactly the same tailoring, sharing the local disks would not present any problems. If tailoring differs, however, some additional work must be done, in order to share them.

Let us clarify this matter using CP as an example. CP tailorable files include HCPRIO, HCPSYS, and HCPBOX. The only way to keep several sets of modifications to these files in the same disk, is to use the source update structure. To distinguish between the several sets, you may use different control files, as shown in Figure 87. To share other part types the mechanism described under "Version Support for Parts" on page 120 must be used.

```
  HCPVMA CNTRL

TEXT  MACS   HCPGPI ......
PAT   AUXPAT  TX$  * LOCAL PATCHES
LCL   AUXLCLA      * SYS A Local Modifications
TEXT  AUXVM        * CP AUX FILE and VVT



  HCPVMB CNTRL

TEXT  MACS   HCPGPI ......
PAT   AUXPAT  TX$  * LOCAL PATCHES
LCL   AUXLCLB      * SYS B Local Modifications
TEXT  AUXVM        * CP AUX FILE and VVT
```

*Figure 87. Using Control Files to Manage Multiple Systems*

In addition, you have to:

- Assign different local tracking numbers to each modification.
- Create the AUX files for each local modification.
- Record in the Software Inventory all local modifications you have made. Use the local tracking number in the :MOD tag.
- Also record in the Software Inventory that you have created a change to the control file.
- Create PPF overrides that include, as a change, the new control file name.

"Local Service" on page 122 contains a detailed example of local service for source maintained parts. See also "Changing GCS" on page 131 for an

example of how to create and manage local service for a replacement maintained part.

## Sharing BASE Disks

As BASE disks contain the product raw materials before any service, they are never changed by service. As a result, they are a prime candidate for sharing, and the complete string can be shared by all copies of the product. Space savings can be quite large.

## Sharing DELTA Disks

Because VMSES/E uses the DELTA string only as a staging area for non-executable parts, the DELTA string may be shared by multiple systems.

If possible, the sharing of DELTA disks should be considered, because these disks may get very large as service piles up on them. The space savings will, hence, be considerable. Also, work savings will occur, because the Receive step is done once for all sharing products.

## Sharing APPLY Disks

If several systems are maintained at the same software level, even the APPLY string, or at least parts of it, could be shared between these systems.

The space gain here will not be as large as with the DELTA string. However, as the product gains stability, fewer parts should be affected by service on the next service level, thus lowering space requirements on the alternate disks. The major savings here will probably be in terms of work.

Let us consider three scenarios:

1. A common situation is to share the production-level APPLY disk, and let each system have separate intermediate and alternate disks. This would allow for independent testing of service in the individual environments. The production disk would contain only “safe” service. This case is illustrated in Figure 88 on page 188.
2. If you keep one or more production systems and a separate, but identical, test system where all service is tested before placing it in production, then the production systems would have no alternate disk and would share the two lowest levels of APPLY disks with the test system. The alternate level would exist in the test system only.
3. In the case of identical (“cloned”) systems all levels could be shared.

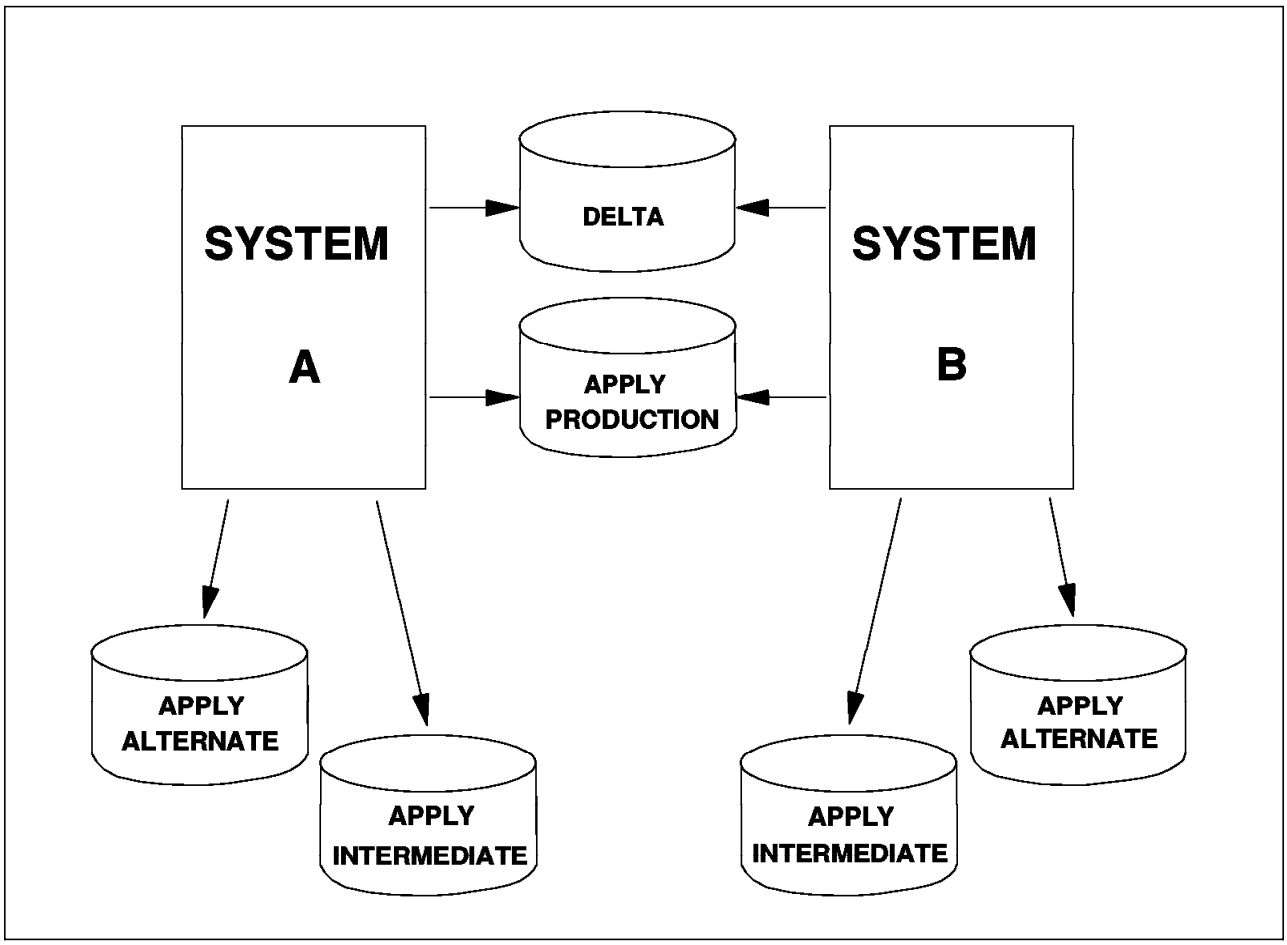



*Figure 88. Partially Sharing the APPLY String*

## Sharing BUILD Disks

Sharing BUILD disks might not be recommended for performance reasons.

If the BUILD disk you wish to share resides in a minidisk, and you are maintaining several systems from a central site, you have to choose from:

- Keeping two copies of each BUILD disk per VM system
- Having a reusable disk area as a BUILD target, then shipping the changed contents to the target system

If you decide to have a reusable area, which can save you disk space, you will have to maintain the test build disks updated. This, of course, means that either you have to get a copy of the BUILD disk from the target system, or you keep an up-to-date tape copy at the central site, restore it before starting the build, and re-save it after the build (see “Refresh” on page 100). Remember that the build process updates the Software Inventory tables, which reside on the APPLY string. Sharing BUILD disks has, then, to take into consideration the sharing of APPLY disks.

If the BUILD disk to be shared resides in an SFS directory, or maintenance for a system is performed at that same system, then the build task would be directly performed on the target disk, with due care for R/W access problems in the case of physical sharing.

## Creating a PPF Override

To implement any of the disk sharing schemes, you have to modify the supplied PPFs by using override PFFs, so that the sharing products have the same definitions for the shared disks. Thus, we will look at the task of creating a PPF override file for SYSTEM B (this system is maintained from SYSTEM A). In "Overview" on page 43, we have seen that the PPF has different areas.

As an example, let us take the PPF for CMS, called 6VMVMA22. Let us introduce a user exit into that PPF. Let us further, as suggested in Scenario 2 of "Sharing APPLY Disks" on page 187, define a new disk address for the Alternate APPLY level.

We will do this in an orderly manner, using the symbolic variable definitions for minidisks. This means replacing the variable &APPLX in the :DCL section with a new symbol, which we will call &APPLXB (for system B).

Finally, we have to update the APPLY string definition in the :MDA section to reflect the changes.

The resulting override file is shown in Figure 89.

```
:OVERLST. CMS
:CMS. CMS 6VMVMA22
:CNTRLOP. UPDATE
:USEREXIT. TESTU
:DCL. UPDATE
./DELETE &APPLX
./INSERT &APPLY BEFORE
&APPLXB LINK MAINT 9A6 9A6 MR  * Aux & software inventory files system A
./END
:MDA. UPDATE
APPLY      &APPLXB &APPLY &APPLZ  * Aux & software inventory
:END.
```

*Figure 89. Sample PPF Override - CMSB $PPF*

The :OVERLST tag must be present. It tells the VMFOVER command which component areas exist in this file.

The next tag (:CMS) starts the override area for the CMS component. The first parameter, CMS, refers to the corresponding tag (component area) in the base (or lower level) PPF file. This PPF has the name 6VMVMA22, as indicated by the second parameter.

Then we identify the sections we want to change. We can replace an existing line by including a new line with the same identification (the first word on the line). The old line will then be commented out and the new line inserted after it. We used this technique to change the :USEREXIT tag.

In the next section we cannot repeat this technique, because we are going to replace the line starting with &APPLX with a line starting with &APPLXB. In this case we use an update control record, similar to the one used in CMS UPDATE files. The ./DELETE command deletes the line staring with &APPLX and the ./INSERT command inserts all the following lines, up to the ./END line, before the line starting with &APPLY.

Finally, we replace the APPLY line in the :MDA section with our updated line.

The :END tag signals the end of the component override area.

Now, to test this new file and generate the usable form of the new PPF, it has to be compiled with the command:

**vmfppf cmsb cms**

This command implicitly invokes VMFOVER to generate the temporary source file that VMFPPF than checks for syntax errors, and compiles into CMSB PPF. Any errors found can be directed to a message log, so you can examine and correct them.

Note that the ESA $PPF actually is an override file. It is a special override file whose sole purpose is to produce a common PPF for all VM/ESA Release 2.2 components. It provides an excellent aliases file, so you do not have to remember, for example, that the component names for CMS and GCS are, respectively, 6VMVMA22 and 6VMVML22. The ESA $PPF is shown in Figure 90.

```
*===============================================================
* Start of Product Header - List of Components in ESA
*===============================================================
:OVERLST. CMS CP AVS REXX TSAF DV VMSES GCS
:OVERLST. AVSSFS TSAFSFS GCSSFS AVS370 AVS370SFS
*===============================================================
* End of Product Header
*===============================================================
:CMS. CMS 6VMVMA22
:END.
:CP. CP 6VMVMB22
:END.
:AVS. AVS 6VMVMD22
:END.
:REXX. REXX 6VMVMF22
:END.
:TSAF. TSAF 6VMVMH22
:END.
:DV. DV 6VMVMI22
:END.
:VMSES. VMSES 6VMVMK22
:END.
:GCS. GCS 6VMVML22
:END.
:AVSSFS. AVSSFS 6VMVMD22
:END.
:TSAFSFS. TSAFSFS 6VMVMH22
:END.
:GCSSFS. GCSSFS 6VMVML22
:END.
:AVS370. AVS370 6VMVMD22
:END.
:AVS370SFS. AVS370SFS 6VMVMD22
:END.
```

*Figure 90. ESA Override File - ESA $PPF*

## Case Study: Central Management

Figure 91 on page 192 shows a Central Management disk layout in a situation where the systems are remote from each other, so disks cannot be physically shared. All disks are managed by system A. However, this same setup could be implemented, with due cautions, for physically close, disk-sharing systems.

In this setup, only the BUILD disks would have to be copied across the network to the remote system. This could be automated using the VMSES/E exit. Once a build has been done for a remote system, the CLEAN-UP user exit could check the Build Status table for new and changed files on the BUILD disks and send them to the remote system.

With the support for CP configurability, even shipping the CP nucleus becomes an easy task. Generating the nucleus as a CMS module file allows it to be sent anywhere by using the SENDFILE command.

Finishing the build process could then be done directly on the remote system, or through PROP, or a similar facility, for example VM/DSNX. The type of functions to be performed would include:

- Placing the CP nucleus on the appropriate PARM disk.
- Generating and saving Saved Systems (for saved segments see “Maintaining Segments for Multiple Systems” on page 175).
- Manually updating the Software Inventory, to reflect the Build Status of the system.

For more information about the VMSES/E user exit, please refer to “Control Options Section” on page 46.

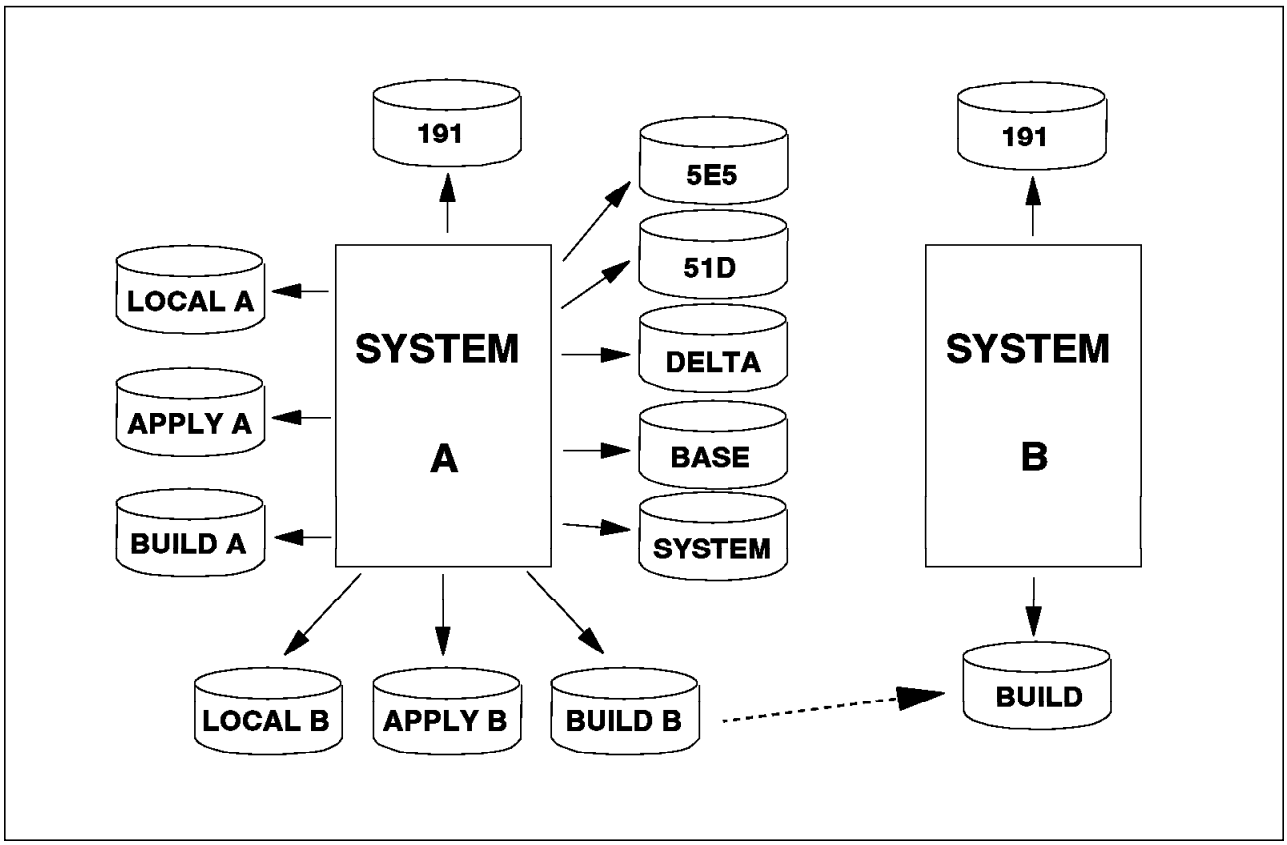



*Figure 91. Central System Management*

**Note:** The RESOURCE option of VMFINS should be used only on the “base” system. The disk and user ID definitions for which there is a need for duplication should be handled manually, as you may want to share some of the disks.

To implement the scheme depicted in Figure 91, new $PPF overrides reflecting that configuration should be defined. In our example we will concentrate only on the CP component (6VMVMB22).

We define CPSYSA $PPF for System A, and CPSYSB $PPF for System B. As a matter of fact, System A is the “base” system, so the ESA $PPF could be used. A sample PPF override for System A, CPSYSA $PPF, is shown in Figure 92.

```
:OVERLST. CP
:CP. CP 6VMVMB22
:END.
```

*Figure 92. Sample PPF Override - CPSYSA $PPF*

As you can see in Figure 92, the CPSYSA $PPF is nothing more than an “alias” $PPF similar to the ESA $PPF itself (Figure 90 on page 190).

Now, in order to get a usable form PPF, enter:

**vmfppf cpsysa cp**

This command produces the usable form, CPSYSA PPF.

Figure 93 on page 193 shows the PPF override file for System B.

```
:OVERLST. CP
:CP. CP 6VMVMB22
*=============================================================
* Control Parameters
*=============================================================
:CNTRLOP. UPDATE
:USEREXIT. SEND_B         * SYS B User exit EXEC called for
                                * setup and cleanup at the
                                * beginning and end of each
                                * service function
:ECNTRLOP.

*=============================================================
* Variable definitions
*=============================================================
:DCL. UPDATE
&LMODZ LINK MAINT B2C4 B2C4 MR  * SYS B Disk for local mods
&APPLX LINK MAINT B2A6 B2A6 MR  * SYS B Aux & software inventory files
&APPLY LINK MAINT B2A4 B2A4 MR  * SYS B Aux & software inventory files
&APPLZ LINK MAINT B2A2 B2A2 MR  * SYS B Aux & software inventory files
&BLD2Z LINK MAINT B193 B193 MR  * SYS B CMS system tools
&BLD5Z LINK MAINT B19D B19D MR  * SYS B Help disk
&BLD6Z LINK MAINT B49O B49O MR  * SYS B Test CMS system disk
&BLD7Z LINK MAINT B493 B493 MR  * SYS B Test CMS system tools
:EDCL.
:END.              * End of the definitions for CP component
```

*Figure 93. Sample PPF Override - CPSYSB $PPF*

Notice that some changes have been made. A user exit EXEC named SEND_B has been included. It allows you, for example, to send to System B any newly built objects. Also, new sets of LOCAL, APPLY, and BUILD minidisks (addresses Bxxx) have been defined. In this way, the systems programmer can service System B in a completely isolated environment.

The :CNTRLOP and :ECNTRLOP tags are not really needed. If you specify them, the only effect is that the original tags are commented out in the usable form PPF. This is the only section where you do not have to specify the starting and ending tags, because all lines in this section either are a comment or start with a tag. This is not the case for other sections.

Remember to create a usable form PPF for CP, in System B. Enter:

**vmfppf cpsysb cp**

This command produces the usable form file CPSYSB PPF. An excerpt of the final PPF for system B is shown in Figure 94 on page 194.

Now all you have to do is to invoke the VMSES/E commands giving the right PPF for each system.

```
:COMPNAME. CP
*=========================================
*====>   * Temporary override of CP component
*====>   * in PPF of product 6VMVMB22.
*=========================================
*====>   * The following overrides have been applied:
*=========================================
*====>   * Override CP from the CPSYSB $PPF.
*=========================================
      ⋮
:CP.
:PRODID.  6VMVMB22%CP
      ⋮
*================================================================
* Control Parameters
*================================================================
:CNTRLOP. UPDATE
*:CNTRLOP.

* TAG      VALUE(S)
*--------- ---------

      ⋮
:USEREXIT. SEND_B              * SYS B User exit EXEC called for
*:USEREXIT.                     * User exit EXEC called for
      ⋮
```

*Figure 94 (Part 1 of 2). CPSYSB PPF*

```
      ⋮
*=============================================================
* Variable definitions
*=============================================================
:DCL.
*&LMODZ LINK MAINT 2C4 2C4 MR  * Disk for local mods
&LMODZ LINK MAINT B2C4 B2C4 MR  * SYS B Disk for local mods
&SAMPZ LINK MAINT 2C2 2C2 MR  * Sample files
&DELTY LINK MAINT 2D6 2D6 MR  * CP service
&DELTZ LINK MAINT 2D2 2D2 MR  * CP service
*&APPLX LINK MAINT 2A6 2A6 MR  * Aux & software inventory files
&APPLX LINK MAINT B2A6 B2A6 MR  * SYS B Aux & software inventory files
*&APPLY LINK MAINT 2A4 2A4 MR  * Aux & software inventory files
&APPLY LINK MAINT B2A4 B2A4 MR  * SYS B Aux & software inventory files
*&APPLZ LINK MAINT 2A2 2A2 MR  * Aux & software inventory files
&APPLZ LINK MAINT B2A2 B2A2 MR  * SYS B Aux & software inventory files
&BAS2Z LINK MAINT 194 194 MR  * Install disk for object files
&BAS3Z LINK MAINT 394 394 MR  * CP source and macro definition
*&BLD2Z LINK MAINT 193 193 MR  * CMS system tools
&BLD2Z LINK MAINT B193 B193 MR  * SYS B CMS system tools
*&BLD5Z LINK MAINT 19D 19D MR  * Help disk
&BLD5Z LINK MAINT B19D B19D MR  * SYS B Help disk
*&BLD6Z LINK MAINT 49O 49O MR  * Test CMS system disk
&BLD6Z LINK MAINT B49O B49O MR  * SYS B Test CMS system disk
*&BLD7Z LINK MAINT 493 493 MR  * Test CMS system tools
&BLD7Z LINK MAINT B493 B493 MR  * SYS B Test CMS system tools
      ⋮
*=============================================================
* Mdisk/Directory Access    -define symbolic disk names
*=============================================================
:MDA.
*STRINGNAME MINIDISKS
*---------- ---------
LOCALMOD    2C4                 * Disk for local mods
LOCALSAM    2C2                 * Sample files
APPLY       B2A6 B2A4 B2A2      * AUX & SOFTWARE INVENTORY
                                * files
DELTA       2D2                 * CP SERVICE
BUILD7      B493                * Test CMS system tools
BUILD6      B49O                * Test CMS system disk
BUILD5      B19D                * Help disk
BUILD2      B193                * CMS system tools
      ⋮
:END.             * End of the definitions for CP component
```

*Figure 94 (Part 2 of 2). CPSYSB PPF*

# Appendix A. Comparing VMSES/E to Previous Systems

This appendix is intended as an update for systems programmers familiar with previous VM systems.

The VM systems considered here are:

- VM/SP (and HPO) Release 5
- VM/SP (and HPO) Release 6
- VM/XA SP Releases 2.0 and 2.1
- VM/ESA Release 1.0 370 Feature and VM/ESA Release 1.5 370 Feature
- VM/ESA Release 1.0 ESA Feature, VM/ESA Releases 1.1, 2, 2.1, and 2.2

These can be grouped into three “generations” of VM operating systems:

- **Pre-VMSES VM:** VM/SP Release 5 and VM/HPO Release 5
- **VMSES VM:** VM/SP Release 6, VM/HPO Release 6, VM/XA SP Release 2 and 2.1, and VM/ESA Release 1.0 (both 370 and ESA features)
- **VMSES/E VM:** VM/ESA Releases 1.1, 2, 2.1, 2.2, and VM/ESA Release 1.5 370 Feature

As we know, VMSES/E has three primary functions:

- Product installation
- Service application
- Software Inventory management

In a task-oriented approach, we briefly inventory the primary tools available in the several VM “generations” to perform these functions. Then we highlight the differences between VMSES and VMSES/E. This appendix also includes two reference tables:

- Table 9 on page 202 compares installation and service tools, by function, and allows you to quickly select the correct VMSES/E tool.
- Table 10 on page 203 shows in which VM/ESA Release VMSES/E functions become available.

The following manuals contain additional information:

- *VM/ESA: Conversion Guide and Notebook for VM/SP, VM/SP HPO and VM/ESA 370 Feature*
- *VM/ESA: Conversion Guide and Notebook for VM/XA SP and VM/ESA*

## Pre-VMSES VM

These systems did not have any consistent form of product installation and service. Also, product and service information was scarce and diluted.

### VM System Installation

ITASK was the main installation tool. It invoked other EXECs such as SPLOAD and SPGEN, and commands such as FORMAT.

VM/IS constitutes an exception because the DDR command and other tools, such as INSTPKG and MIGAID, were used instead.

### Program Product Installation

INSTFPP was the high-level manager for product installation under VM. At installation time it could be used after the (optional) execution of the DIRECGEN EXEC. The DIRECGEN EXEC performed many useful modifications in the VMUSERS DIRECT file, based on the PROGPROD PARMLIST file.

### VM System Service

The tools used depended on the type of service you had, corrective, preventive, ZAPs, or local updates. However, the most common tool was VMSERV EXEC, which was used with the Preventive Service to control the individual service EXECs on the system PUT. In addition, a great deal of manual work had to be done. Each product had its own disks and service methods, which lead to confusion.

### Servicing SNA Products

The service programs used were VMFMERGE, VMFREMOV and VMFZAP. A disk structure similar to that of VMSES was implemented, which resulted in an easier service process.

### Software Inventory

These systems did not have any tools or structure in this area. However, some EXECs generated some useful, yet incomplete, files. For example, DIRECGEN produced a DIRECGEN HISTORY file, INSTFPP produced a PROD LEVEL file, and VMSERV originated a SERVICE DISKMAP file.

---

## VMSES VM

These systems saw the introduction of a service tool, mainly for use by the VM SCP itself.

### VM System Installation

ITASK was the main installation tool. It used other EXECs such as SPLOAD and VMFBLD, and commands such as FORMAT.

Once again, VM/IS constitutes an exception. It mainly used the DDR command and other tools, such as INSTPKG and MIGAID.

### Program Product Installation

INSTFPP was the high level manager for product installation under VM. At installation time it could be used after the (optional) execution of DIRECGEN, which performed many useful modifications in the VMUSERS DIRECT file, based on the PROGPROD PARMLIST file.

### VM System Servicing

This was the first appearance of VMSES. The service process was defined as executing three major tasks:

- Receiving the service
- Applying the service
- Building the serviced objects

VMFREC, VMFAPPLY, and VMFBLD were the main EXECs in this new process.

### Servicing SNA Products

The service programs used were VMFMERGE, VMFREMOV, and VMFZAP. A disk structure similar to the one of VMSES was implemented. This structure was separate from the VMSES one.

### Software Inventory

When compared to previous systems, VMSES provided a substantial quantity of new information. However, there was not a truly reliable software inventory. By using the FILELIST command, one could easily list the PTFs and APARs installed. Also, some EXECs generated some useful, yet incomplete, files. For example, DIRECGEN produced a DIRECGEN HISTORY file and INSTFPP produced a PROD LEVEL file.

---

## VMSES/E VM

This release marks the appearance of a true Installation and Service tool, both for the VM SCP and program products.

### VM System Installation

VMFINS EXEC is used. Refer to “VMFINS Command” on page 83 for a discussion of its many and powerful options. However, current offerings rely on a process named flex-DDR. “Installing VM/ESA” on page 18 discusses the flex-DDR process.

### Program Product Installation

Once again, VMFINS is used. Some of its operands, such as MIGRATE and DELETE, give this tool a scope and flexibility that INSTFPP EXEC never had. Please note that program products that are not supplied in PDI or VMSES formats still use INSTFPP.

### VM System Servicing

VMFREC, VMFAPPLY, and VMFBLD are the main EXECs in this process. However, their roles have been revised and redefined, resulting in easier processing and a net performance improvement.

### Servicing SNA Products

The service programs still used are VMFMERGE, VMFREMOV, and VMFZAP, with a disk structure similar to, but independent of, the VMSES/E structure.

### Software Inventory

Now a real Software Inventory is provided. Its contents are updated during the installation and service tasks, and there is an invaluable tool to help manage the Software Inventory: the VMFSIM EXEC. It is described in Chapter 3, “Software Inventory” on page 37.

## Differences Between VMSES and VMSES/E

A detailed discussion of the differences between VMSES/E and VMSES is beyond the scope of this publication. Here we will highlight only the major differences:

- Many new features and enhancements were added to VMSES/E, when compared with the old VMSES.
- Tools and structures such as VMFINS, VMFSIM, and the Software Inventory, were introduced with VMSES/E.
- The most important differences can be pointed out in the three major service application tools.

### VMFREC

VMFREC has been assigned one simple task: read materials from the installation and service media and place them on a designated disk. During product installation, VMFINS calls VMFREC. During product service, it is directly called by the user, and its job is only to read PTFs from the tape (or electronic envelope) and load them into the DELTA disks.

The heavy I/O involved to obtain PTF/APAR history, and other requisite information, has been reduced with the PTFPART files and Version Vector tables introduced by VMSES/E.

The part handlers called by VMFREC are very selective in what they load from the VMSES/E formatted tapes. To attain that selectivity, the part handlers get their information from the Software Inventory. For example, neither usable forms nor any PTF flagged as “committed” in the Receive Status table is loaded.

Also, the merge function that the old VMFREC performed has been deleted. Judgement of when to perform that operation, and its control, now lies with the user, and a new tool, VMFMRDSK EXEC, is supplied. The VMSES/E commands do not call VMFMRDSK.

### VMFAPPLY

A new algorithm has been introduced in the apply process. Performance is the keyword that characterizes the major enhancements VMFAPPLY received with its new version.

File mode numbers are no longer used as indicators of apply status. That information is kept in the Software Inventory, and VMFAPPLY fetches it from there.

Now there is a true Exclude list. PTFs listed there will **not** be applied even if they are requisites of PTFs on the Apply List.

Another important capability of VMFAPPLY is the TEST option, which enables a very accurate planning of the apply process.

### VMFBLD

VMSES/E implements a very useful file: the product Select Data file, which contains information about which parts have been serviced. This file is built, or updated, during the apply process.

**Note:** A similar concept applies to saved segments support.

VMFBLD gets information from that file, so it can rebuild only the objects with serviced parts. This saves a great deal of time.

Yet another new performance capability is the FASTPATH option, which permits reusing the nucleus load list when only HCPRIO or HCPSYS have been changed.

Also new, build lists specify how to build objects. The old load lists are now called Format 1, but remain unchanged. Format 2 build lists are new and allow the definition of many different objects, and their build options, on the same list. Format 3 build lists, introduced in VM/ESA Release 2, support building of libraries.

VM/ESA Release 2 introduced many other enhancements, such as, support for saved segments, object requisites, global libraries specification, object definition change detection, and building the CP nucleus as a CMS module file.

## Summary Tables

If you are familiar with servicing one of the previous VM systems, you can use Table 9 on page 202 as a quick guide to locate the correct VMSES/E tool for a given action.

Table 10 on page 203 shows major VMSES/E functions per VM/ESA release, so you can easily find whether a function is available on your system.

*Table 9. Comparing VMSES/E and Previous Systems Tools*

| Task | Pre-VMSES | VMSES | VMSES/E |
|---|---|---|---|
| | | **Installation** | |
| *VM System* | | | |
| Install | ITASK | ITASK | VMFINS (•) Flex-DDR (•) |
| Migrate | ✋ | ✋ | ✋ |
| Delete | ✋ | ✋ | VMFINS |
| *Program Products* | | | |
| Plan | ✋ | ✋ | VMFINS |
| Install | DIRECGEN+INSTFPP+ ✋ | DIRECGEN+INSTFPP+ ✋ | VMFINS |
| Migrate | ✋+INSTFPP | ✋+INSTFPP | VMFINS |
| Delete | ✋ | ✋ | VMFINS |
| New Copy | ✋+INSTFPP+ ✋ | ✋+INSTFPP+ ✋ | VMFINS |
| | | **Service** | |
| *VM System* | | | |
| Merge | ✋ | VMFREC | VMFMRDSK |
| Receive | ✋ or VMSERV (•) | VMFREC | VMFREC |
| Apply | ✋ | VMFREC/VMFAPPLY | VMFAPPLY |
| Build | ✋ | VMFBLD + ✋ | VMFBLD (✋) |
| *Program Products* | | | |
| Merge | ✋ | VMFREC | VMFMRDSK (•) |
| Receive | ✋ or VMSERV (•) | ✋ or VMSERV (•) | VMFREC (•) |
| Apply | *service* EXEC + ✋ | *service* EXEC + ✋ | VMAPPLY (•) |
| Build | *service* EXEC + ✋ | *service* EXEC + ✋ | VMBLD (•) |
| | | **Software Information** | |
| Tools | FILELIST + XEDIT | FILELIST + XEDIT | VMFSIM + VMFQOBJ + VMFINFO (•) |

**Notes:**

✋ Manual effort

(✋) Manual effort only on specific parts

- • VM/ESA Release 1.1 and VM/ESA Release 2
- • VM/ESA Release 2 and above
- • If PUT tape
- • If VMSES/E product

*Table 10. VMSES/E Function Availability per VM/ESA Release*

| Function Description | Rel. 1.5 370 Feat. | Rel. 2.2 | Rel. 2.1 | Rel. 2 | Rel. 1.1 |
|---|---|---|---|---|---|
| **Installation** | | | | | |
| File pool flexibility | | √ | | | |
| **Service**<br>*Local service* | | | | | |
| LOGMOD option | √ | √ | √ | | |
| OUTMODE and $SELECT options (VMFASM, VMFHASM, VMFHLASM, VMFNLS, and VMFEXUPD) | | √ | | | |
| VMFEXUPD | | √ | | | |
| CP load list modification (GENCPBLS) | | √ | | | |
| *Standard service* | | | | | |
| High Level ASSEMBLER support | √ | √ | √ | √ | √ (•) |
| PSU/RSU planning, local service identification | | √ | | | |
| VMFPPF multi-compile | | √ | | | |
| *Building* | | | | | |
| MACLIB, TXTLIB, and LOADLIB | √ | √ | √ | √ | √ (•) |
| DOSLIB | | √ | √ | | |
| CSLLIB | √ | √ | √ | | |
| Support of all TXTLIB options | √ (•) | √ | √ (•) | √ (•) | √ (•) |
| Enhanced version support for parts | √ | √ | √ | √ | |
| Object build requirements | | √ | √ | √ | |
| Object definition change detection | √ | √ | √ | √ | |
| Generated object support | √ | √ | √ | | |
| CP configurability | | √ | √ | √ | |
| Test build and VMFBLD LIST operand | | √ | | | |
| **Software Inventory** | | | | | |
| VMFQOBJ and VMFINFO | √ | √ | √ | √ | |
| **Saved segments support** | | | | | |
| Segment build support | | √ | √ | √ | |
| VMFSGMAP | | √ | √ | √ | |

**Notes:**

- Requires APAR VM54803
- Requires APAR VM54804
- Requires APAR VM57453
- Requires APAR VM57429

# Appendix B. Product Packaging and Distribution Media Formats

In today's market, there is a great diversity of vendors and products. The way they are packaged for installation and service has evolved over the years, so that there are many product-unique formats available today.

This appendix summarizes the different product packaging and distribution media formats supported by VMSES/E.

**Note:** The flex-DDR tape format is **not** covered here. See *VM/ESA: Installation Guide* for such information.

## Distribution Media

VMSES/E supports three media types: tape (either reel or cartridge), CD-ROM (see "CD-ROM" on page 17), and electronic envelopes (see "Electronic Envelopes" on page 17).

Using CD-ROM requires special software that makes the CD-ROM appear to the system as a tape. Therefore, for simplicity, in the following sections we employ the term "tape" to refer to tapes, CD-ROMs, and electronic envelopes.

## Product Formats and Naming Conventions

There are several different installation and service conventions existing amongst VM program products. The following sections detail the formats, and review the conventions required to allow a flat VMSES/E name space. Without a single format, and adherence to VMSES/E control files naming standards, VMSES/E will never be able to distinguish the hundreds of program products at the time when they will be all serviced by VMSES/E.

## Product Formats and Product Packaging Formats

A *product format* defines the control information required to automate the installation and servicing of products.

VMSES/E handles the following types of product formats:

- VMSES/E
- Non-VMSES/E (INSTFPP)

The VMSES/E format enables VMSES/E to perform a more extensive set of functions. Products using this format include two files containing product definition and control information: the Product Parameter File (*prodid* $PPF) and the PRODPART file (*prodid* PRODPART). This information enables the VMFINS command to execute the plan, install, migrate, build, and delete functions.

Installation of non-VMSES/E-formatted products is supported via INSTFPP. Thus, VMFINS calls INSTFPP to do all processing, and records some information on the system-level Software Inventory.

A *product packaging installation format* defines the structure and control information of the distribution media used to install the product. VMSES/E handles the following product packaging installation formats:

- VMSES/E
- Parameter Driven Installation (PDI)
- INSTFPP
- Other (for non-VMSES/E-formatted products shipped in separate tapes).

A *product packaging service format* defines the structure and control information of the distribution media used to service the product. VMSES/E handles the following product packaging service formats:

- RSU, for VMSES/E-formatted products only. VMFINS does the processing.
- PUT, for non-VMSES/E-formatted products. VMSES/E calls VMSERV to do the actual processing.
- COR, for VMSES/E-formatted products. VMFREC does the processing.
- COR, for non-VMSES/E-formatted products only. Processing is defined by the product.

## VMSES/E Enabled Program Product Conventions

Each product has a user ID from which installation and service are done. Each product will also have its own set of service disks. Segments can be built from the product user ID or from a central user ID that you can set up. See "Building Segments of Multiple Products from One User ID" on page 180 for these instructions.

There is a set of general rules that program products are encouraged (but not forced) to follow. Here is a brief synopsis for a product:

1. Each product is assigned a 3-character product prefix.
2. The *prodid* for a product is its product number followed by a letter. The letter changes for each release of the product. The *prodid* is the file name of the PPF and the PRODPART files.
3. Each product has its own user ID that it uses for installation and service. The user ID name is a "P" followed by the *prodid* with the first character of the *prodid* removed. The user ID will need to be class E if it will be used to build segments; otherwise, it usually is class G. The user ID also owns the product's disks. If you have set up a separate user ID from which to build segments, that user ID will be given links to the product disks or access to its directories.
4. Each product has 2 components, one for using minidisks for the product disks and one for using SFS directories. The minidisk component name is the product acronym. The SFS component name is formed by concatenating SFS to the product acronym.
5. Each product has the following service disks with the specified default addresses for the minidisks and the default directory names for SFS.

   **Note:** The *comp* variable is the minidisk component name (not the SFS component name).

| Disk Name | Address | Directory |
|---|---|---|
| LOCALSAM | 2C2 | VMSYS:*userid.comp*.LOCAL |
| BASE1 | 2B2 | VMSYS:*userid.comp*.OBJECT |
| DELTA | 2D2 | VMSYS:*userid.comp*.DELTA |
| Alternate APPLY | 2A6 | VMSYS:*userid.comp*.ALTAPPLY |
| Production APPLY | 2A2 | VMSYS:*userid.comp*.PRODAPPLY |
| Test Build | N/A | N/A (no assigned default) |
| Production Build | N/A | N/A (no assigned default) |

In addition,

- Each product shares the 51D and the 19E disks.
- PPF and PRODPART files are placed on the 51D.
- The 19E disk can be used for the product's general user code.

6. PPF values also have an established relationship.

   - Apply, build, receive and product IDs are equal (*appid* = *bldid* = *recid* = *prodid*).
   - Apply and exclude list names equal the product ID.
   - Control file name (*cntrl*) equals the 3-character product prefix followed by VM.
   - Each product chooses their own PTF and APAR prefixes.

7. File names for product parts begin with the 3-character product prefix.

   Build list file names begin with the 3-character product prefix followed by BL for regular build lists or SB for segment build lists.

An example for the program product LE/370 Version 3 Release 2 is as follows:

| Item | Value for product LE/370 Version 3 Release 2 |
|---|---|
| 3-character product prefix | CEE |
| Product number | 5688198 |
| *prodid* | 5688198C |
| PPF file | 5688198C PPF |
| PRODPART file | 5688198C PRODPART |
| *appid*, *bldid*, *recid* | 5688198C |
| *userid* | P688198C |
| *cntrl* | CEEVM |
| PTF prefix | UN |
| APAR prefix | PN |
| Disk component | LE370 |
| SFS component | LE370SFS |
| LOCALSAM disk | 2C2 |
| LOCALSAM directory | VMSYS:P688198C.LE370.LOCAL |
| BASE1 disk | 2B2 |
| BASE1 directory | VMSYS:P688198C.LE370.OBJECT |
| DELTA disk | 2D2 |
| DELTA directory | VMSYS:P688198C.LE370.DELTA |
| Alternate APPLY disk | 2A6 |
| Alternate APPLY directory | VMSYS:P688198C.LE370.ALTAPPLY |
| Production APPLY disk | 2A2 |
| Production APPLY directory | VMSYS:P688198C.LE370.PRODAPPLY |

## Installation Media Formats

There are four ways in which the different formats are packaged within the distribution tapes. They have the following characteristics:

- VMSES/E Installation Tape
  - Logical tape description file
  - Multi-volume logical tape (products can span more than one volume)
  - Only VMSES/E products
- Parameter Driven Installation (PDI) Tape

  There are two types of PDI tapes: Merged, and Stacked Product Tapes. Both offer:
  - System Offering style table of contents
  - A consistent tape format for all products included
  - Support for both VMSES/E and non-VMSES/E products
  - Single volume logical tape (a product must be wholly contained in a single physical tape)
- INSTFPP Installation Tape

  There are two types of INSTFPP tapes: Merged, and Stacked Product Tapes. Both offer:
  - Table of contents
  - A consistent tape format for all products included
  - INSTFPP install format only (no support for VMSES/E products)

This tape format can no longer be ordered. The current offering is SDO.

- Other formats.

  The format and installation method are defined by the product.

## VMSES/E Installation Tape

This tape format is used both for the Refreshed Product Tapes (RPT) and Recommended Service Upgrade (RSU) tapes. The logical tape can have one or more physical tape volumes, and includes special files detailing the logical tape structure:

- Tape Descriptor File

  Is a high-level mapping of the products per tape volume. As an example, Figure 95 on page 210 shows the actual contents of the Tape Descriptor File (TDF) for the VM/ESA Release 1.1 product tape. Although VM/ESA Release 1.1 is used, this structure is valid for all VMSES/E-formatted products.

- Product Contents Directory

  Lists, for a product, the logical files included in each volume. Figure 95 on page 210 also shows the actual Product Contents Directory (PCD) file for the CMS Component in the VM/ESA Release 1.1 product tape.

- Product Identifier File

  Lists, for a product, the logical files included in each volume.

### VMSES/E Product Identifier File

The RPT and RSU contain a file with a file ID of prodid cvrmnns. It is called the Product Identifier File, and the file type is used as a set of indicators:

| Indicators | Description |
|---|---|
| **c** | 0 for an independent product; 1 for a co-requisite product |
| **v** | Version number |
| **r** | Release number |
| **m** | Modification level |
| **nn** | Number of tape files in the product |
| **s** | 1 for a product supported by VMSES/E; blank for a product not supported by VMSES/E |

This file's contents are not important. The VMSES/E functions use only the file's file ID:

- The file name is used to identify the product.
- The file type further characterizes the product and the "nn" value is used to help position the tape. In previous releases, this was not always enforced. Figure 95 on page 210 shows this file for CMS in VM/ESA Release 1.1.

Tape Descriptor File for VM/ESA Release 1.1

```
INS yynn

* VM ESA 1.1 Installation Tape
VOL01 OF 04
:VOL01.
INS FILES        02
6VMVMK11 HDR     02
6VMVMK11 VMSES   10
6VMVMB11 HDR     02
6VMVMB11 CP      12
:VOL02.
INS FILES        02
6VMVMB11 HDR     02
6VMVMB11 CP      02
6VMVMI11 HDR     02
6VMVMI11 DV      10
:VOL03.
INS FILES        02
6VMVMA11 HDR     02
6VMVMA11 CMS     08
:VOL04.
INS FILES        02
6VMVMA11 HDR     02
6VMVMA11 CMS     04
6VMVMF11 HDR     02
6VMVMF11 REXX    09
6VMVML11 HDR     02
6VMVML11 GCS     10
6VMVMH11 HDR     02
6VMVMH11 TSAF    08
6VMVMD11 HDR     02
6VMVMD11 AVS     09
```

Product Contents Directory for CMS

```
6VMVMA11 $INSyynn

*
* Product Contents Directory
*
     :VOL03.
     :CMS.
     SYSSAMP
     AXLIST
     PARTLST
     DELTA
     APPLY
     TOOLS
     BASE
     SYSTEM
     :VOL04.
     HELP
     NCHELP
     MACRO
     SOURCE
```

Product Identifier File for CMS

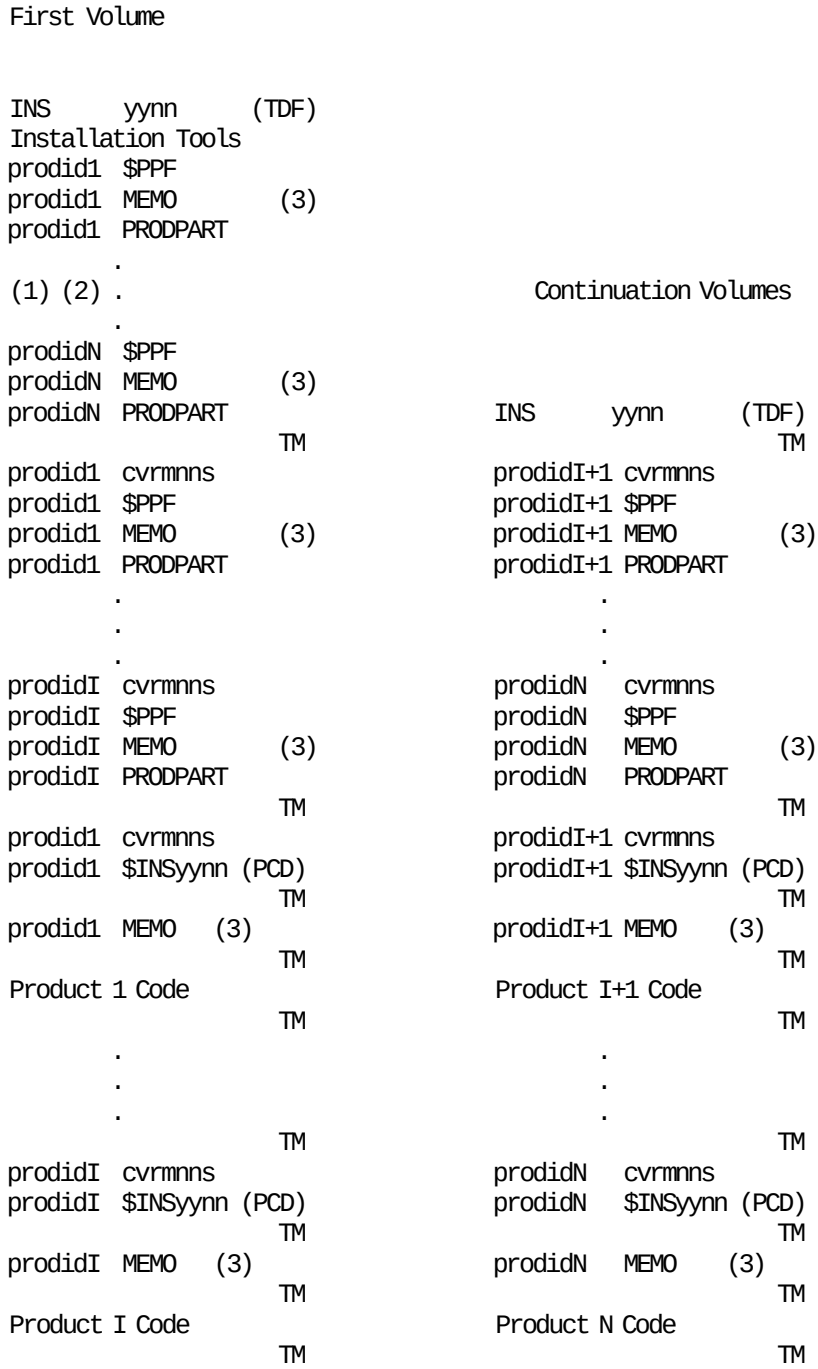
```
6VMVMA11 0101101

6VMVMA11    VM/ESA CMS COMPONENT
* CONTAINS IBM COPYRIGHTED MATERIALS  *
```

*Figure 95. TDF and PCD Files from VM/ESA Release 1.1 Installation Tape*

In the example, the TDF indicates that CMS has eight product files on Volume 3, and four product files on Volume 4. The PCD details the contents of these files. The tape file names in the PCD must match the names defined in the TAPEFILE (first) column of the RECINS section of the PPF.

### VMSES/E Installation Tape Format

Figure 96 on page 211 shows the layout of a VMSES/E-formatted product tape. Please note that, although VM/ESA Release 1.1 is used as an example, the tape structure applies to any VMSES/E-formatted product.




**Notes:**

- On RSU tapes, the first physical tape file contains a *bcompname SRVAPPS* file per product.
- On RSU tapes for VM/ESA Release 2.2, the first physical tape file contains a *appid VVT$PSU$* file per product
- Starting with VM/ESA Release 2.2, the Memo to User files are empty. They are retained for compatibility only. The information they used to contain can now be found in the Program Directory and Product Bucket that accompany the product distribution media.

*Figure 96. Format of VMSES/E Refreshed Product Tape and RSU Tape*

## VM/ESA SDO Installation Tapes (PDI)

SDO tapes can contain both VMSES/E and non-VMSES/E products. Each product is comprised of a set of tape files, here called a *logical tape*. The SDO format allows grouping of several logical tapes in one set of real tapes.

### SDO Product Identifier File

These tapes contain a file with a file ID of "Iprodid 0vrmnnff." It is called the Product Identifier File, and the file type is used as a set of indicators:

| Indicators | Description |
|---|---|
| **0** | Constant (zero) |
| **v** | Version number |
| **r** | Release number |
| **m** | Modification level |
| **nn** | Number of tape files in the product |
| **ff** | Feature ID code. |

**Note:** If a product cannot fit in a single tape volume, it is split into a number of features.

### SDO Tape Formats

SDO tapes may contain one or more products. The set of tape files for a product is known as a "logical tape." Logical tape formats for VMSES/E and INSTFPP products are shown in Figure 97.

**Note:** For VM/ESA Release 2.2 the prodid MEMO files are empty.

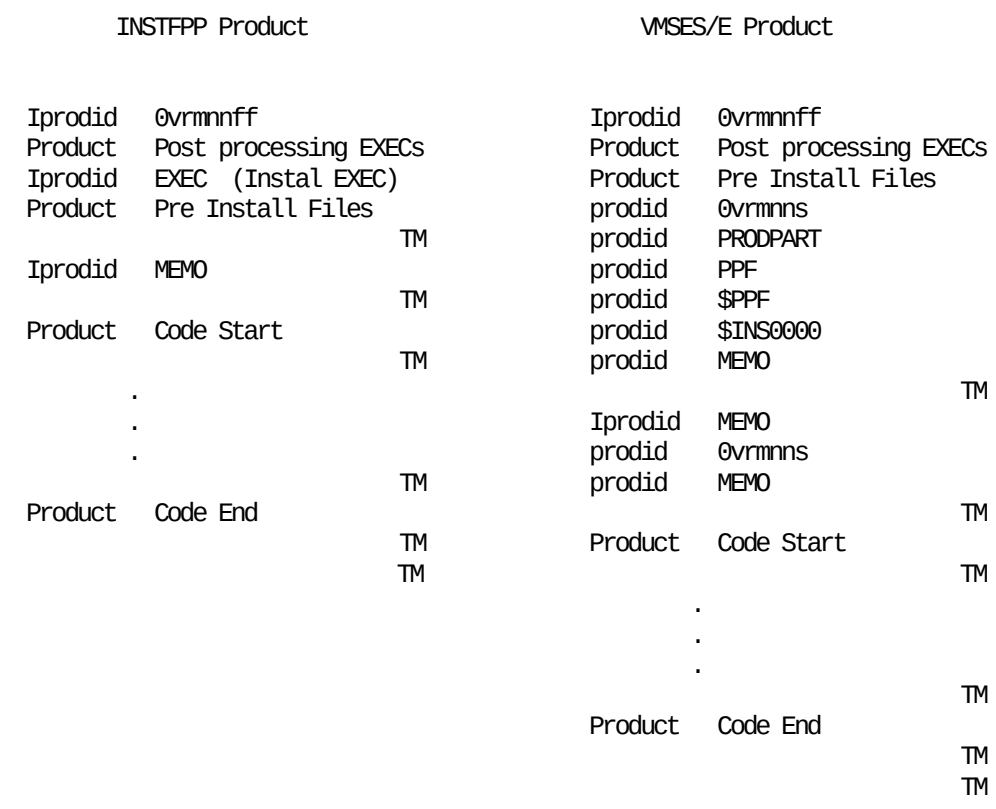



*Figure 97. SDO Logical Tape Formats*

Two SDO sub-formats are supported:

- Merged tape, shown in Figure 98 and
- Stacked tape, shown in Figure 99.

SDO merged tapes look like the second volume of a SDO stacked tape.

```
                                                TM
                        Product 1   Logical Tape
                                                TM
                                .
                                .
                                .
                                                TM
                        Product N   Logical Tape
                                                TM
                                                TM
```

*Figure 98. SDO Merged Tape Format*

```
      First Volume                          Continuation Volumes

Iprodid1  0vrmnnff                                                TM
Product 1 Post processing EXECs          Product I+1 Logical Tape
prodid1   PRODPART                                                TM
prodid1   $PPF                                  .
        .                                       .
        .                                       .
        .                                                         TM
IprodidI  0vrmnnff                       Product N   Logical Tape
Product I Post processing EXECs                                   TM
IprodidI  EXEC                                                    TM
        .
        .
        .
IprodidN  0vrmnnff
Product N Post processing EXECs
prodidN   PRODPART
prodidN   $PPF
                            TM
Iprodid1  MEMO
        .
        .
        .
IprodidN  MEMO
                            TM
Product 1 Logical Tape
                            TM
Product I Logical Tape
                            TM
                            TM
```

*Figure 99. SDO Stacked Tape Format*

## INSTFPP Installation Tapes

INSTFPP tapes have the same format as VM/ESA SDO tapes. Both stacked and merged product tapes exist. The difference is that INSTFPP tapes do not have $PPF or PRODPART files. Therefore, they do not support VMSES/E products.

## VMSES/E Service Tapes

VMSES/E can only service products in VMSES/E format. Therefore, only this format will be discussed here. The VMSES/E recommended service application process is the product service upgrade (PSU), which employs recommended service upgrade (RSU) tapes. This tape has the same format as the installation tape. Some products may in addition use refreshed product tapes (RPTs). However, VMSES/E supported products can also be serviced from Corrective Service (COR) tapes.

**Note:** COR service is not available on CD-ROM.

All other product formats are serviced from COR tapes, and employ their own service procedures. The format of these tapes may differ significantly from the VMSES/E COR format.

## Program Level File

The Program Level File, present in service tapes, resembles the Product Identifier File of the installation tapes. It has a file ID of prodid 0vrmnns, and the file type also serves as a set of indicators:

| Indicators | Description |
|---|---|
| **0** | Constant (zero) |
| **v** | Version number |
| **r** | Release number |
| **m** | Modification level |
| **nn** | Number of tape files in the product |
| **s** | 1 for a product supported by VMSES/E with a PPF; blank for a product not supported by VMSES/E with a PPF |

## VMSES/E Service Tape Format

COR tapes format closely resembles the RPT/RSU format. The differences are:

- Instead of a Tape Descriptor File there is a COR Descriptor File. These files have file IDs of:

  **COR** COR ymdd

  "ymdd" is:

  **y** last digit from the year
  **m** month (in hexadecimal, "1" is January, "C" is December)
  **dd** day

  So, "3C01" means December 1, 1993.

- There is a Tape Document (named COR DOCUMENT) file describing the service procedure.
- There is also one PCD per product. The file ID is prodid $PUTnnnn (or prodid $CORymdd), and the structure is close to that of the installation PCD.

Figure 100 shows the layout of a VMSES/E service tape.

```
COR ymdd                        (TDF)
COR DOCUMENT
                                  TM
prodid1  0vrmnns
prodid1  MEMO
        .
        .
        .
prodidN  0vrmnns
prodidN  MEMO
                                  TM
prodid1  0vrmnns
prodid1  $CORymdd               (PCD)
                                  TM
Product 1 Service Start
                                  TM
        .
        .
        .
                                  TM
Product 1 Service End
                                  TM
        .
        .
        .
                                  TM
prodidN  0vrmnns
prodidN  $CORymdd               (PCD)
                                  TM
Product N Service Start
                                  TM
        .
        .
        .
                                  TM
Product N Service End
                                  TM
```

*Figure 100. Layout of the VMSES/E Service Tape*

Examples of the Tape Descriptor file and Product Contents Directory file are shown in Figure 101 on page 216. Though the examples are for VM/ESA Release 1.1, they remain valid. The tape file names in the PCD must match the names defined in the TAPEFILE (first) column of the RECSER section of the PPF.

Tape Descriptor File for VM/ESA Release 1.1

```
COR ymdd

VM Corrective Service Tape ymdd ...
VOL01 of 01
:VOL01.
COR      FILES    02
6VMVMB11 HDR      01
6VMVMB11 CP       05
6VMVMA11 HDR      01
6VMVMA11 CMS      05
```

Product Contents Directory for CMS

```
6VMVMA11 $CORymdd

:VOL01.
:CMS.
AXLIST
PARTLST
UPDT
TEXT
MISCREPL
```

*Figure 101. TDF and PCD Files from VM/ESA Release 1.1 Corrective Service Tape*

# Appendix C. Removing Service

This appendix describes two possible ways of removing an applied PTF from the system:

- By level
- Selectively

Normally, the need to remove a PTF arises because it is in error, and no corrective or superseding service is available. Further, the PTF may have been applied as part of a functional upgrade, and you may be interested in keeping some of the newly introduced function. The choice of which method to use will thus depend on how urgently you need the service being applied.

The appendix concludes with a discussion on removing local service.

## Back-Out by Level

You will find a discussion of this method, as well as the details of each of the steps required, in *VM/ESA: Service Guide*.

If the product was stable before the service, the easiest method is to simply "back-out" (or remove) the entire new level of service. This approach depends on whether the service applied is:

- In testing mode
- In production mode

In order to better understand the method please refer to Figure 102 as we proceed in the discussion. To improve clarity, a generic product and fictitious "service levels," numbered 101, 102, and 103, are used. Each level is really a set of PTFs. Also, each higher level includes the PTFs in the preceding one. For example, level 102 includes all PTFs in level 101.

| Step | Description | DELTA | | APPLY | | | BUILD | |
|---|---|---|---|---|---|---|---|---|
| | | Alt | Prod | Alt | Int | Prod | Alt | Prod |
| 1 | Installation | | 101 | | | 101 | | 101 |
| 2 | Receive 102 | 102 | 101 | | | 101 | | 101 |
| 3 | Apply 102 | 102 | 101 | 102 | | 101 | | 101 |
| 4 | Build 102 | 102 | 101 | 102 | | 101 | 102 | 101 |
| 5 | Rec/Apply/Build COR | 102+C | 101 | 102+C | | 101 | 102+C | 101 |
| 6 | Merge, move to production | | 102+C | | 102+C | 101 | 102+C | 102+C |
| 7 | Restore build disk | | 102+C | | 102+C | 101 | 102+C | 101 |
| 8 | More COR service | COR 2 | 102+C | 102+C2 | 102+C | 101 | 102+C2 | 102+C2 |
| 9 | Rec/Apply/Bld 103 | 103 | 102+C2 | 103 | 102+C2 | 101 | 103 | 102+C2 |
| 10 | Restore production 101 | 103 | 102+C2 | | | 101 | 103 | 101 |

*Figure 102. Service Removal Steps*

1. Let us suppose the product is installed (from a Refreshed Product Tape) at service level 101, corresponding to the original code plus some service.

2. IBM has sent you some service (at level 102) and you RECEIVE it into your alternate DELTA disk.

3. You APPLY the service (into the alternate APPLY disk).

4. You refresh the alternate build disk (see “Refresh” on page 100) then you BUILD the service (on the alternate build disk).

Now you can test the product. Suppose testing reveals an error. You will correct it in the following steps:

5. Obtain corrective service for the error, and receive, apply, and build it. Note that you do **not** refresh the alternate build disk. Also, do **not** do a merge: you do not need to isolate the COR from level 102. You now resume the normal test.

6. Your test has gone well, so you move the serviced level into production. You use VMFMRDSK to move down one level the disks in the DELTA and APPLY strings, and place the alternate build disks into production (see “Production” on page 108).

Now suppose an error shows up in the production system. It is still easy to correct:

7. Using the backup done in step 4, restore the production build disk.

8. You repeat step 5 and, having done the testing, step 6.

9. Meanwhile a new service level, 103, arrives, so you:

   a. Merge down one level, both the DELTA and APPLY strings, to incorporate the latest COR.

      **Note:** This is important. Be sure to merge only one level, as the intermediate APPLY level is still under test.

   b. Receive, apply, build, and start testing level 103.

Unexpectedly, a new error pops up in the production system! Now you cannot repeat steps 5 and 6: your Alternate APPLY disk has level 103. To complicate the issue, your backup of the production level 101 has been replaced by the backup of level 102+C2. Also, you cannot move the new level 103 into production (either because it has not been tested or because it does not contain the correction to the error), so you will have to recreate level 101. But how?

The levels 101 and 102+C2 differ only by some objects. You will have to rebuild those objects at the 101 level. The intermediate APPLY disk has the key to find those objects: it is the Select Data file (appid $SELECT).

So, you have to run VMFBLD using that Select Data file, but without accessing the changed parts introduced by the 102 and subsequent levels.

10. The recovery has the following operations:

**a** To “hide” the levels you will need to code a PPF override that does not contain them, that is, with those disks removed from the APPLY string definition.

**b** Compile the override (let us name it BACKOUT):

```
vmfppf backout compname
```

**c** As you still need the Select Data File, copy it from the intermediate to the production APPLY disk:

```
copyfile * $select fm-applyint = = fm-applyprod ( olddate replace
```

**d** Now you can rebuild the affected objects. Remember to use the ALL option of VMFBLD, because the objects statuses in the Build Status table are not SERVICED. You should refer to *VM/ESA: Service Guide* for the detailed operations, and possible errors and their solutions.

Note that you were able to restore the 101 production level only because:

- You could identify the objects to rebuild - this is always possible as their parts are listed in the Select Data File.
- You still had the VVT and AUX files for level 101 - you will not keep these forever. Look at Figure 102 on page 217: if you merge down one level the APPLY string, the level 101 in the production disk will be replaced by the level 102+C in the intermediate disk. And you will have to do it, so you can apply newer service.

Therefore we recommend that:

- You do a thorough test before committing a level to production.
- You keep the intermediate apply level for some time, instead of merging it with production at the time you move the alternate build disks to production.
- You keep the backup of the production build disks for one or two extra levels.

## Selective Back-Out

If the problem can be determined to originate with a particular PTF, and no correction is available, you may wish to remove only that PTF. It is not impossible to do it, but you will have to be very careful in following all the required steps.

Once applied, a PTF cannot be re-applied or removed. So, adding it to the exclude list of its service level and re-applying the service does not work. The only way is to go back to the service level, *preceding* the one the PTF belongs to. Then you can reapply the level the bad PTF belongs too, but this time with the PTF added to that level’s exclude list.

Before excluding the PTF you should know what you may be risking. Remember that when a PTF is excluded, all PTFs that have a dependency on it are also excluded, even if they are in the apply list. You can list those service dependencies by using the VMFSIM SRVDEP command.

The section “Impact of Backing Out a PTF” on page 226 includes some sample code that will give you additional information about the dependent PTFs and respective APARs.

Based on the results of the queries, you can then decide whether you want to exclude just the bad PTF, or if it is better to back out the whole service level.

If you do exclude PTFs, you must check very carefully to ensure that none of the PTFs are critical to the stability of your system. If you are in doubt contact the IBM Support Center.

If you wish to exclude the PTF you can perform the following steps:

1. Use the APPLIST option of VMFSIM SRVDEP to create a file (let us name it MYEXCLST) containing the PTF and its dependencies. This file will be used as an exclude list. Here is a sample command:

   ```
   vmfsim srvdep prodid srvreqt * = srvapps * tdata :ptf ptfnum ( applist myexclst
   ```

2. Change this file's file type to $EXCLIST.

3. Find the levels of service that do not include the PTF. Use the Software Inventory query facility to search the Apply Status tables:

   ```
   vmfsim query prodid srvapps fm-apply tdata :ptf ptfnum
   ```

   where:

   **fm-apply** is the access mode for a disk in the APPLY string. Check all disks in the string. Note all disks where the PTF is **not** applied.

4. Create a PPF override to change the APPLY string. The changed APPLY string should have:

   a. A new, empty, disk as the first disk.

   b. All disks, from the original string, that do not have the PTF applied (see 3).

5. Compile the override.

6. Define the new, empty, alternate APPLY disk. This disk replaces all the disks that have been deleted from the original APPLY string. We will refer to those disks as the "replaced" disks.

   **Note:** You should keep those disks for now. Later on you may be able to delete them.

   Please note that the "empty" disk may, as a matter of fact, have files in it: because the disk may be shared by more than one component, it only has to be empty in terms of files belonging to the product you are dealing with. As an example, CP and the Dump View Facility share the same APPLY disks.

   It is because of this sharing possibly that we recommend you replace several disks by one. Another option would be to break that sharing situation, that is, this product would be isolated in the new disk, and the other products would remain undisturbed.

   If the disk is shared, and is to remain shared, we recommend you do the following:

   a. Define a new (target) disk with enough space to contain all of the files on the source disks.

**b** Identify all the products that have files on the replaced disks: edit the VMSES PARTCAT file and make a note of all the different values for the :PRODID. tag.

**c** Because this is essentially a disk merge, you should do your copy starting with the oldest service levels, up to the newest (from RIGHT to LEFT in the APPLY string definition). For each of the replaced disks, and for each product (except the one you are backing out) issue:

```
vmfcopy * * fm-source = = fm-target ( prodid prodid olddate
```

to copy all the product's files from one replaced disk to the target disk.

If the disk is shared but you are creating a private disk for the product, we recommend you do the following:

**a** Determine the space occupied by the product's files in each of the source disks.

**b** Define a new (target) disk with space equal to the greatest of the values above, plus some extra space for future growth.

**7** Manually incorporate in the file created in Step 1 above the exclude list of the current DELTA disk.

**8** Simulate the apply step: run VMFAPPLY specifying your exclude list and the TEST option. An example of the command is:

```
vmfapply ppf backout compid ( test exclist myexclst
```

**9** Check the apply results. Use:

```
vmfview apply
```

**10** Decide whether you wish to proceed.

**This is a critical point in this procedure**.

**11** Run VMFAPPLY specifying your Exclude list. An example of the command is:

```
vmfapply ppf backout compid ( exclist myexclst
```

**12** Run VMFBLD to rebuild the product.

**13** Test the new version created on the alternate build disks.

**14** If test reveals no problems you can place the new product into production.

**15** When you feel confident that the new product is error-free you can delete all the replaced APPLY disks.

**Note:** If the product was sharing the disks and is now isolated, deleting here means only to erase the product's files from the replaced disks. Remember to use VMFERASE, so the parts catalog stays updated.

**16** As a final step you might consider changing the minidisk virtual address (or directory names) so that you will be able to use the original PPF, and discard the override (if possible).

## Removing a Local Modification

Occasionally, you might want to remove a local modification. If you receive a PTF which fixes a problem for which you had a local modification or if you have a local modification that you no longer want applied you could remove the local modification.

The process for removing a local modification depends on whether its parts are update or replacement serviced and whether the modification is the latest (top) modification.

In this example, we will remove the local modification **L0013** previously applied to CMS. We will also assume that the LOCALMOD disk has a file mode of **E**.

**1** Access the CMS disks.

```
vmfsetup esa cms
```

**2** Invoke VMFSIM to determine the parts affected by the local modification and what other modifications have been applied to these parts.

```
vmfsim query 6vmvma22 vvtlcl e tdata :mod lcl0013 :part
     VMFSIP2480I Results for
                 TDATA :MOD LCL0013 :PART
     :PART DMSNGP TXT
        :MOD LCL0015.VL0015DS LCL0013.VL0013DS
     :PART FSOPEN MACRO
        :MOD LCL0013.VL0013DS
     :PART DMSUPD TXT
        :MOD LCL0013.VL0013DS LCL0010.VL0010DS
     :PART GETMAIN CPY
        :MOD LCL0013
     :PART DMSARN TXT
        :MOD LCL0013.VL0013DS
```

**3** Then, for each of these parts do the following:

**a** If the part is update-serviced (for example, DMSNGP TXT, FSOPEN MACRO, DMSUPD TXT, DMSARN TXT):

- Comment out the entry for the modification in the part's AUXLCL file on the LOCALMOD disk. If there are only comment entries left, erase the AUXLCL file. For example, in DMSNGP AUXLCL:

```
   VL0015DS  LCL  LCL0015
*  VL0013DS  LCL  LCL0013
```

- If the part has more recent modifications than the modification being removed (for example, DMSNGP TXT), XEDIT the source part using the CTL option to determine if the updates can still be applied. If XEDIT fails, you need to rework the update that no longer works. Repeat this process until the XEDIT succeeds and the code is correct.

```
xedit dmsngp assemble (ctl dmsvm
```

- If the part is a macro (for example, FSOPEN MACRO) or if the AUXLCL file has only comment records (for example, DMSARN TXT), then invoke VMFSIM to remove the modification from the part in the local VVT.

  ```
  vmfsim logmod 6vmvma22 vvtlcl e tdata :part fsopen macro :mod lcl0013.vl0013ds (delete

  vmfsim logmod 6vmvma22 vvtlcl e tdata :part dmsarn txt :mod lcl0013.vl0013ds (delete
  ```

- If the part is a macro (for example, FSOPEN MACRO), invoke VMFQOBJ to determine the names of the build lists of the MACLIBs which need to be rebuilt.

  ```
  vmfqobj esa cms tdata :object fsopen :libname
      VMFUTL2480I Results for
                  TDATA :OBJECT FSOPEN :LIBNAME
      :OBJECT DMSGPI.FSOPEN
         :LIBNAME DMSGPI
  ```

  The build list name is DMSGPI, which is shown on the output line :OBJECT DMSGPI.FSOPEN.

**b** If the part is replacement-serviced (for example, GETMAIN CPY):

- Invoke VMFSIM to remove the modification from the part in the local VVT.

  ```
  vmfsim logmod 6vmvma22 vvtlcl e tdata :part getmain cpy :MOD LCL0013 (DELETE
  ```

- If the local modification is not the latest modification, then you need to rework all the more recent modifications to remove the changes put in by the deleted local modification.
- If the local modification is for a macro, (for example, GETMAIN CPY) invoke VMFQOBJ to determine the names of the build lists of the MACLIBs which need to be rebuilt

  ```
  vmfqobj esa cms tdata :object getmain :libname
      VMFUTL2480I Results for
                  TDATA :OBJECT GETMAIN :LIBNAME
      :OBJECT OSMACRO.GETMAIN
         :LIBNAME OSMACRO
      :OBJECT MVSXA.GETMAIN
         :LIBNAME MVSXA
  ```

  The build list names are OSMACRO and MVSXA. This is shown on the output lines :OBJECT OSMACRO.GETMAIN and :OBJECT MVSXA.GETMAIN.

**4** Next, recreate the replacement parts without the modification and rebuild any maclibs.

- Add a record for each part to the $SELECT file on the alternate APPLY disk. For our example, in 6VMVMA22 $SELECT add:

  ```
  :APPLYID. mm/dd/yy hh:mm:ss
  DMSNGP TXT
  FSOPEN MACRO
  DMSUPD TXT
  GETMAIN CPY
  DMSARN TXT
  ```

- Rebuild the affected MACLIBs, if any.

```
vmfbld ppf esa cms dmsgpi (serviced
vmfbld ppf esa cms osmacro (serviced
vmfbld ppf esa cms mvsxa (serviced
```

- Re-assemble any ASSEMBLE files if they have a non-comment record in the AUXLCL file (otherwise, the IBM text deck will be used). Use the appropriate VMF*x*ASM command and specify the LOGMOD and OUTMODE LOCALMOD options when running VM/ESA Release 2.2. The LOGMOD option removes the modification from the local VVT entries for these parts and the OUTMODE option places the results of the assembly (DSMNGP TXTL0015 and DMSUPD TXT0010) on the LOCALMOD disk. In prior releases, since the OUTMODE option does not exist, you must manually copy the assembled parts to the LOCALMOD disk.

  ```
  vmfhlasm dmsngp esa cms (logmod outmode localmod
  vmfhlasm dmsupd esa cms (logmod outmode localmod
  ```

- Re-create the replacement parts for the remaining update-serviced parts (excluding ASSEMBLE file and macros). Use the appropriate command (VMFEXUPD, VMFNLS, GENCPBLS) and specify the LOGMOD and OUTMODE LOCAL options when running VM/ESA Release 2.2. This will also remove the modification from these parts in the local VVT. See *VM/ESA: Service Guide* for an introduction on creating replacement parts for local modifications.

**5** Finally, when all modification parts have been updated, rebuild any other affected objects.

- Invoke VMFBLD to rebuild the objects.

  ```
  vmfbld ppf esa cms (serviced
  ```

- If you are **confident** they will not be needed, erase the local modification updates and replacement parts that were generated and placed on the LOCALMOD disk.

  ```
  erase dmsngp vl0013ds e
  erase dmsngp txtl0013 e
  erase fsopen vl0013ds e
  erase dmsupd vl0013ds e
  erase dmsupd txtl0013 e
  erase getmain cpyl0013 e
  erase dmsarn vl0013ds e
  erase dmsarn txtl0013 e
  ```

  **Note:** This example was for CMS. If you are removing a local modification for CP and HCPMDLAT MACRO was affected you could also erase HCPLDL VL0013DS.

**6** You have now completed the removal of a local modification. You will still need to put this into production and rebuild any affected segments. See the *VM/ESA: Service Guide* for these instructions.

# Appendix D. VMFSIM Exploitation Code Examples

This appendix contains several examples of Software Inventory exploitation. You can use REXX and CMS Pipelines to write many useful functions.

Before we go on, we will give you a brief introduction to CMS Pipelines, and also tell you where you can get more information on this function.

## CMS Pipelines Introduction

CMS Pipelines is a CMS command introduced in VM/ESA Release 1.1, and it considerably extends the CMS functional capabilities. For readers familiar with UNIX** systems, the concept of pipes is well known.

The idea of a pipeline is simple. One has a set of small, simple programs, each of which does a specific job. All the programs have a standard input device and a standard output device defined.

To solve a problem, these small programs, called “filters,” are linked together in a pipeline, where data flows from one “stage” to another.

It is possible to have more than one connection (stream) between stages. This can lead to quite sophisticated models.

CMS Pipelines comes with over 140 built-in filters that allow you to access disk files, spool devices and files, and tapes. There are also filters to select, translate, and combine records.

In CMS Pipelines, you also can write your own filters by using REXX. The files for these filters have a file type of REXX.

## Example

Let us look at a simple example. We want to find out which file mode the virtual address “111” is accessed as. The following program will find it:

```
/* ----- QMODE EXEC ---------------------------------------------  */

Address 'COMMAND'                 /* Bypass EXEC search             */

'PIPE'                           ,
'  COMMAND QUERY SEARCH'         , /* Issue CMS command QUERY SEARCH  */
                                 , /* and trap output                */
'| locate / 111 /'               , /* locate all lines with ' 111 '   */
'| specs ,'                      , /* edit the line:                 */
  'The 111 disk is accessed as,', /*   insert some text at the start,*/
  '1 word 3 nextword'            , /*   and pick word 3 from the      */
                                 , /*   result line from query search.*/
                                 , /*   Put this as next word        */
'| cons'                          /* display result on console      */

Exit
```

## Pipeline Documentation

The PIPE command is standard since VM/ESA Release 1.1. The CMS Pipelines manuals are:

- *VM/ESA: CMS Pipelines User's Guide*
- *VM/ESA: CMS Pipelines Reference*

The following manual is highly recommended as an introduction to CMS Pipelines.

- *CMS Pipelines Tutorial*

---

## Impact of Backing Out a PTF

This example consists of two parts:

**PTFREMOV EXEC** Main routine that handles and processes queries to VMFSIM

**JOINLN REXX** Pipeline filter to select certain types of lines from the VMFSIM output, and join lines that have spilled.

The PTFREMOV command also calls the PSIMOUT EXEC, but as this EXEC can be independently used, it is described in "VMFSIM Output Processor" on page 227.

The PTFREMOV command creates a list of all APARs, with their descriptions, for the given PTF and its dependents. In this way, you can better evaluate the impact of removing that PTF from the system.

The syntax of the PTFREMOV command is shown in Figure 103.

.

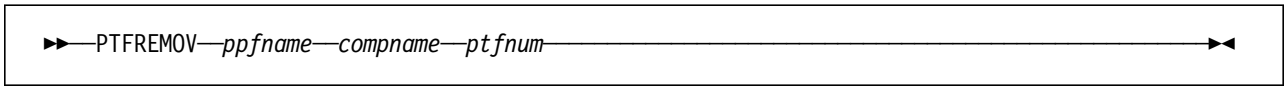



*Figure 103. PTFREMOV Command Syntax*

**Operands**

**ppfname** The filename of the PPF file for this product.

**compname** The name for the component, as defined in the PPF file.

**ptfnum** The PTF number.

Figure 104 shows a sample dialog from executing the PTFREMOV exec. The resulting disk file, ptfnum $BACKOUT, is shown in Figure 105 on page 227.

```
ptfremov esa cp um22636
Getting dependent PTFs...
Looking for corresponding APAR numbers
Looking for APAR descriptions
To back out PTF UM22636: 3 PTFs ( 3 APARs ) must be backed out.
For details, see file UM22636 $BACKOUT A
```

*Figure 104. PTFREMOV EXEC Sample Output*

```
To back out PTF UM22636: 3 PTFs ( 3 APARs ) must be backed out.

UM22637 VM53461 MP DEFER SERIALIZATION DOES NOT WORK W/DEDICATED CPUS.
UM22639 VM53456 ABENDIOL007 DUE TO VMDCTCRT BEING NEGATIVE.
UM22636 VM53455 MAKE HCPLGN AND HCPUSP SCM.
```

*Figure 105. Sample File UM22636 $BACKOUT*

The JOINLN (join line) filter is useful when the value of one or more tag fields is returned by VMFSIM in several lines. The filter is able to identify all lines associated with the tag, or tags, and returns them joined in a single line.

The syntax of the JOINLN filter is shown in Figure 106.

.

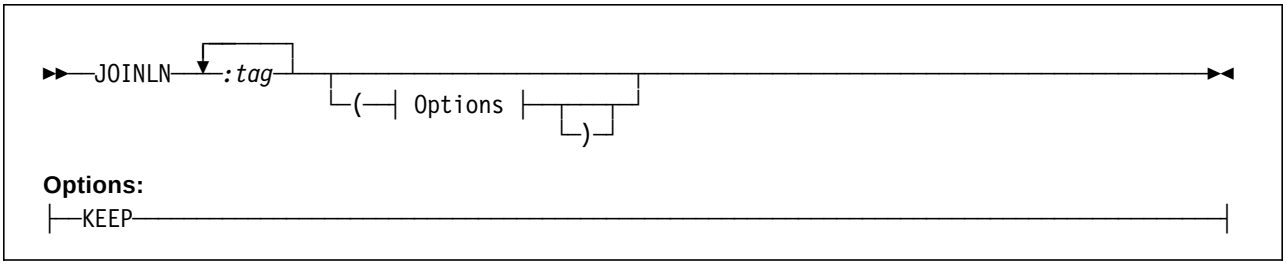



*Figure 106. JOINLN Filter Syntax*

**Operands**

**tags** One or more tags. Only lines with these tags will be selected.

**Options**

**KEEP** The default is to join all spilled lines for the specified tags. The KEEP option forces the filtered lines to be kept as they are, that is, not joined.

## VMFSIM Output Processor

The PSIMOUT EXEC acts as a post-processor for the output of VMFSIM queries. It allows you to reformat the output into an acceptable input format when there is more than one value listed for a tag field.

You may also use this EXEC to filter out only the types of lines that you want; for example, lines containing one particular tag. PSIMOUT also requires the JOINLN REXX CMS Pipelines filter.

The syntax of the PSIMOUT command is shown in Figure 107 on page 228.

.

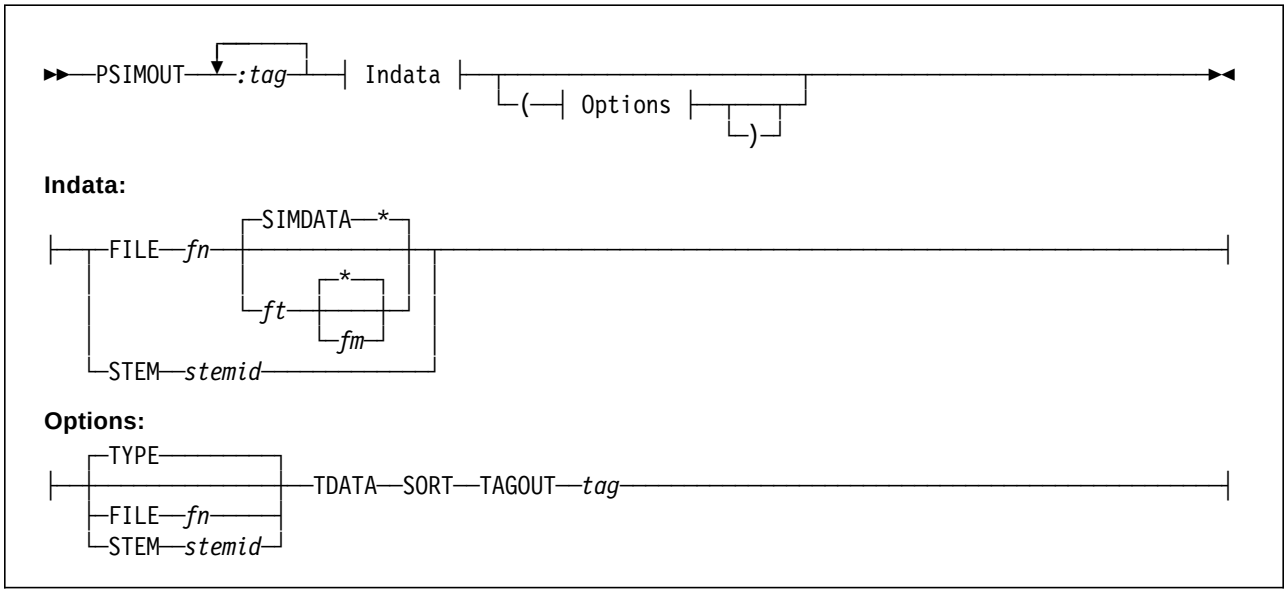



*Figure 107. PSIMOUT Syntax*

Note that the syntax is similar to that of the VMFSIM QUERY command.

The following options are of particular interest:

**TDATA** Means that PSIMOUT is to produce output in TDATA format, which may then be used as input to the next VMFSIM command.

**SORT** Causes the tag values to be sorted in ascending order.

**TAGOUT** Allows you to specify the tag name to be used in place of the original tag, in the generated output.

For example, if the original query listed the DEP tag for a PTF, we could specify TAGOUT :PTF to get the PTF numbers prefixed by :PTF on the output.

## Erasable Parts for Committed PTFs

Committing PTFs was described in “Receive Service Media Definition (RECSER) Section” on page 50. Here we present a sample procedure that, given a PTF number, searches the list of its replacement parts and verifies, for each part, whether it can be safely erased or not. Two lists of replacement parts are created: the ones that cannot be erased, and the ones that you might be able to erase.

The PTF does not have to be committed: this enables you to see what you could erase before actually doing the commit, so you can evaluate if it is really worth doing it. The syntax of the PTFCOMIT command is shown in Figure 108 on page 229.

.

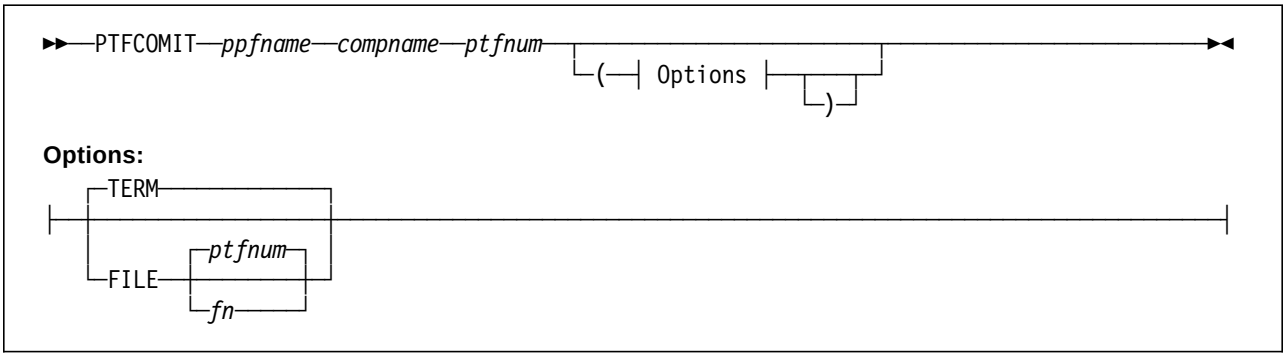



*Figure 108. PTFCOMIT Syntax*

**Operands**

**ppfname** The file name of the PPF for this product.

**compname** The name for the component, as defined in the PPF.

**ptfnum** The PTF number.

**Options**

**TERM** This is the default. The command results are shown on the virtual console.

**FILE** This option directs the command output to a file of file type PTFCOMIT on your A-disk. If a file name is not specified it defaults to “ptfnum.”

This command also exemplifies the use of the ASTEM operand of the VMFSIM command. The command uses an internal routine, called TAGVALUE, that extracts the value for a given tag from an ASTEM reply. You might be able to incorporate this routine in your own programs. Also, the SIM routine, though quite simple, might save you some code. An example of the output obtained when running the PTFCOMIT EXEC is shown in Figure 109.

```
ptfcomit esa cp um22636

PTFCOMIT Verification started.
PTFCOMIT results for PTF UM22636 on 25 May 1993 at 17:13:10.

         Found 3 parts that might be erased.
         Found 4 parts that must be kept.
List of parts that you might be able to erase (3 items):
  HCPMPD TXT22636
  HCPVIR TXT22636
  HCPVOU TXT22636

List of parts that MUST BE KEPT (4 items):
  HCPKFL TXT22636
  CPLOAD EXC22636
  HCPLGN TXT22636
  HCPUSP TXT22636
PTFCOMIT Process completed.
```

*Figure 109. PTFCOMIT Ouput Example*

## Finding the Status of an APAR or PTF

In “5 - List Status of an APAR” on page 152, we showed how to quickly find the service status for a given APAR. The example was, however, incomplete. The sample procedure presented here does a thorough verification and produces a detailed report.

Because the Software Inventory contains status information for PTFs, not for APARs, the PTF containing the given APAR is used. However, if the PTF has been superseded, the superseding PTF may be automatically used in its place. If, instead of the APAR number, you directly specify the PTF number you may inhibit the automatic search for a superseding PTF. Also note that SFS directories are explicitly supported.

The syntax of the PTFSTAT command is shown in Figure 110.

.

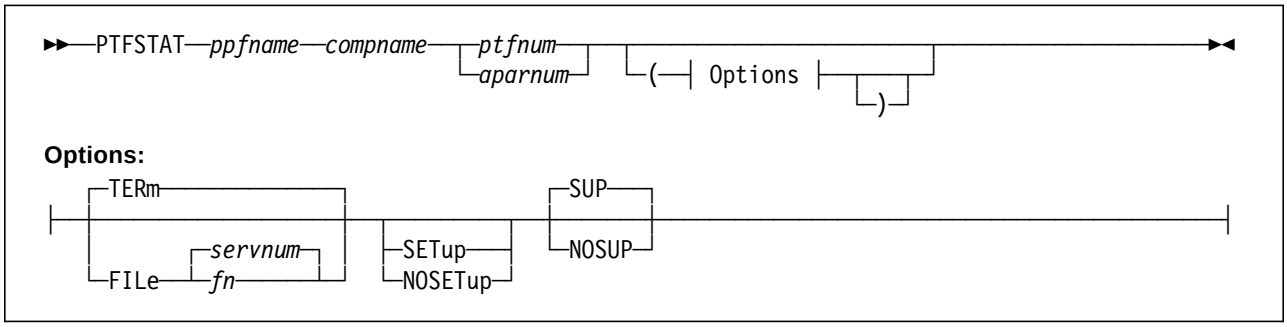



*Figure 110. PTFSTAT Syntax*

**Operands**

**ppfname** The file name of the PPF for this product.

**compname** The name for the component, as defined in the PPF.

**ptfnum** The PTF number; if this PTF has been superseded the superseding PTF will be used instead.

**aparnum** An APAR number; the procedure will find the corresponding PTF number and use it in the search.

**Options**

**TERm** This is the default. The command results are shown on the virtual console.

**FILe** This option directs the command output to a file of file type PTFSTAT on your A-disk. If a file name is not specified it defaults to “servnum,” which stands for the given aparnum or ptfnum.

**SETup** If specified, invokes the VMFSETUP command.

**NOSETup** If specified, the VMFSETUP command is not called. If neither SETUP nor NOSETUP are specified the action depends on the value of the :SETUP tag of the PPF.

**SUP** This is the default. An automatic look for superseding PTFs will be done. If you specify an APAR number this option is automatically invoked, and overrides any NOSUP option.

**NOSUP** If specified, and a PTF number was given, it suppresses the look for superseding PTFs. This allows you to find the status of a PTF in a system level for which the superseding PTFs might not even exist.

Figure 111 shows the console log resulting from running PTFSTAT for APAR VM48166. The associated file generated by PTFSTAT, VM48166 PTFSTAT A, is shown in Figure 112 on page 232.

```
ptfstat esa cp vm48166 ( file

PTFSTAT Verification for VM48166 started.
PTFSTAT Results will be in file "VM48166 PTFSTAT A".
PTFSTAT Finding PTFs that contain APAR VM48166
PTFSTAT Verification will be based on PTF UM18657
PTFSTAT Finding all APPLY system levels.
PTFSTAT Finding all BUILD lists.
PTFSTAT Finding all objects impacted by this PTF.
PTFSTAT Verifying objects status.
PTFSTAT Process completed.
```

*Figure 111. Console Log of PTFSTAT Execution*

```
PTFSTAT ------------------------------------------------------------------------
PTFSTAT Results for APAR VM48166 on 1 Mar 1992 at 01:23:31.
PTFSTAT
PTFSTAT Finding PTFs that contain APAR VM48166
PTFSTAT PTF UM18652 was superseded by  UM18657
PTFSTAT Verification will be based on PTF UM18657
PTFSTAT The system levels in the following disks will be searched:
PTFSTAT   G - 2A6
PTFSTAT   H - 2A4
PTFSTAT   I - 2A2
PTFSTAT
PTFSTAT RECEIVE status is:
PTFSTAT
PTFSTAT   Status     Date     Time     Userid
PTFSTAT   ---------- -------- -------- --------
PTFSTAT   RECEIVED   09/18/91 12:23:58 MAINT
PTFSTAT
PTFSTAT APPLY status by level:
PTFSTAT
PTFSTAT   Level   Status     Date     Time     Userid
PTFSTAT   ------- ---------- -------- -------- --------
PTFSTAT   G(02A6) APPLIED    09/26/91 07:40:35 MAINT
PTFSTAT   H(02A4) This level is empty
PTFSTAT   I(02A2) This level is empty
PTFSTAT
PTFSTAT BUILD status by level:
PTFSTAT
PTFSTAT   Level   Build list Object   Status      Date     Time     Userid
PTFSTAT   ------- ---------- -------- ----------- -------- -------- --------
PTFSTAT   G(02A6) HCPBLSRC   ASSEMBLE NEVER BUILT
PTFSTAT   G(02A6) HCPBLSRC   COPY     NEVER BUILT
PTFSTAT   G(02A6) CPLOAD     -        NEVER BUILT
PTFSTAT   G(02A6) HCPOM1     -        SERVICED    01/22/92 19:00:17 MAINT
PTFSTAT   H(02A4) This level is empty
PTFSTAT   I(02A2) This level is empty
```

*Figure 112. Results File from PTFSTAT Execution*

# Appendix E. Diskette Installation Instructions

The diskette that accompanies this document contains the sample source code for the Software Inventory exploitation examples discussed in Appendix D, “VMFSIM Exploitation Code Examples” on page 225.

This appendix provides guidance on the installation of the files contained on the diskette. Please read it in its entirety before attempting to use the diskette.

## Diskette Contents

The diskette contains the files listed below. Some files, like the README.DOC file, are in ASCII format; others have been transformed to CMS packed format before being placed on the diskette. These files require special handling.

You should have the following files on the diskette:

- ASCII files:

  | | |
  |---|---|
  | README.DOC | A copy of these installation instructions. |
  | OS2UPLD.CMD | An OS/2 procedure to help transfer (upload) the files to the VM host. |
  | DOSUPLD.BAT | An DOS batch file to help transfer (upload) the files to the VM host. |

- BINARY files:

  This file is in fixed record format, with a logical record length of 80 bytes.

  | | |
  |---|---|
  | COPYUNPK.EXC | A VM EXEC to help copy and unpack the other EXEC, REXX, and HELP files |

  These files are in fixed record format, with a logical record length of 1024 bytes.

  | | |
  |---|---|
  | PTFREMOV.EXC | PTFREMOV command |
  | PSIMOUT.EXC | PSIMOUT command |
  | JOINLN.RXX | JOINLN filter |
  | PTFCOMIT.EXC | PTFCOMIT command |
  | PTFSTAT.EXC | PTFSTAT command |
  | PTFREMOV.HCM | Help for the PTFREMOV command |
  | PSIMOUT.HCM | Help for the PSIMOUT command |
  | JOINLN.HCM | Help for the JOINLN filter |
  | PTFCOMIT.HCM | Help for the PTFCOMIT command |
  | PTFSTAT.HCM | Help for the PTFSTAT command |

## Installation Instructions

Regardless of the workstation you are using, its operating system, and 3270 terminal emulator, in order to install the procedures and help files on the VM system, you have to:

1. Transfer the files to the VM system
2. Unpack the transferred files

If your workstation is running OS/2* with the Communications Manager, or DOS 3.30 (or above) with one of the following 3270 emulators, you can take advantage of the sample upload procedures supplied with this diskette. Simply follow the instructions for your situation, below.

The supplied DOS upload procedure should work with any of the following 3270 emulator programs for DOS:

- IBM 3270 Workstation Program
- IBM PC 3270 Emulation Program
- IBM PC 3270 Emulation Program Entry Level
- 3278/79 Emulation Control Program

## Uploading the Files in a OS/2 Environment

Execute the following steps:

**1** Start the Communications Manager, if needed.

**2** Log on to a VM user ID (we suggest you use the MAINT user ID). If you have multiple sessions, use the first defined one.

**3** Insert the diskette in a diskette drive. We will assume it is the A-drive. However, you can use any drive (please adapt the commands below accordingly).

**4** Start, or jump to an OS/2 command window.

The next two steps, though optional, are recommended:

**5** Make the A-drive the current drive. Type:

```
a:
```

**6** Establish the root directory of the current drive as the current directory. Type:

```
cd a:\
```

**7** Invoke the sample upload procedure. If you executed the previous two steps, simply type:

```
os2upld
```

Otherwise, type:

```
drive os2upld inpath session
```

where:

**drive** Is the drive where the OS2UPLD procedure resides (in our example A:)

**inpath** Defaults to A:\; is the path where the files to be uploaded reside

**session** Defaults to blank (first defined session); is the identifier of the emulator session to be used for uploading the files. If you need to specify this operand then you have also to specify the "inpath" operand

8. Jump to the host session to continue the installation process. See "Unpacking the Files (All Environments)" on page 237.

**Note:** During the upload, the file ID of the "README.DOC" file is changed to "GG243851 README."

## Uploading the Files in a DOS Environment

Execute the following steps:

1. Start the appropriate 3270 emulator program, if needed.

2. Log on to a VM user ID (we suggest you use the MAINT user ID). If you have multiple sessions, use the first defined one.

3. Insert the diskette in a diskette drive. We will assume it is the A-drive. However, you can use any drive (please adapt the commands below accordingly).

The next two steps, though optional, are recommended.

4. Make the A-drive the current drive. Type:

   ```
   a:
   ```

5. Establish the root directory of the current drive as the current directory. Type:

   ```
   cd a:\
   ```

6. Invoke the sample upload procedure. If you executed the previous two steps, simply type:

   ```
   dosupld
   ```

   Otherwise, type:

   `path` **`dosupld`** `inpath session`

   where:

   **path** Is the path where the DOSUPLD procedure resides

   **inpath** Defaults to A:\; is the path where the files to be uploaded reside

   **session** Defaults to blank (first defined session); is the identifier of the emulator session to be used for uploading the files. If you need to specify this operand then you have also to specify the "inpath" operand.

   **Note:** If you are using one of the following emulators you **must not** use the session operand:

- IBM PC 3270 Emulation Program Entry Level
- 3278/79 Emulation Control Program

**7** Jump to the host session to continue the installation process. See "Unpacking the Files (All Environments)" on page 237.

**Note:** During the upload, the file ID of the "README.DOC" file is changed to "GG243851 README."

## Uploading the Files in Other Environments

Execute the following steps:

**1** Start your 3270 emulator program, if needed.

**2** Log on to a VM user ID (we suggest you use the MAINT user ID).

**3** Insert the diskette in a diskette drive.

**4** Make that drive the current drive, and the root directory the current directory (optional, but recommended).

**5** Use the file transfer procedures of your 3270 emulator, to transfer (upload) the files in the list below to the VM System. Transferring the README.DOC file is optional. We suggest you place all files on the 191 disk of the MAINT user ID.

Files to upload:

README.DOC

COPYUNPK.EXC

PTFREMOV.EXC
PSIMOUT.EXC
JOINLN.RXX
PTFCOMIT.EXC
PTFSTAT.EXC

PTFREMOV.HCM
PSIMOUT.HCM
JOINLN.HCM
PTFCOMIT.HCM
PTFSTAT.HCM

Except for the README.DOC file, all files should be transferred in BINARY format; that is, without converting them from ASCII to EBCDIC, and preserving the characteristics of fixed record format and logical record length of 1024 bytes (the COPYUNPK.EXC file has a fixed record length of 80 bytes). These might be specified as options of the transfer command, similar to: "RECFM F LRECL 1024."

If you also wish to transfer the README.DOC file, you must specify ASCII to EBCDIC translation, and new line at CR. These might be specified as options of the transfer command, similar to "ASCII CRLF." At the same time, you may also want to change the file's file name and file type, since

"README DOC" is a very common name, and might already exist. We suggest you use the file ID "GG243851 README."

**6** Jump to the host session to continue the installation process. See "Unpacking the Files (All Environments)."

## Unpacking the Files (All Environments)

You will have to unpack the uploaded files, and at the same time change their file types, according to Table 11.

*Table 11. File Type Conversion for Sample Files*

| Original File Type | New File Type |
|---|---|
| EXC | EXEC |
| RXX | REXX |
| HCM | HELPCMS |

We suggest you use the supplied COPYUNPK procedure and place the unpacked files on MAINT's 5E5 disk (normally accessed as file mode B).

The COPYUNPK is ready to be used. To invoke it, enter:

**copyunpk** mode

where:

**mode** Is the access mode of the target minidisk. B is the default.

If you do not wish to use the COPYUNPK procedure, you will have to issue a command similar to the one below, for every packed file:

**copyfile** fn ft fm = newft newfm **(olddate unpack**

where:

**fn ft fm** is the file ID of an uploaded file

**newft** is the new file type, in accordance with Table 11

**newfm** is the file mode of the target minidisk

## Source Listings for the Sample Code

The source listings for the sample code were removed from this document. You should not use the source listings included in the first edition of this document, as some of the EXECs were enhanced, and errors corrected, so those listings do not reflect the current source code.

# Index

## Special Characters



## A



## B

## C

## D

## E



## F



## G



## H



## I

## J



## K



## L

## Q



## R



## S

## T



## U



## V

## X



## Z

# ITSO Technical Bulletin Evaluation RED000

**VMSES/E Primer:**
**Concepts and Experiences**

**Publication No. GG24-3851-02**

Your feedback is very important to help us maintain the quality of ITSO Bulletins. **Please fill out this questionnaire and return it using one of the following methods:**

- Mail it to the address on the back (postage paid in U.S. only)
- Give it to an IBM marketing representative for mailing
- Fax it to: Your International Access Code + 1 914 432 8246
- Send a note to REDBOOK@VNET.IBM.COM

**Please rate on a scale of 1 to 5 the subjects below.**
**(1 = very good, 2 = good, 3 = average, 4 = poor, 5 = very poor)**

| | | | |
|---|---|---|---|
| **Overall Satisfaction** | ____ | | |
| Organization of the book | ____ | Grammar/punctuation/spelling | ____ |
| Accuracy of the information | ____ | Ease of reading and understanding | ____ |
| Relevance of the information | ____ | Ease of finding information | ____ |
| Completeness of the information | ____ | Level of technical detail | ____ |
| Value of illustrations | ____ | Print quality | ____ |

**Please answer the following questions:**

a) If you are an employee of IBM or its subsidiaries:

Do you provide billable services for 20% or more of your time? Yes____ No____

Are you in a Services Organization? Yes____ No____

b) Are you working in the USA? Yes____ No____

c) Was the Bulletin published in time for your needs? Yes____ No____

d) Did this Bulletin meet your needs? Yes____ No____

If no, please explain:

_____

_____

What other topics would you like to see in this Bulletin?

_____

_____

What other Technical Bulletins would you like to see published?

_____

**Comments/Suggestions: ( THANK YOU FOR YOUR FEEDBACK! )**

_____ _____
Name Address

_____ _____
Company or Organization

_____ _____
Phone No.

Cut or Fold Along Line

Fold and Tape **Please do not staple** Fold and Tape

NO POSTAGE NECESSARY IF MAILED IN THE UNITED STATES

# BUSINESS REPLY MAIL

FIRST-CLASS MAIL PERMIT NO. 40 ARMONK, NEW YORK

POSTAGE WILL BE PAID BY ADDRESSEE

IBM International Technical Support Organization
Mail Station P099
522 SOUTH ROAD
POUGHKEEPSIE NY
USA 12601-5400

Fold and Tape **Please do not staple** Fold and Tape



Cut or Fold Along Line

Printed in U.S.A.

GG24-3851-02