

z/VM  
7.3

*VMSES/E Introduction and Reference*



**Note:**

Before you use this information and the product it supports, read the information in [“Notices” on page 787.](#)

This edition applies to version 7, release 3 of IBM® z/VM® (product number 5741-A09) and to all subsequent releases and modifications until otherwise indicated in new editions.

Last updated: 2022-09-07

© **Copyright International Business Machines Corporation 1990, 2022.**

US Government Users Restricted Rights – Use, duplication or disclosure restricted by GSA ADP Schedule Contract with IBM Corp.

---

# Contents

<b>Figures.....</b>	<b>xv</b>
<b>Tables.....</b>	<b>xxvii</b>
<b>About this Document.....</b>	<b>xxix</b>
Intended Audience.....	xxix
Syntax, Message, and Response Conventions.....	xxix
Where to Find More Information.....	xxxii
<b>How to Send Your Comments to IBM.....</b>	<b>xxxiii</b>
<b>Summary of Changes for z/VM: VMSES/E Introduction and Reference.....</b>	<b>xxxv</b>
GC24-6336-73, z/VM 7.3 (September 2022).....	xxxv
GC24-6336-01, z/VM 7.2 (August 2021).....	xxxvi
GC24-6336-01, z/VM 7.2 (September 2020).....	xxxvi
GC24-6336-00, z/VM 7.1 (September 2018).....	xxxvii
<b>Part 1. Introduction.....</b>	<b>1</b>
Chapter 1. Introducing VMSES/E.....	3
Installing Products.....	4
Servicing Products.....	4
Servicing Non-VMSES/E Products.....	4
Managing Saved Segments.....	4
Managing Product Inventories.....	5
Selecting the Correct Method for Installation and Service.....	5
Finding Out Where to Begin.....	5
<b>Part 2. Installing, Migrating, Building, and Deleting Products.....</b>	<b>7</b>
Chapter 2. Introducing the VMFINS EXEC.....	9
Overview.....	9
VMFINS INSTALL.....	9
VMFINS MIGRATE.....	10
VMFINS BUILD.....	10
VMFINS DELETE.....	10
Who Can Use VMFINS?.....	10
What Tape Formats Does VMFINS Support?.....	10
What Type of Processing Does VMFINS Provide?.....	11
What Is the Software Inventory?.....	11
Where Does It Reside?.....	12
Files Shipped on the Product Installation Media.....	13
Files Created and Updated during Installation and Migration.....	13
Chapter 3. Using the VMFINS EXEC.....	17
Determining Which Operand to Use: INSTALL, MIGRATE, BUILD, or DELETE?.....	17
Installing a Product.....	17
Migrating a Product.....	17
Building a Product.....	17

Deleting a Product.....	17
Installing a Recommended Service Upgrade (RSU).....	18
Using the INFO Operand.....	18
The VMFINS PRODLIST File.....	18
Which Products Can You Install?.....	19
Which Products Can You Migrate?.....	20
Using the PROD Operand.....	20
Using the PPF Operand.....	20
Using the LIST Operand.....	20
Using the LIST Operand with the PLAN Option.....	21
Printing the Memo-to-Users.....	21
Using the INFO Operand and the MEMO Option.....	21
Using the PLAN Option.....	23
Missing Requisites.....	24
Determining Whether a Product Can Be Installed.....	24
Determining Whether a Product Can Be Migrated.....	27
Determining Whether a Product Can Be Deleted.....	30
Calculating Space Requirements for Installation and Migration.....	33
Overriding Product Installation Defaults.....	33
Using the Make Override Panel.....	33
Where Do the Product Installation Parameters Come From?.....	33
Understanding the Make Override Panel Information.....	34
Using the Function Keys.....	35
Using the Action Bar.....	37
Understanding the File Options.....	38
Saving a Product Parameter File Override.....	38
Understanding the Help Options.....	42
Getting Help on Help.....	43
Getting Help on the Function Keys.....	44
Getting Basic Instructions on the Make Override Panel.....	45
Online Help.....	45
Changing from Minidisk to SFS Directory Entry Format.....	46
The Default Shared File System Directory Name.....	46
Changing the Shared File System Directory File Pool Name.....	47
Controlling the VMFINS Prompts.....	48
Changing the VMFINS Command Defaults.....	48
Moving a Product from Test Mode to Production Mode.....	48
Chapter 4. Installing Products with VMFINS.....	49
Adding Products to Your System.....	49
Adding a Single Product.....	49
Adding Several Products.....	50
Replacing Products on Your System.....	51
Replacing a Single Product.....	51
Replacing Several Products.....	52
Changing the Product Installation Defaults.....	53
Scenario 1: Installing a Product with the PPF Operand.....	54
Step 1. Create the VMFINS PRODLIST File and Print Memo-to-Users.....	54
Step 2. Find the Product Identifiers for IBM XL C/C++ for z/VM Compiler.....	55
Step 3. Determine if You Have a Usable Form PPF.....	55
Step 4. Run the PLAN Option.....	55
Step 5. Review the 5654A22C PLANINFO File.....	56
Step 6. Install CCXX.....	57
Step 7. Review the Output Files.....	58
Summary.....	60
Using the VMFSIM EXEC during an Installation.....	61
Chapter 5. Migrating Products with VMFINS.....	63

Adding a New Release of a Product to Your System.....	63
Adding a Single Product.....	64
Adding Several Products.....	65
Replacing Products on Your System.....	65
Replacing a Product.....	66
Replacing Several Products.....	67
Changing the Current Product Installation Settings.....	68
Retailing Your Product Files.....	68
Split-Screen XEDIT Session.....	70
View-Screen Session.....	74
Using the VMFSIM EXEC during a Migration.....	76
Chapter 6. Building Products with VMFINS.....	77
When Should You Perform a Build?.....	77
Building Products on Your System.....	77
Scenario 1: Building a Product with the PPF Operand.....	77
Step 1. Find the ppfname for the Product.....	78
Step 2. Build CCXX.....	78
Step 3. Review the Output Files.....	79
Summary.....	80
Chapter 7. Deleting Products with VMFINS.....	81
Deleting Products from Your System.....	81
Scenario 1: Deleting a Product with the PPF Operand.....	81
Step 1. Find the ppfname for the Copy We Want to Delete.....	81
Step 2. Run the PLAN Option.....	82
Step 3. Review the 5654A22C PLANINFO and 5654A22C ERASE Files.....	83
Step 4. Delete CCXX.....	83
Step 5. Review the Output Files.....	84
Summary.....	85
<b>Part 3. Servicing Products.....</b>	<b>87</b>
Chapter 8. z/VM Service Concepts.....	89
What is z/VM Product Service?.....	89
Correcting a Problem.....	89
Circumventing a Problem.....	89
Adding Function.....	89
Applying Local Service or Modifications.....	89
Product and Service Structure.....	89
Usable Forms.....	90
Serviceable Parts.....	90
Updates.....	90
Program Temporary Fixes (PTFs).....	90
Relationship Between Serviceable Parts and the Usable Form.....	90
Types of Service Supported by VMSES/E.....	92
Product Service Upgrade (PSU).....	92
Corrective Service (COR).....	92
Expanded Service Option (ESO).....	93
Chapter 9. Installing Corrective Service.....	95
Chapter 10. Using the Product Service Upgrade (PSU).....	97
Chapter 11. Installing Local Service and Modifications.....	101
Introduction.....	101
Overview for Local Service Procedure.....	102

Overview for Rework Local Service Procedure.....	103
Obtaining File Type Abbreviations.....	103
Chapter 12. Using VMSES/E for Service.....	105
The VMSES/E Database.....	105
Servicing a Product with VMSES/E.....	106
Receiving Service.....	107
Applying Service.....	107
Reapplying Local Service.....	107
Building New Levels.....	108
Placing the Serviced Components into Production.....	108
A Closer Look at the VMFREC EXEC.....	108
Processing Multiple Service Tapes at One Time.....	108
Part Handlers.....	108
Software Inventory Files Used by the VMFREC EXEC.....	109
A Closer Look at the VMFAPPLY EXEC.....	109
Applying a PTF.....	110
Software Inventory Files Used by the VMFAPPLY EXEC.....	111
A Closer Look at the VMFBLD EXEC.....	112
Requisite Processing.....	112
Software Inventory Files Used by the VMFBLD EXEC.....	112
Other VMSES/E EXECs.....	113
Consolidating Levels of the Database.....	114
Managing Product Parameter Files.....	115
The VMFOVER EXEC.....	115
The VMFPPF EXEC.....	115
Managing Disks for the Service Database.....	116
The VMFSETUP EXEC.....	116
The VMFQMDA EXEC.....	116
Managing Objects.....	116
The VMFQOBJ EXEC.....	116
Managing Saved Segments.....	116
The VMFSGMAP EXEC.....	116
Other VMSES/E Functions.....	117
Regenerating Parts Locally.....	117
Viewing Message Logs.....	117
How VMSES/E Uses Control Files.....	117
Control Files.....	117
Control File Extensions.....	119
Auxiliary Control Files.....	120
Version Vector Tables.....	121
Patch Update Files.....	121
Creating Updated Source Files.....	122
Selecting the Latest Version of the Serviceable Part.....	123
Creating Text Decks.....	124
Identifying MACLIBs.....	124
Determining Local Modifications Requiring Rework.....	124
Chapter 13. Other Files Used in the Service Process.....	125
The Tape Document.....	125
The Tape Descriptor File.....	125
For Installation, RSU, and COR Descriptor File.....	125
For Installation and RSU Descriptor File.....	126
For COR Descriptor File.....	126
The Product Contents Directory.....	127
The Memo-to-Users.....	127
The Program Level File.....	127
The Select Data File.....	128

File Syntax.....	128
Example.....	129
Select Data File Used for System Objects.....	130
The VMSESE PROFILE.....	131
File Syntax.....	131
Example.....	132
The Apply List.....	133
File Syntax.....	133
Example.....	133
The Exclude List.....	134
File Syntax.....	134
Example.....	134
Place Into Production Files.....	135
The SERVICE \$PRODS File .....	135
The <i>systemid</i> \$PRODS file.....	137
The Message Log.....	137
The VMFINS DEFAULTS File.....	139
Syntax.....	139
Example.....	140
Usage Notes.....	141
Build Lists.....	141
Syntax Notation.....	142
Format 1 Build List Syntax.....	142
Format 1 Build List Example.....	144
Format 2 and 3 Build List Syntax.....	144
Format 2 Build List Examples.....	152
Format 3 Build List Example.....	154
The National Language Support Table (VMFNLS LANGLIST).....	154
File Syntax.....	154
Example.....	155

**Part 4. Planning and Managing Your Software Inventories..... 157**

Chapter 14. Introduction to the Product Parameter File.....	159
Types of Product Parameter Files.....	159
Source Product Parameter Files.....	159
Override Product Parameter Files.....	160
Temporary Product Parameter Files.....	160
Usable Form Product Parameter Files.....	160
Sections of the Product Parameter File.....	160
The Control Options Section.....	161
The Variable Declarations Section.....	161
The Minidisk/Directory Assignments Section.....	161
The Receive Installation Tape Definition Section.....	161
The Receive Service Media Definition Section.....	161
The Build Product Definitions Section.....	161
The File Type Abbreviations Extensions Section.....	161
Syntax of the Product Parameter File.....	161
Chapter 15. Introduction to the Software Inventory.....	163
Overview of the System-Level Software Inventory.....	164
Types of Information Provided.....	165
The Contents of the System-Level Software Inventory.....	166
Overview of the Service-Level Software Inventory.....	171
Types of Information Provided.....	172
The Contents of the Service-Level Software Inventory.....	173

Chapter 16. Introduction to the VMFSIM EXEC.....	177
Providing Input to VMFSIM.....	177
Receiving Output from VMFSIM.....	177
Querying the Software Inventory.....	177
Querying the System-Level Software Inventory after Receive Processing.....	177
Querying the System-Level Software Inventory after Apply Processing.....	178
Querying the System-Level Software Inventory after Build Processing.....	178
Performing Additional Queries on the System-Level Software Inventory.....	179
Querying the Service-Level Software Inventory after Receive Processing.....	180
Querying the Service-Level Software Inventory after Apply Processing.....	181
Querying the Service-Level Software Inventory after Build Processing.....	183
Performing Additional Queries on the Service-Level Software Inventory.....	184
Updating the Software Inventory.....	186
Adding A Product to the System-Level Software Inventory.....	186
Updating the SRVBLDS Table after Manual Build Processing.....	187
Comparing the Version Vector Table to the AUX File Structure.....	188
Checking that the AUX Files and Version Vector Table Match.....	188
Adding Local AUX File Entries to the Service-Level Software Inventories.....	188
Identifying the Latest Version of a Part.....	189
Determining the Current Service Level of a Part.....	189
Comparing Two Software Inventory Tables.....	189
Building an APPLY List of All PTFs Received and Not Applied.....	189
Creating an APPLY List from Two SRVAPPS Tables.....	190
Building Apply and Exclude Lists after Receive Processing.....	190
Building an APPLY List Containing All Requisites of a PTF.....	190
Building an EXCLUDE List Containing All Dependents of a PTF.....	191
Listing the Requisites for PTFs.....	191
Determining the Prerequisites for a PTF.....	191
Listing the Requisites for a Product.....	192
Determining the Prerequisites for a Product.....	192
Listing the Dependent PTFs for Another PTF.....	192
Determining the PTFs Dependent on a PTF.....	193
Listing the Dependent Products for Another Product.....	193
Determining the Dependents of a Product.....	193
Adding Local Modifications to the Software Inventory.....	193
Adding Local Source Update Modifications to Service-Level Software Inventories.....	194
Adding Local Replacement Files to Service-Level Software Inventories.....	194
Initializing and Recovering the Software Inventory Tables.....	195
Recovering the System-Level Software Inventories.....	195
Recovering Service-Level Software Inventories.....	196
 Chapter 17. Using the VMFINFO Panels.....	 199
Where Does the Information Come From?.....	199
Understanding the VMFINFO Panel Information.....	199
Getting Started.....	201
Selection Panels.....	201
VMFINFO Main Panel.....	203
Product Description Query.....	204
Product Status Query.....	204
Product Requisites Query.....	205
Product Dependencies Query.....	205
PTF/APAR Query Panel.....	206
Serviceable Parts/Usable Forms Query Panel.....	211
Miscellaneous Queries Panel.....	216
Compare Table Contents Panel.....	218
File Type Abbreviations Panel.....	219



Chapter 18. Changing the Software Inventory to an SFS Directory.....	221
Create the SFS Directory.....	221
Initialize the VMPSFS:MAINTvrm.SIDISK Directory.....	221
Change the Software Inventory Default from Minidisk to SFS Directory.....	221
Enroll Users and Give Them Access Authority.....	223

**Part 5. Reference..... 225**

Chapter 19. Using the VMSES/E Reference Information.....	227
Understanding Syntax Diagrams.....	227
Using the Online HELP Facility.....	227
Using the VMSES/E Commands.....	227

Chapter 20. VMSES/E Exec and Command Format Summaries.....	229
Using Tools for Service and System Generation.....	229
CHKAPARS EXEC.....	234
GENCPBLS EXEC.....	237
LOCALMOD EXEC.....	245
PRODUTL EXEC.....	249
PRODUTL File Exclusion Support.....	250
Messages and Return Codes.....	251
CATALOG Files.....	252
PUT2PROD EXEC.....	258
SERVICE EXEC.....	261
SERVMGR EXEC.....	269
SERVMGR INITIALIZE.....	270
SERVMGR SYSTEM.....	272
SERVMGR SRVLVL.....	279
SERVMGR REMOVE.....	285
SERVMGR MANAGED.....	288
VMFAPPLY EXEC.....	291
VMFASM EXEC.....	297
ASSEMBLE Options Supported by VMFASM.....	303
VMFBLD EXEC.....	305
VMFBLD BLDDATA File.....	311
Build List Options.....	312
Usage Notes.....	320
Examples.....	321
Input and Output Files.....	322
Messages and Return Codes.....	325
Recovery Information.....	325
Creating Objects with VMFBLD.....	325
Callable Services Libraries (CSL).....	326
Restore from DDR Image Files.....	330
CMS/DOS Phase Libraries (DOSLIB).....	331
Generated Objects.....	333
Load to the Byte File System.....	336
Objects Serviced by Complete Replacement.....	337
Text Objects Serviced by Complete Replacement (Format 1 Build List).....	340
LOADLIBs.....	341
MACLIBs.....	344
Executable Modules.....	347
Nuclei.....	352
Identifying System Objects to be Built.....	354
Saved Segments.....	355
TXTLIBs.....	359

SMAPI Appliance Servers and Stand-Alone Dump Utility.....	362
VMFBTMAP EXEC.....	364
The BITMAP File Structure.....	365
VMFCNVT EXEC.....	367
VMFCOPY EXEC.....	369
VMFENRPT EXEC.....	373
VMFERASE EXEC.....	377
VMFEXUPD EXEC.....	381
EXECUPDT Options Supported by VMFEXUPD.....	386
UPDATE Options Supported by VMFEXUPD.....	386
VMFHASM EXEC.....	387
VMFHLASM EXEC.....	394
VMFINFO EXEC.....	402
VMFINS EXEC.....	404
VMFINS BUILD Command.....	405
VMFINS DELETE Command.....	412
VMFINS DISABLE Command.....	417
VMFINS ENABLE Command.....	421
VMFINS INSTALL Command.....	425
VMFINS MIGRATE Command.....	433
Step 1. Re-IPL CMS.....	440
Step 2. Try to Determine What Stopped Your Migration.....	440
Step 3. Correct the Problem.....	441
Step 4. Finish Your Migration.....	441
Step 5. Manually Complete Your Migration (Optional).....	442
VMFMRDSK EXEC.....	444
VMFNLS EXEC.....	448
VMFOVER EXEC.....	456
VMFPPF EXEC.....	458
VMFPSU EXEC.....	462
VMFQMDA EXEC.....	469
VMFQOBJ EXEC.....	472
VMFREC EXEC.....	477
Filespec Operands.....	484
VMFREM EXEC.....	485
VMFREPL EXEC.....	493
VMFSETUP EXEC.....	500
VMFSGMAP EXEC.....	506
VMFSIM EXEC.....	525
VMFSIM: Tagged Data (TDATA).....	527
Using File Input.....	527
Querying Multiple Tables Using the File Option.....	527
Using the STEM Variable.....	528
Considerations for Using Stem Input and Output.....	530
VMFSIM CHKLVL.....	531
VMFSIM COMPTBL.....	538
VMFSIM GETLVL.....	543
VMFSIM INIT.....	550
VMFSIM LOGMOD.....	556
VMFSIM MODIFY.....	562
VMFSIM QUERY.....	567
VMFSIM SRVDEP.....	573
VMFSIM SRVREQ.....	579
VMFSIM SYSDEP.....	585
VMFSIM SYSREQ.....	591
VMFSUFIN EXEC.....	597
VMFSUFTB EXEC.....	602
VMFUPDAT EXEC.....	604

VMFVIEW EXEC.....	614
Chapter 21. Product Parameter File Syntax.....	621
Structure of the Data in Product Parameter Files.....	621
Data Records.....	621
Comments.....	621
File Structure of Product Parameter Files.....	621
Product Parameter File Processing.....	622
Source Product Parameter File Syntax.....	623
Header Area.....	623
Component Area.....	624
Control Options Section.....	625
Variable Declarations Section.....	631
Minidisk/Directory Assignments Section.....	633
Place Into Production Section.....	635
Receive Installation Tape Definition Section.....	637
Receive Service Media Definition Section.....	639
Build Product Definition Section.....	641
File Type Abbreviations Extensions Section.....	644
Override Area.....	644
Override Product Parameter File Syntax.....	653
Header Area.....	653
Override Area.....	653
Temporary Product Parameter File Syntax.....	654
Usable Form Product Parameter File Syntax.....	654
Component Area.....	654
Examples of Overrides.....	655
Inserting a Record.....	655
Deleting Records.....	656
Updating a Record - Single-Level Override.....	656
Updating a Record - Multi-Level Override.....	657
Chapter 22. Software Inventory Syntax.....	659
Structure of the Data in the Software Inventory Tables.....	659
Delimiters.....	659
The System-Level Software Inventory.....	659
The Software Inventory Defaults.....	660
Changing the Software Inventory Defaults.....	660
The Product Parts (PRODPART) File.....	660
The Saved Segment Data (SEGDATA) File.....	677
The Migration Parts Table ( <i>prodid</i> MIGPvrm).....	682
The System-Level Description Table (VM SYSDESCT).....	684
The System-Level Memo Table (VM SYSMEMO).....	685
The System-Level Requisite Table (VM SYSREQT).....	686
The System-Level Receive Status Table (VM SYSRECS).....	688
The System-Level Apply Status Table (VM SYSAPPS).....	690
The System-Level Build Status Table (VM SYSBLDS).....	692
The System-Level Service Update Facility Table (VM SYSSUF).....	693
The System-Level Product Inventory Table (VM SYSPINV).....	696
The System-Level Restart Table (VM SYSREST).....	696
The System-Level Local Modification Table (VM SYSLMOD).....	699
The System-Level Base APAR Table (VM SYSAPARS).....	701
The File Type Abbreviation Table (VM SYSABRVT).....	702
The Parts Catalog (VMSES PARTCAT).....	703
The Service-Level Software Inventory.....	704
The PTF Parts (\$PTFPART) File.....	705
The Service-Level Description Table (recid SRVDESCT).....	712
The Service-Level Requisite Table (recid SRVREQT).....	713

The Service-Level Receive Status Table (recid SRVRECS).....	715
The Service-Level Apply Status Table (appid SRVAPPS).....	716
The Service-Level Build Status Table (bldid SRVBLDS).....	717
The Service-Level Production Status Table (prodid SRVPROD).....	720
The Version Vector Table (appid VVTlvld).....	721
<b>Appendix A. Related Commands and EXECs.....</b>	<b>723</b>
INSTFPP EXEC.....	723
The Patch Facility.....	728
Controlling Patches.....	728
SNTINFO EXEC.....	733
<b>Appendix B. Input/Output Files.....</b>	<b>735</b>
<b>Appendix C. VMSES/E Sample Files.....</b>	<b>747</b>
<b>Appendix D. Module Identifiers for VMSES/E Messages.....</b>	<b>749</b>
<b>Appendix E. Tape Formats Supported by VMSES/E.....</b>	<b>753</b>
VMSES/E Product Media Format.....	753
z/VM System Delivery Offering Format.....	754
VMSES/E Service Tape Formats.....	754
<b>Appendix F. Servicing Non-VMSES/E SNA Products.....</b>	<b>757</b>
Types of Disks.....	757
Service Control File.....	758
Product Parameter File.....	759
How VMFMERGE, VMFREMOVE, and VMFZAP Use the PPF.....	760
Merge Log.....	760
ZAP Log.....	761
Reqby Log.....	762
Service Log.....	762
Apply List.....	763
Remove List.....	763
Exclude List.....	764
ZAP List.....	764
Object Code Service Processing.....	765
Applying Emergency Fixes Using ZAPs.....	765
Applying Corrective Service to Object Code.....	765
Applying Preventive Service to Object Code.....	767
Merge Service.....	767
Merging a Single PTF (No Dependents or Supersedes).....	767
Merging Multiple PTFs (with Dependents and Supersedes).....	768
Remove Service.....	770
Removing a Single PTF (No Dependents or Supersedes).....	770
Removing Multiple PTFs (with Dependents and Supersedes).....	772
Prevent Regression.....	773
Removing a Fix-in-Error.....	774
VMFMERGE EXEC.....	774
VMFREMOVE EXEC.....	777
VMFZAP EXEC.....	779
ZAPTEXT EXEC.....	781
EXPAND Command.....	783
<b>Notices.....</b>	<b>787</b>
Programming Interface Information.....	788

Trademarks.....	788
Terms and Conditions for Product Documentation.....	788
IBM Online Privacy Statement.....	789
<b>Bibliography.....</b>	<b>791</b>
Where to Get z/VM Information.....	791
z/VM Base Library.....	791
z/VM Facilities and Features.....	792
Prerequisite Products.....	794
Related Products.....	794
<b>Index.....</b>	<b>797</b>



---

# Figures

- 1. Introducing VMSES/E..... 1
- 2. VMSES/E Overview..... 3
- 3. VMSES/E - Installing, Migrating, Building, and Deleting Products.....7
- 4. The System-Level and Service-Level Software Inventories..... 12
- 5. A VMFINS PRODLIST File.....18
- 6. The VMFINS PRODLIST File after Updates by the LIST Operand and PLAN Option..... 21
- 7. Selecting the Memo-to-Users for Printing.....22
- 8. PLANINFO File Created by a VMFINS INSTALL Command with the PLAN Option..... 26
- 9. PLANINFO File Created by a MIGRATE with the PLAN Option..... 29
- 10. PLANINFO File Created by a VMFINS DELETE with the PLAN Option.....32
- 11. Make Override Panel..... 33
- 12. Reading the Make Override Panel.....34
- 13. The Function Keys on the Make Override Panel.....35
- 14. Using the Action Bar on the Make Override Panel..... 37
- 15. Using the Action Bar to Save Product Parameter File Overrides..... 38
- 16. The Options for File..... 39
- 17. Selecting a File Option..... 40
- 18. The Save as... Window..... 40
- 19. Saving the New Product Parameter File Override with a New Name..... 41
- 20. Returning to the Make Override Panel.....41
- 21. The Options for Help..... 42
- 22. Selecting an Option in Help..... 43
- 23. Help on Help Window.....43

24. Help on Function Keys Window.....	44
25. Selecting Help on the F6=Mdisk or SFS Dir Function Key.....	44
26. Help Description for the F6=Mdisk or SFS Dir Function Key.....	45
27. Selecting Basic Instructions.....	45
28. Changing from Minidisk to Shared File System Directory Entry.....	46
29. Expand Dirid Window.....	47
30. Entering the Complete Shared File System Directory ID.....	47
31. The Make Override Panel.....	53
32. Printing the Memo-to-Users.....	54
33. VMFINS PRODLIST File.....	55
34. Finding the Usable Form Product Parameter File for CCXX.....	55
35. 5654A22C PLANINFO File Created by the PLAN Option (1 of 2).....	56
36. 5654A22C PLANINFO File Created by the PLAN Option (2 of 2).....	57
37. \$VMFINS \$MSGLOG Created during Installation Processing.....	59
38. The VMFINS CONSOLE File Created during Installation Processing.....	60
39. Sample Make Override Panel.....	68
40. The VMFINS MIGRATE Restore Tailorings Phase Screen.....	69
41. The VMFINS MIGRATE Message Screen Asking You If You Want to Tailor Your Customized Files.....	69
42. Sample Split-Screen XEDIT Session.....	70
43. Using the CUTC Prefix Command in a Split-Screen File.....	71
44. Using the PLACE Prefix Command in a Split-Screen File.....	72
45. Split-Screen File after a CUTC and PLACE.....	72
46. Final Tailorings Phase Screen.....	73
47. Final Tailorings Phase Screen.....	73
48. Final Tailorings Phase Screen.....	74



49. Sample View-Screen Session.....	75
50. Final Tailorings Phase Screen.....	75
51. Final Tailorings Phase Screen.....	76
52. PPF Fileid - Help Panel.....	78
53. The \$VMFINS \$MSGLOG File Created during BUILD Processing.....	79
54. The VMFINS CONSOLE File Created during BUILD Processing.....	80
55. PPF Fileid - Help Panel.....	82
56. 5654A22C ERASE File Created during the PLAN Processing.....	83
57. The \$VMFINS \$MSGLOG File Created during DELETE Processing.....	84
58. The VMFINS CONSOLE File Created during DELETE Processing.....	85
59. VMSES/E - Servicing Your System.....	87
60. Serviceable Part to Usable Form Relationship.....	90
61. CMS MODULE Example (Serviced by Updates).....	91
62. CMS MODULE Example (Serviced by Text Replacement).....	91
63. CMS MODULE Example (Serviced by Module Replacement).....	91
64. MACRO Example (Parts Serviced by Updates Only).....	92
65. Installing Corrective Service.....	96
66. Recommended Service Upgrade, Files.....	97
67. Service Application Flowchart using PSU.....	98
68. Servicing a Product.....	107
69. The Apply Algorithm.....	110
70. Software Inventory Tables Updated During VMFAPPLY Processing.....	111
71. VMFMRDSK EXEC Example.....	114
72. VMFPPF EXEC Examples.....	115
73. Example of a Patch Update File, DMKMNT VM12345.....	121

74. Control File Structure.....	122
75. Control File Structure Using Version Vector Tables.....	123
76. Example of a Program Level File - 1VMVMC23 0123081.....	128
77. A Sample Select Data File.....	130
78. Example of the VMSBR \$SELECT File.....	130
79. Example of the SEGBLD \$SELECT File.....	130
80. Example of an Apply List–DMSVM \$APPLIST.....	134
81. Example of an Exclude List–DMSVM \$EXCLIST.....	135
82. A Sample Message Log: \$VMFREC \$MSGLOG.....	139
83. The VMFINS DEFAULTS File.....	141
84. Example of a Format 1 Build list.....	144
85. Example of Format 3 Build List for TXTLIB.....	147
86. Format 2 Build List Example.....	153
87. System Saved Segment Build List Example.....	153
88. Product Saved Segment Build List Example.....	153
89. Format 3 Build List Example.....	154
90. The VMFNLS LANGLIST File.....	155
91. VMSES/E - Software Inventory Management.....	158
92. Product Parameter File Relationship.....	159
93. Software Inventory Files in the VMSES/E Database.....	164
94. System-Level Description Table Example.....	167
95. System-Level Requisite Table Example.....	167
96. System-Level Receive Status Table Example.....	168
97. System-Level Apply Status Table Example.....	168
98. System-Level Build Status Table Example.....	169

99. System-Level Service Update Facility Table.....	169
100. System-Level Restart Table Example.....	170
101. File Type Abbreviation Table Example.....	170
102. Parts Catalog Table Example.....	171
103. Service-Level Description Table Example.....	173
104. Service-Level Requisite Table Example.....	174
105. Service-Level Receive Status Table Example.....	174
106. Service-Level Apply Status Table Example.....	175
107. Service-Level Build Status Table Example.....	175
108. Version Vector Table Example.....	176
109. Reading the VMFINFO Main Panel.....	199
110. The Function Keys on the VMFINFO Panel.....	200
111. PPF Fileid - Help Panel.....	202
112. Component Name - Help Panel.....	202
113. VMFINFO Main Panel.....	203
114. Product Description Query Output.....	204
115. Product Status Query Output.....	204
116. Product Requisite Query Output.....	205
117. Product Dependencies Query Output.....	206
118. PTF/APAR Query Panel.....	206
119. PTF Status Query Output.....	208
120. PTF Requisites/Supersedes Query Output.....	208
121. PTF Dependencies/Superseding Query Output.....	209
122. PTF User Memo Query Output.....	209
123. PTF Serviceable Parts Query Output.....	210

124. APAR Abstract Query Output.....	210
125. Serviceable Parts/Usable Forms Query Panel.....	211
126. Object Status Query Output.....	214
127. Object Build Requisites Query Output.....	214
128. Object Build Dependencies Query Output.....	215
129. Object Part Handler/Target Query Output.....	215
130. Part Service Level Query Output.....	216
131. Part Service History Query Output.....	216
132. Miscellaneous Queries Panel.....	217
133. Minidisk/Directory Access Query Output.....	217
134. Compare Table Contents.....	218
135. Compare Table Contents Query Output.....	219
136. Build Requirements Query Output.....	219
137. File Type Abbreviations Panel.....	220
138. File Type Abbreviations Query Output.....	220
139. VMSES/E - Reference Information.....	225
140. Syntax for VMFBLD LIST Input File (VMFBLD BLDDATA).....	311
141. Example Build List Used by VMFBDCLB.....	327
142. Example Build List Used by VMFBDDDR.....	330
143. Example Build List Used by VMFBDDL B.....	331
144. Example Build List Used by VMFBDGEN.....	334
145. Example Build List Used by VMFBDBFS.....	336
146. Example Build List Used by VMFBDCOM.....	338
147. Example Build List Used by VMFBDCPY.....	340
148. Example Build List Used by VMFBDLLB.....	341

149. Example Build List Used by VMFBTMLB.....	344
150. Example Build List Used by VMFBDMOD.....	347
151. Example Build List Used by VMFBPMD.....	349
152. Example Build List Used by VMFBNUC.....	352
153. Example Product Saved Segment Build List Used by VMFBDSBR and VMFBSEG.....	354
154. Example System Saved Segment Build List Used by VMFBSEG.....	355
155. Example Build List Used by VMFBTLB.....	360
156. Example of Format 3 Build List for TXTLIB.....	360
157. Example Build List Used by VMFBSSP.....	363
158. Product Enablement Report.....	375
159. PSUPLAN File.....	465
160. VMFQMDA Sample Output.....	469
161. VMFSETUP Sample Output.....	503
162. VMFSGMAP Segment Map Panel with Status Codes.....	508
163. VMFSGMAP Segment Map Panel with Spool File Classes.....	511
164. VMFSGMAP Segment Definition Panel.....	514
165. VMFSGMAP Query NSS Map Panel.....	521
166. Example of the VMFSGMAP Segment Map Panel.....	523
167. Example of the VMFSGMAP Change Segment Definition Panel.....	523
168. Software Inventory Data Structure.....	525
169. Tree Structure Format.....	530
170. VMFUPDAT Function Selection Panel.....	606
171. SRVBLDS Update Panel.....	607
172. SYSSUF Update Panel.....	608
173. SYSLMOD Update Panel.....	609

174. SYSREST Update Panel.....	610
175. SYSMEMO Update Panel.....	611
176. Product Parameter File Relationship.....	622
177. Sample Header Area of a Source PPF.....	624
178. Sample Component Area of a Source PPF.....	625
179. Sample :CNTRLOP Section of a PPF.....	631
180. Sample :DCL Section of a PPF.....	633
181. Sample :MDA Section of a Source PPF.....	635
182. Sample :MDA Section of a Usable Form PPF.....	635
183. Sample :P2P Section of a PPF.....	637
184. Sample :RECINS Section of a PPF.....	639
185. Sample :RECSER Section of a PPF.....	641
186. Sample :BLD Section of a PPF.....	643
187. Sample :DABBV Section of a PPF.....	644
188. Source Product Parameter File (\$PPF), MDA Section Override Syntax.....	649
189. Sample Header Area of an Override PPF.....	653
190. Sample Component Area of a Usable Form PPF.....	655
191. Software Inventory Data Structure.....	659
192. Requisite OR Format Syntax.....	665
193. :PREREQ Tag Using OR Format.....	665
194. System-Level Requisite Table Example.....	688
195. System-Level Receive Status Table Example.....	690
196. System-Level Apply Status Table Example.....	692
197. System-Level Build Status Table Example.....	693
198. System-Level Service Update Facility Table Example.....	695

199. System-Level Product Inventory Table Example.....	696
200. System-Level Restart Table Example.....	699
201. System-level Base APAR Table Example.....	702
202. File Type Abbreviation Table Example.....	703
203. Parts Catalog Table Example.....	704
204. \$PTFPART File Header Section Example.....	706
205. \$PTFPART File Requisite Section Example.....	708
206. \$PTFPART File Parts Section Example.....	711
207. Example \$PTFPART File.....	712
208. Service-Level Description Table Example.....	713
209. Service-Level Requisite Table Example.....	715
210. Service-Level Receive Status Table Example.....	716
211. Service-Level Apply Status Table Example.....	717
212. Service-Level Build Status Table Example.....	720
213. Service-Level Apply Status Table Example.....	721
214. Version Vector Table Example.....	722
215. Version Vector Table Local Modification Entry Example.....	722
216. INSTFPP Installation Options Panel.....	725
217. A Sample INSTFPP Product Selection Panel.....	726
218. Sample PROD LEVEL File Entries.....	726
219. Example of a Service Control File (UV00006 SCF).....	758
220. Example of a Product Parameter File.....	759
221. Example of a Merge Log Produced by VMFMERGE.....	761
222. Example of a ZAP Log (5664175 VMFZPLOG).....	761
223. Example of a Reqby Log (5664167 VMFREQBY).....	762

224. Example of a Service Log (5664175 VMFSVLOG).....	763
225. Example of an Apply List (5664175 APPLIST).....	763
226. Example of a Remove List (5664175 REMLIST).....	764
227. Example of an Exclude List (5664175 EXCLIST).....	764
228. Example of a Zap List (5664175 ZAPLIST).....	764
229. Merge Single PTF—Sample Merge Log for 5664167.....	767
230. Merge Single PTF—Sample Exclude List for 5664167.....	767
231. Merge Single PTF—Sample SCF for UV00001.....	768
232. Merge Single PTF—Sample SCF for UV00002.....	768
233. Merge Single PTF—Changed merge log for 5664167.....	768
234. Merge Multiple PTFs—Sample Merge Log for 5664167.....	768
235. Merge Multiple PTFs—Sample SCF for UV00001.....	768
236. Merge Multiple PTFs—Sample SCF for UV00002.....	769
237. Merge Multiple PTFs—Sample SCF for UV00003.....	769
238. Merge Multiple PTFs—Sample SCF for UV00004.....	769
239. Merge Multiple PTFs—Sample SCF for UV00005.....	769
240. MYLIST APPLIST.....	769
241. Merge Multiple PTFs—Sample Changed Merge Log for 5664167.....	770
242. Merge Multiple PTFs—Changed Reqby Log for 5664167.....	770
243. Remove Single PTF—Merge Log for 5664167.....	770
244. Remove Single PTF—SCF for UV00001.....	770
245. Remove Single PTF—SCF for UV00004.....	771
246. Remove Single PTF—SCF for UV00005.....	771
247. Remove Single PTF—SCF for UV00006.....	771
248. Remove Single PTF—SCF for UV00007.....	771



249. Remove Single PTF—SCF for UV00008.....	771
250. Remove Single PTF—SCF for UV00009.....	771
251. Remove Single PTF—Sample Reqby Log for 5664167.....	772
252. Remove Single PTF—Changed Merge Log for 5664167.....	772
253. Remove Single PTF—Changed Reqby Log for 5664167.....	772
254. MYLIST REMLIST.....	773
255. Remove Multiple PTFs—Changed Merge Log for 5664167.....	773
256. Remove Multiple PTFs—Changed Reqby Log for 5664167.....	773



---

# Tables

1. Examples of Syntax Diagram Conventions.....	xxix
2. Methods of Installation and Service.....	5
3. Processing by Product Format.....	11
4. Function Key Assignments for the Make Override Panel.....	35
5. Program Function (PF) Keys for Split-Screen XEDIT Session.....	70
6. Program Function (PF) Keys for View-Screen Session.....	75
7. The VMSES/E Database Defaults.....	105
8. Tape Descriptor Files.....	126
9. Product Contents Directory.....	127
10. VMSES \$MSGLOG Message Codes.....	138
11. Build List Formats Used by the VMFBLD Part Handlers.....	141
12. Function Key Assignments for the VMFINFO Panels.....	200
13. Example PTF/APAR Query Inputs and Results.....	207
14. Serviceable Parts/Usable Forms Query Inputs and Results for Objects.....	211
15. Serviceable Parts/Usable Forms Query Inputs and Results for Parts.....	212
16. z/VM Service and System Generation Tools.....	229
17. Additional z/VM Service and System Generation Tools.....	231
18. VMFBLD EXEC Parameter Specifications and Objects Built.....	309
19. Build List Options.....	312
20. VMFBLD EXEC Parameter Specifications and Objects Built.....	408
21. Valid Tags for VMFQOBJ.....	474
22. Program Function (PF) Keys on the VMFSGMAP EXEC Segment Map Panel.....	511
23. Program Function (PF) Keys on the VMFSGMAP EXEC Segment Definition Panel.....	519

24. Program Function (PF) Keys on the VMFSGMAP EXEC Query NSS Map Panel.....	521
25. Program Function (PF) Keys on the VMFUPDAT Function Update Panel.....	607
26. Program Function (PF) Keys on the VMFUPDAT EXEC SRVBLDS Update Panel.....	607
27. Program Function (PF) Keys on the VMFUPDAT EXEC SYSSUF Update Panel.....	608
28. Program Function (PF) Keys on the VMFUPDAT EXEC SYSLMOD Update Panel.....	609
29. Program Function (PF) Keys on the VMFUPDAT EXEC SYSREST Update Panel.....	610
30. Program Function (PF) Keys on the VMFUPDAT EXEC SYSMEMO Update Panel.....	611
31. Default Program Function (PF) Key Assignments for the VMFVIEW EXEC.....	618
32. Control Structure Containing Patches at Multiple Levels.....	732
33. Control Structure Containing Temporary Patch Over Local Source Update.....	733
34. Input/Output Files.....	735
35. Message IDs.....	749

# About this Document

---

This document introduces you to the Virtual Machine Serviceability Enhancements Staged/Extended (VMSES/E) installation and service tool for IBM z/VM 7.3. It is intended to supplement information about the specific product you are installing or servicing. Reference information about servicing non-VMSES/E products is also included.

VMSES/E provides tools for installing, migrating, building, deleting, and servicing software on your system. VMSES/E also provides tools to help you manage your system software.

## Intended Audience

---

This information is intended for anyone responsible for installing, migrating, building, deleting, or servicing products on z/VM and those managing the z/VM software inventory.

A general knowledge of the z/VM operating system and z/VM commands is required for getting the most out of this information. It should also be understood and that the word "product" refers to both products and components, unless otherwise specified.

## Syntax, Message, and Response Conventions





---

The following topics provide information on the conventions used in syntax diagrams and in examples of messages and responses.

### How to Read Syntax Diagrams

Special diagrams (often called *railroad tracks*) are used to show the syntax of external interfaces.

To read a syntax diagram, follow the path of the line. Read from left to right and top to bottom.

- The  symbol indicates the beginning of the syntax diagram.
- The  symbol, at the end of a line, indicates that the syntax diagram is continued on the next line.
- The  symbol, at the beginning of a line, indicates that the syntax diagram is continued from the previous line.
- The  symbol indicates the end of the syntax diagram.

Within the syntax diagram, items on the line are required, items below the line are optional, and items above the line are defaults. See the examples in [Table 1 on page xxix](#).



Syntax Diagram Convention	Example
<b>Keywords and Constants</b> A keyword or constant appears in uppercase letters. In this example, you must specify the item KEYWORD as shown.  In most cases, you can specify a keyword or constant in uppercase letters, lowercase letters, or any combination. However, some applications may have additional conventions for using all-uppercase or all-lowercase.	 KEYWORD 

Table 1. Examples of Syntax Diagram Conventions (continued)

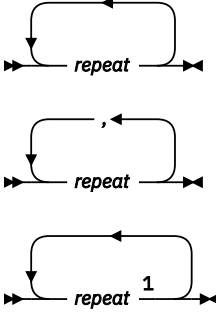
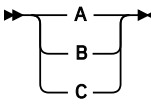
Syntax Diagram Convention	Example
<p><b>Abbreviations</b></p> <p>Uppercase letters denote the shortest acceptable abbreviation of an item, and lowercase letters denote the part that can be omitted. If an item appears entirely in uppercase letters, it cannot be abbreviated.</p> <p>In this example, you can specify KEYWO, KEYWOR, or KEYWORD.</p>	<p>▶▶ KEYWOrd ▶▶</p>
<p><b>Symbols</b></p> <p>You must specify these symbols exactly as they appear in the syntax diagram.</p>	<ul style="list-style-type: none"> <li>* Asterisk</li> <li>:</li> <li>,</li> <li>= Equal Sign</li> <li>- Hyphen</li> <li>() Parentheses</li> <li>.</li> <li>Period</li> </ul>
<p><b>Variables</b></p> <p>A variable appears in highlighted lowercase, usually italics.</p> <p>In this example, <i>var_name</i> represents a variable that you must specify following KEYWORD.</p>	<p>▶▶ KEYWOrd — <i>var_name</i> ▶▶</p>
<p><b>Repetitions</b></p> <p>An arrow returning to the left means that the item can be repeated.</p> <p>A character within the arrow means that you must separate each repetition of the item with that character.</p> <p>A number (1) by the arrow references a syntax note at the bottom of the diagram. The syntax note tells you how many times the item can be repeated.</p> <p>Syntax notes may also be used to explain other special aspects of the syntax.</p>	 <p>Notes:</p> <p><sup>1</sup> Specify <i>repeat</i> up to 5 times.</p>
<p><b>Required Item or Choice</b></p> <p>When an item is on the line, it is required. In this example, you must specify A.</p> <p>When two or more items are in a stack and one of them is on the line, you must specify one item. In this example, you must choose A, B, or C.</p>	<p>▶▶ A ▶▶</p> 

Table 1. Examples of Syntax Diagram Conventions (continued)

Syntax Diagram Convention	Example
<p><b>Optional Item or Choice</b></p> <p>When an item is below the line, it is optional. In this example, you can choose A or nothing at all.</p> <p>When two or more items are in a stack below the line, all of them are optional. In this example, you can choose A, B, C, or nothing at all.</p>	
<p><b>Defaults</b></p> <p>When an item is above the line, it is the default. The system will use the default unless you override it. You can override the default by specifying an option from the stack below the line.</p> <p>In this example, A is the default. You can override A by choosing B or C.</p>	
<p><b>Repeatable Choice</b></p> <p>A stack of items followed by an arrow returning to the left means that you can select more than one item or, in some cases, repeat a single item.</p> <p>In this example, you can choose any combination of A, B, or C.</p>	
<p><b>Syntax Fragment</b></p> <p>Some diagrams, because of their length, must fragment the syntax. The fragment name appears between vertical bars in the diagram. The expanded fragment appears in the diagram after a heading with the same fragment name.</p> <p>In this example, the fragment is named "A Fragment."</p>	

## Examples of Messages and Responses

Although most examples of messages and responses are shown exactly as they would appear, some content might depend on the specific situation. The following notation is used to show variable, optional, or alternative content:

**xxx**

Highlighted text (usually italics) indicates a variable that represents the data that will be displayed.

[ ]

Brackets enclose optional text that might be displayed.

{ }

Braces enclose alternative versions of text, one of which will be displayed.

|

The vertical bar separates items within brackets or braces.

...

The ellipsis indicates that the preceding item might be repeated. A vertical ellipsis indicates that the preceding line, or a variation of that line, might be repeated.

## Where to Find More Information

---

For information about related documents, see [“Bibliography” on page 791](#).



## How to Send Your Comments to IBM

---

We appreciate your input on this publication. Feel free to comment on the clarity, accuracy, and completeness of the information or give us any other feedback that you might have.

To send us your comments, go to [z/VM Reader's Comment Form \(https://www.ibm.com/systems/campaignmail/z/zvm/zvm-comments\)](https://www.ibm.com/systems/campaignmail/z/zvm/zvm-comments) and complete the form.

### **If You Have a Technical Problem**

Do not use the feedback method. Instead, do one of the following:

- Contact your IBM service representative.
- Contact IBM technical support.
- See [IBM: z/VM Support Resources \(https://www.ibm.com/vm/service\)](https://www.ibm.com/vm/service).
- Go to [IBM Support Portal \(https://www.ibm.com/support/entry/portal/Overview\)](https://www.ibm.com/support/entry/portal/Overview).



# Summary of Changes for z/VM: VMSES/E Introduction and Reference

---

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

## GC24-6336-73, z/VM 7.3 (September 2022)

---

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

### **z/VM Centralized Service Management enhancements**

z/VM Centralized Service Management (z/VM CSM) includes the following usability enhancements:

- QUERY processing for service levels now allows queries that are specific to individual components in a service level.
- Wildcard support is added to the SERVMGR SRVLVL QUERY command.
- The ability to query z/VM CSM managed systems for current PUT2PROD status is added.
- Additional details are provided when querying local modifications.

The following topics are updated:

- [“SERVMGR SYSTEM” on page 272](#)
- [“SERVMGR SRVLVL” on page 279](#)

### **Language Environment upgrade**

The z/VM Language Environment runtime libraries have been upgraded to z/OS® 2.5 equivalence.

The following topics are updated:

- [“Step 1. Find the ppfname for the Product” on page 78](#)
- [“Step 1. Find the ppfname for the Copy We Want to Delete” on page 81](#)
- [“VMFSETUP EXEC” on page 500](#)

### **System APAR Evaluation Utility: CHKAPARS EXEC**

The CHKAPARS EXEC evaluates system APAR data for a selected z/VM component or all components. APAR data for the subject z/VM system is acquired by using the VMSES/E SERVICE STATUS ALLAPARS command. The acquired data is compared to APAR reference data that is acquired from a specified CMS file. A report of the evaluation is produced in a program-created report file.

The following topic is new:

- [“CHKAPARS EXEC” on page 234](#)

The following topics are updated:

- [“Step 1. Find the ppfname for the Product” on page 78](#)
- [“Step 1. Find the ppfname for the Copy We Want to Delete” on page 81](#)
- [“VMFSETUP EXEC” on page 500](#)

## Miscellaneous updates for z/VM 7.3

The following topic is new:

- [“SERVMGR REMOVE” on page 285](#)

The following topics are updated:

- [“SERVMGR EXEC” on page 269](#)
- [“Object Parameters \(Replacement Objects\)” on page 339](#)
- [“Part Options \(Replacement Objects\)” on page 339](#)
- [Appendix B, “Input/Output Files,” on page 735](#)

## GC24-6336-01, z/VM 7.2 (August 2021)

---

This edition includes terminology, maintenance, and editorial changes.

The following topic is updated:

- [“SERVMGR SRVLVL” on page 279](#)

## GC24-6336-01, z/VM 7.2 (September 2020)

---

This edition includes changes to support the general availability of z/VM 7.2.

### **z/VM Centralized Service Management (z/VM CSM) for non-SSI environments**

z/VM provides support to deploy service to multiple systems, regardless of geographic location, from a centralized primary location that manages distinct levels of service for a select group of traditional z/VM systems. One system is designated as a principal system and uses the z/VM Shared File System (SFS) to manage service levels for a set of defined managed systems. The principal system builds service levels using the new service management command, SERVMGR, and existing VMSES/E SERVICE commands. This centralized service process keeps track of available service levels and manages the files needed to supply a customer-defined service level to a managed system.

#### **Attention:**

Before you initialize z/VM CSM, the PTF for APAR VM66428 *must* be:

1. Installed on the principal system and all remote systems in your z/VM CSM environment
2. Applied to any customer-defined z/VM CSM service level that is based on the BASE z/VM CSM service level (the service level that incorporates the initial z/VM 720 RSU).

See the [z/VM: Service Guide](#) for more information.

The following topics are new:

- [“SERVMGR EXEC” on page 269](#)
- [“SERVMGR INITIALIZE” on page 270](#)
- [“SERVMGR SYSTEM” on page 272](#)
- [“SERVMGR SRVLVL” on page 279](#)
- [“SERVMGR MANAGED” on page 288](#)
- [Appendix C, “VMSES/E Sample Files,” on page 747](#)

The following topics are updated:

- [“Viewing Message Logs” on page 117](#)
- [“The Message Log” on page 137](#)
- [“Using Tools for Service and System Generation” on page 229](#)

- [“SERVICE EXEC” on page 261](#)
- [“VMFSUFIN EXEC” on page 597](#)
- [“VMFVIEW EXEC” on page 614](#)
- [“The System-Level Service Update Facility Table \(VM SYSSUF\)” on page 693](#)
- [Appendix B, “Input/Output Files,” on page 735](#)
- [Appendix D, “Module Identifiers for VMSES/E Messages,” on page 749](#)

### **Miscellaneous updates for z/VM 7.2**

The following topics are updated:

- [“The SERVICE \\$PRODS File ” on page 135](#)
- [“PUT2PROD EXEC” on page 258](#)

## **GC24-6336-00, z/VM 7.1 (September 2018)**

---

This edition includes changes to support the general availability of z/VM 7.1.

### **VMSESE PROFILE support for multi-token variables added**

The following topic is updated:

- [“Example” on page 132](#)

### **VMSES/E MIGRATE command support withdrawn**

The VMSES/E MIGRATE command and related commands, first supplied with z/VM 5.2, are no longer provided or supported in z/VM 7.1. The upgrade installation process that was introduced with z/VM 6.3 can be used to upgrade supported z/VM levels to z/VM 7.1. The migration of customized data for components, features, or products from z/VM levels prior to those supported by the upgrade installation process for z/VM 7.1 now must be performed using locally-developed procedures.

The following VMSES/E commands are no longer supported:

- MIGCLEAN
- MIGLINK
- MIGRATE
- MIGSETUP

The following VMSES/E system software inventory file is no longer supported:

- *prodid* MIGDvrm (the migration disk table)

The following topics are updated:

- [Chapter 20, “VMSES/E Exec and Command Format Summaries,” on page 229](#)
- [“Using Tools for Service and System Generation” on page 229](#)
- [“The System-Level Software Inventory” on page 659](#)
- [“The System-Level Service Update Facility Table \(VM SYSSUF\)” on page 693](#)
- [“The System-Level Local Modification Table \(VM SYSLMOD\)” on page 699](#)
- [Appendix D, “Module Identifiers for VMSES/E Messages,” on page 749](#)
- [Appendix B, “Input/Output Files,” on page 735](#)

Various CP, TCP/IP, and VMSES/E messages are updated or deleted.

For more information, see:

- [z/VM: CP Messages and Codes](#)

- [z/VM: Other Components Messages and Codes](#)
- [z/VM: TCP/IP Messages and Codes](#)

## **VMSES/E ITNVTSTR command support withdrawn**

The ITNVTSTR EXEC is no longer provided or supported in z/VM 7.1.

The following topics are updated:

- [“Example” on page 170](#)
- [“Using Tools for Service and System Generation” on page 229](#)
- [“Example” on page 699](#)
- [Appendix D, “Module Identifiers for VMSES/E Messages,” on page 749](#)

## **Packaging update**

The Open Systems Adapter / Support Facility (OSA/SF) is no longer shipped with z/VM.

The following topics are updated:

- [“Step 1. Find the ppfname for the Product” on page 78](#)
- [“Step 1. Find the ppfname for the Copy We Want to Delete” on page 81](#)
- [“PUT2PROD EXEC” on page 258](#)
- [“SERVICE EXEC” on page 261](#)

## **Miscellaneous updates for z/VM 7.1**

The following topics are updated:

- [Chapter 9, “Installing Corrective Service,” on page 95](#)
- [Chapter 11, “Installing Local Service and Modifications,” on page 101](#)
- [“Using Tools for Service and System Generation” on page 229](#)
- [“SERVICE EXEC” on page 261](#)

# Part 1. Introduction

In this part of the book, you will learn about the z/VM Installation and Service Tool (VMSES/E). VMSES/E consists of two user interfaces, the VMFINS and VMFSIM EXECs; enhanced service execs, for example, VMFREC and VMFBLD; and the system-level and service-level Software Inventories. [Figure 1 on page 1](#) shows a high-level overview of VMSES/E.

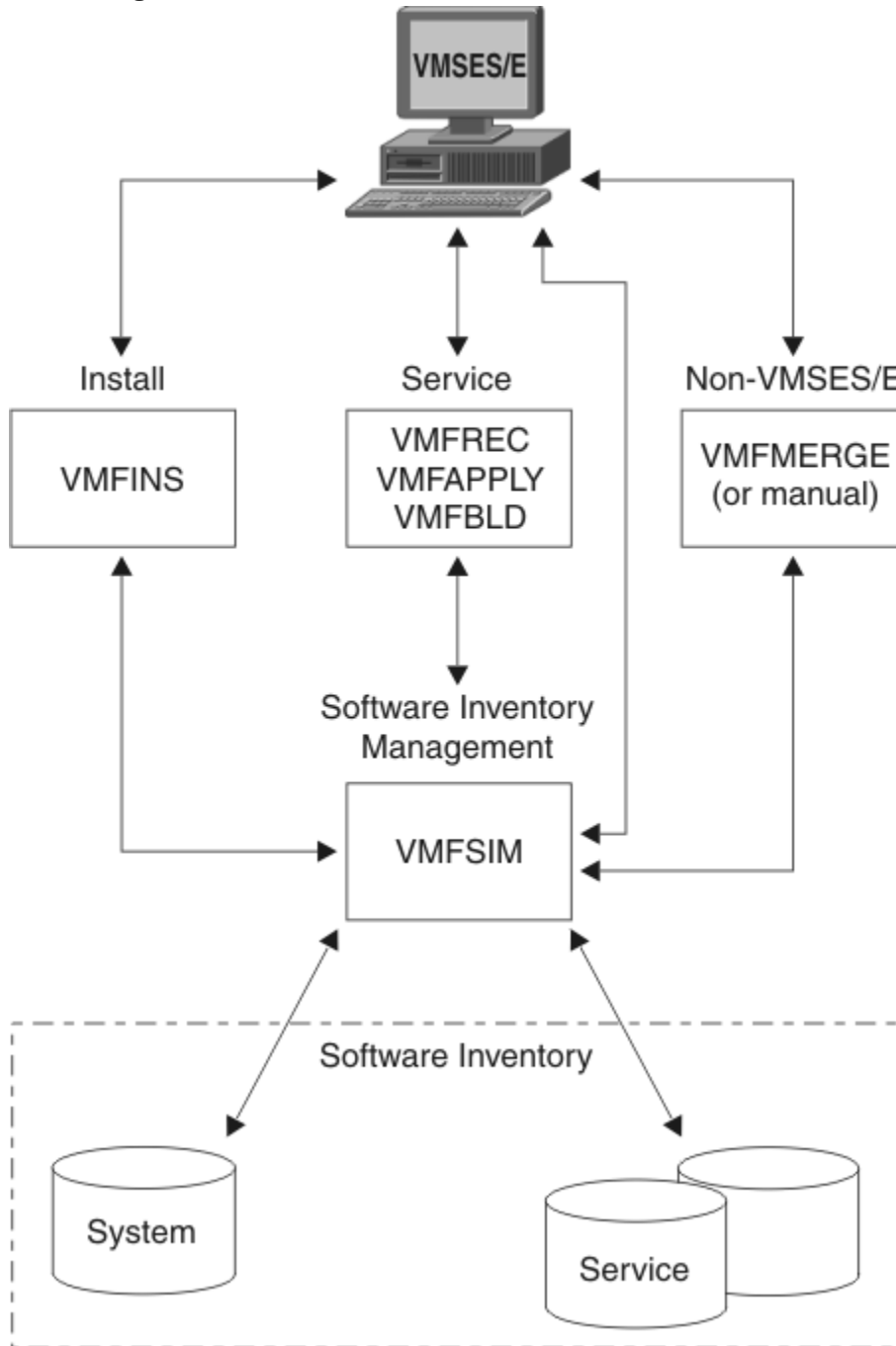


Figure 1. Introducing VMSES/E

In each part of this book, you will learn about a different aspect of VMSES/E. To help you, this figure appears at the beginning of each part of the book. We have highlighted the features and functions in the diagram that are discussed in each part.





# Chapter 1. Introducing VMSES/E

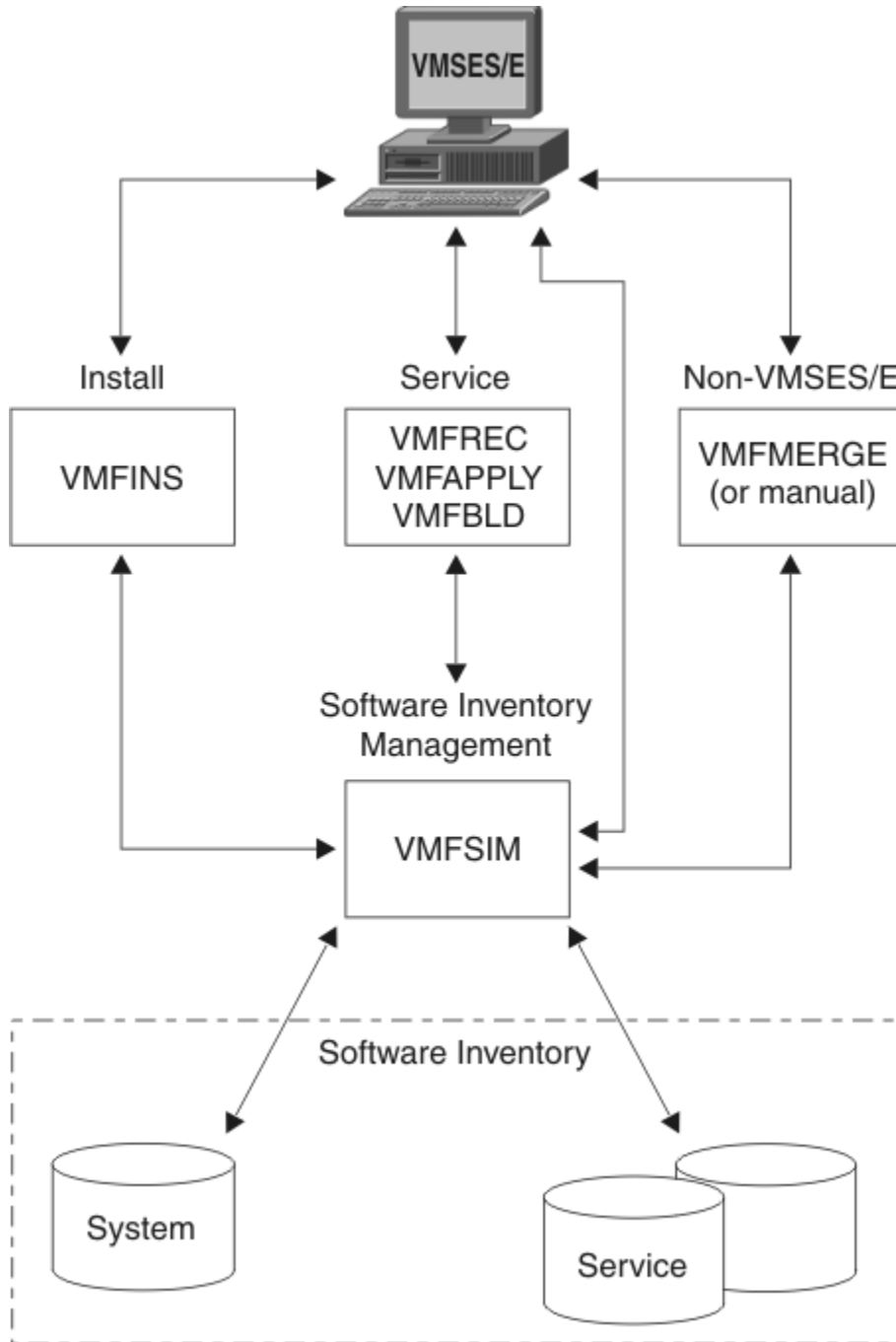


Figure 2. VMSES/E Overview

VMSES/E is a component of z/VM. VMSES/E provides:

- An exec for installing, migrating, building, and deleting products. [Part 2, “Installing, Migrating, Building, and Deleting Products,” on page 7](#) describes this aspect of VMSES/E.
- Execs for receiving service, applying service, and building the serviced usable forms, files to direct the operation of these execs and to save the status of their execution, and a database structure that isolates executable code from the control structure used to manage it. [Part 3, “Servicing Products,” on page 87](#) describes this aspect of VMSES/E.

- A Software Inventory that stores information on the status of installed products, the Program Temporary Fixes (PTFs) applied to products, the requisite relationships among products and PTFs, information about saved segments, as well as other information. [Part 4, “Planning and Managing Your Software Inventories,” on page 157](#) describes this aspect of VMSES/E.
- An exec for managing saved segments. The supporting structure for saved segments in VMSES/E is described in [Part 3, “Servicing Products,” on page 87](#) and [Part 4, “Planning and Managing Your Software Inventories,” on page 157](#). Saved segment management tasks using the VMSES/E function are described in [z/VM: CP Planning and Administration](#).

## Installing Products

---

In [Figure 2 on page 3](#), the VMFINS EXEC helps you install products. The VMFINS EXEC can also be used to:

- Migrate products while keeping previously tailored files
- Build products and update the Software Inventory tables
- Delete products you no longer need on your system

The VMFINS EXEC provides a planning option to help you check product requisites and resource requirements before you install, migrate, and delete products. The VMFINS planning option also lets you:

- Specify the installation location for a product
- Specify installation related parameters

The VMFINS EXEC can process products from installation media formatted for:

- VMSES/E
- z/VM System Delivery Offering

The command syntax for the VMFINS EXEC is consistent across functions, yet flexible, to provide for ease-of-use and personal preference in how a task is completed. For more information on the VMFINS EXEC, see [Part 2, “Installing, Migrating, Building, and Deleting Products,” on page 7](#).

## Servicing Products

---

In [Figure 2 on page 3](#), the VMSES/E EXECs, VMFREC, VMFAPPLY, and VMFBLD, help you service products. For more information on servicing products with VMSES/E, see [Part 3, “Servicing Products,” on page 87](#).

## Servicing Non-VMSES/E Products

---

Use the VMFMERGE EXEC to apply service to Systems Network Architecture products. For more information on servicing these products, see [Appendix F, “Servicing Non-VMSES/E SNA Products,” on page 757](#).

## Managing Saved Segments

---

The VMFSGMAP EXEC is a saved segment mapping and management interface. VMFSGMAP provides a full-screen segment map that shows you the saved segments defined on your system and in the Software Inventory. Using VMFSGMAP, you can change, add, and delete saved segment definitions and display the results. You can then use the VMFBLD EXEC to build or delete the saved segments.

For information about VMFSGMAP, see [“The Source Product Parameter File” on page 13](#). For information about VMFBLD, see [“VMFBLD EXEC” on page 305](#). For information about defining, building, and managing saved segments, see [z/VM: CP Planning and Administration](#).

## Managing Product Inventories

The VMFSIM EXEC is the single interface between you, the end user who is installing, migrating, building, deleting, and servicing products, and the system-level and service-level Software Inventories. The Software Inventory contains control and status information which is used when products are installed, migrated, built, deleted, and serviced. The service-level Software Inventory contains information on the service applied to each product on the system, if it is serviced with VMSES/E. The Software Inventory information resides in a series of tables on the Software Inventory minidisk or Shared File System directory.

With VMFSIM, you can easily manage the Software Inventories on your systems. The VMFSIM EXEC creates and updates the Software Inventory and provides queries so you can check the status of products and service at any time. For more information on Software Inventory management with VMSES/E, see [Part 4, “Planning and Managing Your Software Inventories,”](#) on page 157.

## Selecting the Correct Method for Installation and Service

With the many products available today, it is sometimes difficult to know which installation or service process to use for each type of product. The following table is provided to help you select the appropriate method of installation and service for each product type.

In [Table 2](#) on page 5:

- VMSES/E refers to products that are in VMSES/E install and service format.
- INSTFPP refers to products that use the VMFINS command but not all of the VMFINS capabilities. They are not in VMSES/E service format.
- SNA refers to products that use the VMFINS command but not all of the VMFINS capabilities. SNA products are not in VMSES/E install or service format.
- Other refers to products that are not in VMSES/E install or service format.

**Note:** Before you install or service products using VMSES/E, you should always see the documentation for the product.

*Table 2. Methods of Installation and Service*

Product Type	Install	Service
VMSES/E	VMFINS	VMFREC, VMFAPPLY, VMFBLD
INSTFPP	VMFINS or INSTFPP	Method defined by product
SNA	VMFINS, INSTFPP, or Method defined by product	See Appendix F, “ <a href="#">Servicing Non-VMSES/E SNA Products,</a> ” on page 757
Other	Method defined by product	Method defined by product

## Finding Out Where to Begin

The following information can help you get started with VMSES/E.

- For changes to VMSES/E, see [z/VM: Migration Guide](#) for information on what is new in this release.
- To learn how to use the VMFINS EXEC to install, migrate, build, and delete products on your z/VM system and perform related tasks, see [Part 2, “Installing, Migrating, Building, and Deleting Products,”](#) on page 7.
- If you are using VMSES/E to install a product on your z/VM system, see [Chapter 4, “Installing Products with VMFINS,”](#) on page 49 and the documentation for that product.
- If you are using VMSES/E to migrate a product on your z/VM system, see [Chapter 5, “Migrating Products with VMFINS,”](#) on page 63 and the documentation for that product.

## Introducing VMSES/E

- If you are using VMSES/E to build products you have installed on your z/VM system, see [Chapter 6, “Building Products with VMFINS,”](#) on page 77 and the documentation for that product. You may also want to see [“VMFBLD EXEC”](#) on page 305.
- If you are using VMSES/E to delete a product from your z/VM system, see [Chapter 7, “Deleting Products with VMFINS,”](#) on page 81.
- To learn about service concepts and how VMSES/E is used for service, see [Part 3, “Servicing Products,”](#) on page 87.
- To learn about the product parameter file, Software Inventory, the VMFSIM EXEC, and the VMFINFO EXEC and how you can use them to manage the software on your system, see [Part 4, “Planning and Managing Your Software Inventories,”](#) on page 157.
- If you are servicing licensed program products, such as Systems Network Architecture (SNA) products, see [Appendix F, “Servicing Non-VMSES/E SNA Products,”](#) on page 757.
- If you are defining, building, or deleting saved segments, see [z/VM: CP Planning and Administration](#).

## Part 2. Installing, Migrating, Building, and Deleting Products

In this part of the book, you will learn how VMSES/E can be used to install, migrate, build, and delete products and components on your z/VM system.

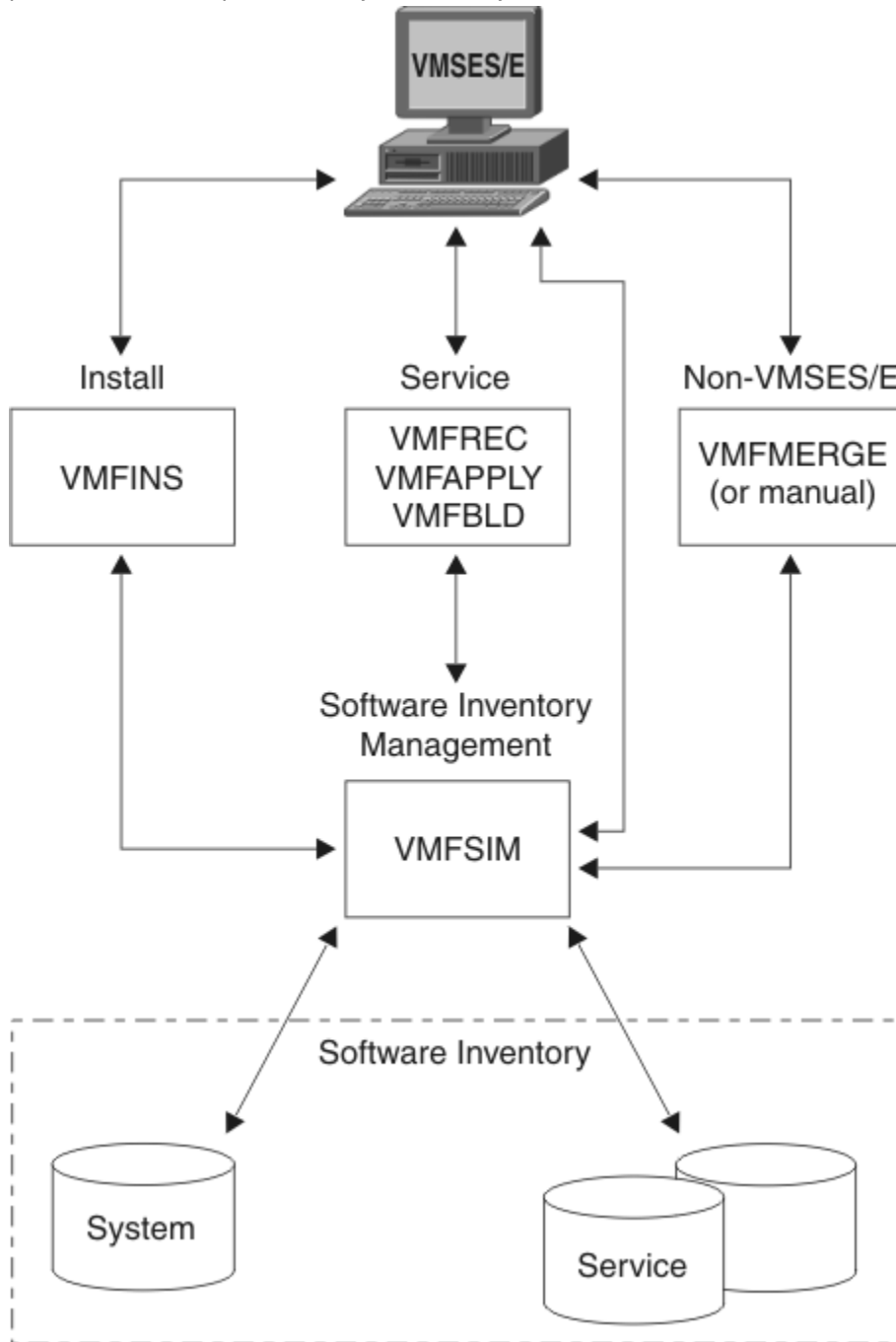


Figure 3. VMSES/E - Installing, Migrating, Building, and Deleting Products



## Chapter 2. Introducing the VMFINS EXEC

The **VMFINS EXEC** provides a single, consistent, and flexible process for installing the z/VM base product, the licensed programs and components that run on z/VM, and **preventive service** by product replacement. VMFINS combines resource allocation, requisite checking, and service level support to allow you to:

- Print the *Memo-to-Users* for each product on the installation media.
- Plan to install, migrate, and delete products.

With VMFINS, you can use the PLAN option to perform requisite checks, determine the amount of minidisk space required to install or migrate a product, and change the default product installation parameters

- Install new products.
- Upgrade previously installed products and components to a new version, new release, or a current release at a new service level by replacing the current copy or installing an additional copy.

This means you can maintain multiple levels of products and components on a system in production mode, test mode, or multiple production environments.

- Migrate products from one version or release to another.

This includes new versions, new releases, or current releases at a new service level.

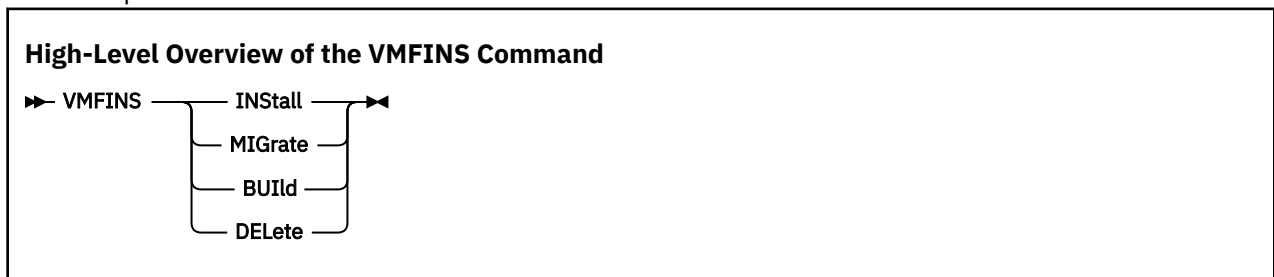
- Build products and components.
- Delete previously installed products and components that you no longer need or want.
- Create **product parameter file** overrides for products that are in VMSES/E format.

When you create a product parameter file override, you can:

- Specify the product installation location.
- Specify installation related parameters.
- Install multiple copies of a product.
- Select a default installation path.

### Overview

“High-Level Overview of the VMFINS Command” on page 9 shows a high-level overview of the VMFINS command syntax. To use the VMFINS EXEC, you must select either the INSTALL, MIGRATE, BUILD, or DELETE operand.



For help reading the syntax diagram, see “Understanding Syntax Diagrams” on page 227.

### VMFINS INSTALL

When you select the INSTALL operand, you can add new copies of a product to your system. You can also add a new copy of a product and replace the existing copy of that product. You cannot, however, save the existing files that have been tailored for the copy that is already on your system.

### VMFINS MIGRATE

When you select the MIGRATE operand, you can add a new copy of a product to your system and use the files that have been tailored for an existing copy of the product. You can also add a new copy of a product and replace the existing copy of that product and keep the files that have been tailored for the copy you are replacing.

### VMFINS BUILD

When you select the BUILD operand, you can build a product you have added to your system by either a VMFINS INSTALL or MIGRATE. The BUILD operand updates the Software Inventory tables to show the product has been built.

### VMFINS DELETE

When you select the DELETE operand, you can remove a product from your system if it was installed using VMSES/E. The DELETE operand updates the Software Inventory tables to show the product has been removed from the system.

## Who Can Use VMFINS?

---

The VMFINS EXEC can be used by any user who wants to install, migrate, build, or delete licensed products on a z/VM system and has:

- The installation documentation for the products being installed or migrated.
- A running z/VM 6.4 (or later) system. For information on installing a z/VM operating system, see [z/VM: Installation Guide](#).
- Installation media in acceptable format. See [“What Tape Formats Does VMFINS Support?”](#) on page 10 to find the tape formats you can use.
- Access to a tape drive.
- Read access to the VMSES/E build disk (MAINTvrm 5E5 by default).
- Read-write access to the Software Inventory minidisk or Shared File System directory (MAINTvrm 51D by default).
- Access to the product resources for the product you are installing.

If you are migrating a product, you must:

- Have administrative authority for the VMFINS default Shared File System directory (*filepoolid:userid.VMFINS*) for the user ID running the VMFINS MIGRATE command. VMPSFS: is the default file pool ID for VMFINS. *userid* is the user ID of the person using VMFINS. You can use another SFS file pool during a VMFINS MIGRATE, as long as you have administrative authority for that file pool.
- Make sure the file pool is available in interactive mode.
- Create the *filepoolid:userid.VMFINS* directory or enroll a user in the file pool. This directory is used to store migration information.

## What Tape Formats Does VMFINS Support?

---

VMFINS can install and migrate products from *product tapes* in the following formats:

- VMSES/E

VMSES/E tapes contain only products that are formatted for VMSES/E.

- INSTFPP

INSTFPP tapes can contain products that are formatted for VMSES/E and INSTFPP. INSTFPP tapes can be received as part of the z/VM System Delivery Offering (SDO). They can also be received as stand alone tapes.



If you have questions on the format of your product tape, contact your IBM representative.

## What Type of Processing Does VMFINS Provide?

Table 3 on page 11 shows the amount of processing performed for each product format. The tasks you may need to complete manually are listed in the **You Must Manually** column. In this table, *prodid* is the 7- or 8-alphanumeric identifier assigned to the product by IBM.

Table 3. Processing by Product Format

Product Format	Processing Performed	You Must Manually	VMSES/E Files for Product Control and Management
VMSES/E	Plans, installs, migrates, builds, and deletes; updates system-level and service-level Software Inventories  Compares service levels if product serviced by VMSES/E  Manages multiple copies of products	Tailor files and allocate or delete product resources	<i>prodid</i> \$PPF <i>prodid</i> PRODPART
INSTFPP	Plan: None. See the product documentation for minidisk requirements.  Install: Calls INSTFPP to install the product. See the product documentation for resource requirements.  Build: Updates system-level Software Inventory tables only; does not perform build  Migrate: None  Delete: None	Plan, generate and allocate resources, migrate, and delete  Compare service levels  Tailor files  Manage multiple copies of products  Service product using method defined by the product (non-VMSES/E service)	There are no \$PPF or PRODPART files for INSTFPP products.

**Note:** Previously, some products were packaged in Parameter Driven Installation (PDI) format, which was a subset of the VMSES/E product format. PDI-formatted products were found on VMSES/E- or SDO-formatted installation media. Products in this format are still supported by VMSES/E, but they are not used widely.

## What Is the Software Inventory?

Managing a wide variety of products and levels of products on a system can be quite a job. VMSES/E can help you manage your system software.

The data required by VMFINS to process a product is distributed with each product on the product installation media. When products are installed or migrated, VMSES/E creates a series of tables to identify the products, the levels of the products, and the status of the products on your system. These tables, known as the Software Inventory tables, are stored on the Software Inventory minidisks or Shared File System directories. The following information can be found in the Software Inventory tables:

## Introducing VMFINS

- The requisite relationships between the products and components
- How product identifiers, the 7 or 8 alphanumeric identifier assigned to a product by IBM (also known as the *prodid* by VMSES/E execs), map to the names of the product parameter files and descriptions created and used during installation and migration
- The status of each product or component on the system.

In Figure 4 on page 12, there are two levels to the Software Inventory. There is a system-level Software Inventory and a service-level Software Inventory.

The Software Inventory minidisks and Shared File System directories are managed by the VMFSIM EXEC. The VMFSIM EXEC is the single interface between the users and the Software Inventory minidisks and Shared File System directories. To see how you can use the VMFSIM EXEC to manage your Software Inventory, see Chapter 16, “Introduction to the VMFSIM EXEC,” on page 177.

## Where Does It Reside?

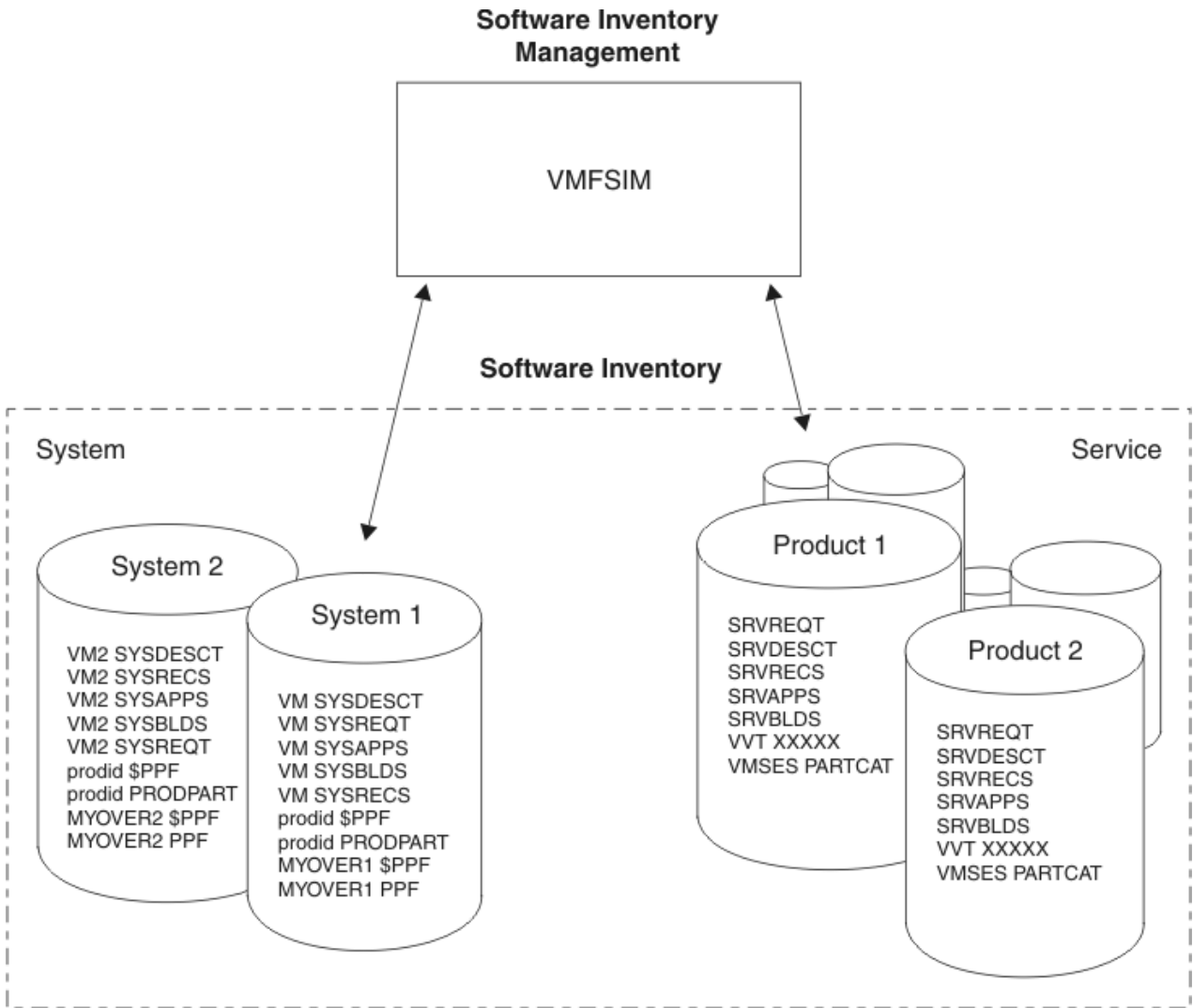


Figure 4. The System-Level and Service-Level Software Inventories

When products are installed or migrated, the system-level Software Inventory tables are created and stored on the system-level Software Inventory minidisk, the MAINT $\nu$ rm 51D minidisk by default. The system-level Software Inventory tables are updated when install, migrate, build, and delete processing are successfully completed. As you can see in Figure 4 on page 12, there can be a number of system-level Software Inventory minidisks. There is a system-level Software Inventory minidisk for each system the

user maintains. See [Chapter 15, “Introduction to the Software Inventory,” on page 163](#) for additional information on the system-level Software Inventory tables.

The service-level Software Inventory tables reside on the DELTA and APPLY minidisks for each product (shown in the lower right of [Figure 4 on page 12](#)). There can be multiple sets of these minidisks, and they can contain either the tables for other products or the tables for additional copies of a product. See [“Overview of the Service-Level Software Inventory” on page 171](#) for more information on the service-level Software Inventory tables. For more information on the DELTA and APPLY minidisks, see [“The VMSES/E Database” on page 105](#).

## Files Shipped on the Product Installation Media

To process a VMSES/E-formatted product with VMFINS, certain files are shipped on the installation media with the product. The following files are the most important.

### The Product Parts File

The product parts file contains important information that is used during VMFINS processing. This information includes requisites, tailorable files, and product resources such as:

- Product user IDs, including the directory statements to define a user ID such as IPL options, link statements, and spool statements
- Minidisk size requirements
- Shared File System requirements
- Saved segment definitions

The file name and file type of the product parts file is *prodid* PRODPART. *prodid* is the 7- or 8-alphanumeric identifier assigned to the product by IBM. For more information on the contents of the file, see [“The Product Parts \(PRODPART\) File” on page 660](#).

### The Source Product Parameter File

Products in VMSES/E format are shipped with a source product parameter file, the *prodid* \$PPF file. *prodid* is the 7 or 8 alphanumeric identifier assigned to the product by IBM. The *prodid* \$PPF file is used during VMFINS processing, and it defines the:

- Information necessary to build the usable form of the product
- Variables used in the \$PPF and PRODPART files
- Symbolic names for the target minidisks
- Target minidisk or SFS directory for each tape file and the **part handler** that transfers the files from the tape to the target
- Product processing exits to execute during the loading of a product

Product processing exits perform installation related product customization or enable products and components. An example of a product processing exit task is copying and renaming sample files.

You may also receive a usable form product parameter file (*ppfname* PPF) that has been compiled for you by IBM. For more information on product parameter files, see [Chapter 14, “Introduction to the Product Parameter File,” on page 159](#). For a complete description of the contents of the file, see [Chapter 21, “Product Parameter File Syntax,” on page 621](#).

## Files Created and Updated during Installation and Migration

The following files are created and updated during installation and migration to record completed processing and additional information for problem determination.

Within the file identifiers below, *prodid* is the 7 or 8 alphanumeric identifier assigned to the product by IBM. *ppfname* is the unique name assigned to the product parameter file override for a specific copy of a product.

### Product-Level Files

When you install or migrate a VMSES/E-formatted product with the VMFINS EXEC, you can create a product parameter file override for each copy of the product that is installed or migrated. The product parameter file override may contain either the default product installation parameters that are provided with the product by IBM or new product installation parameters you have entered.

The following files are created during VMFINS install and migrate processing:

#### ***ppfname* \$PPF**

The *ppfname* \$PPF file contains the user's overrides to the product supplied source product parameter file (*prodid* \$PPF).

#### ***ppfname* PPF**

The *ppfname* PPF file is the usable form of the product parameter file with all overrides applied.

**Note:** The *ppfname* PPF and \$PPF files represent only one copy of a product, and they may or may not reflect the product installation defaults that were originally shipped with the product.

Once install and migrate processing is complete, the *ppfname* PPF and *ppfname* \$PPF files are moved to the Software Inventory minidisks or SFS directories. If processing ends unsuccessfully, you may find *ppfname* PPF and *ppfname* \$PPF files on your A-disk.

For more information on the product parameter file, see [Chapter 14, “Introduction to the Product Parameter File,”](#) on page 159.

### Files for Your Information and Use

The following files are created by VMFINS during INSTALL, MIGRATE, BUILD, and DELETE processing.

#### **\$VMFINS \$MSGLOG**

The \$VMFINS \$MSGLOG is created during VMFINS processing and stored on your A-disk. The \$VMFINS \$MSGLOG contains a record of the VMFINS commands that have been entered and the results of the VMFINS processing. The information from the latest session appears at the top of the file.

You can use the VMFVIEW command to view messages in the \$VMFINS \$MSGLOG file by entering:

```
vmfview install
```

For more information on the VMFVIEW command, see [“VMFVIEW EXEC”](#) on page 614.

An example of a \$VMFINS \$MSGLOG file can be found in [“Scenario 1: Installing a Product with the PPF Operand”](#) on page 54.

#### **VMFINS CONSOLE**

The VMFINS CONSOLE file contains the console log listing that is created during VMFINS processing. If you have not previously spooled your console, it is automatically spooled to your reader when VMFINS processing ends. If you have already spooled the console, VMFINS does not close the console; and the VMFINS CONSOLE file is not created. The VMFINS CONSOLE file contains a record of the entries you made, the system prompts you received, and the messages that were issued. The CONSOLE file can be helpful if you need to determine where an error may have occurred. An example of a VMFINS CONSOLE file can be found in [“Scenario 1: Installing a Product with the PPF Operand”](#) on page 54.

The following files may be created when a VMFINS INSTALL, MIGRATE, or DELETE command is entered with the PLAN option:

#### ***prodid* PLANINFO**

The *prodid* PLANINFO file contains important information on product dependencies and requisites, as well as information on system resource requirements. The *prodid* PLANINFO file is created when you use the PLAN option, and it is stored on your A-disk. For more information on using the PLAN option and the *prodid* PLANINFO file, see [“Using the PLAN Option”](#) on page 23. See [“5654A22C PLANINFO”](#)

File” on page 25, “5654A22C PLANINFO File” on page 28, and “5654A22C PLANINFO File” on page 31 for *prodid* PLANINFO file examples.

### ***ppfname* ERASE**

The *ppfname* ERASE file is created when you enter a:

- VMFINS INSTALL command with the REPLACE and PLAN options
- VMFINS MIGRATE command with the REPLACE and PLAN options
- VMFINS DELETE command with the PLAN option

The *ppfname* ERASE file contains a list of the files that will be erased when the actual processing occurs, and it is stored on your A-disk. An example of a *ppfname* ERASE file can be found in [“Scenario 1: Deleting a Product with the PPF Operand”](#) on page 81.

### ***prodid* \$APPLIST**

The *prodid* \$APPLIST file contains a list of program temporary fixes (PTFs) that need to be reapplied. This file is only created during a VMFINS MIGRATE when reach-ahead service has been applied to the copy of the product already on your system. Reach-ahead service is **corrective service** that has been applied to a product but is not on the preventive service vehicle.

The files just mentioned are not the only files used during VMFINS EXEC processing. For more information on the files used and created by VMFINS, see [“VMFINS EXEC”](#) on page 404. For a list of the files used and created by other VMSES/E EXECs, see [Part 5, “Reference,”](#) on page 225.



---

## Chapter 3. Using the VMFINS EXEC

In this chapter you will see how you can use VMFINS commands to install, migrate, build, and delete products. The VMFINS command syntax is flexible, and it can be used in many combinations. The commands in this chapter are examples only, and they may not reflect the way you would organize your tasks.

Although VMFINS has defined many command defaults, the commands in the following examples have been entered in their longest form to show you exactly which options to use. The complete command syntax for VMFINS can be found in [“VMFINS EXEC” on page 404](#). Information on changing the VMFINS command option defaults can be found in [“Changing the VMFINS Command Defaults” on page 48](#).

---

### Determining Which Operand to Use: INSTALL, MIGRATE, BUILD, or DELETE?

---

The first thing you need to do is decide whether you are installing, migrating, building, or deleting a product. The following information can help you make that decision.

#### Installing a Product

You choose INSTALL when you are:

- Adding the first copy of a product to the system or adding a new copy of a product to the system and you do not want to copy **tailorings** from a previously installed copy of the product. Tailorings are the changes you add to tailorable files to customize them for your own environment or **reach-ahead service**. Reach-ahead service is corrective service that has been applied to a product that is not on the preventive service vehicle. If you want to preserve tailorings, you should use MIGRATE instead of INSTALL.
- Replacing a currently installed version of a product and you do not want to save tailorings from a previously installed copy of the product.
- Installing a new version or release of a product and keeping an existing version or release of the same product.

For example, you might want to install a new version in test mode and keep the existing copy in production mode.

#### Migrating a Product

You choose MIGRATE when you are:

- Adding a new copy of a product to the system and you want to copy tailorings from a previously installed copy of the product (that was installed using VMSES/E).
- Replacing a currently installed version of a product (that was installed using VMSES/E) and you want to save tailorings from the copy of the product you are replacing.

#### Building a Product

You choose BUILD when you are:

- Building a product on your system and you want to update the Software Inventory tables to show the current status.

#### Deleting a Product

You choose DELETE when you are:

- Removing a copy of a product (that was installed using VMSES/E) from your system.

## Installing a Recommended Service Upgrade (RSU)

You choose INSTALL when you are:

- When you install an RSU with the VMFINS INSTALL command, tailorings are saved.

## Using the INFO Operand

You can use the INFO operand to obtain information for a number of decisions. For example, you can use the INFO operand with the VMFINS INSTALL command to get a list of the products on the product installation media and decide which ones to install. You can also use the INFO operand with the VMFINS MIGRATE command to get a list of products that are on the installation media and on your system and are available to be migrated.

The list created by the INFO operand is stored on your A-disk in a file called VMFINS PRODLIST by default. The VMFINS PRODLIST file is rewritten each time you use the INFO operand.

You can pass the VMFINS PRODLIST file to a VMFINS INSTALL or VMFINS MIGRATE command by using the LIST operand. For more information on the LIST operand, see [“Using the LIST Operand” on page 20](#).

## The VMFINS PRODLIST File

The VMFINS PRODLIST file in [Figure 5 on page 18](#) was created when we entered this command:

```
vmfins install info (add
```

This file shows us which products are on our product installation media.

```
VMFINS  PRODLIST A1  V 84  Trunc=84 Size=7 Line=0 Col=1 Alt=0
====>
* * * Top of File * * *
PPF 5654A22C CCXX      PRODID 5654A22C%CCXX IBM XL C/C++ for z/VM Compiler
PPF 5654A22C CCXSFS   PRODID 5654A22C%CCXSFS IBM XL C/C++ for z/VM Compiler in SFS
PPF 5654A22C CCXXK   PRODID 5654A22C%CCXXK IBM XL C/C++ for z/VM Compiler
PPF 5654A22C CCXXKSFS PRODID 5654A22C%CCXXKSFS IBM XL C/C++ for z/VM Compiler in SFS
PROD 5668812 NONE     GDDM-PGF
PPF 5684100E PVMINS   PRODID 5684100E%PVMINS Installing PVM 2.1.1
PPF 5684100E PVMUCENG PRODID 5684100E%PVMUCENG Servicing PVM 2.1.1 Upper Case English help
PPF 5684100E PVMISFS  PRODID 5684100E%PVMISFS Installing PVM 2.1.1 Optional source
PPF 5684100E PVMUSFS  PRODID 5684100E%PVMUSFS Servicing PVM 2.1.1 Upper Case English help
PPF 5684100E PVMSSFS  PRODID 5684100E%PVMSSFS Installing PVM 2.1.1 Optional source using SFS
using SFS directories
directories
PPF 5684100E PVMSSFS  PRODID 5684100E%PVMSSFS Installing PVM 2.1.1 Optional source using SFS
directories
.
.
.
* * * End of File * * *
```

Figure 5. A VMFINS PRODLIST File

## Product Parameter File Entries

When you see the PPF keyword, it means a usable form product parameter file has been shipped with the product by IBM. The usable form product parameter file is a product parameter file that has been compiled for you by IBM, and it contains the recommended installation parameters for the product. For example, here is one line from the file.

```
PPF 1 5654A22C 2 CCXX 3 PRODID 4 5654A22C%CCXX 5 IBM XL C/C++ for z/VM Compiler 6
```

In this example:



- The first field contains the PPF keyword (1).
- The second field of information is the 7- or 8-character alphanumeric identifier (the *ppfname*) assigned to the usable form product parameter file for the product (2).
- The third field contains the name assigned to the component by IBM (3). In this example, the component name is CCXX. In the remainder of this book, you may see the component name referred to as the *compname*.
- The fourth field contains the keyword PRODIG (4) to indicate the next field contains the product identifier.
- The fifth field contains the product identifier (*prodid*) followed by a percent sign (a delimiter) and the component name (5).
- The last field contains a description for the product (6).

PPF level entries can be entered in the PRODLIST file by VMFINS when you enter a VMFINS INSTALL with the INFO operand. They can also be automatically entered in the VMFINS PRODLIST file when you enter a VMFINS INSTALL or VMFINS MIGRATE command with the LIST operand and the PLAN option. You can also enter them manually.

Let's take a look at another example. In [Figure 5 on page 18](#), you can see there are six entries for *prodid* 5684100E. Look to the right of each entry to see the component name for each. On this particular installation media, product 5684100E has six components - PVMINS, PVMUCENG, PVMSRC, PVMISFS, PVMUSFS, and PVMSSFS. The description to the right of the component name tells you that PVMINS is for the installation of PVM 2.1.1, whereas PVMISFS is for the installation of PVM 2.1.1 in the Shared File System. The descriptions for the other components likewise convey the content that is associated with each one.

If a product has multiple components on the installation media, you can specify that only certain components be processed. If you do not specify a component, VMFINS will prompt you to select one.

## Product-Level Entries

In [Figure 5 on page 18](#), you can also see lines of information that begin with the keyword PROD. The PROD keyword tells you the information is base-level product information. This information does not change as long as you are working with a new copy of the product each time.

Let's look at one line from the file shown in [Figure 5 on page 18](#) to see what each field represents.

```
PROD 1 5668812 2 NONE 3 GDDM-PGF 4
```

The first field of information is the PROD keyword (1).

The second field of information is the 7 or 8 alphanumeric product identifier assigned to the product by IBM, the *prodid* (2).

The third field of information is the component name (3). If you see the word, NONE, the product is not in VMSES/E format; and it is installed by VMFINS using the INSTFPP EXEC.

The description of the product (4) appears last.

Now, let's see how you can use the INFO operand with the different VMFINS commands.

## Which Products Can You Install?

To create a list of the products on the installation media, you can enter the following VMFINS INSTALL command:

```
vmfins install info (add
```

You see some messages; and, when processing is complete, VMFINS creates a file called VMFINS PRODLIST and stores it on your A-disk.

You can also enter this command:

```
vmfins install info (replace
```

to find out which products on the system can be replaced by products on the installation media. The format of the VMFINS PRODLIST file does not change for the different options. [Figure 5 on page 18](#) shows you what a typical VMFINS PRODLIST file looks like.

### Which Products Can You Migrate?

Even though we used the INFO operand with the INSTALL operand in the previous example, the INFO operand can also be used with the MIGRATE operand. Because a product must already be installed on your system to be able to migrate it, the INFO operand can tell you which products are installed and available for processing.

When you use the INFO operand with the MIGRATE operand, you receive a list of products on the installation media that can be used to migrate products on the system.

### Using the PROD Operand

---

You can use the PROD operand to install, migrate, build, and delete a product. You specify which product to process by entering the PROD operand and the *prodid*, the 7- or 8-character alphanumeric identifier assigned to the product by IBM. For example, 5684100E is the *prodid* for PVM 2.1.1.

You can find the *prodid* for products on the product installation media in the VMFINS PRODLIST file, which is created when you enter a VMFINS INSTALL or VMFINS MIGRATE command with the INFO operand.

As an example, to install a product using the PROD operand, you can enter:

```
vmfins install prod prodid (options...
```

The format is the same for VMFINS BUILD, MIGRATE, and DELETE.

### Using the PPF Operand

---

You can use the PPF operand to install, migrate, build, and delete a specific copy of a single product. Before you can use the PPF operand, however, you must have a usable form product parameter file for the copy of the product you are installing, migrating, building, or deleting. The usable form product parameter file is a product parameter file after all overrides have been applied and variables have been resolved. The usable form product parameter file can be:

- Shipped with the product by IBM
- Created manually using the VMFPPF EXEC

The usable form product parameter file contains the product installation parameters for one copy of the product. A usable form product parameter file also has a unique name, which you may assign, to identify the specific product installation parameters for this copy of the product. In the VMFINS command syntax, this unique name is referred to as the *ppfname*.

As an example, to install a product using the PPF operand, you can enter:

```
vmfins install ppf ppfname (options
```

The format is the same for build, migrate, and delete. For examples of how to use the PPF operand, see [“Scenario 1: Installing a Product with the PPF Operand” on page 54](#), [“Scenario 1: Building a Product with the PPF Operand” on page 77](#), and [“Scenario 1: Deleting a Product with the PPF Operand” on page 81](#).

### Using the LIST Operand

---

You can use the LIST operand with the INSTALL or MIGRATE operand to process a product or group of products. If you are processing a number of products, it is easier and faster to use the LIST operand. With

the LIST operand, you only have to enter a command once; you do not have to enter the command for each product.

The LIST operand uses the VMFINS PRODLIST file as input. You can use other files as input by entering the file name (*fn*), file type (*ft*), and file mode (*fm*) after the LIST operand. These files, however, must be in the proper format and contain the required information. See [“The VMFINS PRODLIST File” on page 18](#) for more information on the correct format for and contents of the VMFINS PRODLIST file.

The z/VM SDO stacked product tapes and related products are created so requisites are installed prior to the products requiring them. The VMFINS PRODLIST file lists the products in the order that they appear on the product tape. If you edit the VMFINS PRODLIST file and remove products, you may be removing a requisite for another product. If a requisite is missing, processing is interrupted; and you are asked if you want to continue the installation even though all requisites are not installed. For an example, see [“Missing Requisites” on page 24](#).

## Using the LIST Operand with the PLAN Option

When you enter a VMFINS INSTALL or MIGRATE command with the LIST operand and the PLAN option, VMFINS automatically updates the VMFINS PRODLIST file to include the new product parameter file override names that are assigned during PLAN processing. For example, in [Figure 6 on page 21](#), a product parameter file override was created for the IBM XL C/C++ for z/VM Compiler product, and it was given the *ppfname* MYCCXX. VMFINS comments out the existing PPF-level entry and enters the new product parameter file override information on the line below.

```
VMFINS  PRODLIST A1  V 82  Trunc=82 Size=1 Line=0 Col=1 Alt=1
====>
|...+...1...+...2...+...3...+...4...+...5...+...6...+...7..
* * * Top of File * * *
*PPF 5654A22C CCXX      PROPID 5654A22C%CCXX IBM XL C/C++ for z/VM Compiler
PPF  MYCCXX  CCXX      PROPID 5654A22C%CCXX IBM XL C/C++ for z/VM Compiler (Copy1)
```

Figure 6. The VMFINS PRODLIST File after Updates by the LIST Operand and PLAN Option

## Printing the Memo-to-Users

Before you install or migrate products, you should read the *Memo-to-Users* for each product. The *Memo-to-Users* contains important information that can help you as you prepare to install or migrate a product. All *Memo-to-Users* files are stored on the Software Inventory disk when you enter a VMFINS INSTALL or MIGRATE command.

You can print the *Memo-to-Users* for a group of products or a single product. To print the *Memo-to-Users*, you can use any of the following operands with the MEMO option:

- INFO
- LIST
- PROD
- PPF

When you enter the MEMO option, you are asked to select the *Memo-to-Users* you want to print; and they are sent to the system-designated printer.

## Using the INFO Operand and the MEMO Option

You can print the *Memo-to-Users* and create a list of the products that are on the installation media at the same time. You simply enter:

```
vmfins install info (memo
```

When you enter this command, the *Memo-to-Users* for each product on the installation media are listed; and you can select the ones you would like to print. For example, in [Figure 7 on page 22](#), there are 3 *Memo-to-Users* on the product installation media.

```

VMFUTL2767I Reading VMFINS DEFAULTS B for additional options
VMFINS1909I VMFINS PRODLIST created on your A-disk 1
VMFINS2602R The following memos are available to be printed 2
           Enter the numbers of your choices separated by blanks
(0) Do not print any memos
(1) I5664281 MEMO      D1 Description: AMERICAN ENGLISH
(2) I5686037 MEMO      D1 Description: AMERICAN ENGLISH
(3) 5684100E MEMO      D1 Description: AMERICAN ENGLISH
1 2
PRT FILE 0028 TO MAINTvrm COPY 001 NOHOLD 3
PRT FILE 0029 TO MAINTvrm COPY 001 NOHOLD 4

```

Figure 7. Selecting the Memo-to-Users for Printing

When you look at [Figure 7 on page 22](#), you can see:

- The VMFINS PRODLIST file has been created and stored on your A-disk **(1)**. The VMFINS PRODLIST file contains a list of the products on the installation media.
- The *Memo-to-Users* that are available for printing **(2)**.

For example, in this line:

```
(3) 5684100E MEMO      D1 Description: AMERICAN ENGLISH
```

5684100E is the *prodid* for PVM 2.1.1.. The American English version of the *Memo-to-Users* for PVM 2.1.1. is stored in a file, 5684100E MEMO, on the D-disk.

- You select the *Memo-to-Users* to print by entering the corresponding number. As you can see in the example, you can select as many as you like **(3)**.
- The files containing the *Memo-to-Users* are sent to the system-designated printer **(4)**.

## Using the LIST Operand and the MEMO Option

You can use the LIST operand to print the *Memo-to-Users* for a single product or many products. Before you can use the LIST operand, however, you need to create the file that will be used as input to the LIST operand.

Use the INFO operand to create a list of the products that are on the product installation media. You can name the output file anything you like, or you can use the default name, VMFINS PRODLIST. To create the list of the products on the installation media and use the default file identifier, enter this command:

```
vmfins install info
```

Next, edit the VMFINS PRODLIST file and comment out the products for which you do not want to print the *Memo-to-Users*. (You comment out a line by putting an asterisk (\*) in column 1.) Save the file.

To print the *Memo-to-Users*, enter this command:

```
vmfins install list (memo plan
```

It's important to remember to include the PLAN option when you use the LIST operand to print the *Memo-to-Users*. **If you do not include the PLAN option, VMFINS automatically installs the products listed in the VMFINS PRODLIST file.**

**Note:** When you run the PLAN option, you can also create a product parameter file override for this copy of the product.

The MEMO option can also be used with the VMFINS MIGRATE command.

## Using the PROD Operand and the MEMO Option

To print the *Memo-to-Users* for only a certain product or component, enter:

```
vmfins install prod prodid (memo plan
```

You can find the *prodid* (the 7 or 8 alphanumeric identifier assigned to the product by IBM) in the output file created by the VMFINS INSTALL INFO command. For an example of this file, see [“The VMFINS PRODLIST File”](#) on page 18.

If there is more than one *Memo-to-Users* for the product, a list of available *Memo-to-Users* is displayed, and you can select the one to print.

The MEMO option can also be used with the VMFINS MIGRATE command.

## Using the PPF Operand and the MEMO Option

When you use the PPF operand, you must enter the name of a usable form product parameter file (*ppfname*). The *ppfname* identifies a specific copy of a product. The usable form product parameter file is either shipped with the product by IBM or created manually using the VMFPPF EXEC. For more information, see [“VMFPPF EXEC”](#) on page 458.

When you have the name of the usable form product parameter file, you can print the *Memo-to-Users* using the PPF operand and the *ppfname* by entering:

```
vmfins install ppf ppfname (memo plan
```

The MEMO option can also be used with the VMFINS MIGRATE command.

## Using the PLAN Option

You can use the PLAN option before you install, migrate, or delete products to:

- Determine if you have the necessary requisites to install or migrate a product.
- Determine how much minidisk or Shared File System directory space you need to install or migrate a product.
- Determine reach-ahead corrective service for the product you want to migrate to a new service level within the same release.
- Determine if any products depend on the product you want to delete.
- Determine the resources currently being used by the product you want to delete.
- Determine how deleting a product or component will affect other products installed on the system and the system resources.
- Decide if you want to install, migrate, or delete the product based on the output provided.

This information is stored in a file, called *prodid* PLANINFO, on your A-disk. The *prodid* is the 7- or 8-character alphanumeric identifier assigned to each product by IBM. For example, 5684100E is the *prodid* for PVM 2.1.1.

Each time you run the PLAN option for a specific product, the current information is placed at the top of the existing *prodid* PLANINFO file. For example, if you run the command:

```
vmfins install ppf 5684100E pvmins (plan nomemo
```

three times and you name the default override something different each time, you will find the planning information in the same file, 5684100E PLANINFO. Each entry to the PLANINFO file includes the name of the *ppfname* \$PPF file that was used to create it. **The PLAN option does not generate, allocate, or commit resources.**

## Missing Requisites

The VMFINS command performs requisite checking during PLAN processing, as well as during INSTALL and MIGRATE processing.

### During PLAN Processing

If the product is in VMSES/E format, the PLAN option checks the requisites for the products you are installing or migrating. If requisites are missing, a list of the missing products is displayed.

### During Installation and Migration Processing

If you are installing or migrating a product and requisites are missing, a list of missing products is displayed; and you are asked if you want to continue.

As you can see by the following sample messages, which are displayed on the screen during VMFINS processing, the requisites for this product are not satisfied.

```

VMFUTL2767I Reading VMFINS DEFAULTS B for additional options
VMFINS2767I Reading VMFINS PRODLIST A for list of products to process
VMFINS2760I VMFINS processing started
:
VMFREQ2806W The following requisites for product :PPF ABC0001 ABCTEXT :PRODID
ABC0001%ABCTEXT are not satisfied:

VMFREQ2806W Type          Product          Component          PTF
-----
VMFREQ2806W Requisite    ABC0001         ABC
:
VMFINS2604W Product :PPF ABC0001 ABCTEXT :PRODID ABC0001%ABCTEXT cannot be
processed because its requisites are not satisfied

VMFINS2605R How would you like to proceed? Enter the number of your choice:
(0) Bypass this product
(1) Install this product without its requisites
(2) Exit

```

To continue, you enter 1. VMFINS loads the product to the system even though requisites are missing. VMFINS does not build any portion of the product that is not already built when it is received on the installation media.

To complete the installation or migration, you should install the missing requisite and enter the VMFINS BUILD command to build the product on the system. See [Chapter 6, “Building Products with VMFINS,”](#) on page 77 to see how to use the VMFINS BUILD command.

## Determining Whether a Product Can Be Installed

You can use the PLAN option with the VMFINS INSTALL command to obtain information to help you decide if you can install a product on your system.

When you use the PLAN option with the VMFINS INSTALL ... (ADD command, VMFINS:

- Checks the product requisites
- Creates a PPF override file for products in VMSES/E format, if you choose to create one
- Determines the requirements for the product resources, for example, the user IDs, the SFS directory and minidisk addresses, and the amount of space required on the minidisks and SFS directories

When you use the PLAN option with the VMFINS INSTALL ... (REPLACE command, VMFINS:

- Checks the product requisites
- Determines which copy of the product you wish to replace
- Creates a PPF override file for products in VMSES/E and PDI format, if you choose to create one
- Determines the resources being used for the copy of the product you are replacing

- Determines the resource requirements for the product you are installing, for example, the user IDs, the SFS directory and minidisk addresses, and the amount of space required on the minidisks and SFS directories
- Determines which files to delete for the product being replaced

The PLAN option also creates a *prodid* PLANINFO file for each product specified and stores the file on your A-disk.

## What Information Does the PLANINFO File Provide?

The following example shows you the type of information you can find in the *prodid* PLANINFO file. In this example, we entered:

```
vmfins install prod 5654a22c ccxx (add plan nomemo
```

We created a PPF override file for the CCXX component of product 5654A22C and we named the PPF override file MYCCXX.

### **5654A22C PLANINFO File**

[Figure 8 on page 26](#) shows the PLANINFO file created for *prodid* 5654A22C during the PLAN processing.

```

*****
****      VMFINS  INSTALL                USERID: 5654A22C      ****
*****
****      Date: 2022-06-28                Time: 16:25:07        ****
*****
VMFINS2195I VMFINS INSTALL PPF 5654A22C CCXX ( SYSTEM VM SIDISK 51D SIMODE
          D PLAN NORESOURCE LINK DFNAME USER DFTYPE DIRECT DFMODE *
          NOMEMO ADD ENV 5654A22C SETUP
*****
*              Requisite Planning Information                *
*****
*              PPF: MYCCXX CCXX      PROID: 5654A22C%CCXX    *
*              DATE: 06/28/22      TIME: 16:25:07      USERID: 5654A22C    *
*****
VMFREQ2805I Product :PPF MYCCXX CCXX :PROID 5654A22C%CCXX has passed
          requisite checking
*****
*              Resource Allocation Planning Information        *
*****
*              PPF: MYCCXX CCXX      PROID: 5654A22C%CCXX    *
*              DATE: 06/28/22      TIME: 16:25:07      USERID: 5654A22C    *
*****
Resource requirements for product 5654A22C component CCXX
*****
OWNER: 5654A22C
TARGID:      191
SIZE:        22500
BLKSIZE:     4K
FORMAT:      CMS
RECOMPED:    NO
PREFERRED:   NO
SEPARATED:   NONE

TARGID:      2C2
SIZE:        900
BLKSIZE:     4K
FORMAT:      CMS
RECOMPED:    NO
PREFERRED:   NO
SEPARATED:   NONE

.
.
.
REPLACE USER: 5654A22C
USER 5654A22C XXXXX 256M 2G EG
ACCOUNT xxxxxx
IPL CMS
MACHINE ESA
CONSOLE 009 3215 T
SPOOL 00C 2540 READER *
SPOOL 00D 2540 PUNCH A
SPOOL 00E 1403 A
LINK MAINT 190 190 RR
LINK MAINT 19D 19D RR
LINK MAINT 19E 19E RR
LINK MAINT 51D 51D MR
LINK MAINT 5E5 5E5 RR

```

Figure 8. PLANINFO File Created by a VMFINS INSTALL Command with the PLAN Option

In Figure 8 on page 26, the information that is most important to you is numbered for reference purposes.

**1** shows where you can find the name of the product parameter file that was created when the PLAN option was run (PPF: MYCCXX CCXX).

**2** identifies the section in the PLANINFO file that tells you if all of the requisites have been met for this product. VMFREQ2805I Product :PPF MYCCXX CCXX :PROID 5654A22C%CCXX has passed requisite checking means all of the requisites necessary to install this product are met. There are no missing requisites to report.

**3** identifies the section in the PLANINFO file that tells you what this file contains. For example, Resource requirements for product 5654A22C component CCXX tells you the resources required to install this product are listed below.



**4** identifies the user ID of the owner of the target minidisks. OWNER: 5654A22C, in our example, tells us the minidisks are owned by the 5654A22C user ID.

**5** identifies the first target minidisk for this product. TARGID: 191 means some of the code for this product will be loaded to the 5654A22C 191 minidisk. If we were using the Shared File System, the SFS directory ID would be after TARGID.

**6** and **7** tell us how much space we need on the minidisk to install the product. For example:

```
SIZE:      22500
BLKSIZE:   4K
```

tells us we need 22500 4K blocks available on this minidisk to be able to install the necessary code and information for this product.

You can use the VMFCNVT command to convert this information into the space requirements for the type of DASD you are using. See [“VMFCNVT EXEC” on page 367](#) for more information on using the VMFCNVT command.

If you are installing a product into a Shared File System file pool, the SIZE and BLKSIZE represent the amount of physical space you need in the file pool.

**8** identifies how this minidisk must be formatted. FORMAT: CMS tells us the minidisk must be formatted for CMS.

**9** specifies if this minidisk is **recomped**. In our example, RECOMPED: NO tells us the minidisk has not been recompd. If the minidisk has been recompd, the number of blocksize blocks that are recompd will appear here. In most cases, this will be NO.

**10** specifies whether the minidisk will be stored in a **preferred** location on the full-pack DASD. The middle one third of a full-pack DASD is considered to be a preferred location because information can be retrieved faster by the hardware if it is stored in this location. In our example, PREFERRED: NO, indicates this product will not be stored in a preferred location.

**11** specifies whether a minidisk should be **separated** from another minidisk. If an entry in this format, user ID address, appears after the word SEPARATED, the product should not be put on the same DASD volume with user ID address. In our example, SEPARATED: NONE means we do not have to worry about placing this minidisk on a DASD volume with any other minidisk.

**12** provides information on the server machine, 5654A22C, that is required by the IBM XL C/C++ for z/VM Compiler product. This information is used to update the CP directory during VMFINS processing. If this is a new installation, this information is added to the CP directory to define the 5654A22C server machine. If another version of this server machine is already defined, it will be redefined by the information in this section.

**Note:** If we had changed the defaults on the override panel and saved them, the 5654A22C PLANINFO file would contain the new values defined on the Make Override Panel.

## Determining Whether a Product Can Be Migrated

You can use the PLAN option with the VMFINS MIGRATE command to obtain information to help migrate a product on your system. When you use the PLAN option with the MIGRATE operand, VMFINS reads the product installation media, compares the list of products with the Software Inventory tables, and determines which products can be migrated. To migrate a product with VMFINS, the product must be in VMSES/E format.

When you use the PLAN option with the VMFINS MIGRATE ... (ADD command, VMFINS:

- Determines which tailorings to use for the product you are adding
- Checks the product requisites
- Determines reach-ahead service for the product being migrated
- Creates a product parameter file override file, if you choose to create one

- Determines the requirements for the product resources, for example, the user IDs, the SFS directory and minidisk addresses, and the amount of space required on the minidisks and SFS directories

When you use the PLAN option with the VMFINS MIGRATE ... (REPLACE command, VMFINS:

- Determines which copy of the product you want to replace
- Checks the product requisites
- Determines reach-ahead service for the product being replaced
- Creates a product parameter file override file, if you choose to create one
- Determines the resources being used for the copy of the product you are replacing
- Determines the resource requirements for the product you are installing, for example, the user IDs, the SFS directory and minidisk addresses, and the amount of space required on the minidisks and SFS directories
- Determines which files to delete for the product being replaced

The PLAN option also creates a *prodid* PLANINFO file and a *ppfname* ERASE file for each product specified and stores the files on your A-disk.

### What Information Does the PLANINFO File Provide?

The following example shows a typical *prodid* PLANINFO file. We entered:

```
vmfins migrate prod 5654a22c ccxx (add plan nomemo
```

We created a PPF override file for the CCXX component of product 5654A22C and we named the PPF override file CCXXMIG.

#### **5654A22C PLANINFO File**

Figure 9 on page 29 shows the PLANINFO file created for *prodid* 5654A22C during the PLAN processing.

```

****      VMFINS  MIGRATE                USERID: 5654A22C      ****
*****
****      Date: 2022-06-28              Time: 16:25:07      ****
*****
VMFINS2195I VMFINS MIGRATE PPF 5654A22C CCXX ( SYSTEM VM SIDISK 51D
SIMODE D PLAN NORESOURCE LINK DFNAME USER DFTYPE DIRECT
DFMODE * NOMEMO ADD ENV 5654A22C SETUP
*****
*          Requisite Planning Information          *
*****
*          PPF: CCXXMIG CCXX      PROIDID: 5654A22C%CCXX      * 1
*          DATE: 06/28/22      TIME: 16:25:07      USERID: 5654A22C      *
*****
VMFREQ2805I Product :PPF CCXXMIG CCXX :PROIDID 5654A22C%CCXX has passed 2
requisite checking
*****
*          Reconciliation Planning Information          *
*****
*          PPF: CCXXMIG CCXX      PROIDID: 5654A22C%CCXX      *
*          DATE: 06/28/22      TIME: 16:25:07      USERID: 5654A22C      *
*****
VMFREQ2770I No PTF's need to be re-applied to 5654A22C 3
*****
*          Resource Allocation Planning Information          *
*****
*          PPF: CCXXMIG CCXX      PROIDID: 5654A22C%CCXX      *
*          DATE: 06/28/22      TIME: 16:25:07      USERID: 5654A22C      *
*****
Resource requirements for product 5654A22C component CCXX 4
*****
OWNER: 5654A22C
TARGID: 191 5
SIZE: 22500 6
BLKSIZE: 4K 7
FORMAT: CMS 8
RECOMPED: NO 9
PREFERRED: NO 10
SEPARATED: NONE 11

TARGID: 2C2
SIZE: 900
BLKSIZE: 4K
FORMAT: CMS
RECOMPED: NO
PREFERRED: NO
SEPARATED: NONE

...
TARGID: 2B2
SIZE: 45000
BLKSIZE: 4K
FORMAT: CMS
RECOMPED: NO
PREFERRED: NO
SEPARATED: NONE

REPLACE USER: 5654A22C 13
USER 5654A22C XXXXX 256M 2G EG
ACCOUNT xxxxx
IPL CMS
MACHINE ESA
CONSOLE 009 3215 T
SPOOL 00C 2540 READER *
SPOOL 00D 2540 PUNCH A
SPOOL 00E 1403 A
LINK MAINT 190 190 RR
LINK MAINT 19D 19D RR
LINK MAINT 19E 19E RR
LINK MAINT 51D 51D MR
LINK MAINT 5E5 5E5 RR

REPLACE USER: CCXXESA
USER CCXXESA XXXXX 256M 2G EG
ACCOUNT xxxx
IPL CMS
MACHINE ESA
CONSOLE 0009 3215 T
SPOOL 00C 2540 READER *
SPOOL 00D 2540 PUNCH A
SPOOL 00E 1403 A
LINK 5654A22C 401 191 MR

```

Figure 9. PLANINFO File Created by a MIGRATE with the PLAN Option

In Figure 9 on page 29, the information that is most important to you is numbered for reference purposes.

**1** tells you the CCXXMIG \$PPF file was created during PLAN processing.

**2** is the section that tells you if all the requisites have been met for this product. VMFREQ2805I Product :PPF CCXXMIG CCXX :PROIDID 5654A22C%CCXX has passed requisite checking means you have all the necessary requisites to migrate the product.

**3** tells you there is no service to be reapplied. If there was service to reapply, a file (*prodid \$APPLIST*) would be created on your A-disk. This file contains information on the service that needs to be reapplied.

**4** tells you the resource requirements for the product you want to migrate. The sample PLANINFO file lists all of the target minidisks or SFS directories that CCXX requires and gives the individual requirements for each minidisk or directory.

**5** is the user ID of the owner of the minidisks and SFS directories. In this example, OWNER: 5654A22C, tells you the minidisks and directories are owned by the 5654A22C user ID.

**6** tells you the first target minidisk or SFS directory for this product. In this example, TARGID: 191 tells you CCXX needs the 191 minidisk to migrate some of the product code.

**7** and **8** tell you how much space you need on the minidisk or SFS directory to migrate your product. For example,

```
SIZE:      22500
BLKSIZE:   4K
```

tells you that you need 22500 4K blocks on this minidisk to migrate the product code and information.

You can use the VMFCNVT command to convert this information into the space requirements for the type of DASD you are using. See [“VMFCNVT EXEC” on page 367](#) for more information about using the VMFCNVT command.

If you are migrating a product into a Shared File System file pool, the SIZE and BLKSIZE represent the amount of physical space you need in the file pool. For more information about determining the amount of available space in a SFS file pool, see [z/VM: CMS User's Guide](#).

**9** tells you how the minidisk must be formatted. FORMAT: CMS tells you the minidisk needs to be formatted for CMS.

**10** tells you whether the minidisk has been recomped. In our example, RECOMPED: NO tells us the minidisk had not been recomped. If the minidisk has been recomped, the number of blocks that were recomped will appear here. In most cases, this will be NO.

**11** tells you whether the minidisk will be stored in a preferred location on the full-pack DASD. The middle one-third of the full-pack DASD is considered to be preferred because information can be retrieved faster by the hardware if it stored in this location. In our example, PREFERRED: NO indicates this product will not be stored in a preferred location.

**12** tells you whether a minidisk should be separated from other minidisks. If you see SEPARATED: userid address, the minidisk should not be defined on the same DASD volume with userid address. In our example, SEPARATED: NONE means we do not need to worry about placing this minidisk on a DASD volume with another product.

**13** tells you the CP directory information for the server machine for this product. This information is used to update the CP directory when you use VMFINS. If this information is different from what was defined for the *old* version of the product, it is replaced with the *new* information in this section.

**Note:** If we had changed any product installation parameters during our PLAN, the new parameter installation defaults would appear in the PLANINFO file.

## Determining Whether a Product Can Be Deleted

You can use the PLAN option with the VMFINS DELETE command to obtain information to help you decide if you want to delete a product from your system.

When you use the PLAN option with the VMFINS DELETE command, VMFINS:

- Checks the Software Inventory tables to see which products are installed
- Determines the resources being used by the product you are deleting
- Determines product dependencies
- Determines which files to delete for the product you are removing

VMFINS also creates a *prodid* PLANINFO file and a *ppfname* ERASE file on your A-disk.

The *ppfname* ERASE file shows you the files that will be erased when you delete the product. The *prodid* PLANINFO file shows you how much minidisk or SFS directory space will be available when the product is deleted. It also tells you if other products depend on the product you want to delete.

## What Information Does the PLANINFO File Provide?

The following example shows a typical *prodid* PLANINFO file. We entered:

```
vmfins delete ppf myccxx ccxx (plan
```

### **5654A22C PLANINFO File**

[Figure 10 on page 32](#) shows the PLANINFO file created for the MYCCXX copy of *prodid* 5654A22C.

```

****      VMFINS  DELETE                      USERID: 5654A22C      ****
*****
****      Date: 2022-06-28                    Time: 16:25:07          ****
*****
VMFINS2195I VMFINS DELETE PPF MYCCXX CCXX ( SYSTEM VM SIDISK 51D SIMODE
          D PLAN NORESOURCE NOLINK
*****
*                Requisite Planning Information                *
*****
*                PPF: MYCCXX CCXX      PRODID: 5654A22C%CCXX      * 1
*                DATE: 06/28/22      TIME: 16:25:07      USERID: 5654A22C      *
*****
VMFDEP2805I No other products depend on product :PPF 5654A22C CCXX :PRODID
          5654A22C%CCXX                                          2
*****
*                File Deletion Planning Information                *
*****
*                PPF: MCCXX CCXX      PRODID: 5654A22C%CCXX      *
*                DATE: 06/28/22      TIME: 16:25:07      USERID: 5654A22C      *
*****
VMFDEF1909I MYCCXX ERASE created on your A-disk                    3
*****
*                Resource Allocation Planning Information                *
*****
*                PPF: MYCCXX CCXX      PRODID: 5654A22C%CCXX      *
*                DATE: 06/28/22      TIME: 16:25:07      USERID: 5654A22C      *
*****
Resource requirements for product 5654A22C component CCXX        4
*****
OWNER: 5654A22C
  TARGID:      191          5
  SIZE:        22500
  BLKSIZE:     4K
  FORMAT:      CMS
  RECOMPED:    NO
  PREFERRED:   NO
  SEPARATED:   NONE

  TARGID:      2C2
  SIZE:        900
  BLKSIZE:     4K
  FORMAT:      CMS
  RECOMPED:    NO
  PREFERRED:   NO
  SEPARATED:   NONE

  TARGID:      2D2
  SIZE:        81000
  BLKSIZE:     4K
  FORMAT:      CMS
  RECOMPED:    NO
  PREFERRED:   NO
  SEPARATED:   NONE

...
REPLACE USER: 5654A22C
USER 5654A22C XXXXX 256M 2G EG
ACCOUNT xxxxxx
IPL CMS
MACHINE ESA
CONSOLE 009 3215 T
SPOOL 00C 2540 READER *
SPOOL 00D 2540 PUNCH A
SPOOL 00E 1403 A
LINK MAINT 190 190 RR
LINK MAINT 19D 19D RR
LINK MAINT 19E 19E RR
LINK MAINT 51D 51D MR
LINK MAINT 5E5 5E5 RR

```

Figure 10. PLANINFO File Created by a VMFINS DELETE with the PLAN Option

The important parts of the 5654A22C PLANINFO file are noted in [Figure 10 on page 32](#).

**1** tells you this is the PLANINFO file for the MYCCXX copy of the IBM XL C/C++ for z/VM Compiler product.

**2** lists the products that depend on the product you are deleting. In this example, no other products depend on product 5654A22C, component CCXX.

**3** tells you the MYCCXX ERASE file has been created on your A-disk. This file contains a list of the product files that will be deleted.

**4** tells you what the resource requirements currently are for the product you are deleting. This sample PLANINFO file lists all the target minidisks or SFS directories that 5654A22C owns and gives the individual requirements set up for each minidisk or directory.

**5** tells you the first target minidisk or SFS directory for this product. This example, TARGID: 191, tells you that some of the product code will be deleted from the 191 minidisk.

**Note:** You should note each of the minidisks and SFS directories that will be affected by the delete.

## Calculating Space Requirements for Installation and Migration

You can use the VMFCNVT EXEC to convert the size and blocksize information from the *prodid* PLANINFO file into the number of cylinders that are required to install or migrate a product. For more information on the VMFCNVT EXEC, see “VMFCNVT EXEC” on page 367.

## Overriding Product Installation Defaults

When you use VMFINS to install and migrate products in VMSES/E format, you have an opportunity to override the default installation parameters that have been provided for each product by IBM. The **Make Override Panel** is displayed (Figure 11 on page 33), and you can change the information on the panel.

The Make Override Panel shows you the default minidisks, user IDs, or Shared File System directories that will be used when the product is installed or migrated, unless you choose to override them.

```

File  Help
-----
MKOVR1                      Make Override Panel                      More:
Storage resource for product 1234567 component COMP
PRODUCT Userid..... LPUSERID
COMP Samp & Loadlibs. 59F          Link as..... 59F
COMP Text..... 49F             Link as..... 49F
IPCS Text..... 193             Link as..... 193
COMP Source..... 39F           Link as..... 39F
COMP CMS Help Files.. 19D       Link as..... 19D
COMP Service Disk.... 29F       Link as..... 29F

COMP Service Machine... COMP

Command===> _____

F1=Help F2=Command F3=Exit F4=Expand Dirid F5=Save as... F6=Mdisk or SFS
F7=Backward F8=Forward F9=Retrieve F10=Action F11=Conflict F12=Cancel

```

*Figure 11. Make Override Panel*

You can also use the Make Override Panel to change product parameter file overrides that you have created during previous product installations and migrations.

## Using the Make Override Panel

When the Make Override Panel is displayed, you can enter new product installation parameters by typing over the parameters that appear on the panel (Figure 12 on page 34).

## Where Do the Product Installation Parameters Come From?

When you enter a VMFINS INSTALL or VMFINS MIGRATE command with the ADD option, the product installation parameters are taken from the source product parameter file.

When you enter a VMFINS INSTALL or VMFINS MIGRATE command with the REPLACE option, the product installation parameters are taken from the product parameter file for the copy of the product you are replacing.

## Understanding the Make Override Panel Information

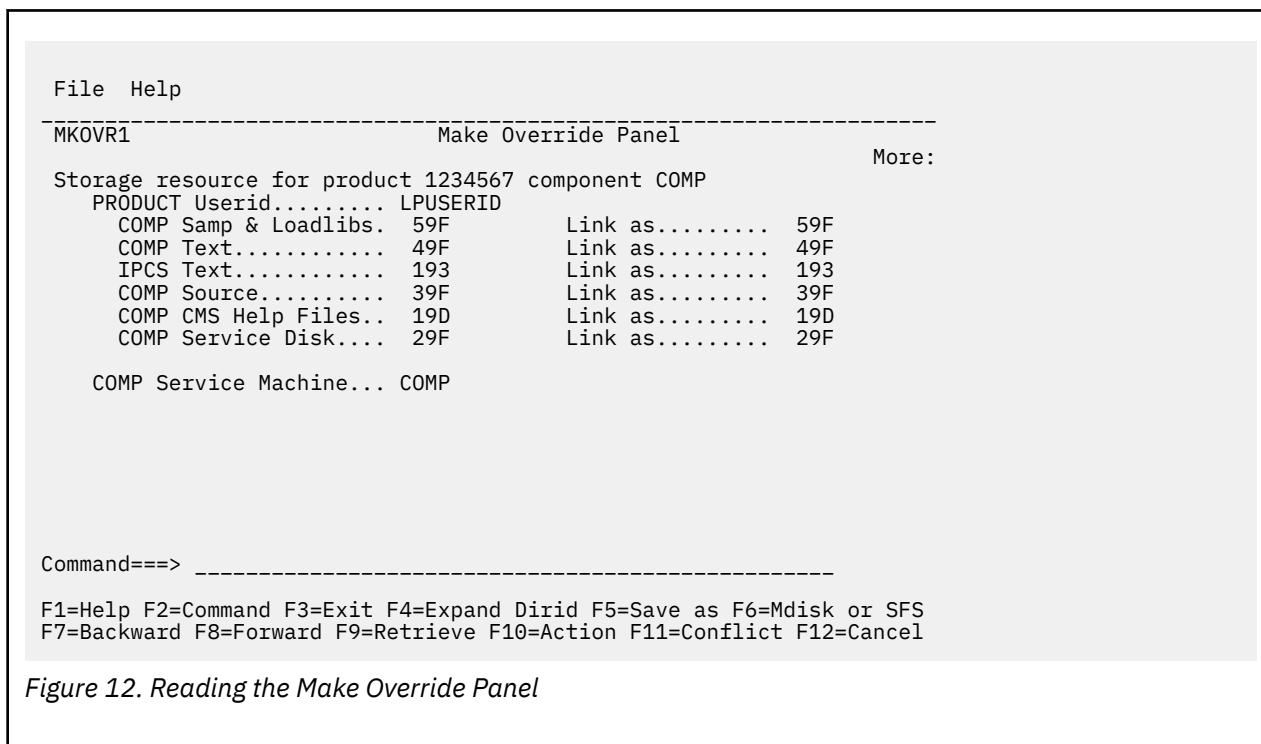


Figure 12. Reading the Make Override Panel

In [Figure 12 on page 34](#), The default product installation parameters are shown to the right of the descriptions of the input fields. For example, 59F is the default production installation parameter for the COMP Samp & Loadlibs field.

Keep the following information in mind when you use the Make Override Panel:

- When a colon (: ) precedes an input field, you cannot make changes to the information in that field.
- When a period ( . ) precedes an input field, you may change the information in that field.
- When an ellipsis ( ... ) follows an option, you will need to provide additional information to complete the action.
- When using variables and their values as defined in the VMSESE PROFILE, they must follow the CMS naming conventions and lengths defined for the field you are changing.
- Entering an asterisk ( \* ) in the PRODUCT Userid input field tells VMFINS to use the user ID on which you are currently logged on.
- On color displays:
  - Fields highlighted in white are in conflict with information in an existing product parameter file and should be changed.
  - Fields highlighted in yellow are in error and **must** be changed.
- On monochrome displays:
  - Fields highlighted with high-intensity characters contain either errors or information that is in conflict with information in another product parameter file override on your system. If there is an error, the information **must** be changed. High-intensity characters appear to be brighter than the other characters on the display.
- When you see **More: +** in the upper right corner of the panel, there is more information available. You may scroll through it with the F8=Forward key.



- When you see **More:** - in the upper right corner of the panel, there is more information available. You may scroll through it with the F7=Backward key.
- Any valid CP or CMS command can be entered on the command line.
- If you are using F1=Help on the Make Override Panel:
  - And the cursor is on an input field, F1=Help tells you if the characters you entered are valid for that field
  - And the cursor is on an area other than an input field, F1=Help provides general help on the Make Override Panel.
- The F12=Cancel key closes the window and returns to the previous window or panel. The F12=Cancel key can be used in any pull-down or pop-up window.

For HELP on the Make Override Panel, enter:

```
help vmses vmfmkovr
```

on the Make Override Panel command line or the CMS command line.

## Using the Function Keys

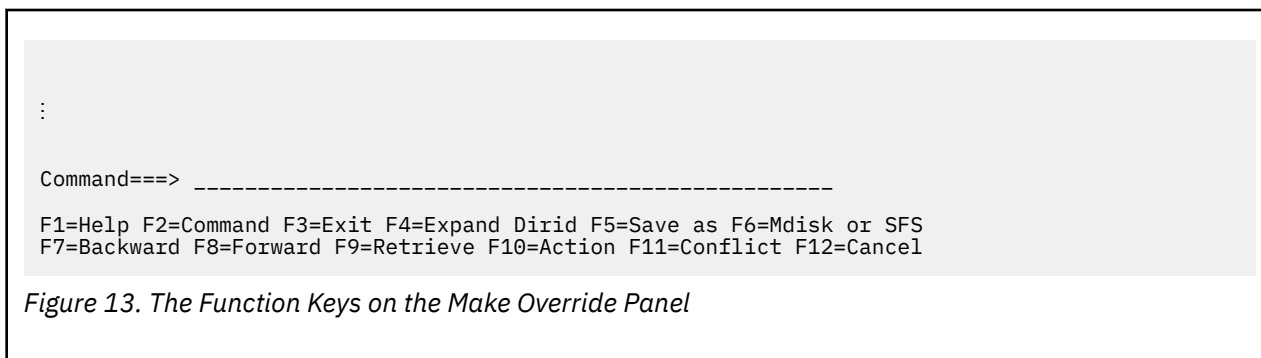


Figure 13 on page 35 shows the Function key assignments. Table 4 on page 35 explains the function each key provides.

Table 4. Function Key Assignments for the Make Override Panel

Function Key	Function Provided
F1=Help	Indicates if the information entered is valid for that input field. Move the cursor to the input field in question and press the F1 key. For example, if =39F is entered as a minidisk address, F1=Help tells you the entry is incorrect because '=' is not a valid character for a minidisk address.  If the cursor is not on an input field when you press F1=Help, general help on the Make Override Panel is provided.
F2=Command	Moves the cursor to the command line and moves the cursor from the command line to its previous position.

Table 4. Function Key Assignments for the Make Override Panel (continued)

Function Key	Function Provided
F3=Exit	<p>Exits from the Make Override Panel.</p> <p><b>If the information on the Make Override Panel has been changed</b> and the changes have not been saved, F3=Exit gives you an Exit Confirmation window with the following options:</p> <ol style="list-style-type: none"> <li>1. Save</li> <li>2. Save as</li> <li>3. Quit</li> </ol> <p>Option 1 (Save) and option 2 (Save as...) allow you to save the product installation parameters, exit from the Make Override Panel, and continue VMFINS processing.</p> <p>Option 3 (Quit) exits from VMFINS processing and does not save changes or allow you to provide a product parameter file override name.</p> <p><b>If you do not change the product installation parameters</b> and you do not save the original product installation defaults in a product parameter file override, F3=Exit exits from the Make Override Panel. You are asked how you would like to proceed. You can bypass processing for the product, you can exit immediately, or you can continue processing the product.</p> <ul style="list-style-type: none"> <li>• If you used the PPF operand, you continue to process the product using the existing product parameter file override (PPF).</li> <li>• If you used the PROD operand, you continue to process the product using the product parameter file (\$PPF) that was shipped with the product.</li> </ul> <p>There must be a product parameter file override for each copy of the product you install or migrate. The product parameter file override can contain either the original product installation parameter defaults provided by IBM or new product installation parameters that you have entered.</p>
F4=Expand dirid	<p>Either provides more space to enter a Shared File System directory ID, or displays an existing Shared File System directory ID. Shared File System directory IDs can be up to 153 characters long, and they typically include the file pool, user ID, and subdirectory names.</p>
F5=Save as...	<p>Lets you save the updated information in a new file with a name you assign. The Save as . . . panel is displayed, just as it would be if you selected Save as . . . from the action bar File option.</p>
F6=Mdisk or SFS dir	<p>Changes the entry format for the input field from minidisk format to Shared File System directory format. When you select Shared File System directory format, VMFINS provides a default SFS directory name. See <a href="#">“Changing from Minidisk to SFS Directory Entry Format”</a> on page 46 for more information on entering Shared File System directory information and the default SFS directory names provided by VMFINS. F6=Mdisk or SFS dir is a toggle key.</p>
F7=Backward	<p>Moves backward through the information. More: - in the upper right corner of the panel means there is more information, and you may scroll backward through it.</p>
F8=Forward	<p>Moves forward through the information. More: + in the upper right corner of the panel means there is more information to display; and you may scroll forward through it.</p>

Table 4. Function Key Assignments for the Make Override Panel (continued)

Function Key	Function Provided
F9=Retrieve	Retrieves up to 5 previous entries for the input field indicated by the cursor position. When a blank entry is displayed, you have either retrieved all previous entries (if there were less than 5) or you have already retrieved the fifth entry. If you press F9=Retrieve after a blank entry, you can cycle through the previous entries again.
F10=Action	Moves the cursor to the action bar at the top left of the panel or moves the cursor from the action bar to the previous cursor position.
F11=Conflict	Takes you to the first highlighted field that was found to be in conflict with a value in an existing product parameter override file. A message, which is displayed at the bottom of the panel, tells you which product parameter file override ( <i>ppfname</i> ) contains the information that is in conflict with the information currently on the panel.
F12=Cancel	Exits from the Make Override Panel and ends VMFINS processing or cancels the current window.

## Using the Action Bar

In Figure 14 on page 37, on the upper left of the panel, you see the words **File** and **Help** and a line that runs all the way across the panel. The area above the line is called the **action bar**. To move the cursor to the action bar, press F10=Action. To move the cursor back to its previous position, press F10 again.

To select an action, move the cursor to the word that indicates the action you would like to perform. For example, if you want to save the changes you made, move the cursor to **File**. The tab key, which is located on your keyboard, can be used to move between the selections on the action bar. To activate your selection, press Enter.

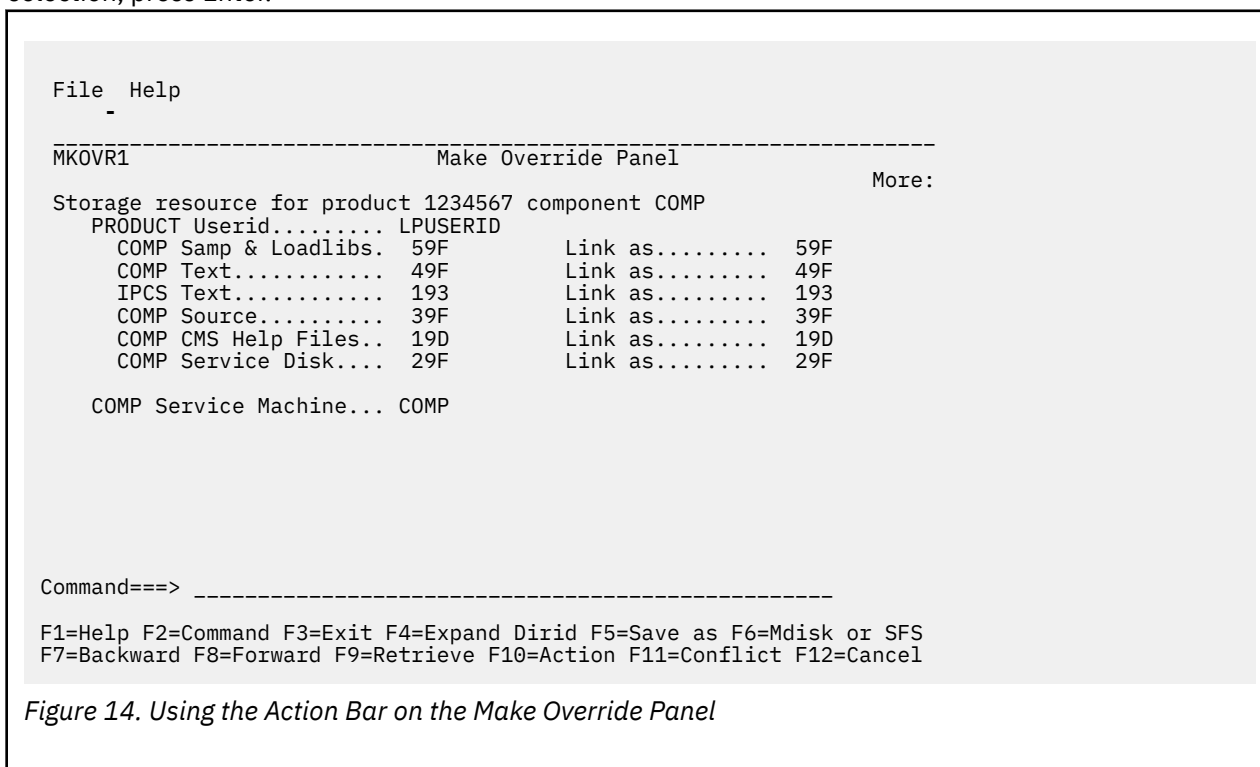


Figure 14. Using the Action Bar on the Make Override Panel

## Understanding the File Options

To see the options available under the File action key, move the cursor to File and press Enter. A pull-down window is displayed (Figure 15 on page 38).

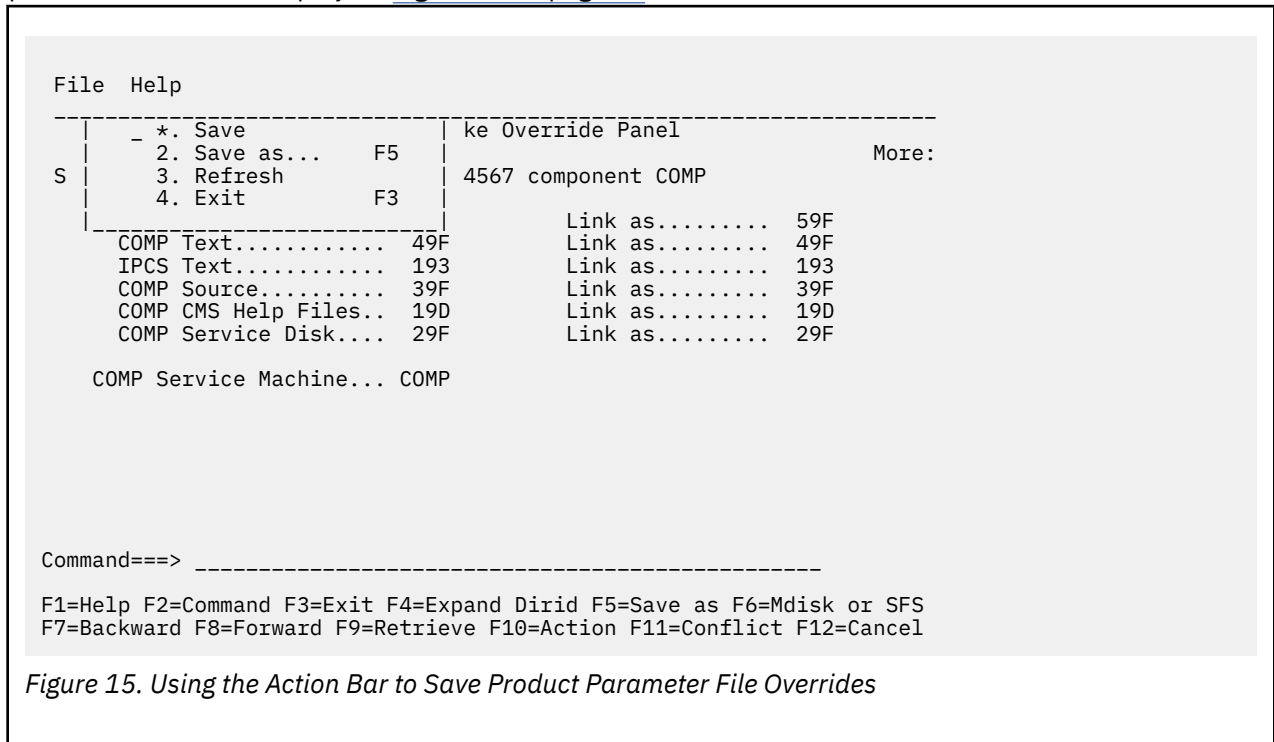


Figure 15. Using the Action Bar to Save Product Parameter File Overrides

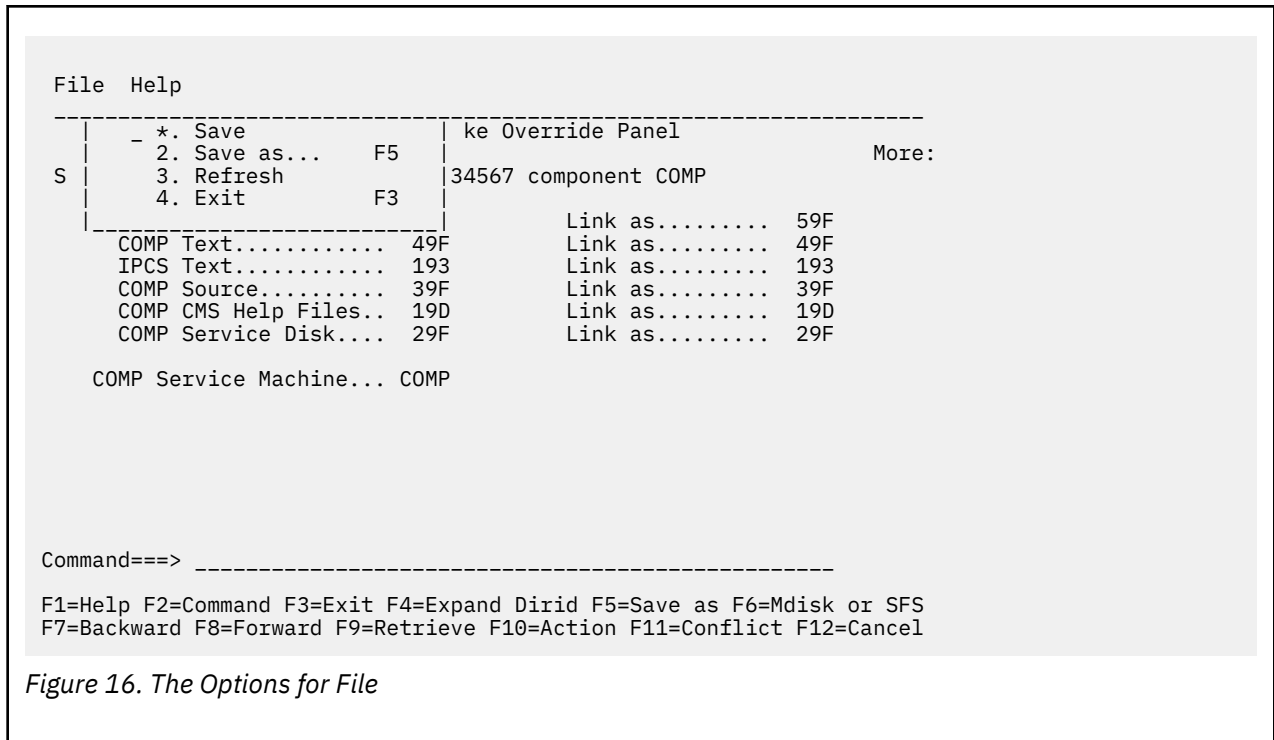
To select an option, either:

- Enter the number of the option in the blank which appears to the left of the first option in the window.
- Move the cursor so it appears on the number of the option and press Enter.

If you want to use the Save as... option, you may also press F5=Save as.... To select the Exit option, you may also use the F3 key.

## Saving a Product Parameter File Override

As you can see in [Figure 16 on page 39](#), there are a number of options available to you within File. Any option preceded by an asterisk (\*), however, is not available at the present time. In [Figure 16 on page 39](#), there is an asterisk (\*) to the left of Save.



Option 1, **Save**, lets you save the information on the Make Override Panel in a current working file. New information that has been entered overwrites the existing information, and you reuse the existing current working file name. **Save** assumes you are working with an existing product parameter file for one copy of the product.

In this example, we are working with this product for the first time; and we have not created a current working PPF override file. **Save**, therefore, is not an available option at this time; and it is preceded by an asterisk (\*).

Option 2, **Save as . . .**, lets you save the information updated on the override panel in another file with a new name. When you select this option, the **Save as...** window opens; and you can name the new PPF override file. You may also decide to keep the existing name. If you do, the updated information, which is currently shown on the panel, is saved in the existing file; and it overwrites the existing information.

Option 3, **Refresh**, erases all new entries on the panel and returns it to its original form.

Option 4, **Exit**, lets you exit from the Make Override Panel. If the information on the Make Override Panel has been changed and the changes have not been saved, **F3=Exit** gives you an Exit Confirmation window with the following options:

1. Save
2. Save as ...
3. Quit

In the Exit Confirmation window, option 3 (**Quit**) exits from VMFINS processing and does not save changes or allow you to provide a product parameter file override name.

## Note

If you do not change the product installation parameters and you do not save the original product installation defaults in a product parameter file override, **F3=Exit** ends VMFINS processing. There must be a product parameter file override for each copy of the product you install or migrate.

In [Figure 17](#) on page 40, option 2, **Save as . . .**, is selected.

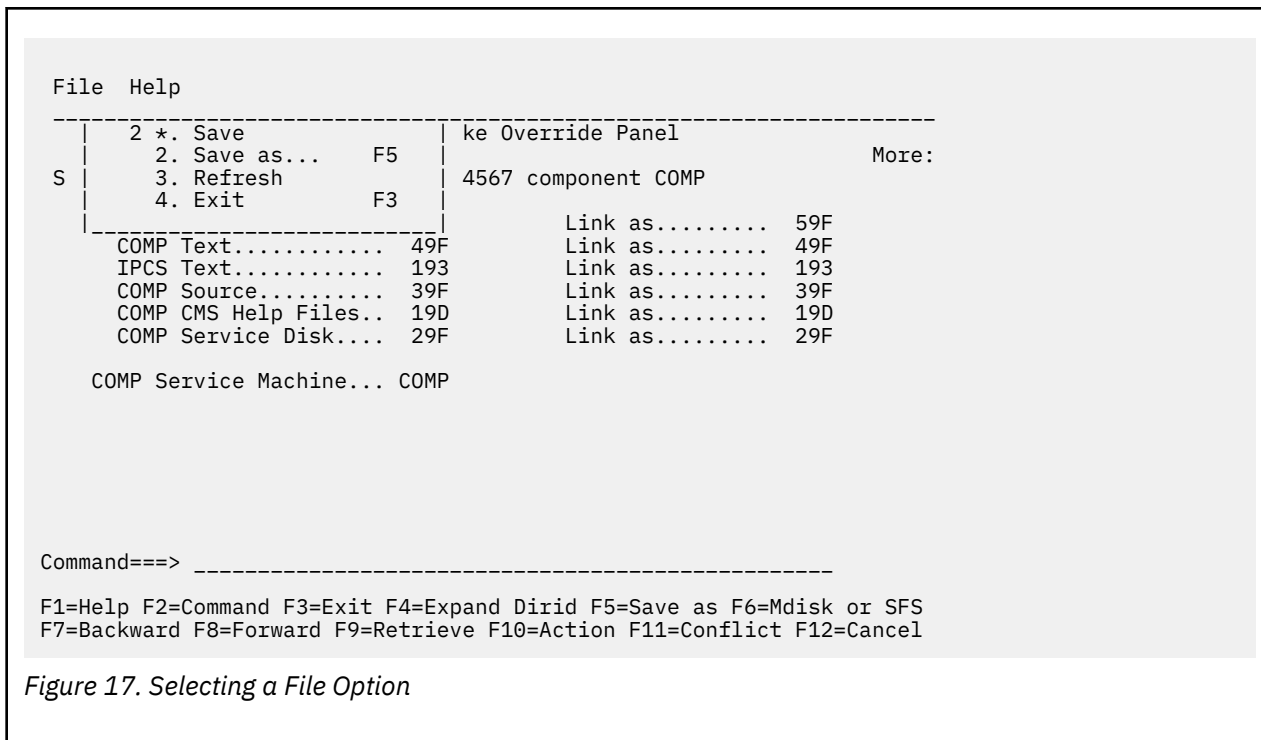


Figure 17. Selecting a File Option

In Figure 18 on page 40, the Save as . . . window is open; and you see the information you need to provide to save the new product parameter override file.

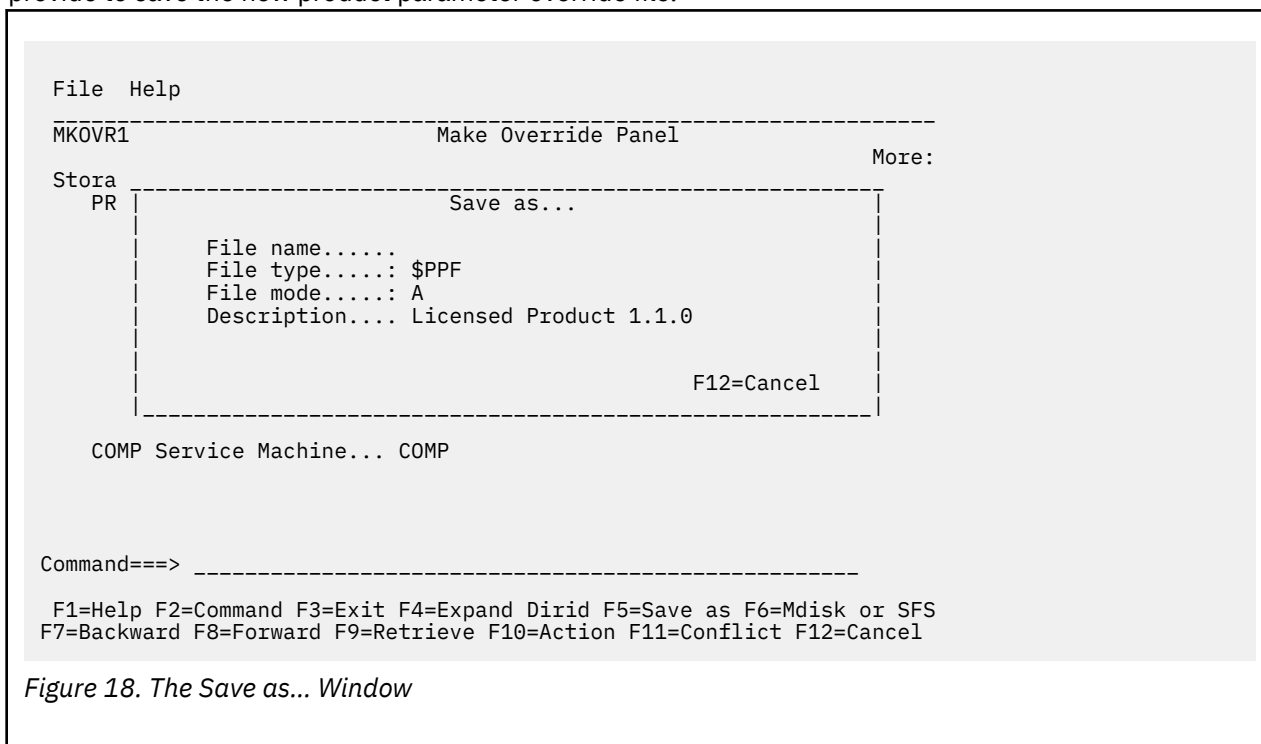


Figure 18. The Save as... Window

In Figure 18 on page 40, there is a colon (:) to the left of \$PPF, the file type, and A, the file mode. We cannot change the information in these fields. We can, however, enter a new file name and a new description.

In Figure 19 on page 41, we entered a new file name (LP110CP1) for the product parameter override file.

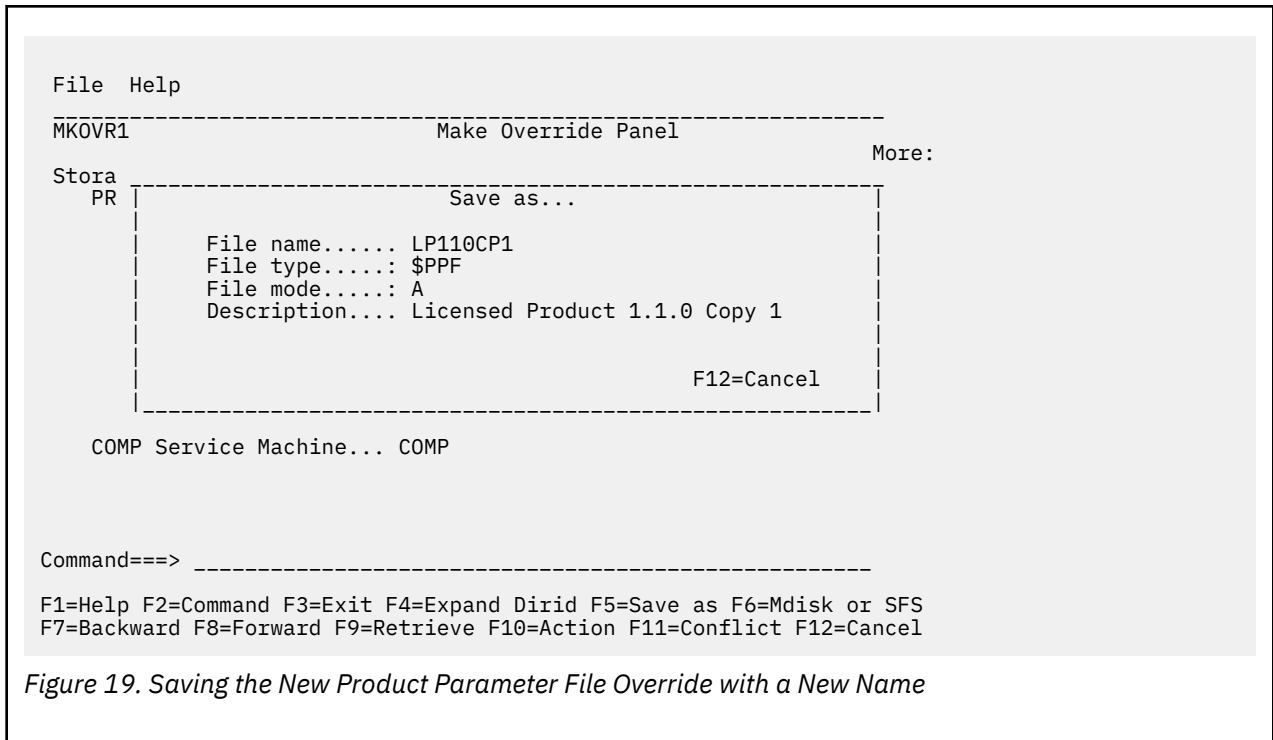


Figure 19. Saving the New Product Parameter File Override with a New Name

Once you save the new override file, you return to the Make Override Panel (Figure 20 on page 41). If you look at the bottom left of the panel, you see two messages. One tells you the product parameter file has been saved as LP110CP1 \$PPF on the A-disk. The other message tells you that you are now working with the LP110CP1 \$PPF file, which is the name of the new product parameter override file you just created.

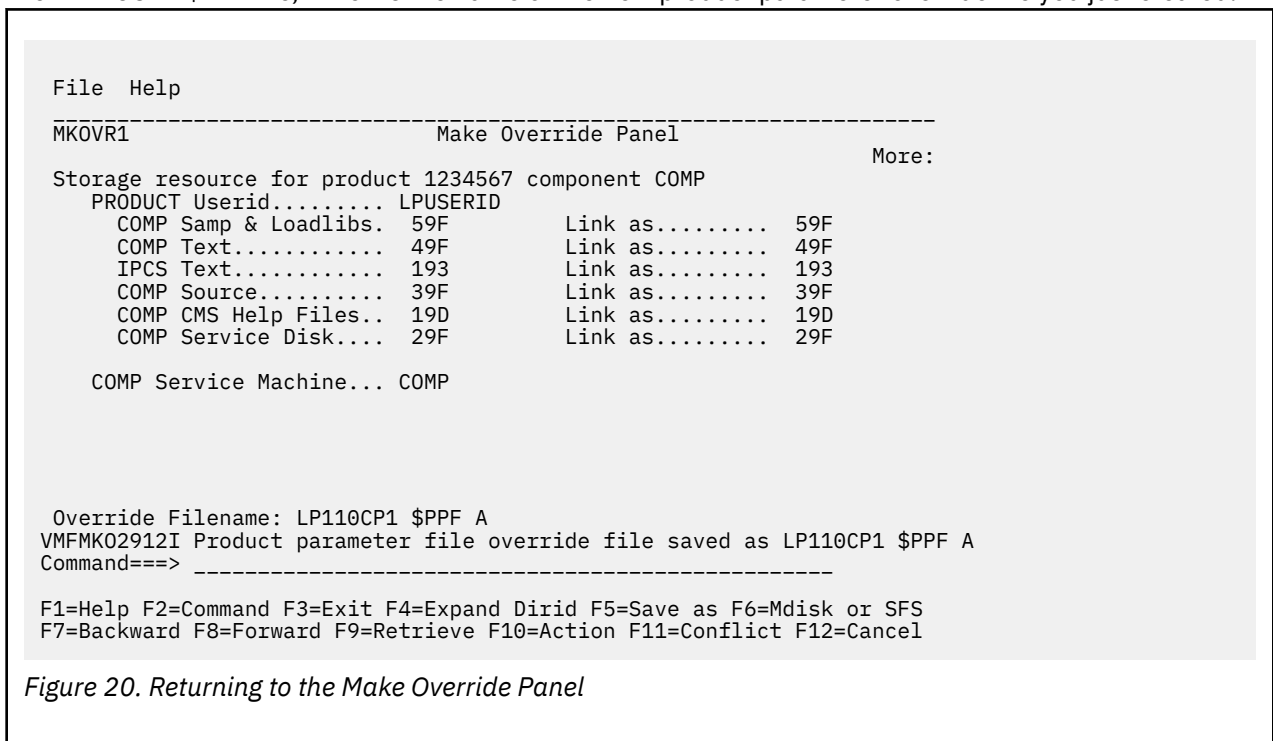


Figure 20. Returning to the Make Override Panel

From this point on, all processing is completed using the LP110CP1 \$PPF file. VMFINS uses the last override file created, unless otherwise specified.

If you enter F3=Exit at this point, you leave the Make Override Panel; and processing continues.

If you enter F12=Cancel, you have an opportunity to confirm your selection. The Exit Confirmation window is displayed, and you may select:

1. Save
2. Save as ...
3. Quit

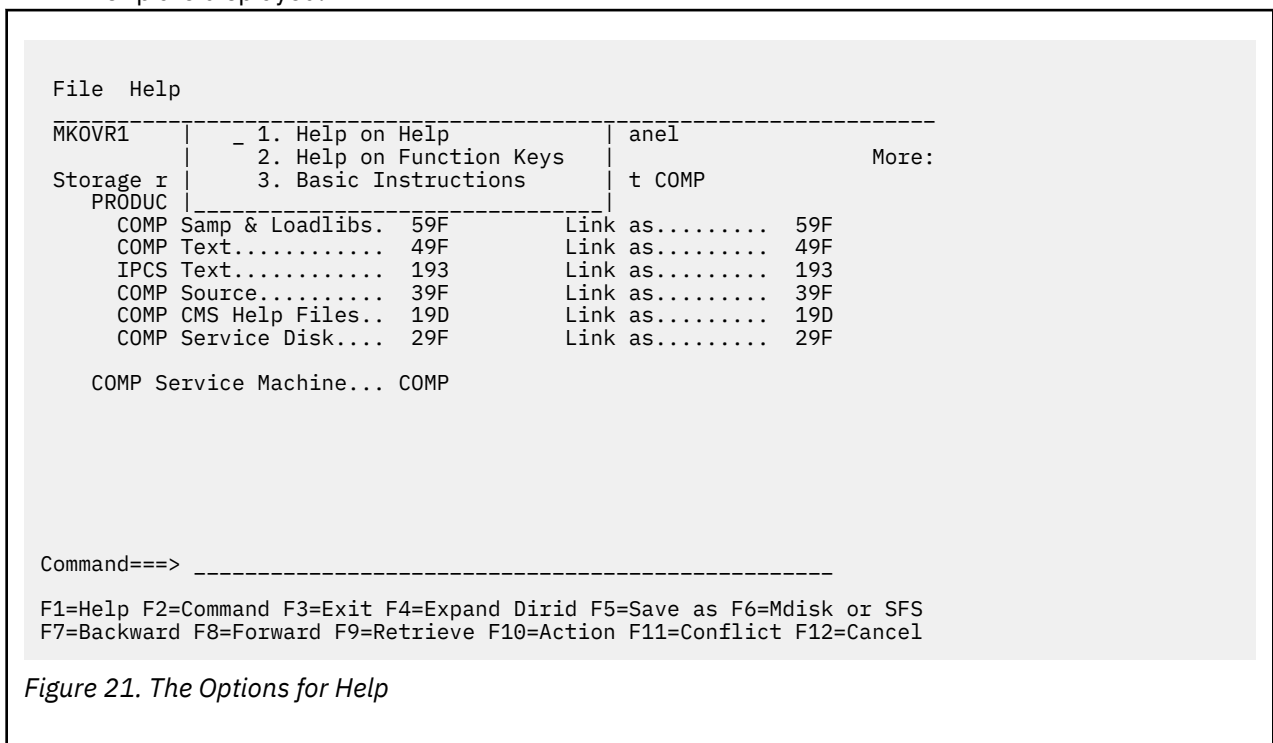
Option 1, Save, lets you save the current working file and end VMFINS processing.

Option 2, Save as ..., lets you save the current working file with a new file name and end VMFINS processing.

Option 3, Quit, ends VMFINS processing and does not save the current working file.

## Understanding the Help Options

To get to the Help options, use F10=Action to move the cursor to the action bar. Place the cursor under Help and press Enter. As shown in [Figure 21 on page 42](#), a window opens, and the options available within Help are displayed.



*Figure 21. The Options for Help*

To select an option, either:

- Enter the number of the option in the blank which appears to the left of the first option in the window.
- Move the cursor so it appears on the number of the option and press Enter.

In [Figure 22 on page 43](#), we select option 1, Help on Help.



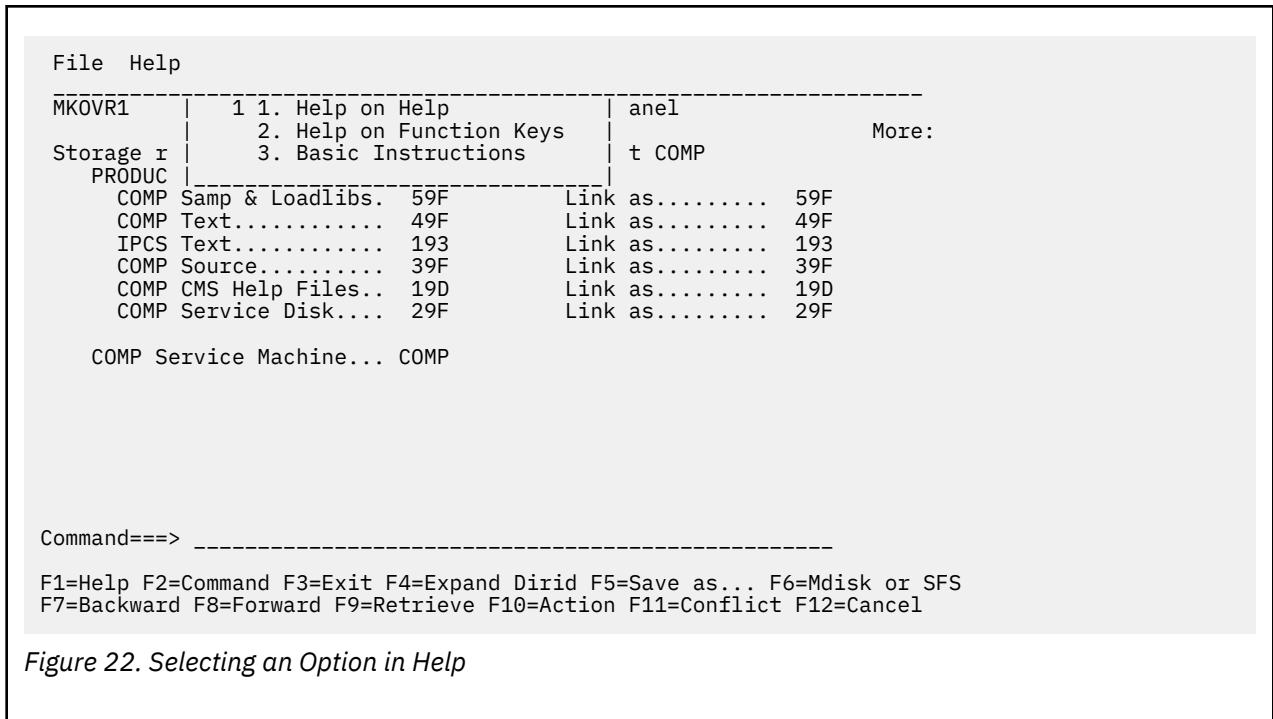


Figure 22. Selecting an Option in Help

## Getting Help on Help

When you select option 1, the following window is displayed.

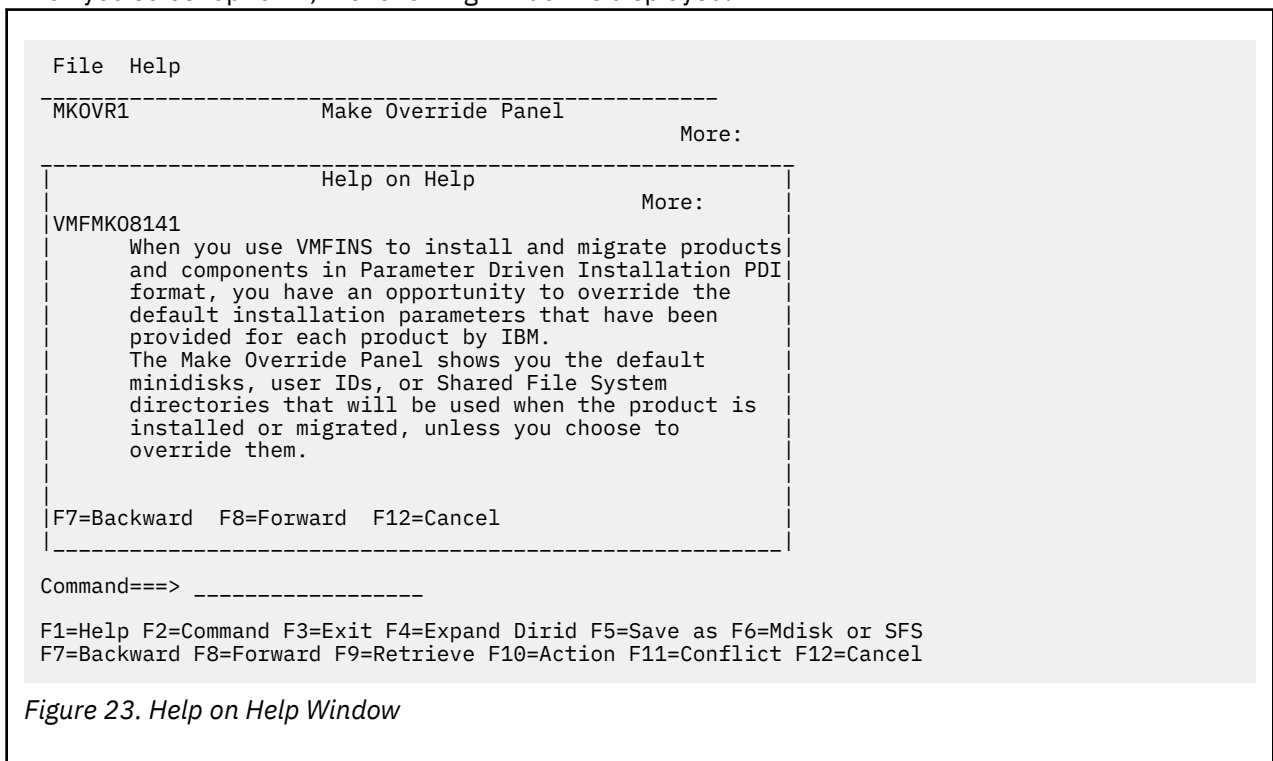


Figure 23. Help on Help Window

Help on Help gives you general help information on the Make Override Panel and includes:

- The purpose of the Make Override Panel
- Instructions for creating an override file using this panel
- Instructions on saving files.

## Getting Help on the Function Keys

To get help on a Function Key, select option 2. The Help on Function Keys window is displayed (Figure 24 on page 44).

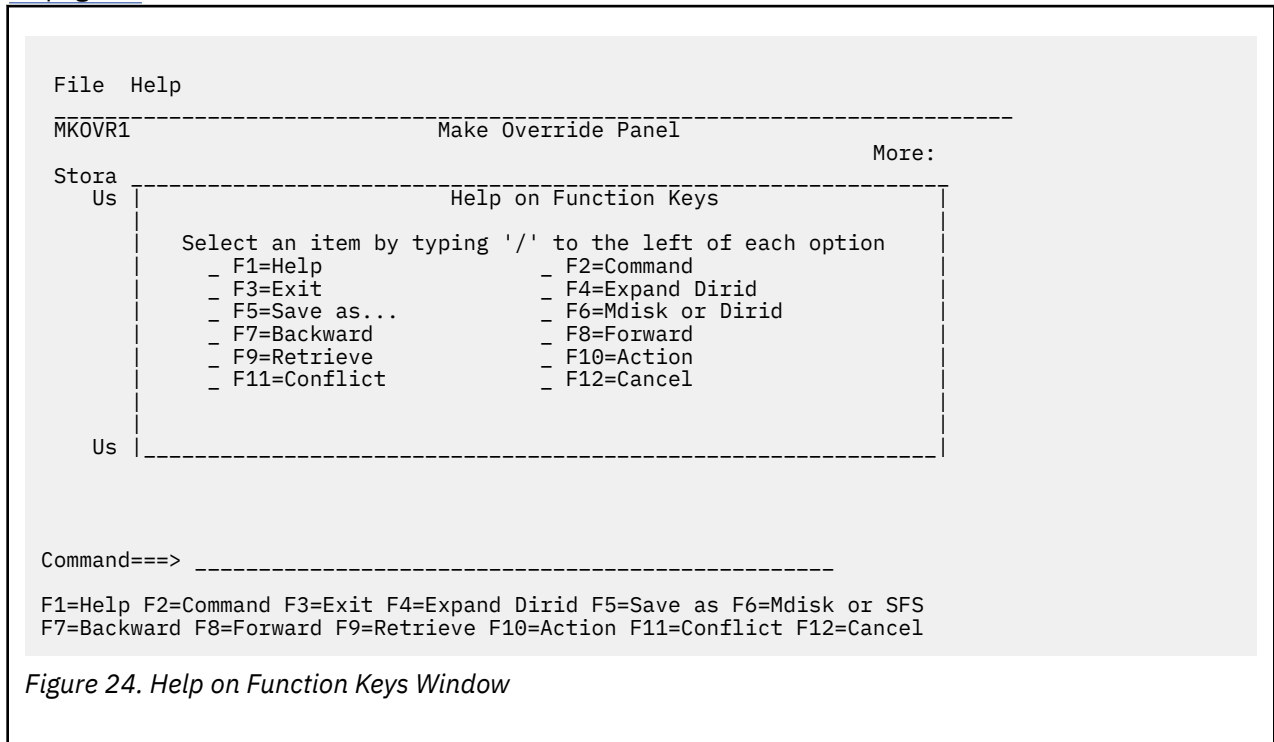


Figure 24. Help on Function Keys Window

You can select help on an individual function key by placing a slash (/) to the left of the function key in question. For example, in Figure 25 on page 44, we have asked for help on the F6 function key.

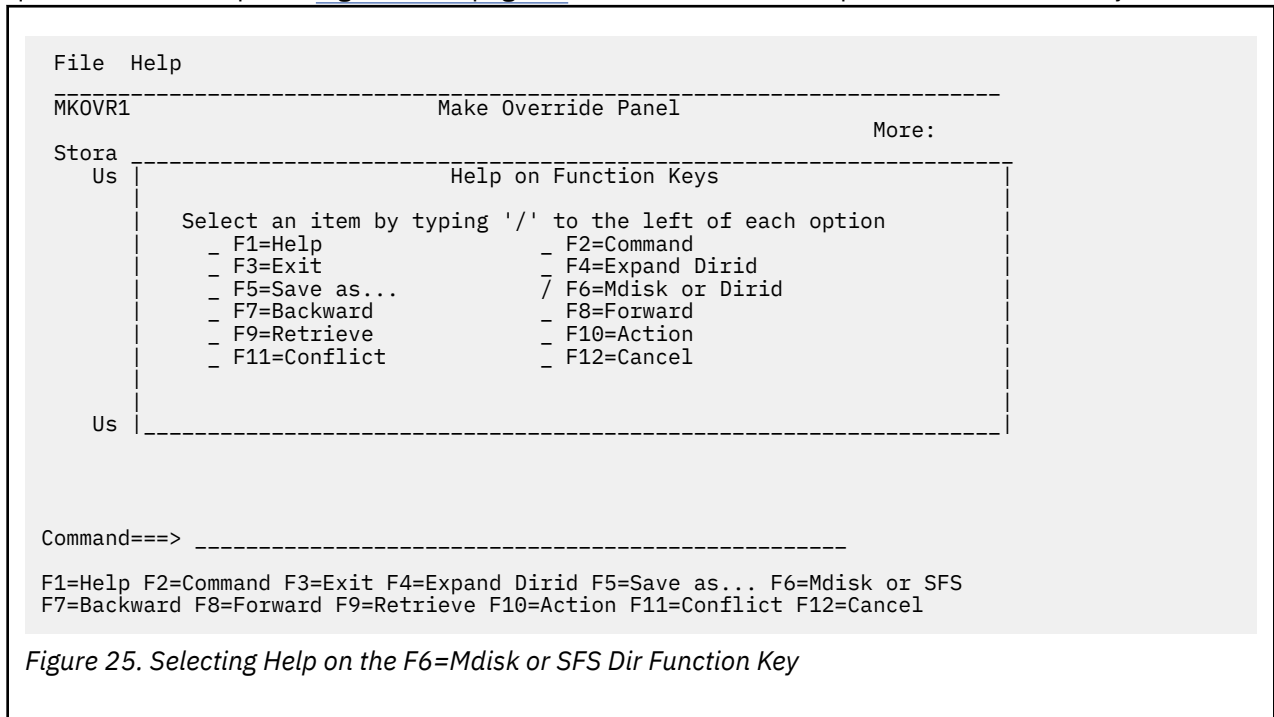


Figure 25. Selecting Help on the F6=Mdisk or SFS Dir Function Key

Figure 26 on page 45 shows you the help information for the F6=Mdisk or SFS dir function key.

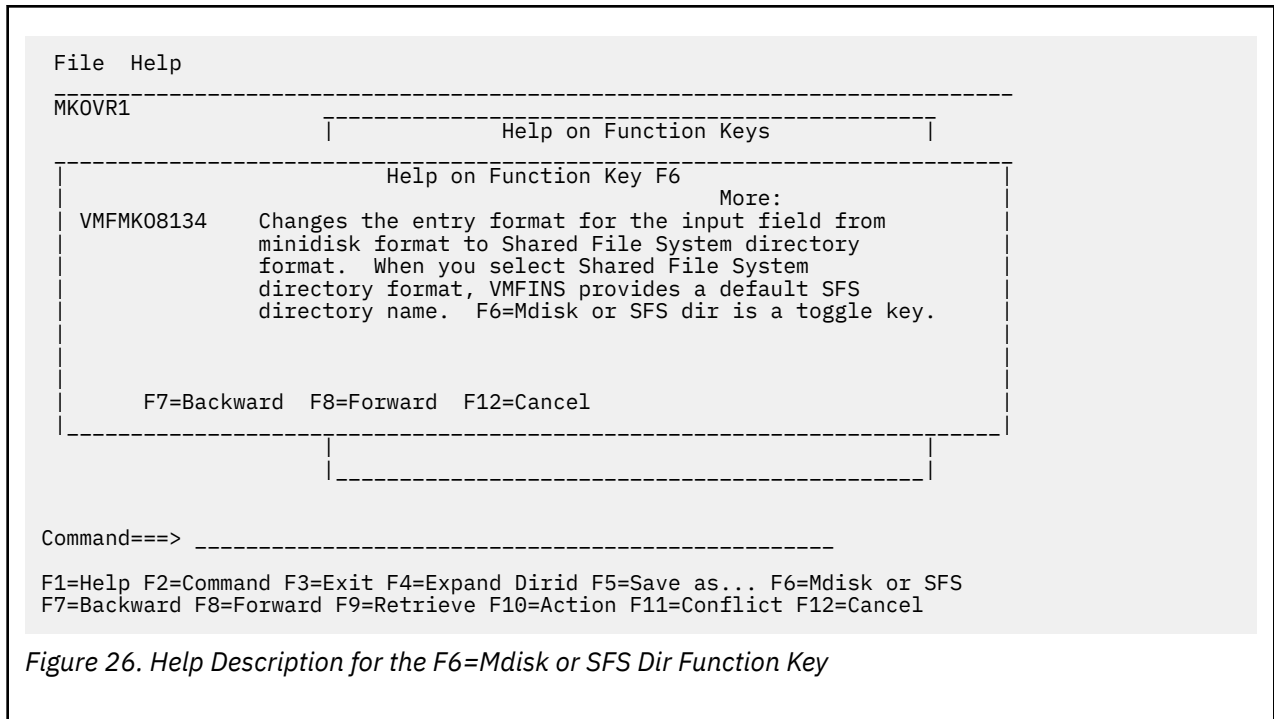


Figure 26. Help Description for the F6=Mdisk or SFS Dir Function Key

## Getting Basic Instructions on the Make Override Panel

To get general help on using the Make Override Panel, select option 3, Basic Instructions, from the Help options pull-down window (shown in Figure 27 on page 45). To see the general help provided by the Basic Instructions option, see “Understanding the Make Override Panel Information” on page 34.

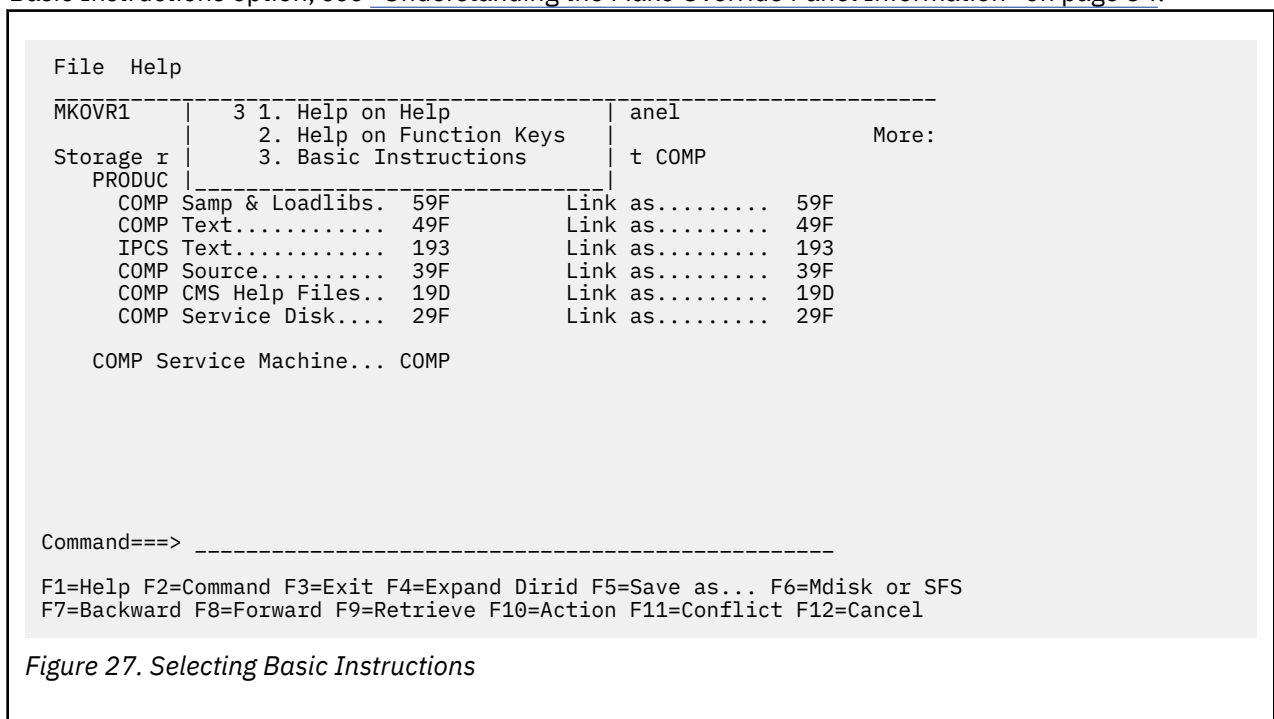


Figure 27. Selecting Basic Instructions

## Online Help

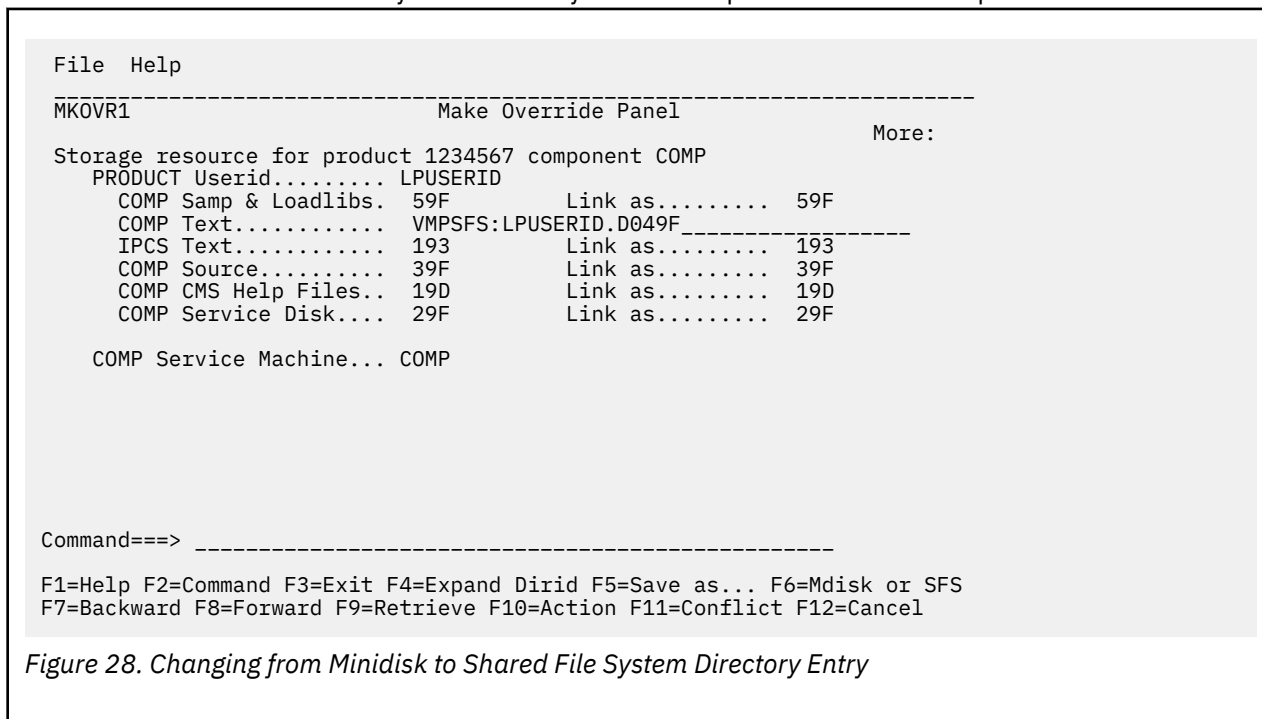
To get HELP online, enter `help vmses vmfmkovron` on the Make Override Panel command line or the CMS command line.

## Changing from Minidisk to SFS Directory Entry Format

You can use the F6=Mdisk or SFS dir function key to change the format of the input fields from minidisk format to Shared File System directory format. To change the format of the input field, move the cursor to the field in question, and press F6. F6 removes the existing entry and provides a default Shared File System directory name, which you may use or change.

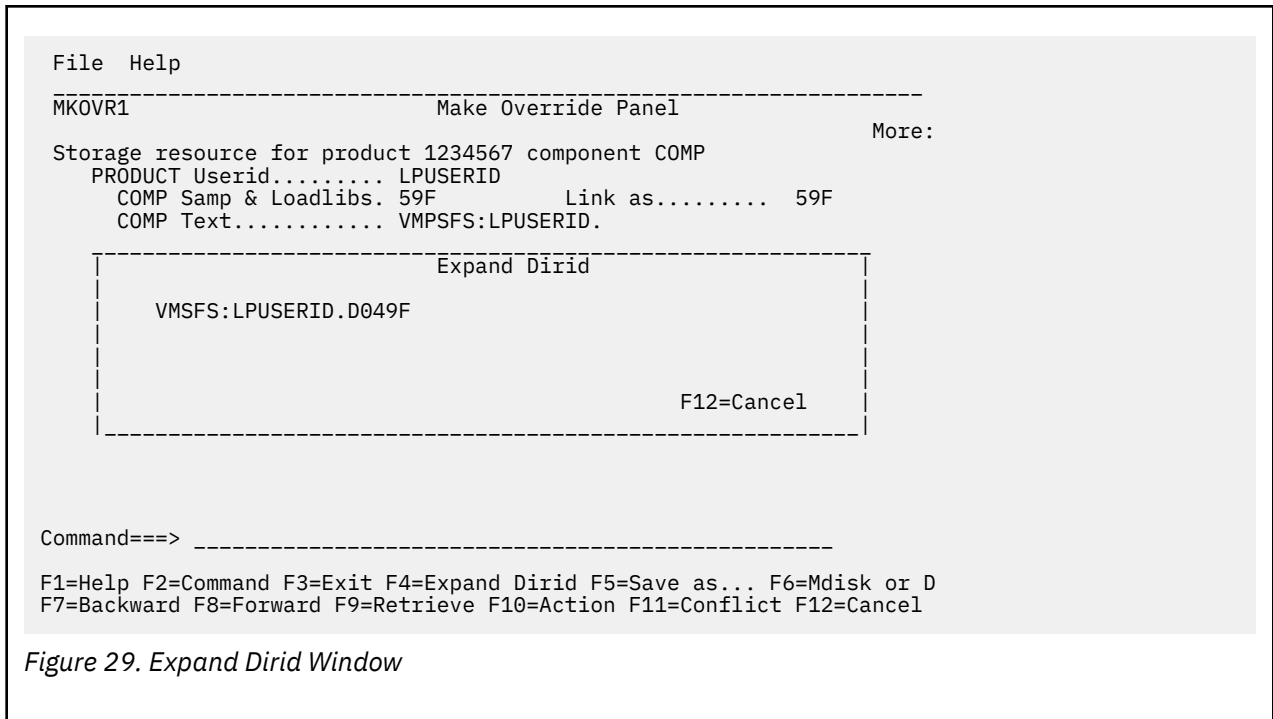
## The Default Shared File System Directory Name

The default SFS directory name is defined by the file pool name, the user ID that owns the target minidisk, and the target minidisk address prefixed by the letter D. (VMPSFS: is the default file pool name defined for VMSES/E, but you can change the default.) As you can see in [Figure 28 on page 46](#), the field after COMP Text has a default Shared File System directory name and space for additional input.

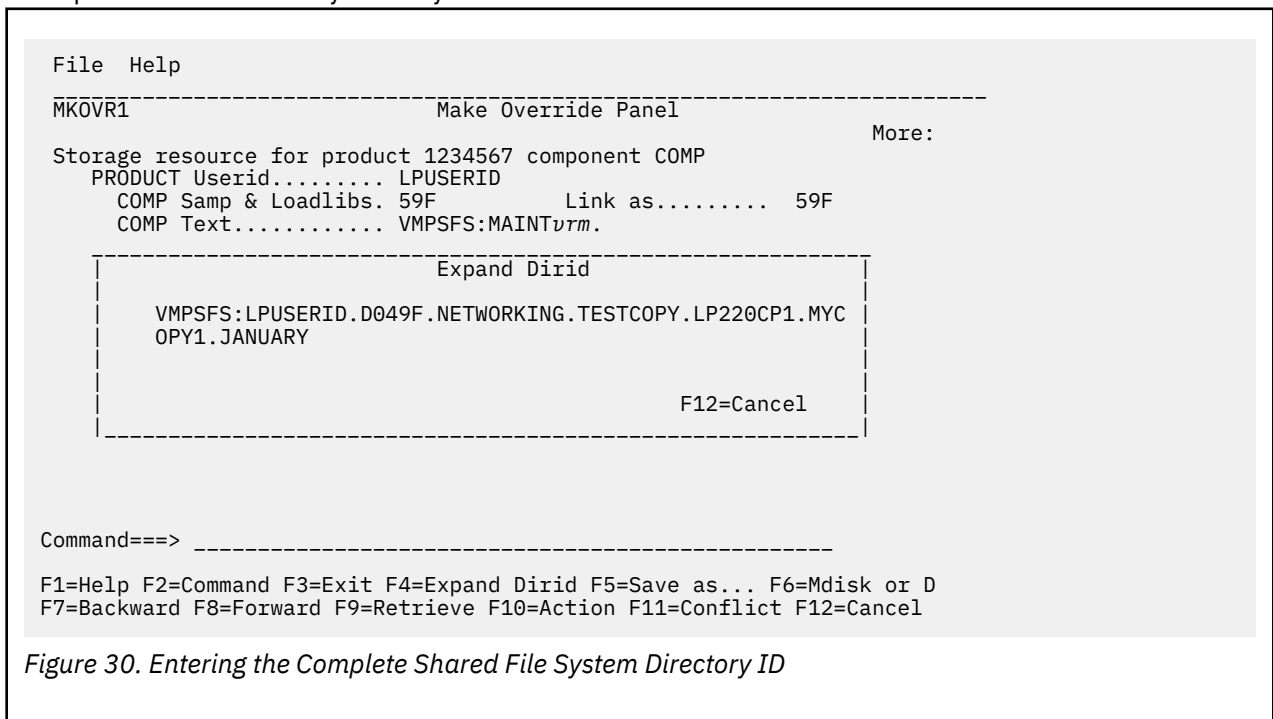


*Figure 28. Changing from Minidisk to Shared File System Directory Entry*

If you need more space to enter the complete Shared File System directory ID, press F4=Expand Dirid. [Figure 29 on page 47](#) shows the Expand Dirid window which provides additional space for your input.



To complete the SFS directory ID, just type the remainder of the information in the window provided. You can enter up to 153 characters, including the colon and the periods. [Figure 30 on page 47](#) shows you an example of an SFS directory ID entry.



To save the new Shared File System directory name, select File from the action bar or press F5=Save as.... See [“Saving a Product Parameter File Override” on page 38](#) for details on using File.

## Changing the Shared File System Directory File Pool Name

You can change the Shared File System directory file pool name by:

- Entering the FILEPOOL option on the VMFINS command.
- Changing the value for the FILEPOOL option in the VMFINS DEFAULTS file.

- Using the F6 key on the Make Override Panel. [Figure 28 on page 46](#) shows the Make Override Panel after using the F6 key. If ALTSYS: had been entered as the file pool ID on the VMFINS command, the panel in [Figure 28 on page 46](#) would show ALTSYS:LPUSERID.D049F instead.

When you change the SFS directory file pool, make sure:

- The file pool you are specifying is available in interactive mode.
- You have administrative authority for the file pool for the user ID performing the VMFINS operation. You need administrative authority to enroll yourself in the file pool or increase the amount of space allocated.
- You create the *filepoolid:userid.VMFINS* directory or enroll a user in the file pool.

## Controlling the VMFINS Prompts

---

You can change the way prompts are issued during VMFINS INSTALL and MIGRATE processing by using the OVERRIDE option when you issue a VMFINS INSTALL or MIGRATE command. You can also change the way the prompts are issued by changing the default for OVERRIDE in the VMFINS DEFAULTS file. For more information on the OVERRIDE option, see [“VMFINS INSTALL Command” on page 425](#) and [“VMFINS MIGRATE Command” on page 433](#). For more information on changing the VMFINS DEFAULTS file, see [“Changing the VMFINS Command Defaults” on page 48](#).

## Changing the VMFINS Command Defaults

---

The VMFINS EXEC has established defaults for each command option, as described in [“VMFINS EXEC” on page 404](#). These defaults are obtained from the definitions in the VMFINS DEFAULTS file (a copy of which is supplied with VMSES/E).

VMFINS command defaults can be changed to meet requirements for your specific environment, either by supplying appropriate options and values directly as part of a VMFINS command, or via customized definitions within a VMFINS DEFAULTS file. For more information about supported statements, syntax, and the use of the VMFINS DEFAULTS file for configuration purposes, see [“The VMFINS DEFAULTS File” on page 139](#).

## Moving a Product from Test Mode to Production Mode

---

You may want to establish a test environment to test new products and new levels of products before you put them into production. Once you are satisfied with the product's performance, manually move the product from test mode to production mode to make it available to your users.

For more information on specific tasks that may need to be completed when you move a product from test mode to production mode, see the documentation for that product.

## Chapter 4. Installing Products with VMFINS

This chapter describes, in general terms, what happens when you use the VMFINS INSTALL command to install new products formatted for VMSES/E on your system or replace products that are already on your system. It is intended as an overview only. See the documentation for the product you are installing for specific installation instructions.

The topics covered in this chapter include:

- Adding new products to your system
- Replacing products on your system
- Changing the Parameter Driven Installation defaults
- Installing a product with the PPF operand (scenario)
- Using the VMFSIM EXEC during an installation

### Adding Products to Your System

When you use the VMFINS INSTALL command with the ADD option to install products,

- The VMSES/E product management files (for example, the *Memo-to-Users*, product parameter files, and PRODPART files) are loaded from the product tape to the Software Inventory disk.
- The system-level requisite and description Software Inventory tables are updated.
- If the product is in VMSES/E format:
  - The requisites for the product are checked.
    - If the requisites are not satisfied, a list of missing products is displayed; and you are asked if you want to continue. To see a sample message, see [“Missing Requisites” on page 24](#). (You should install any missing requisite products.)
    - If you want to continue, VMFINS loads the product to the system even though requisites are missing.
  - You are given an opportunity to use the current product-defined installation parameters or change them in the product parameter file. If you choose to change them, the Make Override Panel is displayed. On the Make Override Panel, you see the parameters that may be changed.
- The product is installed, using the information from the product parameter file.
- The Software Inventory is updated to show the files have been received and the product is installed.

For more information on VMFINS processing for products in different formats, see [Table 3 on page 11](#).

### Adding a Single Product

The easiest way to install a single product is to use the VMFINS INSTALL command with the PPF operand. This section provides an overview of the steps involved when you use the PPF operand and the ADD option.

**Note:** For specific instructions, see the documentation for the product you are installing.

Before you can install a product, you should enter:

```
vmfins install info (add
```

to find out which products are on the installation tape. As you saw in [Chapter 3, “Using the VMFINS EXEC,” on page 17](#), the INFO operand creates a list of the products on the installation tape. It also stores the *Memo-to-Users*, which you should read, on the Software Inventory disk.

Next, look in the VMFINS PRODLIST file (on your A-disk) that was created when you used the INFO operand. Find the product parameter file name (*ppfname*), the product identifier (*prodid*), and the component name (*compname*) for the product you want to install. These identifiers are explained in [“The VMFINS PRODLIST File”](#) on page 18.

When you have the *ppfname*, run the VMFINS INSTALL command with PPF operand and the PLAN option to check the product requisites and see what resources are required to install the product. To run the PLAN option, enter:

```
vmfins install ppf ppfname compname (add plan
```

You are asked if you want to create an override for the product you are installing. If you answer yes, you are asked if you want to use the default product installation parameters that have been supplied by IBM. You may change the installation parameters if the product is in VMSES/E format. The Make Override Panel provides, for your review, the specific product resource user IDs and minidisk addresses supplied by IBM, and you can enter new addresses and user IDs each time you install a copy of the product. You must always provide a name for the product parameter file installation defaults, however, even if you do not make any changes. This is how VMFINS identifies each copy of a product and maintains multiple copies of a product on one system.

When the PLAN processing is complete, a *prodid* PLANINFO file is stored on your A-disk.

Check the *prodid* PLANINFO file on your A-disk to see if you are missing any requisites and to see how much minidisk space is required to install the product you have identified. You must check the minidisk space yourself and allocate the required amount of available space.

Once you have reviewed the PLANINFO file, you can install the product using the VMFINS PPF operand and the installation parameters that were entered during PLAN processing. You can also change the installation parameters that were entered during PLAN processing, if you want to.

To install the product using the PPF operand, enter:

```
vmfins install ppf ppfname compname (add noplan nomemo
```

The options used in this command are:

### **add**

ADD puts a new copy of a product on the system.

### **noplan**

NOPLAN installs the product.

### **nomemo**

NOMEMO does not ask you if you want to print the *Memo-to-Users*.

For more information on the VMFINS options, see [“VMFINS INSTALL Command”](#) on page 425.

When install processing is complete, you should run the VMFINS BUILD command to build the product on the system and update the Software Inventory tables.

## Adding Several Products

You might want to use the LIST operand if you are installing several products at the same time. The basic steps are the same. You would:

- Enter the VMFINS INSTALL command with the INFO operand.
- Read the *Memo-to-Users* for the products you are installing.
- Edit the VMFINS PRODLIST file and comment out the products you do not want to install.
- Enter the VMFINS INSTALL command with the LIST operand and the ADD and PLAN options.
- Enter the new PPF override file names when you are asked to supply them during PLAN processing (optional).
- Review the *prodid* PLANINFO files that are created and stored on your A-disk.



- Review the VMFINS PRODLIST file to see where the new PPF override names have been entered to aid in further processing with the LIST operand.
- Enter the VMFINS INSTALL command with either:
  - The LIST operand (to install all of the products in the VMFINS PRODLIST file)
  - The PPF operand, a *ppfname*, and a *compname* (to install only one of the products in the VMFINS PRODLIST file).

If you use the PPF operand, you must enter the VMFINS INSTALL command with the PPF operand and the *ppfname* for each product in the VMFINS PRODLIST file that you want to install.

- Run the VMFINS BUILD command for each product to build the products and update the Software Inventory tables.

See [“Scenario 1: Installing a Product with the PPF Operand” on page 54](#) for an example of how to use VMFINS to install products.

## Replacing Products on Your System

When you enter the VMFINS INSTALL command with the REPLACE option:

- The VMSES/E product management files (for example, the *Memo-to-Users*, PRODPART file, and product parameter files) are loaded from the product tape to the Software Inventory disk.
- The system requisite and description tables are updated.
- If the product is in VMSES/E format, the requisites for the product are checked.
  - If the requisites are not satisfied, a list of missing products is displayed; and you are asked if you want to continue. To see a sample message, see [“Missing Requisites” on page 24](#).
  - If you want to continue, VMFINS loads the product to the system even though requisites are missing.
- The Software Inventory is checked to see which copies of the product are installed.
- You are asked which copy you want to replace.
- If the product is in VMSES/E format:
  - You are given an opportunity to change the product installation parameters.
 

The Make Override Panel is displayed, and the product installation parameters for the product you are replacing are provided. You can enter new addresses and user IDs if new minidisks or SFS directories have been added for the copy of the product you are installing.
- The files for the product being replaced are **erased**.
- The product is installed, using the information in the product parameter file.
- The Software Inventory is updated to show the files have been received and the product is installed.

## Replacing a Single Product

The easiest way to install a single product and replace an existing copy is to use the VMFINS INSTALL command with the PPF operand. This section provides an overview of the steps you complete when you replace a product using VMFINS.

**Note:** For specific instructions, see the documentation for the product you are installing.

Before you enter the VMFINS INSTALL command with the PPF operand, you need to complete a few simple steps. Begin by entering:

```
vmfins install info (replace
```

to get a list of the products you can replace. As you saw in Chapter 3, [“Using the VMFINS EXEC,” on page 17](#), the INFO operand creates a list of the products that are on the installation tape and also on the system. Details on using the INFO operand can be found in [“Which Products Can You Install?” on page 19](#).

## Installing Products

Check the *prodid* PLANINFO file on your A-disk to see if you are missing any requisites and to see how much minidisk space is required to install the product you have identified. You must check the minidisk space yourself and create the required amount of available space.

Once you have this information, run the VMFINS INSTALL command with PPF operand and the PLAN option to check the product requisites and see what resources are required to install the product. The PLAN option will tell you if you are missing any requisites. (You may also want to include the MEMO option to print the *Memo-to-Users*). To run the PLAN option, enter:

```
vmfins install ppf ppfname compname (replace plan memo
```

During the PLAN processing, the Make Override Panel is displayed, and the specific product resource user IDs and minidisk addresses for the copy of the product you are replacing are provided. You can enter new addresses and user IDs if new minidisks or SFS directories have been added for the copy of the product you are installing. You should always provide a name for the product parameter file installation defaults even if you do not make any changes. This is how VMFINS identifies each copy of a product and maintains multiple copies of a product on one system.

When the PLAN processing is complete, a *prodid* PLANINFO file and a *ppfname* ERASE file are stored on your A-disk. The *ppfname* ERASE file lists the files that will be erased when you replace this copy of the product.

Check the *prodid* PLANINFO file on your A-disk to see if you are missing any requisites and to see how much minidisk space is required to install the product you have identified. If you are going to use the NORESOURCE option when you install the product, you must check the minidisk space yourself and create the required amount of available space.

Once you have reviewed the PLANINFO file, you can install the product using the PPF operand. With the PPF operand, you can use the PPF installation default overrides that were entered during PLAN processing. You can also change the installation parameters that were entered during PLAN processing, if you want to.

To install the product using the PPF operand, enter:

```
vmfins install ppf ppfname compname (replace noplan nomemo
```

The options used in this command are:

### **replace**

REPLACE puts a new copy of a product on the system and replaces the existing copy.

### **noplan**

NOPLAN installs the product.

### **nomemo**

NOMEMO does not ask you if you want to print the *Memo-to-Users*.

For more information on the VMFINS options, see [“VMFINS INSTALL Command”](#) on page 425.

When install processing is complete, you should run the VMFINS BUILD command to build the product on the system and update the Software Inventory tables.

## Replacing Several Products

You may want to use the LIST operand if you are installing several products at once and replacing existing copies. The basic steps are:

- Enter the VMFINS INSTALL command with the INFO operand and the REPLACE option.
- Edit the VMFINS PRODLIST file and comment out the products you do not want to install.
- Enter the VMFINS INSTALL command with the LIST operand and the REPLACE, PLAN, and MEMO options.
- Answer yes (1) when you are asked if you want to create an override.

- Enter new product installation parameters for any new minidisk or SFS directory requirements when the Make Override Panel is displayed. When the Make Override Panel is displayed for each product, the product installation parameters for the copy of the product you are replacing are provided. You may only change parameters for minidisk or SFS directory requirements that have changed or been added since the previous copy was installed.
- Enter the new PPF override file names when you are asked to supply them during the PLAN processing.
- Review the *prodid* PLANINFO files and the *ppfname* ERASE files that were created and stored on your A-disk.
- Review the VMFINS PRODLIST file to see where the new PPF override names have been entered to aid in further processing with the LIST operand.
- Enter the VMFINS INSTALL command with either:
  - The LIST operand and the REPLACE option (to install all of the products in the VMFINS PRODLIST file)
  - The PPF operand, a *ppfname*, and the REPLACE option (to install only one of the products in the VMFINS PRODLIST file).

If you use the PPF operand, you must enter the VMFINS INSTALL command with the PPF operand and the *ppfname* for each product in the VMFINS PRODLIST file.

- Specify which products you want to replace when you are asked to do so.
- Run the VMFINS BUILD command to build the products and update the Software Inventory tables.

For a complete description of the command syntax, see “VMFINS INSTALL Command” on page 425.

## Changing the Product Installation Defaults

When you use VMFINS to install products in VMSES/E format, you have an opportunity to override the product installation defaults that have been provided for each product by IBM. The Make Override Panel, shown in Figure 31 on page 53, is displayed; and you can change the information on the panel.

The Make Override Panel shows you the default minidisks, user IDs, or Shared File System directories that will be used when the product is installed, unless, of course, you override them.

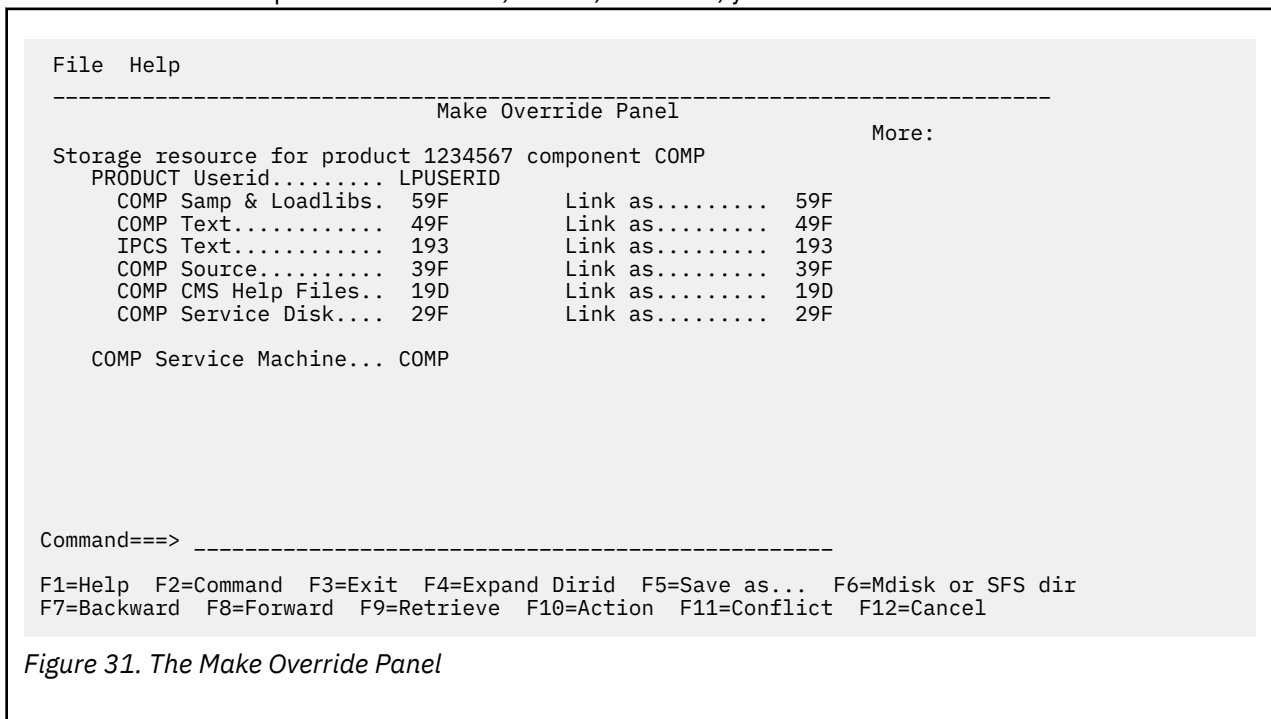


Figure 31. The Make Override Panel

For more information on using the Make Override Panel, see “Overriding Product Installation Defaults” on page 33.

## Scenario 1: Installing a Product with the PPF Operand

---

In this scenario we show you how to use the PPF operand with the VMFINS INSTALL command to install a single product. We also show you examples of the types of output we receive. The information you receive will depend on the products and system you are using.

### Note:

This scenario is only an example. When you install a product, you should always refer to the installation instructions for that product.

Before we can install a product using the PPF operand, we must have a usable form product parameter file for this particular copy of the product. The product parameter file can be:

- Shipped with the product by IBM
- Created using the VMFPPF command

We have a z/VM system installed and running. The product installation tape is mounted and attached as virtual address 181. We check [“Who Can Use VMFINS?”](#) on page 10 to make sure we have everything we need. Now we want to install the IBM XL C/C++ for z/VM Compiler product for the first time. We are going to use the NORESOURCE option.

### Step 1. Create the VMFINS PRODLIST File and Print Memo-to-Users

The first thing we need to do is find the information we need to install this product. We can use the INFO operand with the VMFINS INSTALL command to create a file containing a list of the products on the installation media.

With VMFINS, we can print the *Memo-to-Users* and run the INFO operand at the same time. To create the list of products, which will be stored in the VMFINS PRODLIST file, and print the *Memo-to-Users*, we enter:

```
vmfins install info (memo)
```

VMFINS reads the product installation media, creates the VMFINS PRODLIST file, and stores it on our A-disk. As you can see from the VMFINS CONSOLE file shown in [Figure 32 on page 54](#), we are asked if we want to print the *Memo-to-Users* for each product on the installation media, if one is available.

```
VMFUTL2767I Reading VMFINS DEFAULTS B for additional options
VMFINS2760I VMFINS processing started
VMFINS1909I VMFINS PRODLIST created on your A-disk
VMFINS2601R The following memos are available to be printed
              Enter the numbers of your choices separated by blanks
(0) Do not print any memos
(1) 5654A22C MEMO      D1 Description: AMERICAN ENGLISH
1
PRT FILE 0028 TO 5654A22C COPY 001 NOHOLD
VMFINS2760I VMFINS processing completed successfully
```

Figure 32. Printing the Memo-to-Users

We enter 1 to print the *Memo-to-Users* for CCXX. There is information in this document to help us during our installation. The *Memo-to-Users* prints on our system printer. [Figure 33 on page 55](#) shows the VMFINS PRODLIST file that was created when we entered this command.

```

VMFINS  PRODLIST A1  V 85  Trunc=85 Size=4 Line=0 Col=1 Alt=0
====>
* * * Top of File * * *
PPF  5654A22C CCXX      PROID 5654A22C%CCXX IBM XL C/C++ for z/VM Compiler
PPF  5654A22C CCXXSFS  PROID 5654A22C%CCXXSFS IBM XL C/C++ for z/VM Compiler in SFS
PPF  5654A22C CCXXK    PROID 5654A22C%CCXXK  IBM XL C/C++ for z/VM Compiler
PPF  5654A22C CCXXKSFS PROID 5654A22C%CCXXKSFS IBM XL C/C++ for z/VM Compiler in SFS
* * * End of File * * *

```

Figure 33. VMFINS PRODLIST File

## Step 2. Find the Product Identifiers for IBM XL C/C++ for z/VM Compiler

In this scenario, we are going to install the CCXX component. We look in the VMFINS PRODLIST file (shown in Figure 33 on page 55). Following the keyword PPF, we find 5654A22C. This is the file name of the product parameter file, *ppfname*, for CCXX. Following the PROID keyword, we find 5654A22C%CCXX. The characters before the %, 5654A22C, are the product ID, *prodid*, for CCXX. The characters after the %, CCXX, are the component name, *compname*, for CCXX.

## Step 3. Determine if You Have a Usable Form PPF

When we entered the VMFINS INSTALL command with the INFO operand, the product parameter files were saved on the D-disk. To determine if we have a usable form product parameter file for CCXX (file type is PPF), we enter:

```
filelist 5654a22c ppf d
```

As you can see in Figure 34 on page 55, we have a usable form product parameter file for CCXX.

```

5654A22C FILELIST A0  V 169  Trunc=169 Size=1 Line=1 Col=1 Alt=0
Cmd  Filename Filetype Fm Format Lrecl  Records  Blocks  Date  Time
     5654A22C PPF      D1 V          73      637      12  8/29/11  9:52:20

```

Figure 34. Finding the Usable Form Product Parameter File for CCXX

When the usable form product parameter file is shipped with the product by IBM, we can use the PPF operand to run the PLAN option. If the usable form product parameter file is not shipped with the product by IBM, we need to use the VMFPPF command to create one before we can run the VMFINS INSTALL command with the PPF operand. For more information on this command, see [“VMFPPF EXEC” on page 458](#).

## Step 4. Run the PLAN Option

To run PLAN, we enter:

```
vmfins install ppf 565aA22c ccxx (nomemo plan
```

We are asked if we want to create an override for this copy of CCXX. We answer no (0) to use the product installation parameters provided in the usable form product parameter file.

When PLAN processing is complete, 5654A22C PLANINFO file is stored on our A-disk. We need to review this file to see if there are any missing requisites, which minidisks will be affected, and how much space is required on each. The 5654A22C PLANINFO file is shown in [Figure 35 on page 56](#).

## Step 5. Review the 5654A22C PLANINFO File

We check the contents of the 5654A22C PLANINFO file to make sure all of the requisites are satisfied and we have the correct minidisks to install this product.

### 5654A22C PLANINFO File

Figure 35 on page 56 and Figure 36 on page 57 show the PLANINFO file created for 5654A22C (*prodid* 5654A22C) during the PLAN processing.

```

*****
****      VMFINS  INSTALL                USERID: 5654A22C      ****
*****
****      Date: 2022-06-28              Time: 16:25:07        ****
*****
VMFINS2195I VMFINS INSTALL PPF 5654A22C CCXX ( SYSTEM VM SIDISK 51D SIMODE
          D PLAN NORESOURCE LINK DFNAME USER DFTYPE DIRECT DFMODE *
          NOMEMO ADD ENV 5654A22C SETUP
*****
*          Requisite Planning Information                      *
*****
*          PPF: 5654A22C CCXX      PROPID: 5654A22C%CCXX      *
*          DATE: 06/28/22      TIME: 16:25:07      USERID: 5654A22C      *
*****
VMFREQ2805I Product :PPF 5654A22C CCXX :PROPID 5654A22C%CCXX has passed
          requisite checking
*****
*          Resource Allocation Planning Information            *
*****
*          PPF: 5654A22C CCXX      PROPID: 5654A22C%CCXX      *
*          DATE: 06/28/22      TIME: 16:25:07      USERID: 5654A22C      *
*****
Resource requirements for product 5654A22C component CCXX
*****
OWNER: 5654A22C
  TARGID:      191
  SIZE:        22500
  BLKSIZE:     4K
  FORMAT:      CMS
  RECOMPED:    NO
  PREFERRED:    NO
  SEPARATED:    NONE

  TARGID:      2C2
  SIZE:        900
  BLKSIZE:     4K
  FORMAT:      CMS
  RECOMPED:    NO
  PREFERRED:    NO
  SEPARATED:    NONE

  TARGID:      2D2
  SIZE:        81000
  BLKSIZE:     4K
  FORMAT:      CMS
  RECOMPED:    NO
  PREFERRED:    NO
  SEPARATED:    NONE

  TARGID:      2A6
  SIZE:        900
  BLKSIZE:     4K
  FORMAT:      CMS
  RECOMPED:    NO
  PREFERRED:    NO
  SEPARATED:    NONE

```

Figure 35. 5654A22C PLANINFO File Created by the PLAN Option (1 of 2)

```

TARGID:      2A2
SIZE:        900
BLKSIZE:     4K
FORMAT:      CMS
RECOMPED:    NO
PREFERRED:   NO
SEPARATED:   NONE

TARGID:      29E
SIZE:        81000
BLKSIZE:     4K
FORMAT:      CMS
RECOMPED:    NO
PREFERRED:   NO
SEPARATED:   NONE

TARGID:      2B2
SIZE:        45000
BLKSIZE:     4K
FORMAT:      CMS
RECOMPED:    NO
PREFERRED:   NO
SEPARATED:   NONE

REPLACE USER: 5654A22C 4
USER 5654A22C XXXXX 256M 2G EG
ACCOUNT xxxxx
IPL CMS
MACHINE ESA
CONSOLE 009 3215 T
SPOOL 00C 2540 READER *
SPOOL 00D 2540 PUNCH A
SPOOL 00E 1403 A
LINK MAINT 190 190 RR
LINK MAINT 19D 19D RR
LINK MAINT 19E 19E RR
LINK MAINT 51D 51D MR
LINK MAINT 5E5 5E5 RR

```

Figure 36. 5654A22C PLANINFO File Created by the PLAN Option (2 of 2)

We have checked the 5654A22C PLANINFO file and see we are not missing any requisites. VMFREQ2805I Product :PPF 5654A22C CCXX :PRODID 5654A22C%CCXX has passed requisite checking **(2)** says there are no missing requisites to report.

If we were missing requisites, we could still install the product. We would, however, receive messages during the installation telling us which products we were missing and asking if we wanted to continue. See “Missing Requisites” on page 24 for more information on installing a product when a requisite is missing.

In 5654A22C PLANINFO, we can also see where:

- The 5654A22C product parameter file was created **(1)**.
- The target minidisks belong to the 5654A22C user ID **(3)**. You may have to update the CP directory with these minidisks. If you do, make sure you put the directory online after you update it.
- A server machine is required by CCXX **(4)**.

You must allocate and generate the resources before you begin the installation. You must also add the server machine information to the CP directory.

## Step 6. Install CCXX

Before we enter the install command, we make sure our CP directory is updated and online. Then, to install CCXX, we enter the PPF operand and the *ppfname* (5654A22C) in the following command:

```
vmfins install ppf 5654a22c ccxx (add nolink nomemo
```

We have entered the ADD option, even though it is the default, for illustrative purposes.

When VMFINS INSTALL processing begins, we see the following message:

```
VMFINS2760I VMFINS processing started
```

We are prompted for additional information during the installation processing. For example, we are asked if we want to create an override. We answer no (0). We could have answered yes (1), however, if we wanted to change any of the installation parameters that were saved during PLAN processing.

For a complete list of the messages issued by VMFINS, we can check the \$VMFINS \$MSGLOG, which is created during the installation processing and stored on our A-disk. For a complete listing of the commands we enter, the responses we made to system prompts, and the messages issued during install processing, we can check the VMFINS CONSOLE file that is spooled to our reader when processing is complete.

[Figure 38 on page 60](#) shows the console listing that was created when we installed this copy of CCXX.

When processing is complete, we see the message:

```
VMFINS2760I VMFINS processing completed successfully
```

### Step 7. Review the Output Files

Now that we have completed our installation, we review the \$VMFINS \$MSGLOG and VMFINS CONSOLE files to see if we need to perform additional tasks.

#### The \$VMFINS \$MSGLOG File

The \$VMFINS \$MSGLOG is created by VMFINS during the installation. The status of the installation can be found in this file, as well as information for determining where an error may have occurred. The \$VMFINS \$MSGLOG that was created during our example installation is shown in [Figure 37 on page 59](#). [Table 10 on page 138](#) explains the different types of messages in a message log.



```

*****
****          VMFINS  INSTALL                      USERID: 5654A22C          ****
*****
****          Date: 2022-09-12                    Time: 16:30:34                    ****
*****
ST:VMFINS2195I VMFINS INSTALL PPF 5654A22C CCXX ( SYSTEM VM SIDISK 51D SIMODE
ST:          D NOPLAN NORESOURCE NOLINK DFNAME USER DFTYPE DIRECT DFMODE *
ST:          NOMEMO ADD SETUP                      1
ST:VMFINS2760I VMFINS processing started
ST:VMFINS2603I Processing product :PPF 5654A22C CCXX :PRODID 5654A22C%CCXX
ST:VMFREQ2805I Product :PPF 5654A22C CCXX :PRODID 5654A22C%CCXX has passed
ST:          requisite checking
ST:VMFINT2603I Installing product :PPF 5654A22C CCXX :PRODID 5654A22C%CCXX
ST:VMFSET2760I VMFSETUP processing started for 5654A22C CCXX
ST:VMFUTL2205I Minidisk|Directory Assignments: 2
ST:          String      Mode  Stat  Vdev  Label  (OwnerID Odev : Cyl/%Used)
ST:          -or-          SFS Directory Name
ST:VMFUTL2205I LOCALSAM  E     R/W  2C2  CCX2C2 (5654A22C 02C2 : 5/01)
ST:VMFUTL2205I APPLY     F     R/W  2A6  CCX2A6 (5654A22C 02A6 : 5/01)
ST:VMFUTL2205I          G     R/W  2A2  CCX2A2 (5654A22C 02A2 : 5/01)
ST:VMFUTL2205I DELTA    H     R/W  2D2  CCX2D2 (5654A22C 02D2 : 450/00)
ST:VMFUTL2205I BUILD0   I     R/W  29E  CCX29E (5654A22C 029E : 450/00)
ST:VMFUTL2205I BASE1    J     R/W  2B2  CCX2B2 (5654A22C 02B2 : 250/00)
ST:VMFUTL2205I -----  A     R/W  191  CCX191 (5654A22C 0191 : 125/01)
ST:VMFUTL2205I -----  B     R/O  5E5  MNT5E5 (MAINT730 05E5 : 18/48)
ST:VMFUTL2205I -----  D     R/W  51D  MNT51D (MAINT730 051D : 26/36)
ST:VMFUTL2205I -----  S     R/O  190  MNT190 (MAINT 0190 : 207/64)
ST:VMFUTL2205I -----  Y/S   R/O  19E  MNT19E (MAINT 019E : 500/38)
ST:VMFSET2760I VMFSETUP processing completed successfully
ST:VMFREC2760I VMFREC processing started
ST:VMFREC1852I Volume 1 of 1 of INS ENVELOPE 1100
ST:VMFREC1851I (1 of 8) VMFRCAXL processing AXLIST
ST:VMFRCX2159I Loading 0 part(s) to DELTA 2D2 (H)
ST:VMFREC1851I (2 of 8) VMFRCPTF processing PARTLST
ST:VMFRCP2159I Loading 0 part(s) to DELTA 2D2 (H)
ST:VMFREC1851I (3 of 8) VMFRCOM processing DELTA
ST:VMFRC2159I Loading 0 part(s) to DELTA 2D2 (H)
ST:VMFREC1851I (4 of 8) VMFRCALL processing APPLY
ST:VMFRC2159I Loading part(s) to APPLY 2A6 (F)
ST:VMFRC2159I Loaded 1 part(s) to APPLY 2A6 (F)
ST:VMFREC1851I (5 of 8) VMFRCALL processing BASE
ST:VMFRC2159I Loading part(s) to BASE1 2B2 (J)
ST:VMFRC2159I Loaded 66 part(s) to BASE1 2B2 (J)
ST:VMFREC1851I (6 of 8) VMFRCALL processing SAMPLE
ST:VMFRC2159I Loading part(s) to LOCALSAM 2C2 (E)
ST:VMFRC2159I Loaded 180 part(s) to LOCALSAM 2C2 (E)
ST:VMFREC1851I (7 of 8) VMFRCALL processing BUILD
ST:VMFRC2159I Loading part(s) to BUILD0 29E (I)
ST:VMFRC2159I Loaded 24 part(s) to BUILD0 29E (I)
ST:VMFREC1851I (8 of 8) VMFRCALL processing BUILDENG
ST:VMFRC2159I Loading part(s) to BUILD0 29E (I)
ST:VMFRC2159I Loaded 6 part(s) to BUILD0 29E (I)
ST:VMFREC2760I VMFREC processing completed successfully
ST:VMFINT2603I Product installed
ST:VMFINS2760I VMFINS processing completed successfully

```

Figure 37. \$VMFINS \$MSGLOG Created during Installation Processing

We check the \$VMFINS \$MSGLOG and see:

- The complete VMFINS command that was processed (1)
- The required minidisks are accessed (2).

## The VMFINS CONSOLE File

The following console file was created during VMFINS INSTALL processing and spooled to our reader. VMFINS spools the console so the information can be saved.

If you encounter an error during processing, you can use the VMFINS CONSOLE file and the \$VMFINS \$MSGLOG file to find where the error occurred. The console file contains a record of the entries we made, the system prompts, and the messages that were issued.

```

vmfins install ppf 5654A22C CCXX ( add nolink nomemo
VMFUTL2767I Reading VMFINS DEFAULTS B for additional options
VMFINS2760I VMFINS processing started
VMFINS2601R Do you want to create an override for :PPF 5654A22C CCXX :PRODID
5654A22C%CCXX? 1
Enter 0 (No), 1 (Yes) or 2 (Exit)
0
VMFINS2603I Processing product :PPF 5654A22C CCXX :PRODID 5654A22C%CCXX
VMFREQ2805I Product :PPF 5654A22C CCXX :PRODID 5654A22C%CCXX has passed
requisite checking 2
VMFINT2603I Installing product :PPF 5654A22C CCXX :PRODID 5654A22C%CCXX
VMFSET2760I VMFSETUP processing started for 5654A22C CCXX
VMFUTL2205I Minidisk|Directory Assignments:
String Mode Stat Vdev Label (OwnerID Odev : Cyl/%Used)
-or- SFS Directory Name
VMFUTL2205I LOCALSAM E R/W 2C2 CCX2C2 (5654A22C 02C2 : 5/01)
VMFUTL2205I APPLY F R/W 2A6 CCX2A6 (5654A22C 02A6 : 5/01)
VMFUTL2205I G R/W 2A2 CCX2A2 (5654A22C 02A2 : 5/01)
VMFUTL2205I DELTA H R/W 2D2 CCX2D2 (5654A22C 02D2 : 450/00)
VMFUTL2205I BUILD0 I R/W 29E CCX29E (5654A22C 029E : 450/00)
VMFUTL2205I BASE1 J R/W 2B2 CCX2B2 (5654A22C 02B2 : 250/00)
VMFUTL2205I ----- A R/W 191 CCX191 (5654A22C 0191 : 125/01)
VMFUTL2205I ----- B R/O 5E5 MNT5E5 (MAINT730 05E5 : 18/48)
VMFUTL2205I ----- D R/W 51D MNT51D (MAINT730 051D : 26/36)
VMFUTL2205I ----- S R/O 190 MNT190 (MAINT 0190 : 207/64)
VMFUTL2205I ----- Y/S R/O 19E MNT19E (MAINT 019E : 500/38)
VMFSET2760I VMFSETUP processing completed successfully
VMFREC2760I VMFREC processing started
VMFREC1852I Volume 1 of 1 of INS ENVELOPE 1100
VMFREC1851I (1 of 8) VMFRCAXL processing AXLIST
VMFRCX2159I Loading 0 part(s) to DELTA 2D2 (H)
VMFREC1851I (2 of 8) VMFRCPTF processing PARTLST
VMFRCP2159I Loading 0 part(s) to DELTA 2D2 (H)
VMFREC1851I (3 of 8) VMFRCCOM processing DELTA
VMFRCC2159I Loading 0 part(s) to DELTA 2D2 (H)
VMFREC1851I (4 of 8) VMFRCALL processing APPLY
VMFRCA2159I Loading part(s) to APPLY 2A6 (F)
VMFRCA2159I Loaded 1 part(s) to APPLY 2A6 (F)
VMFREC1851I (5 of 8) VMFRCALL processing BASE
VMFRCA2159I Loading part(s) to BASE1 2B2 (J)
VMFRCA2159I Loaded 66 part(s) to BASE1 2B2 (J)
VMFREC1851I (6 of 8) VMFRCALL processing SAMPLE
VMFRCA2159I Loading part(s) to LOCALSAM 2C2 (E)
VMFRCA2159I Loaded 180 part(s) to LOCALSAM 2C2 (E)
VMFREC1851I (7 of 8) VMFRCALL processing BUILD
VMFRCA2159I Loading part(s) to BUILD0 29E (I)
VMFRCA2159I Loaded 24 part(s) to BUILD0 29E (I)
VMFREC1851I (8 of 8) VMFRCALL processing BUILDENG
VMFRCA2159I Loading part(s) to BUILD0 29E (I)
VMFRCA2159I Loaded 6 part(s) to BUILD0 29E (I)
VMFREC2760I VMFREC processing completed successfully
VMFINT2603I Product installed
VMFINS2760I VMFINS processing completed successfully

```

Figure 38. The VMFINS CONSOLE File Created during Installation Processing

We look at the VMFINS CONSOLE file, shown in [Figure 38](#) on page 60, and see:

- We are asked if we want to create an override for the 5654A22C product parameter file override **(1)**.
- All of the requisites have been met for this product **(2)**.

## Summary

We have just installed CCXX using the VMFINS INSTALL command with the PPF operand. After you complete an installation, you need to perform two additional tasks.

Second, run the VMFINS BUILD command after each product installation even if there is nothing to build. Running the VMFINS BUILD command:

- Updates the build status table to show the product has been built.

- Runs the *Bponum* EXEC, which is a product-specific build exec that may be shipped with the product. (*ponum* is the product order number.)
- Runs the *Vponum* EXEC, which is a product-specific verification exec that may also be shipped with the product.

See [Chapter 6, “Building Products with VMFINS,” on page 77](#) for more information on using the VMFINS BUILD command.

## Using the VMFSIM EXEC during an Installation

---

The VMFSIM EXEC:

- Maintains system level inventories of all products installed on the system.
- Maintains service level inventories of all maintenance installed to products supported by VMSES/E.
- Allows queries of both the system-level and service-level Software Inventories to identify products and maintenance installed on the system.

If you have questions on the status or levels of products during an installation, you can use the VMFSIM QUERY command to access the information in the Software Inventory. See [“Querying the Software Inventory” on page 177](#) for examples of how to use the VMFSIM QUERY command. You can also use the VMFINFO EXEC to query the Software Inventory tables. For example, you can use VMFINFO to get a list of the product parameter files for the products on your system. VMFINFO provides easy-to-use panels and a variety of predefined queries for both product and service information. For more information on the VMFINFO EXEC, see [Chapter 17, “Using the VMFINFO Panels,” on page 199](#).

For more information on the Software Inventory, see [Chapter 15, “Introduction to the Software Inventory,” on page 163](#) and [Chapter 22, “Software Inventory Syntax,” on page 659](#).



## Chapter 5. Migrating Products with VMFINS

This chapter tells you, in general terms, what happens when you use the VMFINS MIGRATE command to migrate products in VMSES/E format.

The topics covered in this chapter include:

- Adding a copy of a product to your system while preserving the tailored files from an existing copy
- Replacing products on your system while preserving the tailorings from older versions of the products
- Changing the product installation defaults
- Retailoring your product files and Shared File System (SFS) attributes and aliases
- Using the VMFSIM commands during a migration.

After you have used VMFINS to install a product, you can use VMFINS MIGRATE to put additional copies of the product on your system and preserve the tailorable files that were modified for special features, SFS file authorizations, and SFS aliases so you can use them with the new copies of the product.

With the VMFINS MIGRATE command, you can also add new products to your system and replace products that are already on your system, while preserving the tailorings for the current copy on your system. VMFINS lets you work with your tailored files, files you have modified for your system, and gives you the chance to transfer the tailorings to a new copy of a product.

The VMFINS MIGRATE command uses a Shared File System file pool directory (*filepoolid:userid.VMFINS*) for its migration save area. The default SFS name (shipped with VMSES/E) is *VMPSFS:userid.VMFINS* (*userid* is the user ID of the person running VMFINS). You can change the Shared File System file pool using the FILEPOOL option on the VMFINS MIGRATE command. Or, you can redefine the file pool by changing the value specified in the VMFINS DEFAULTS file. For more information, see [“Changing the Shared File System Directory File Pool Name” on page 47](#).

Also, the product tape must be in VMSES/E or INSTFPP format. For more information on tape formats and VMFINS processing for those tape formats, see [“What Tape Formats Does VMFINS Support?” on page 10](#).

If you have no tailored files or you do not want to keep the tailored files, use the VMFINS INSTALL command to install your product. See Chapter 4, [“Installing Products with VMFINS,” on page 49](#) for information on how to install a product.

### Adding a New Release of a Product to Your System

When you use the VMFINS MIGRATE command to migrate VMSES/E-formatted products, VMFINS:

- Loads the first tape file from the product tape
- Checks the requisites
  - If the requisites are not satisfied, a list of the missing products is displayed; and you are asked if you want to continue
  - Loads the product to the system even though requisites may be missing
- Queries the Software Inventory for all previously installed or migrated copies of the product
- Asks you which copy of the product tailorings you want to use, if there is more than one copy of the product on your system
- Gives you the opportunity to change installation parameters
  - The Make Override Panel is displayed. On the Make Override Panel, you see the parameters that may be changed.
  - When you use the VMFINS MIGRATE command with the ADD option, you must change the default product installation parameters to install additional copies of the product.
- Calculates the space requirements for the new product

## Migrating Products

- Preserves any tailored files, SFS file authorizations, and SFS aliases and determines reach-ahead service
  - Lets you customize, with a split screen XEDIT session, any tailorable files that have changed since the previous version or release of that product. You are shown the new file and the current file on your system, so you can compare and update the new file with the data from your file.
  - Lets you see other tailored files on the system you have changed, with a view screen session, so you can save them. These files may no longer be needed by the new version or release of the product.
  - Copies the tailored files that have not been changed by the product to the new product minidisk or SFS directory and logs message VMFRES2818I in the \$VMFINS \$MSGLOG file for each preserved file.
- Updates the Software Inventory to show the files were received and the product was migrated

## Adding a Single Product

The easiest way to migrate a single product is to use the VMFINS MIGRATE command with the PPF operand. This section provides an overview of the steps involved in migrating a product using the PPF operand.

**Note:** For specific instructions, see the install documentation for the product you are migrating.

Before you enter the VMFINS MIGRATE command with the PPF operand, you need to complete a few simple steps. Begin by entering:

```
vmfins migrate info (add
```

As you saw in [Chapter 3, “Using the VMFINS EXEC,”](#) on page 17, the INFO operand creates a list of the products that are on the installation tape and also on the system. This list is stored in the VMFINS PRODLIST file on your A-disk. Details on using the INFO operand can be found in [“Which Products Can You Install?”](#) on page 19.

Look in the VMFINS PRODLIST file to find the product identifier (*prodid*), product parameter file name (*ppfname*), and the component name (*compname*).

Next, use the PLAN option to see how much minidisk space is required to migrate the product. The PLAN option also tells you whether you are missing any requisites and whether you need to reapply any reach-ahead service.

To run the PLAN option, enter:

```
vmfins migrate ppf ppfname compname (plan add
```

During plan processing:

- You are asked to select the copy of the product from which you want to migrate.
- You are also asked to select a product parameter file (\$PPF) to use to create the PPF override for the new copy of the product to which you are migrating.

When you add a new copy of a product, you should change the installation parameters so you do not overlay the existing copy of the product with the new copy of the product. The Make Override Panel provides the specific product resource user IDs, minidisks addresses, and SFS directory names used for the current copy of the product. You must enter new user IDs, addresses, and SFS directory names. You are asked to provide a name for the new PPF override (the usable form product parameter file). VMFINS identifies each copy of a product and maintains multiple copies of a product on one system with the usable form product parameter files.

- A *prodid* PLANINFO file is created for the product listed in the VMFINS PRODLIST file.

Check the *prodid* PLANINFO file on your A-disk to see if you are missing any requisites, if you need to reapply reach-ahead service, and to see how much minidisk space is required to migrate the product.

After you have reviewed the PLANINFO file, you can migrate the product using the VMFINS MIGRATE command with the PPF operand. With the PPF operand, you can use the PPF override information that was entered during PLAN processing. You can also change the PPF override information, if you want to.

To migrate a single product using the PPF operand, enter:

```
vmfins migrate ppf ppfname compname (add noplan nomemo
```

The options are:

**add**

ADD puts a new copy of a product on the system.

**noplan**

NOPLAN migrates the product.

**nomemo**

NOMEMO does not ask you if you want to print the *Memo-to-Users*.

You may need to tailor some of the tailored files for the product near the end of the migration processing. SFS file authorizations and aliases are restored automatically.

When migrate processing is complete, you should run the VMFINS BUILD command to build the product on the system and update the Software Inventory tables.

See [Chapter 6, “Building Products with VMFINS,”](#) on page 77 for more information on using the VMFINS BUILD command.

## Adding Several Products

You may want to use the LIST operand if you are migrating several products at the same time. The basic steps are:

- Enter the VMFINS MIGRATE command with the INFO operand.
- Edit the VMFINS PRODLIST file and comment out the products you do not want to migrate.
- Enter the VMFINS MIGRATE command with the LIST operand and the ADD, PLAN, and MEMO options.
- Enter the new PPF override file names when you are asked to supply them during the plan processing.
- Review the *prodid* PLANINFO files that are created and stored on your A-disk.
- Review the VMFINS PRODLIST file to see where the new PPF override names have been entered to aid in further processing with the LIST operand.
- Enter the VMFINS MIGRATE command with either:
  - The LIST operand (to migrate all the products in the VMFINS PRODLIST file).
  - The PPF operand, a *ppfname*, and a *compname* (to migrate only one of the products in the VMFINS PRODLIST file). When you use the PPF operand, you must enter the VMFINS MIGRATE command for each product in the VMFINS PRODLIST file you want to migrate.
- Retain the product files with the new information from the product default files, and restore SFS file authorizations and aliases.
- Reaccess the minidisk with the CP Directory.
- Reapply any reach-ahead service that was identified during the VMFINS processing.
- Run the VMFINS BUILD command to build the products and update the Software Inventory tables.

For specific instructions, see the documentation for the products you are migrating.

## Replacing Products on Your System

When you enter VMFINS MIGRATE command with the REPLACE option, VMFINS:

- Loads the first tape file from the product tape.
- Checks the requisites.

- If the requisites are not satisfied, a list of the missing products is displayed; and you are asked if you want to continue.
- VMFINS will load the product to the system, even though requisites may be missing.
- Queries the Software Inventory for all previously installed or migrated copies of your product.
- Asks which copy of the product tailorings you want to use, if there is more than one copy of the product on your system.
- Gives you a chance to override the current product parameter file.  
**Note:** You do not want to change these settings because you are replacing the new version or release of the product, but you may want to review them.
- Gives you the opportunity to review product installation parameters.
  - The Make Override Panel is displayed. On the Make Override Panel, you see the parameters that will be used for this migration. Parameters that may **not** be changed are preceded by a colon (:).
  - The product installation parameters should be the same as the ones for the existing copy of the product you are replacing, so the new copy will overlay the old copy.
  - If you realize you did not want to migrate and replace your product, you can exit from the migration while in the Make Override Panel.
- Calculates the space requirements for the new product.
- **Erases** the files for the copy of the product you are replacing.
- Loads the files for the copy of the product you are installing.
- Copies over any tailored files, SFS file authorizations and SFS aliases, and determines any reach-ahead service.
  - Lets you customize any of the product tailored files that have changed with a split screen XEDIT session. You are shown the new file and the current file on your system so you can compare and update the new file with the data in your file.
  - Lets you see, with a view screen session, the other tailored files on your system you have changed so you can save them. These files may no longer be needed by the new version or release of the product.
  - Copies the tailored files that have not been changed by the product to the new product minidisk or SFS directory and logs message VMFRES2818I in the \$VMFINS \$MSGLOG file for each file copied over.
- Updates the Software Inventory to show the files were received and the product was migrated.

## Replacing a Product

The easiest way to migrate a single product and replace the existing copy is to use the VMFINS MIGRATE command with the PPF operand and the REPLACE option. This section provides an overview of the steps involved when you migrate a product and replace the existing copy.

**Note:** For specific instructions, see the install documentation for the product you are migrating.

Before you can enter the VMFINS MIGRATE command with the PPF operand and the REPLACE option, you need to complete a few simple steps. To begin, you enter:

```
vmfins migrate info (replace
```

As you saw in Chapter 3, “Using the VMFINS EXEC,” on page 17, the INFO operand creates a list of the products on the system that can be replaced by a product on the installation tape. Details on using the INFO operand can be found on page “Which Products Can You Install?” on page 19.

Next, look in the VMFINS PRODLIST file on your A-disk. The VMFINS PRODLIST file was created when you used the INFO operand. Find the *ppfname*, *prodid*, and *compname* for the product you want to migrate.

After you have found the *prodid*, *ppfname*, and *compname*, run the VMFINS MIGRATE command with the PLAN option to check the product requisites and see what resources are required to migrate the product.



The PLAN option will tell you if you are missing any requisites. You may also want to include the MEMO option to print the *Memo-to-Users*. To run the PLAN option, enter:

```
vmfins migrate ppf ppfname compname (replace plan memo
```

During the plan processing, you are asked to select the copy of the product (and its tailorings) you are replacing, if there is more than one copy of the product on your system. You are asked if you want to create an override for the copy you are replacing. You should answer no (0), so you can overlay the current copy of the product.

When plan processing is complete, a *prodid* PLANINFO file is stored on your A-disk.

Check the *prodid* PLANINFO file on your A-disk to see if you are missing any requisites, whether there is any reach-ahead service, and to see how much minidisk space is required to migrate the product you have identified. You must check the minidisk space yourself and create the required amount of available space.

After you have reviewed the PLANINFO file, you can migrate the product using the PPF operand. With the PPF operand, you can use the PPF installation overrides that were entered during PLAN processing. You can also change the installation overrides, if you want to.

To migrate the product using the PPF operand, enter:

```
vmfins migrate ppf ppfname compname (replace noplan nomemo
```

The options used in this command are:

#### **replace**

REPLACE puts a new copy of a product on the system by replacing the existing copy and keeps the current product tailorings.

#### **noplan**

NOPLAN migrates the product.

#### **nomemo**

NOMEMO does not ask you if you want to print the *Memo-to-Users*.

When migrate processing is complete, you should run the VMFINS BUILD command to build the product on the system and update the Software Inventory tables.

See [Chapter 6, “Building Products with VMFINS,”](#) on page 77 for more information on using the VMFINS BUILD command.

## Replacing Several Products

You may want to use the LIST operand, if you are migrating several products at one time and replacing existing copies and you want to keep the product tailorings for the current copies on your system. The basic steps are:

- Enter the VMFINS MIGRATE command with the INFO operand and the REPLACE option.
- Edit the VMFINS PRODLIST file and comment out the products you do not want to migrate.
- Enter the VMFINS MIGRATE command with the LIST operand and the REPLACE, PLAN, and MEMO options.
- Enter the new PPF override file names when you are asked to supply them during the plan processing.
- Review the *prodid* PLANINFO files that were created and stored on your A-disk.
- Review the VMFINS PRODLIST file to see where the new PPF override names have been entered.
- Enter the VMFINS MIGRATE command with either:
  - The LIST operand (to migrate all of the products in the VMFINS PRODLIST file)
  - The PPF operand, a *ppfname*, and a *compname* (to migrate only one of the products in the VMFINS PRODLIST file). When you use the PPF operand, you must enter the VMFINS MIGRATE command for each product in the VMFINS PRODLIST file that you want to replace.
- Specify which products you want to replace when you are asked to do so.

- Retain the product files with the new information from the product default files and restore the SFS file authorizations and aliases.
- Reaccess the minidisk where the CP Directory resides.
- Reapply any reach-ahead service that was identified during the VMFINS processing.
- Run the VMFINS BUILD command to build the products and update the Software Inventory tables.

**Note:** For specific instructions, see the documentation for the products you are migrating. For a complete description of the VMFINS MIGRATE command syntax, see [“VMFINS MIGRATE Command”](#) on page 433.

## Changing the Current Product Installation Settings

When you use VMFINS to migrate products in VMSES/E format, you have an opportunity to override the current installation settings that have been provided for each product. The Make Override Panel, shown in [Figure 39](#) on page 68 is displayed, and you can change the information on the panel.

If you are adding a new copy, the current settings displayed on the Make Override Panel are the defaults for the product. You may change these settings. If you are replacing a product, the settings displayed are the current ones for the product, and you cannot change them.

The Make Override Panel shows you the minidisks, user IDs, or Shared File System directories that will be used when the product is migrated, unless you override them.

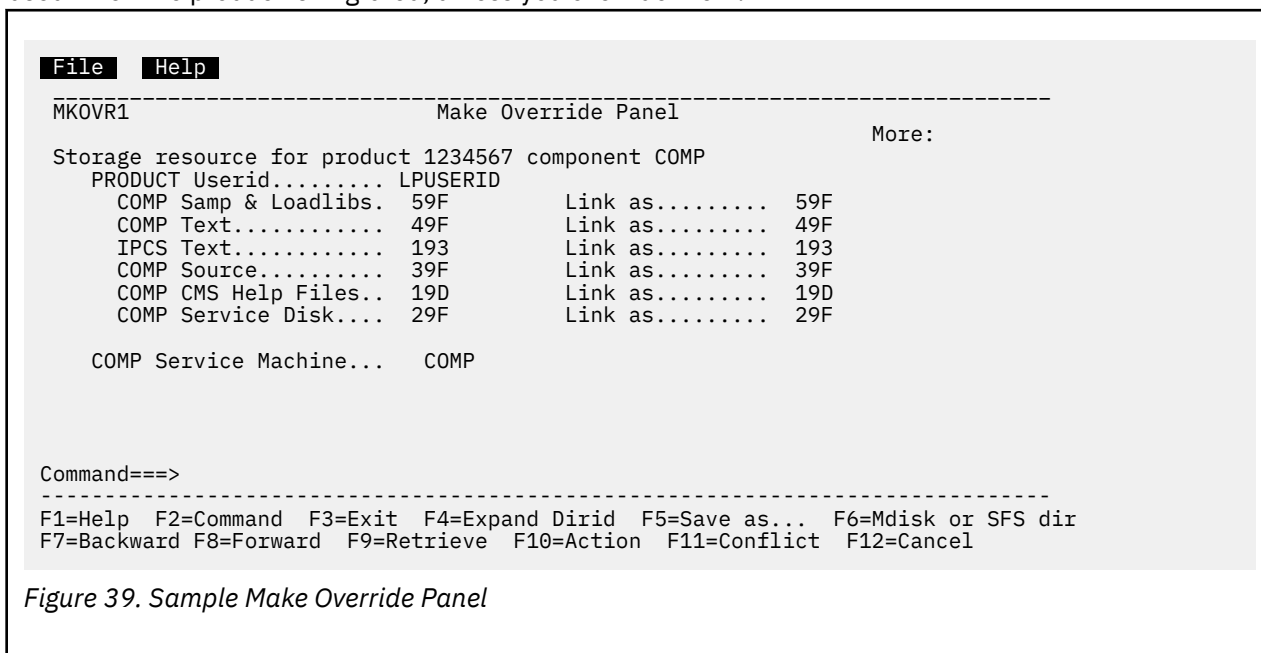


Figure 39. Sample Make Override Panel

For more information on using the Make Override Panel, see [“Overriding Product Installation Defaults”](#) on page 33.

## Retailing Your Product Files

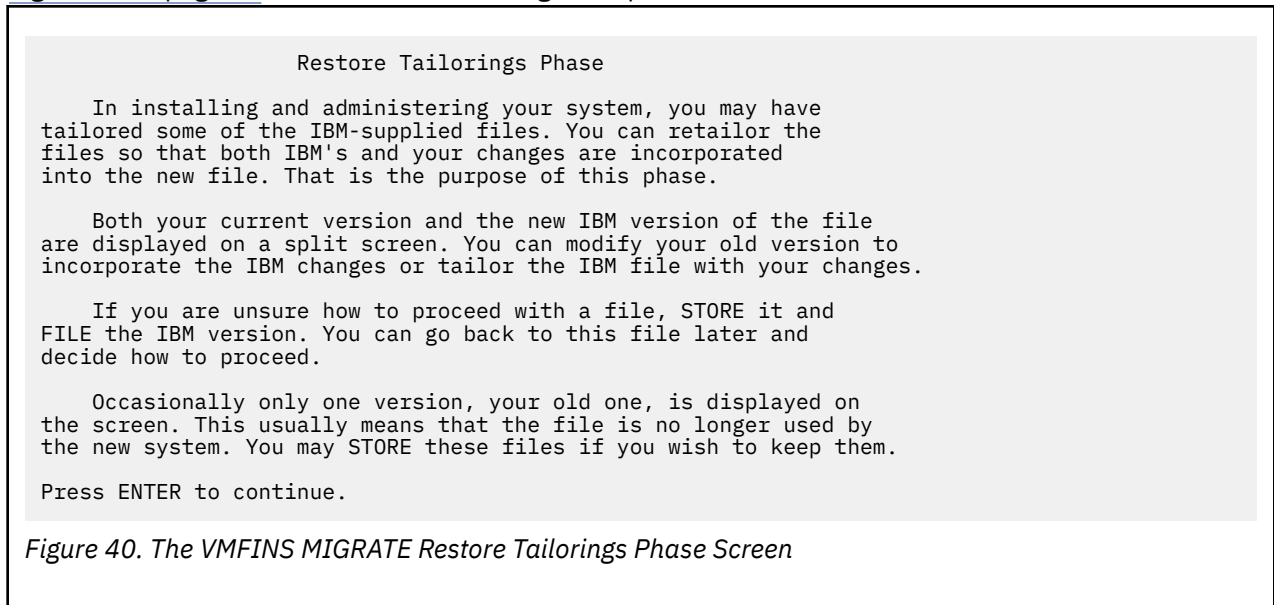
First, you need to understand how to identify a tailorable file. A **tailorable file** is any source-level product file that requires your input to work correctly. After you have modified a tailorable file, it is considered **tailored**. An example of a source-level file is your PROFILE EXEC. You can add information to these files in several ways by using:

- Data entry panels
- File editing
- Template files

The VMFINS MIGRATE process identifies only tailorable files found in the :PARTS section of the PRODPART file. For more information on the PRODPART file, see [“The Product Parts \(PRODPART\) File” on page 660](#).

If you tailored files for a product and they were not changed by the product for this new version or release, the files are automatically copied to the new product minidisk or SFS directory; and message VMFRES2818I is logged in the \$VMFINS \$MSGLOG file to tell you this was done.

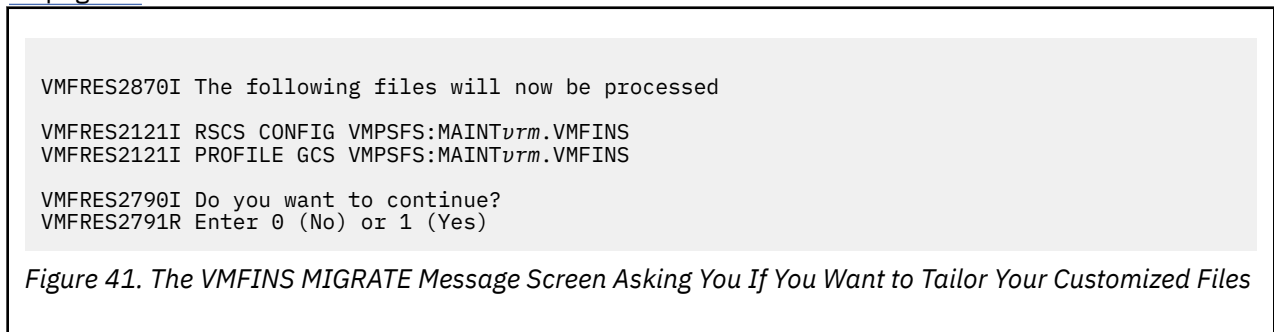
If any of the tailorable files were changed by you or the product, you may see a screen, like the one in [Figure 40 on page 69](#), near the end of the migration process.



*Figure 40. The VMFINS MIGRATE Restore Tailorings Phase Screen*

This screen tells you there are one or more tailored files on the system for your product.

You should read this initial screen and press the Enter key. Now you see a screen like the one in [Figure 41 on page 69](#).



*Figure 41. The VMFINS MIGRATE Message Screen Asking You If You Want to Tailor Your Customized Files*

This screen lists the tailored files on your system that are associated with your copy of the product and asks if you want to continue. We recommend you choose yes (1).

If you choose no (0), the files will be left unchanged in the migration save area (usually the VMPSFS:userid.VMFINS directory with file mode C); and the default product files are installed and used with this copy of the product. If you say no (0), but you realize afterward that you wanted to tailor your files, see [“Step 5. Manually Complete Your Migration \(Optional\)” on page 442](#) for information on how to recover your tailored files from the migration save area.

If you choose yes (1), you are automatically put into either a split screen XEDIT session (see [“Split-Screen XEDIT Session” on page 70](#)) or a view screen session (see [“View-Screen Session” on page 74](#)). The files will be processed in the order they appear in the list shown in [Figure 41 on page 69](#).

After VMFINS MIGRATE has processed all the tailored files, the migration process is complete.

## Split-Screen XEDIT Session

You are placed in a split-screen XEDIT session because you have tailored the file and the file provided with the product for this version or release has also changed. See [Figure 42 on page 70](#) for an example of a session.

You can get online help for the split-screen XEDIT session by entering:

```

help vm ses vmfsplit

RSCS CONFIG      (IBM Version - Unmodified)  Trunc= 80 Size= 379
00000 * * * Top of File * * *
00001 *****
00002 *
00003 * Following are the "rules" for defining an RSCS Configuration File:
00004 *
00005 * 1. The LOCAL statement, if included, MUST be the FIRST valid
00006 *      (non-commented, non-blank) statement in the Configuration File.
-----
1-Help 2-Jump 3-Quit 4-Zoom 5-BBack 6-BNext 7-Back 8-Next 10-Store 12-File
====>
RSCS CONFIG      (Your Version - Modified)  Trunc= 80 Size= 244
00000 * * * Top of File * * *
00001 *****
00002 *
00003 *      modified by system programmer
00004 *
00005 *
00006 * Following are the "rules" for defining an RSCS Configuration File:
-----
1-Help 2-Jump 3-Quit 4-Zoom 5-BBack 6-BNext 7-Back 8-Next 10-Store 12-File
====>

```

*Figure 42. Sample Split-Screen XEDIT Session*

The split-screen editing session lets you retailer the files so that both your changes and the product changes can be incorporated into one file. In this task, you can either merge your changes into the new product files or modify your files to contain the product changes. **We recommend you modify and save the product version of the file** so you have the new changes for the product for that file.

When you are in the split-screen XEDIT session, use the PF keys (explained in [Table 5 on page 70](#)) to work with the file.

*Table 5. Program Function (PF) Keys for Split-Screen XEDIT Session*

PF Keys	Function	Explanation
PF1 / PF13	Help	Displays online help information.
PF2 / PF14	Jump	Moves the cursor between the two files' command lines.
PF3 / PF15	Quit	Quits the XEDIT session without saving your changes.
PF4 / PF16	Zoom	Expands the screen in which the cursor is positioned to full screen mode. Press PF4 (ZOOM) again to return the split-screen session.
PF5 / PF17	BBack	Scrolls both files in split-screen mode backward one screen image.
PF6 / PF18	BNext	Scrolls both files in split-screen mode forward one screen image.
PF7 / PF19	Back	Scrolls the file where the cursor is positioned backward one screen image. Leaves the other screen as is.
PF8 / PF20	Next	Scrolls the file where the cursor is positioned forward one screen image. Leaves the other screen as is.

Table 5. Program Function (PF) Keys for Split-Screen XEDIT Session (continued)

PF Keys	Function	Explanation
PF9 / PF21		Undefined PF key.
PF10 / PF22	Store	Saves a copy of either file to your A-disk or your reader.
PF11 / PF23		Splits or joins the line at the position of the cursor. (This PF key is not displayed with the other PF key settings.)
PF12 / PF24	File	Copies either file to the product disk and exits the XEDIT session.

When you are in the split-screen session, you can use the CUT, CUTC, and PLACE prefix commands. The commands are similar to the XEDIT COPY prefix commands.

**CUT**

Selects a single line to be copied to another file.

**CUTC**

Selects a block of lines to be copied to another file.

**PLACE**

Specifies where to place the selected lines from a CUT or CUTC.

For example, assume you are editing both versions of the RSCS CONFIG file on a split screen in XEDIT mode and you want to change the product version of the file. You have added three lines to your version of the file and you would like to copy them to the product version.

In your version of the file, type cutc in the prefix area of the first line you would like to copy and cutc in the prefix area of the last line you would like to copy. Press Enter or PF2 (JUMP) to move the cursor to the other file. It should look like the split-screen file in Figure 43 on page 71.

```

RSCS CONFIG      (IBM Version - Unmodified)  Trunc= 80 Size= 379
00000 * * * Top of File * * *
00001 *****
00002 *
00003 * Following are the "rules" for defining an RSCS Configuration File:
00004 *
00005 * 1. The LOCAL statement, if included, MUST be the FIRST valid
00006 *      (non-commented, non-blank) statement in the Configuration File.
-----
  1-Help 2-Jump 3-Quit 4-Zoom 5-BBack 6-BNext 7-Back 8-Next 10-Store 12-File
====>
RSCS CONFIG      (Your Version - Modified)  Trunc= 80 Size= 244
00000 * * * Top of File * * *
00001 *****
00002 *
CUTC* *      modified by system programmer
00004 *
CUTC* *
CUTC* *
00006 * Following are the "rules" for defining an RSCS Configuration File:
-----
  1-Help 2-Jump 3-Quit 4-Zoom 5-BBack 6-BNext 7-Back 8-Next 10-Store 12-File
====>

```

Figure 43. Using the CUTC Prefix Command in a Split-Screen File

Then, in the product version of the file, type place in the prefix area of the line before the line you want the new lines to be copied to. It should look the screen in Figure 44 on page 72.

```

RSCS CONFIG      (IBM Version - Unmodified)  Trunc= 80 Size= 379

00000 * * * Top of File * * *
00001 *****
place *
00003 * Following are the "rules" for defining an RSCS Configuration File:
00004 *
00005 * 1. The LOCAL statement, if included, MUST be the FIRST valid
00006 *      (non-commented, non-blank) statement in the Configuration File.
-----
1-Help 2-Jump 3-Quit 4-Zoom 5-BBack 6-BNext 7-Back 8-Next 10-Store 12-File
====>
RSCS CONFIG      (Your Version - Modified)  Trunc= 80 Size= 244

00000 * * * Top of File * * *
00001 *****
00002 *
CUTC* *      modified by system programmer
00004 *
CUTC* *
00006 * Following are the "rules" for defining an RSCS Configuration File:
-----
1-Help 2-Jump 3-Quit 4-Zoom 5-BBack 6-BNext 7-Back 8-Next 10-Store 12-File
====>

```

Figure 44. Using the PLACE Prefix Command in a Split-Screen File

When you press Enter, the lines you specified in your version are copied to the specified location in the product version. After you have completed the CUTC and PLACE commands, your split-screen file should look like the one in [Figure 45 on page 72](#).

```

RSCS CONFIG      (IBM Version - Unmodified)  Trunc= 80 Size= 379

00000 * * * Top of File * * *
00001 *****
00002 *
00003 *      modified by system programmer
00004 *
00005 *
00006 * Following are the "rules" for defining an RSCS Configuration File:
-----
1-Help 2-Jump 3-Quit 4-Zoom 5-BBack 6-BNext 7-Back 8-Next 10-Store 12-File
====>
RSCS CONFIG      (Your Version - Modified)  Trunc= 80 Size= 244

00000 * * * Top of File * * *
00001 *****
00002 *
00003 *      modified by system programmer
00004 *
00005 *
00006 * Following are the "rules" for defining an RSCS Configuration File:
-----
1-Help 2-Jump 3-Quit 4-Zoom 5-BBack 6-BNext 7-Back 8-Next 10-Store 12-File
====>

```

Figure 45. Split-Screen File after a CUTC and PLACE

You can ask for help at any time by pressing the PF1 key. You will get an explanation of how to use the split-screen editor.

If you want to go back to the original file that you began with, you can press PF3 (QUIT); and you can restore both files to their original condition. When you press PF3, you see the screen in [Figure 46 on page 73](#).

```

VMFRES2858R Choose an option and enter the corresponding number
VMFRES2854I (0) Return to edit session without saving
VMFRES2854I      any changes made in this session
VMFRES2854I (1) Return to edit session saving
VMFRES2854I      all changes made this session
VMFRES2854I (2) File IBM version with changes
VMFRES2854I      and continue migration processing
VMFRES2854I (3) File your version with changes
VMFRES2854I      and continue migration processing

```

*Figure 46. Final Tailorings Phase Screen*

You need to choose an option.

Choosing:

**0**

brings you back into the split-screen XEDIT session and saves none of the changes you made in either file.

**1**

brings you back into the split-screen XEDIT session and saves all the changes you made to both files.

**2**

files the product version of the tailored file with the changes you made to the product disk, erases the copy of the old version of the tailored file from the migration save area, and continues the migration processing.

**3**

files your version of the tailored file with the changes you made to the product disk, erases the copy of the old version of the tailored file from the migration save area, and continues the migration processing.

If you would like to save a copy of the file after you have made changes to it, but you are not finished making all the changes, press PF10 (STORE) to save a copy of either file to your A-disk or your reader. When you press PF10, you see the screen in Figure 47 on page 73.

```

VMFRES2858R Choose an option and enter the corresponding number
VMFRES2854I (1) Return to edit session saving
VMFRES2854I      all changes made this session
VMFRES2854I (2) Send IBM version with changes to reader
VMFRES2854I      and return to edit session
VMFRES2854I (3) Send your version with changes to reader
VMFRES2854I      and return to edit session
VMFRES2854I (4) Copy IBM version with changes to filemode A
VMFRES2854I      and return to edit session
VMFRES2854I (5) Copy your version with changes to filemode A
VMFRES2854I      and return to edit session

```

*Figure 47. Final Tailorings Phase Screen*

You need to choose an option.

Choosing:

**1**

brings you back into the split-screen XEDIT session and saves all the changes you made in both files.

**2**

sends a copy of the product version of the tailored file, with the changes you made, to your reader and puts you back into the editing session.

**3**

sends a copy of your version of the tailored file, with the changes you made, to your reader and puts you back into the editing session.

- 4** copies the product version of the tailored file, with the changes you made, to your A-disk and returns you to the editing session.
- 5** copies your version of the tailored file, with the changes you made, to your A-disk and returns you to the editing session.

Once you have finished making any changes to the files, press PF12 to save your changes and keep one of the files for your new migrated copy of the product to use. If you press PF12, you will see the screen in [Figure 48](#) on page 74.

```
VMFRES2858R Choose an option and enter the corresponding number
VMFRES2854I (1) Return to edit session saving
VMFRES2854I     all changes made this session
VMFRES2854I (2) File IBM version with changes
VMFRES2854I     and continue migration processing
VMFRES2854I (3) File your version with changes
VMFRES2854I     and continue migration processing
```

*Figure 48. Final Tailorings Phase Screen*

You need to choose an option.

Choosing:

- 1** brings you back into the split-screen XEDIT session and saves all the changes you made to both files.
- 2** files the product version of the tailored file with the changes you made to the product disk, erases the copy of the old version of the tailored file from the migration save area, and continues the migration processing.
- 3** files your version of the tailored file with the changes you made to the product disk, erases the copy of the old version of the tailored file from the migration save area, and continues the migration processing.

You will get a chance to retailer each of the product files you tailored and each file that has been changed by the product for this new version or release.

## View-Screen Session

If you have tailored files associated with your product and the product has packaged these files a different way for this new version or release, you are placed in this view-screen session. See [Figure 49](#) on page 75 for an example of a session.

You can get online help for the view-screen session by entering:

```
help vmses vmfbrwse
```



```

PROFILE GCS      (Your Version - Modified)  Trunc= 80 Size= 32

00000 * * * Top of File * * *
00001 /* Procedure to load and initialize RSCS */
00002 'CP SET IMSG OFF'
00003 'CP SET EMSG ON'
00004 'CP SP CON START'
00005
00006 /*****
00007 customer added tailorings
00008 *****/
00009
00010 'GLOBAL LOADLIB RSCS'           /* LOADLIB where RSCS lives */
00011 'FILEDEF CONFIG DISK RSCS CONFIG *' /* RSCS configuration file */
00012 'FILEDEF EVENTS DISK EVENTS CONFIG *' /* RSCS events file */
00013 'LOADCMD RSCS DMTMAN'         /* Load RSCS module */
00014
00015 'RSCS INIT'                     /* RSCS initialize command */
00016 if rc -= 0 then exit rc         /* If failed then exit */
00017
00018 'RSCS ENABLE 125'               /* Enable auto-answer */
-----
  1-Help      3-Quit      7-Back      8-Next      10-Store
=====>

```

Figure 49. Sample View-Screen Session

The files might no longer be needed by the new copy of the product, but this session allows you to see the files so you can save them if you do not want them erased from your system.

When you are in the view-screen session, use the PF keys (explained in [Table 6 on page 75](#)) to work with the file.

Table 6. Program Function (PF) Keys for View-Screen Session

PF Keys	Function	Explanation
PF1 / PF13	Help	Displays online help information.
PF3 / PF15	Quit	Quits the view session without saving the file.
PF7 / PF19	Back	Scrolls the file backward one screen image.
PF8 / PF20	Next	Scrolls the file forward one screen image.
PF10 / PF22	Store	Saves a copy of the file to your A-disk or your reader.

**Note:** The other PF keys are not defined during the view-screen session.

You can ask for help at any time by pressing the PF1 key. You will get an explanation of how to use the view screen.

If you no longer need the file and want to erase it from your system, you can press PF3 (QUIT); and it will be erased from the migration save area. When you press PF3, you see the screen in [Figure 50 on page 75](#).

```

VMFRES2858R Choose an option and enter the corresponding number
VMFRES2855I (0) Return to view session
VMFRES2855I (1) Erase the saved version and continue
VMFRES2855I      the migration processing

```

Figure 50. Final Tailorings Phase Screen

You need to choose an option.

Choosing:

**0**

brings you back into the view-screen session and allows you to continue viewing the file.

**1**

erases the file from the migration save area and continues the migration processing.

If you are finished viewing the file and you would like to save a copy, press PF10 (STORE). A copy of the file is stored on your A-disk or in your reader. If you press PF10, you will see the screen in [Figure 51 on page 76](#).

```
VMFRES2858R Choose an option and enter the corresponding number
VMFRES2855I (0) Return to view session
VMFRES2855I (1) Send the saved version to the reader
VMFRES2855I      and continue migration processing
VMFRES2855I (2) Copy the saved version to filemode A
VMFRES2855I      and continue migration processing
```

*Figure 51. Final Tailorings Phase Screen*

You need to choose an option.

Choosing:

**0**

brings you back into the view-screen session and allows you to continue viewing the file.

**1**

sends a copy of the file to your reader, erases the file from the migration save area, and continues your migration.

**2**

Copies the file to your A-disk, erases the file from the migration save area, and continues your migration.

You will get a chance to view each of the product files that you have tailored, but the product may no longer need them.

## Using the VMFSIM EXEC during a Migration

If you have questions on the status or levels of products during a migration, you can use the VMFSIM queries to access the information in the system-level Software Inventory. The VMFSIM EXEC:

- Maintains system-level inventories of all products installed or migrated on the system.
- Maintains service-level inventories of all maintenance installed to products supported by VMSES/E.
- Allows queries of both the service-level and system-level Software and Service Inventories to identify products and maintenance installed on the system.

You can also use the VMFINFO EXEC to query the software inventory tables. For example, you can use VMFINFO to get a list of the product parameter files for the products on your system. VMFINFO provides easy-to-use panels and a variety of predefined queries for both product information and service information. For more information, see [Chapter 17, “Using the VMFINFO Panels,” on page 199](#).

For more information on the Software Inventory tables, see [Chapter 15, “Introduction to the Software Inventory,” on page 163](#) and [Chapter 22, “Software Inventory Syntax,” on page 659](#). [Chapter 16, “Introduction to the VMFSIM EXEC,” on page 177](#) shows you how to use the VMFSIM EXEC to query the Software Inventory tables.

## Chapter 6. Building Products with VMFINS

This chapter tells you, in general terms, what happens when you use the VMFINS BUILD command to build products you have installed or migrated using VMSES/E.

The topics covered in this chapter include:

- When you would build a product
- Building a product using the PPF operand (scenario)

### When Should You Perform a Build?

You should perform a build on any product you install or migrate. VMFINS BUILD:

- Builds the product
- Updates the Software Inventory
- Verifies the product was installed or migrated and built correctly

Even if the product does not need to be built, it is important to update the Software Inventory. Other functions you perform on the system with VMFINS need to know the product was built.

### Building Products on Your System

Before you can build a product, you must install or migrate that product using VMSES/E. After you have installed or migrated the product, use the VMFINFO EXEC or look through the VMFINS PRODLIST file to find the product identifiers (*ppfname* and *prodid*) for the product you want to build. For more information on the VMFINFO EXEC, see Chapter 17, “Using the VMFINFO Panels,” on page 199. For more information on how to read the VMFINS PRODLIST file, see “The VMFINS PRODLIST File” on page 18.

To build a product on your system, enter:

```
vmfins build ppf ppfname compname (serviced
```

See “Scenario 1: Building a Product with the PPF Operand” on page 77 for an example of how to use VMFINS to build products.

Two execs, *Bponum* and *Vponum*, may also be sent as part of the product and are handled by VMFINS (*ponum* is the product order number). The *Bponum* EXEC is a product-specific build exec and the *Vponum* EXEC is a product-specific verification exec. VMFINS does not need these files to run, but it does call these execs as part of its process, if they exist. For more information on these execs, see the installation documentation for the specific product. For a complete description of the VMFINS BUILD command syntax, see “VMFINS BUILD Command” on page 405.

### Scenario 1: Building a Product with the PPF Operand

In this scenario, we show you how to use the PPF operand to build a product. We also show you examples of the types of output we receive. The information you receive will depend on the products and system you are using. In this scenario, we use the SERVICED option to build the products. The default, STATUS, only provides information and does not build the product.

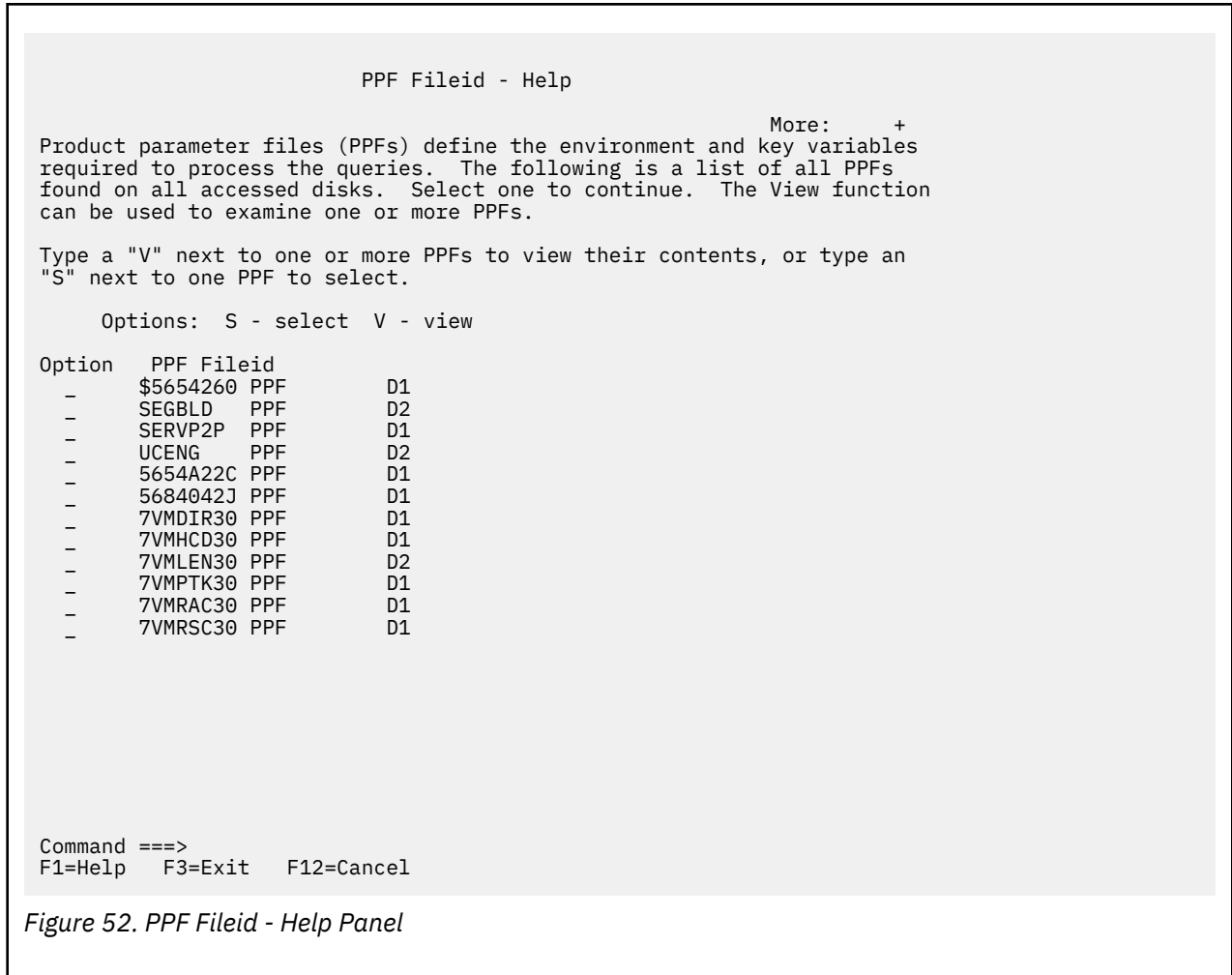
A z/VM system is installed and running. A copy of IBM XL C/C++ for z/VM Compiler has been installed on the system. We check “Who Can Use VMFINS?” on page 10 to make sure we have everything we need. We are ready to build our product.

## Step 1. Find the ppfname for the Product

First, we need to find the *ppfname* for the copy of the product we want to build. We use the VMFINFO EXEC to get a list of the product parameter file names for the products installed on our system. We enter:

```
vmfinfo
```

As we can see in [Figure 52 on page 78](#), the PPF Fileid - Help panel provides us with a list of the PPFs for the products installed.



We can see the *ppfname* for the copy of CCXX that we installed (5654A22C) is in the list. We want to build this copy of CCXX, 5654A22C PPF.

**Note:** If we wanted to find the *prodid* for any entry, we could enter an "S" in the blank to the left of the entry and press Enter. The *prodid* would appear on the upper left side of the next panel displayed.

In this scenario, the *ppfname* is 5654A22C and the *compname* is CCXX.

## Step 2. Build CCXX

Now we are ready to build CCXX. We enter:

```
vmfins build ppf 5654a22c ccxx (link serviced
```

We specify the LINK option because we need to link to the minidisks, and access the minidisks and SFS directories necessary to build the product.

As the build process begins, we see the message:

```
VMFINS2760I VMFINS processing started
```

We see several messages during the build, and when build processing has completed successfully, we see the message:

```
VMFINS2760I VMFINS processing completed successfully
```

### Step 3. Review the Output Files

Now that we have completed the build, we review the \$VMFINS \$MSGLOG and VMFINS CONSOLE files to see if we need to perform additional tasks.

#### The \$VMFINS \$MSGLOG File

A log of the commands and the messages issued during the build processing is created and stored in the \$VMFINS \$MSGLOG file on our A-disk. [Figure 53 on page 79](#) shows the \$VMFINS \$MSGLOG file. [Table 10 on page 138](#) explains the different types of messages in a message log.

```
*****
****          VMFINS   BUILD                USERID: 5654A22C          ****
*****
****          Date: 2022-09-12             Time: 16:45:14          ****
*****
ST:VMFINS2195I VMFINS BUILD PPF 5654A22C CCXX ( SYSTEM VM SIDISK 51D SIMODE D
ST:
ST:VMFINS2760I VMFINS processing started
ST:VMFINS2603I Processing product :PPF 5654A22C CCXX :PRODID 5654A22C%CCXX
ST:VMFREQ2805I Product :PPF 5654A22C CCXX :PRODID 5654A22C%CCXX has passed
ST:
ST:VMFINB2603I Building product :PPF 5654A22C CCXX :PRODID 5654A22C%CCXX
ST:VMFSET2760I VMFSETUP processing started for 5654A22C CCXX
ST:VMFUTL2205I Minidisk|Directory Assignments:
ST:
ST:          String      Mode  Stat  Vdev  Label  (OwnerID 0dev : Cyl/%Used)
ST:          -or-          SFS Directory Name
ST:VMFUTL2205I LOCALSAM  E    R/W  2C2  CCX2C2 (5654A22C 02C2 : 5/23)
ST:VMFUTL2205I APPLY      F    R/W  2A6  CCX2A6 (5654A22C 02A6 : 5/01)
ST:VMFUTL2205I           G    R/W  2A2  CCX2A2 (5654A22C 02A2 : 5/01)
ST:VMFUTL2205I DELTA     H    R/W  2D2  CCX2D2 (5654A22C 02D2 : 450/00)
ST:VMFUTL2205I BUILD0    I    R/W  29E  CCX29E (5654A22C 029E : 450/54)
ST:VMFUTL2205I BASE1     J    R/W  2B2  CCX2B2 (5654A22C 02B2 : 250/97)
ST:VMFUTL2205I ----- A    R/W  191  CCX191 (5654A22C 0191 : 125/01)
ST:VMFUTL2205I ----- B    R/O  5E5  MNT5E5 (MAINT730 05E5 : 18/48)
ST:VMFUTL2205I ----- C    R/W  1700 MJD700 (5654A22C 1700 : 3330/30)
ST:VMFUTL2205I ----- D    R/W  51D  MNT51D (MAINT730 051D : 26/36)
ST:VMFUTL2205I ----- S    R/O  190  MNT190 (MAINT 0190 : 207/64)
ST:VMFUTL2205I ----- Y/S  R/O  19E  MNT19E (MAINT 019E : 500/38)
ST:VMFSET2760I VMFSETUP processing completed successfully
ST:VMFSET2760I VMFSETUP processing completed successfully
ST:VMFBLD2760I VMFBLD processing started
ST:VMFBLD1851I Reading build lists
ST:VMFBLD2182I Identifying new build requirements
ST:VMFBLD2182I No new build requirements identified
ST:VMFBLD2179I There are no build requirements matching your request at this
ST:
BD:VMFBLD2180I There are 0 build requirements remaining
ST:VMFBLD2760I VMFBLD processing completed successfully
ST:VMFINB2603I Product built
ST:VMFINB2173I Executing verification exec V5654A22
ST:VMFINS2760I VMFINS processing completed successfully 1
```

Figure 53. The \$VMFINS \$MSGLOG File Created during BUILD Processing

As we can see in [Figure 53 on page 79](#), the VMFINS processing completed successfully (1).

#### The VMFINS CONSOLE File

We can also look in the VMFINS CONSOLE file ([Figure 54 on page 80](#)), which is spooled to our reader, for any system messages and the responses we made during the build.

```

vmfins build ppf 5654a22c ccxx ( link serviced
VMFUTL2767I Reading VMFINS DEFAULTS B for additional options
VMFINS2760I VMFINS processing started
VMFINS2603I Processing product :PPF 5654A22C CCXX :PROPID 5654A22C%CCXX
VMFREQ2805I Product :PPF 5654A22C CCXX :PROPID 5654A22C%CCXX has passed
requisite checking
VMFINB2603I Building product :PPF 5654A22C CCXX :PROPID 5654A22C%CCXX
VMFSET2760I VMFSETUP processing started for 5654A22C CCXX
VMFUTL2205I Minidisk|Directory Assignments:
String      Mode Stat Vdev Label (OwnerID Odev : Cyl/%Used)
-or-
SFS Directory Name
VMFUTL2205I LOCALSAM E R/W 2C2 CCX2C2 (5654A22C 02C2 : 5/23)
VMFUTL2205I APPLY F R/W 2A6 CCX2A6 (5654A22C 02A6 : 5/01)
VMFUTL2205I G G R/W 2A2 CCX2A2 (5654A22C 02A2 : 5/01)
VMFUTL2205I DELTA H R/W 2D2 CCX2D2 (5654A22C 02D2 : 450/00)
VMFUTL2205I BUILD0 I R/W 29E CCX29E (5654A22C 029E : 450/54)
VMFUTL2205I BASE1 J R/W 2B2 CCX2B2 (5654A22C 02B2 : 250/97)
VMFUTL2205I ----- A R/W 191 CCX191 (5654A22C 0191 : 125/01)
VMFUTL2205I ----- B R/O 5E5 MNT5E5 (MAINT730 05E5 : 18/48)
VMFUTL2205I ----- C R/W 1700 MJD700 (5654A22C 1700 : 3330/30)
VMFUTL2205I ----- D R/W 51D MNT51D (MAINT730 051D : 26/36)
VMFUTL2205I ----- S R/O 190 MNT190 (MAINT 0190 : 207/64)
VMFUTL2205I ----- Y/S R/O 19E MNT19E (MAINT 019E : 500/38)
VMFSET2760I VMFSETUP processing completed successfully
VMFUTL2767I Reading VMFINS DEFAULTS B for additional options
VMFBLD2760I VMFBLD processing started
VMFBLD1851I Reading build lists
VMFBLD2182I Identifying new build requirements
VMFBLD2182I No new build requirements identified
VMFBLD2179I There are no build requirements matching your request at this
time. No objects will be built
VMFBLD2180I There are 0 build requirements remaining
VMFBLD2760I VMFBLD processing completed successfully
VMFINB2603I Product built
1
VMFINB2173I Executing verification exec V5654A22 2
*** V5654A22: Installation Verification Beginning...

C/C++ for z/VM Installation Verification Test, for OPT(0) RENT
Product Name: 5694A01
Version 01 Release 12 Modification 00
Text Creation Date: 22:179
VALIDATION SUCCESSFUL

C/C++ for z/VM Installation Verification Test, for OPT(1)
Product Name: 5694A01
Version 01 Release 12 Modification 00
Text Creation Date: 22:179
VALIDATION SUCCESSFUL
VMFINS2760I VMFINS processing completed successfully

```

Figure 54. The VMFINS CONSOLE File Created during BUILD Processing

In Figure 54 on page 80, we check to see whether the product was built (1) and whether a product verification exec was called (2).

**Note:** If something goes wrong during a build, you should check the previous two files to see where the error occurred.

See online help or the appropriate messages documentation for additional information on error messages and recommendations for fixing the errors. Then, rerun the VMFINS BUILD command as you originally entered it.

## Summary

We have just built CCXX using the VMFINS BUILD command with the PPF operand.

---

## Chapter 7. Deleting Products with VMFINS

This chapter tells you, in general terms, what happens when you use the VMFINS DELETE command to delete products in VMSES/E format from your system. It also provides a detailed scenario to show you how to delete a product using the PPF operand.

---

### Deleting Products from Your System

You can delete products from your system if they were installed using VMSES/E. When you use the VMFINS DELETE command to delete products, VMFINS:

- Asks you which copy of the product you want to delete, if there is more than one copy on your system.
- Checks to see which products depend on the product you are deleting. If other products rely on the product, a list of products affected by the delete is displayed.

**Note:** When you are deleting licensed programs, you will have to manually delete files if they reside on a minidisk that is listed in the :MDA section of the product parameter file with a label beginning with 'LOCAL'.

---

### Scenario 1: Deleting a Product with the PPF Operand

In this scenario we show you how to use the PPF operand to delete a single product. We also show examples of the types of output we receive. The information you receive will depend on the products and system you are using.

We have a z/VM system installed and running, and there is a copy of IBM XL C/C++ for z/VM Compiler on our system. We want to delete this product, but not its resources, from the system. We check [“Who Can Use VMFINS?”](#) on [page 10](#) to make sure we have everything we need.

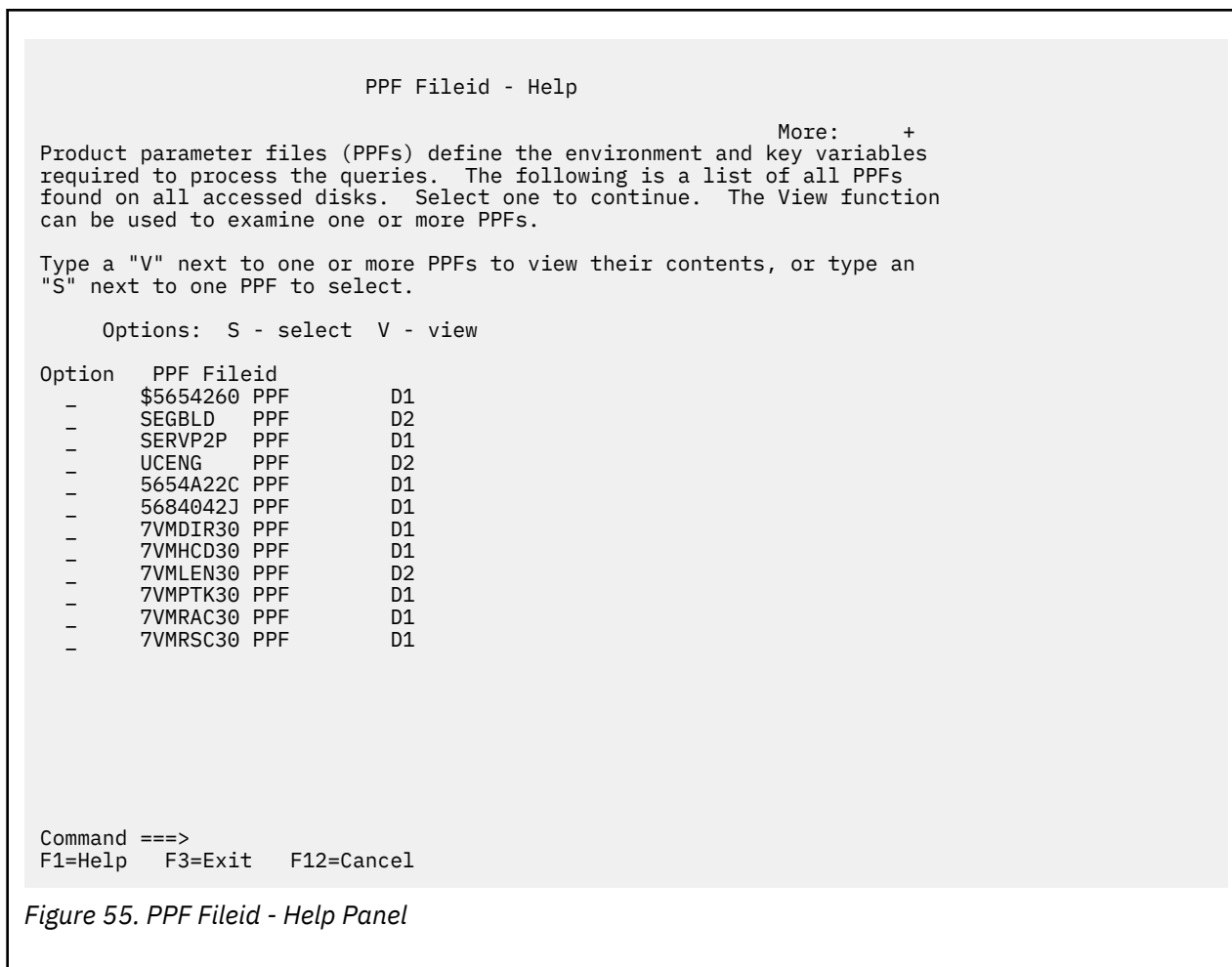
#### Step 1. Find the ppfname for the Copy We Want to Delete

The first thing we need to do is find the *ppfname* for the copy of CCXX that we want to delete.

We use the VMFINFO EXEC to get a list of the product parameter files for the products installed on our system. We enter:

```
vmfinfo
```

As we can see in [Figure 55 on page 82](#), the PPF Fileid - Help panel provides us with a list of the PPFs for the products installed.



We can see the *ppfname* for the copy of CCXX that we installed (5654A22C) is in the list. We want to build that copy of CCXX, 5654A22C PPF.

**Note:** If we wanted to find the *prodid* for any entry, we could enter an "S" in the blank to the left of that entry and hit enter. The *prodid* would appear on the upper left side of the next panel displayed.

## Step 2. Run the PLAN Option

Now, we run the PLAN option. PLAN processing creates two files, 5654A22C PLANINFO and 5654A22C ERASE.

The 5654A22C PLANINFO file shows us:

- Minidisks, user IDs, and Shared File System (SFS) directories used by the product, as well as the amount of space the product is currently using
- Products that depend on the product we want to delete

The 5654A22C ERASE file lists the names of the files that will be erased when we delete the product. 5654A22C is the name that was given to the product parameter file when the product was installed.

We want to plan for the deletion of the product, so we enter:

```
vmfins delete ppf 5654a22c ccxx (plan nolink
```

When PLAN processing is complete, two files, 5654A22C PLANINFO and 5654A22C ERASE, are stored on our A-disk.



### Step 3. Review the 5654A22C PLANINFO and 5654A22C ERASE Files

We check the contents of the 5654A22C PLANINFO file to make sure no products depend on this product and to see which minidisks, user IDs, and SFS directories can be deleted.

We also make sure the 5654A22C ERASE file was created on our A-disk. This file shows a list of the files owned by the product we want to delete. VMFINS gets this information from the VMSES PARTCAT tables on each target minidisk and SFS directory owned by CCXX. (For more information on the VMSES PARTCAT table, see [“The Parts Catalog \(VMSES PARTCAT\)”](#) on page 170.)

Figure 56 on page 83 shows portions of the 5654A22C ERASE file.

```
* VMFDEF2716I The following files can be erased from 2A6
TDATA
:PARTID DUMMY FILE
:PRODID 5654A22C%CCXX
:STAT VMFREC.06/28/22.16:30:49.5654A22C
* VMFDEF2716I The following files can be erased from 29E
TDATA
:PARTID CCNDRVR MODULE
:PRODID 5654A22C%CCXX
:STAT VMFREC.06/28/22.16:31:33.5654A22C
TDATA
:PARTID CCNEP MODULE
:PRODID 5654A22C%CCXX
:STAT VMFREC.06/28/22.16:31:33.5654A22C
TDATA
:PARTID CCNETBY MODULE
:PRODID 5654A22C%CCXX
:STAT VMFREC.06/28/22.16:31:33.5654A22C
TDATA
:PARTID CBXFINIT MODULE
:PRODID 5654A22C%CCXX
:STAT VMFREC.06/28/22.16:31:33.5654A22C
.
.
* VMFDEF2716I The following files can be erased from 2B2
TDATA
:PARTID 5654A22C $PPF
:PRODID 5654A22C%CCXX
:STAT VMFREC.06/28/22.16:31:07.5654A22C
TDATA
:PARTID CBXFINIT TEXT
:PRODID 5654A22C%CCXX
:STAT VMFREC.06/28/22.16:31:07.5654A22C
TDATA
:PARTID CBXIDYNA TEXT
:PRODID 5654A22C%CCXX
:STAT VMFREC.06/28/22.16:31:07.5654A22C
TDATA
:PARTID CBXIDYNF TEXT
:PRODID 5654A22C%CCXX
:STAT VMFREC.06/28/22.16:31:07.5654A22C
TDATA
:PARTID CBXIFOPN TEXT
:PRODID 5654A22C%CCXX
:STAT VMFREC.06/28/22.16:31:07.5654A22C
.
.
.
```

Figure 56. 5654A22C ERASE File Created during the PLAN Processing

Now that we have checked the necessary files and we know that our files contain the right information, we can delete the product.

### Step 4. Delete CCXX

To delete CCXX, we enter:

```
vmfins delete ppf 5654a22c ccxx (noplan nolinek
```

As the delete process begins, we see the message:

```
VMFINS2760I VMFINS processing started
```

We are asked if we want to delete this product. We answer yes (1).

When delete processing completes successfully, we see the message:

```
VMFINS2760I VMFINS processing completed successfully
```

**Note:** When you are deleting licensed programs, you will have to manually delete files if they reside on a minidisk that is listed in the :MDA section of the product parameter file with a label beginning with "LOCAL".

## Step 5. Review the Output Files

Now that we have completed our delete, we review the \$VMFINS \$MSGLOG and VMFINS CONSOLE files to see if we need to perform additional tasks.

### The \$VMFINS \$MSGLOG File

The \$VMFINS \$MSGLOG file contains a list of the commands we entered and the messages issued by VMFINS during the delete process. This file is stored on our A-disk. [Figure 57 on page 84](#) shows the \$VMFINS \$MSGLOG file that was created during the delete process. [Table 10 on page 138](#) explains the different types of messages in a message log.

```
*****
****      VMFINS   DELETE                USERID: 5654A22C      ****
*****
****      Date: 2022-09-12             Time: 16:58:46       ****
*****
ST:VMFINS2195I VMFINS DELETE PPF 5654A22C CCXX ( SYSTEM VM SIDISK 51D SIMODE
ST:
ST:VMFINS2760I VMFINS processing started
ST:VMFINS2603I Processing product :PPF 5654A22C CCXX :PRODID 5654A22C%CCXX
ST:VMFDEF2805I No other products depend on product :PPF 5654A22C CCXX :PRODID
ST:
ST:VMFDEL2603I Deleting product :PPF 5654A22C CCXX :PRODID 5654A22C%CCXX
ST:VMFSET2760I VMFSETUP processing started for 5654A22C CCXX
ST:VMFUTL2205I Minidisk|Directory Assignments:
ST:
ST:
ST:VMFUTL2205I LOCALSAM E R/W 2C2 CCX2C2 (5654A22C 02C2 : 5/23)
ST:VMFUTL2205I APPLY F R/W 2A6 CCX2A6 (5654A22C 02A6 : 5/01)
ST:VMFUTL2205I G R/W 2A2 CCX2A2 (5654A22C 02A2 : 5/01)
ST:VMFUTL2205I DELTA H R/W 2D2 CCX2D2 (5654A22C 02D2 : 450/00)
ST:VMFUTL2205I BUILD0 I R/W 29E CCX29E (5654A22C 029E : 450/54)
ST:VMFUTL2205I BASE1 J R/W 2B2 CCX2B2 (5654A22C 02B2 : 250/97)
ST:VMFUTL2205I ----- A R/W 191 CCX191 (5654A22C 0191 : 125/01)
ST:VMFUTL2205I ----- B R/O 5E5 MNT5E5 (MAINT730 05E5 : 18/48)
ST:VMFUTL2205I ----- C R/W 1700 MJD700 (5654A22C 1700 : 3330/30)
ST:VMFUTL2205I ----- D R/W 51D MNT51D (MAINT730 051D : 26/36)
ST:VMFUTL2205I ----- S R/O 190 MNT190 (MAINT 0190 : 207/64)
ST:VMFUTL2205I ----- Y/S R/O 19E MNT19E (MAINT 019E : 500/38)
ST:VMFSET2760I VMFSETUP processing completed successfully
ST:VMFSET2760I VMFSETUP processing completed successfully
ST:VMFDEF2739I No files to erase from 2D2
ST:VMFDEF2739I Erasing files from 2A6 1
ST:VMFSIP2480I Results for
ST:
ST:VMFSIP2462I Table VMSES PARTCAT F is empty, it will be erased
ST:VMFDEF2739I No files to erase from 2A2
ST:VMFDEF2739I Erasing files from 29E
ST:VMFSIP2480I Results for
ST:
ST:VMFSIP2462I Table VMSES PARTCAT I is empty, it will be erased
ST:VMFDEF2739I Erasing files from 2B2
ST:VMFSIP2480I Results for
ST:
ST:VMFSIP2462I Table VMSES PARTCAT J is empty, it will be erased
ST:VMFDEL2725I Product files have been deleted
ST:VMFDEL2603I Product deleted
ST:VMFINS2760I VMFINS processing completed successfully
```

Figure 57. The \$VMFINS \$MSGLOG File Created during DELETE Processing

When we look at Figure 57 on page 84, we can see the command we entered (1), including the default values. We can also see when VMFINS began to delete the files (2).

## The VMFINS CONSOLE File

We can also look in the VMFINS CONSOLE file (Figure 58 on page 85), which is spooled to our reader, for any system messages and the responses we entered during the delete processing.

```
vmfins delete ppf 5654a22c ccxx ( noplan nolink
VMFUTL2767I Reading VMFINS DEFAULTS B for additional options
VMFINS2760I VMFINS processing started
VMFINS2601R Do you want to delete :PPF 5654A22C CCXX :PRODID 5654A22C%CCXX?
Enter 0 (No), 1 (Yes) or 2 (Exit) 1
1
VMFINS2603I Processing product :PPF 5654A22C CCXX :PRODID 5654A22C%CCXX
VMFDEP2805I No other products depend on product :PPF 5654A22C CCXX :PRODID
5654A22C%CCXX
2
VMFDEL2603I Deleting product :PPF 5654A22C CCXX :PRODID 5654A22C%CCXX
VMFSET2760I VMFSETUP processing started for 5654A22C CCXX
VMFUTL2205I Minidisk|Directory Assignments:
String      Mode  Stat  Vdev  Label  (OwnerID Odev : Cyl/%Used)
-or-
SFS Directory Name
VMFUTL2205I LOCALSAM  E    R/W   2C2   CCX2C2 (5654A22C 02C2 : 5/23)
VMFUTL2205I APPLY      F    R/W   2A6   CCX2A6 (5654A22C 02A6 : 5/01)
VMFUTL2205I          G    R/W   2A2   CCX2A2 (5654A22C 02A2 : 5/01)
VMFUTL2205I DELTA      H    R/W   2D2   CCX2D2 (5654A22C 02D2 : 450/00)
VMFUTL2205I BUILD0    I    R/W   29E   CCX29E (5654A22C 029E : 450/54)
VMFUTL2205I BASE1     J    R/W   2B2   CCX2B2 (5654A22C 02B2 : 250/97)
VMFUTL2205I -----  A    R/W   191   CCX191 (5654A22C 0191 : 125/01)
VMFUTL2205I -----  B    R/O   5E5   MNT5E5 (MAINT730 05E5 : 18/48)
VMFUTL2205I -----  C    R/W   1700  MJD700 (5654A22C 1700 : 3330/30)
VMFUTL2205I -----  D    R/W   51D   MNT51D (MAINT730 051D : 26/36)
VMFUTL2205I -----  S    R/O   190   MNT190 (MAINT 0190 : 207/64)
VMFUTL2205I -----  Y/S  R/O   19E   MNT19E (MAINT 019E : 500/38)
VMFSET2760I VMFSETUP processing completed successfully
VMFDEF2739I No files to erase from 2D2
VMFDEF2739I Erasing files from 2A6
VMFDEF2739I No files to erase from 2A2
VMFDEF2739I Erasing files from 29E
VMFDEF2739I Erasing files from 2B2
VMFDEL2725I Product files have been deleted
VMFDEL2603I Product deleted
VMFINS2760I VMFINS processing completed successfully
```

Figure 58. The VMFINS CONSOLE File Created during DELETE Processing

We check the VMFINS CONSOLE file (Figure 58 on page 85) to see the system prompts and the responses we entered during delete processing (1) and to see if any other products depend on the product we deleted (2).

### Note

If something goes wrong during the delete, you should check these two files first to see where the error occurred.

See online help or the appropriate messages book for additional information on error messages and recommendations for fixing the errors. For more information on what to do when something goes wrong during a delete, see [“Recovery Information”](#) on page 416.

## Summary

We have just deleted 5654A22C from our system using the VMFINS DELETE command with the PPF operand.

If any files are left on minidisks or SFS directories after a delete, you may want to look at those files to see if you can delete them.



## Part 3. Servicing Products

In this part of the book, you will learn concepts you need to know to service your z/VM system (and related products) and how VMSES/E supports the service task. You will learn about the primary VMSES/E execs, the control and database structures, the files used for input and output, and other VMSES/E execs that support the service process.

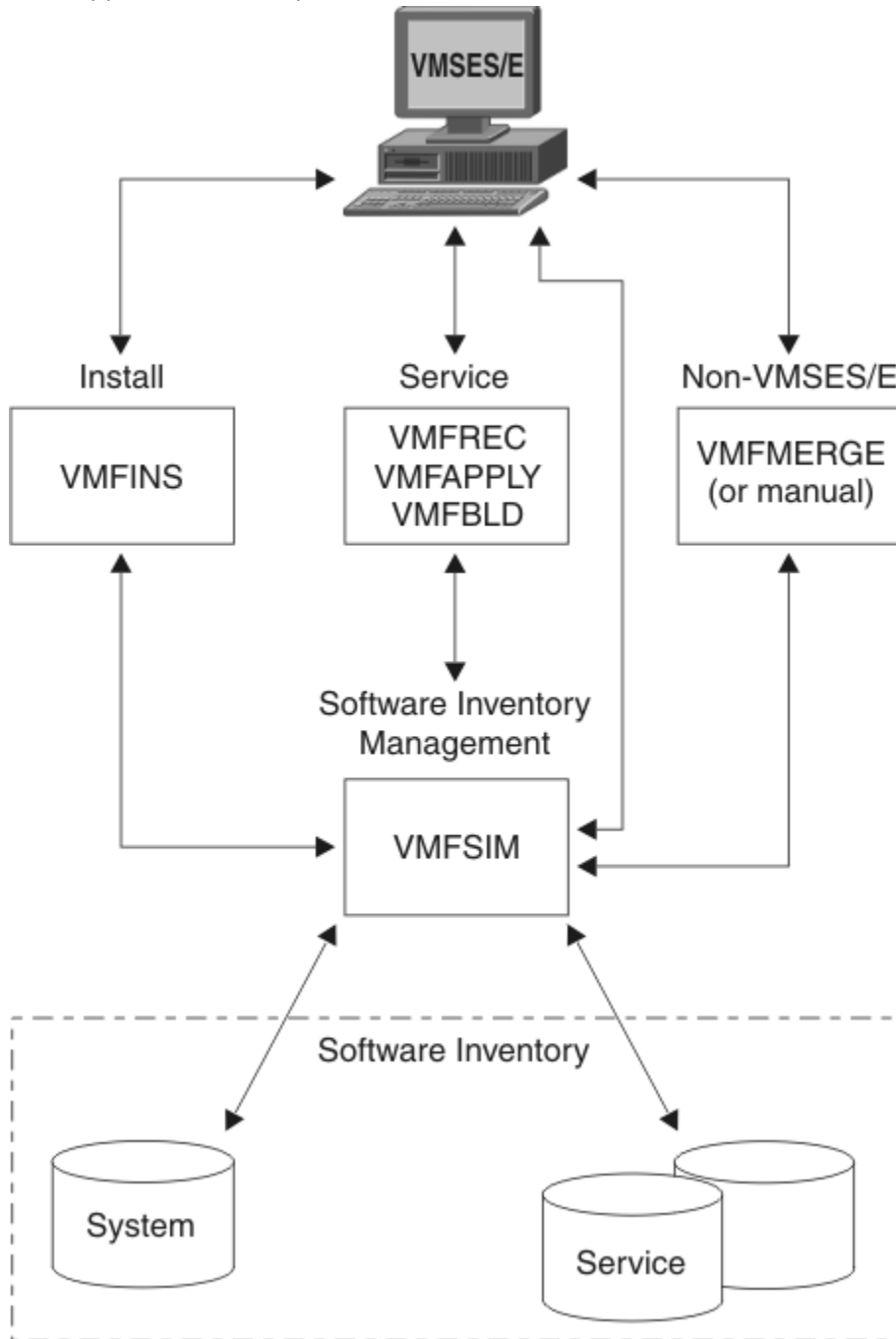


Figure 59. VMSES/E - Servicing Your System

**Note:** See the inside of the front cover for other sources of information you may need when you service products.



---

## Chapter 8. z/VM Service Concepts

This chapter introduces concepts and terminology you need to know to understand how products are serviced using VMSES/E.

---

### What is z/VM Product Service?

Service is the process of changing a particular release of a software product. There are a number of reasons for servicing a product, such as:

- Correcting a problem
- Circumventing a problem
- Adding function
- Applying local service or modifications

### Correcting a Problem

Problems that you report to IBM are called authorized program analysis reports (APARs). APARs provide a formal method of tracking problems for a specific version of a product. An APAR can also affect several releases of a product. IBM fixes these problems for a particular release through program temporary fixes (PTFs). A PTF contains fixes for one or more problems (APARs) on a particular release. Each release of a product has a unique set of PTFs, because the fixes may be different on each release.

### Circumventing a Problem

For expediency, IBM may provide a circumvention for a problem until a PTF can be developed. A circumvention is meant to be a temporary solution to the problem, and it may be in the form of a procedural or software change. The method for delivery depends on the form of the solution, and it is determined by you and the IBM support center. Once the APAR number is established for the problem, you can use that number to track the fix and see when a PTF is available.

### Adding Function

New functions can be delivered in a new release or version of a product or as service. When new function is delivered as service it is referred to as a small programming enhancement (SPE).

SPEs are delivered and tracked the same way as problems. An authorized program analysis report number (APAR number) is assigned to the SPE, and it is delivered as a program temporary fix (PTF).

### Applying Local Service or Modifications

Local service and local modifications are defined as any service or software change that is applied to your z/VM system that was not supplied by IBM as a COR or as part of an RSU.

When it is absolutely necessary to apply service from IBM before it is available as a COR, or when you need a local modification to tailor your system environment, you must apply the service locally. This includes updates supplied to you by other vendors or licensed products.

You should assign a unique local tracking number to each local modification. As subsequent PTFs are installed, circumventions and local modifications might require rework. For more information about adding local modifications, see [Chapter 11, “Installing Local Service and Modifications,” on page 101](#).

---

## Product and Service Structure

In terms of service, when we speak about a product, we are referring to the software entity that is serviced with a unique set of PTFs. This may be a separately orderable product (such as PVM), a

component within a product (such as CMS in z/VM), or a feature of a product (such as DFSMS/VM in z/VM).

## Usable Forms

A product consists of usable forms. Usable forms are the objects that make up the running software of a product, such as MODULEs, execs, LOADLIBs, and nuclei. Usable forms are created using:

- One or more serviceable parts
- A build process (for example, GENMOD, LOAD, HCPLDR, or COPYFILE)

## Serviceable Parts

Serviceable parts are the individual parts of a product that can be serviced separately. Serviceable parts have the file name of the source or replacement part and a file type that includes the PTF number and a unique three character abbreviation that describes the part type. Examples of serviceable part file identifiers are DMSABC TXT12345, SENDFILE EXC12345, and VMFLDS MOD12345.

Serviceable parts are maintained by both source updates and replacement service. When serviceable parts are maintained with **source updates**, the usable form can be generated using an update/compile facility, such as the VMFHLASM EXEC.

## Updates

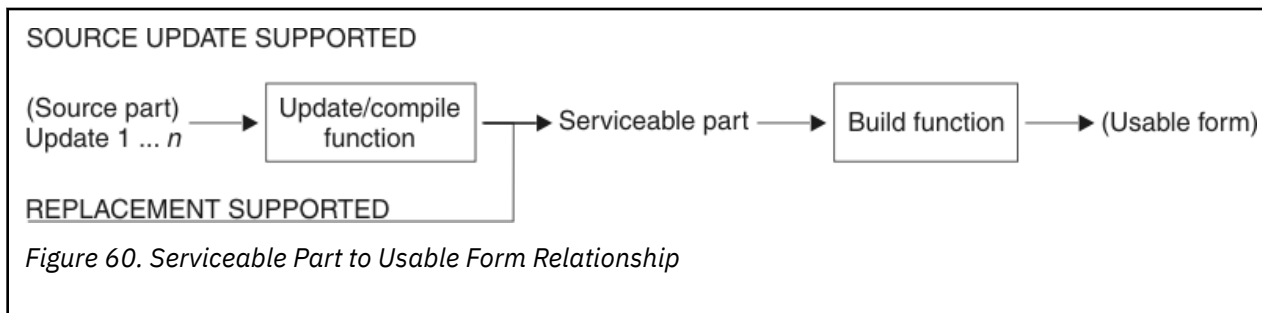
Updates are changes to the original source code that is provided with a product. There is an update for each problem or APAR that is fixed by a PTF. The APAR number is included in the file type for updates.

## Program Temporary Fixes (PTFs)

PTFs contain serviceable parts, updates, and information on the PTF. PTFs do not contain usable forms. The usable forms are built by VMSES/E from the serviceable parts and updates.

## Relationship Between Serviceable Parts and the Usable Form

Figure 60 on page 90 shows the general relationship between updates, serviceable parts, and usable forms.



In Figure 60 on page 90, parts shown in parentheses (source part and usable form) are not shipped with service. Source parts are delivered as part of the initial installation of a product. (An exception is when a new part is shipped as part of service).

Updates are applied to the source part to create serviceable parts. Updates are named *filename XaaaaaZZ*, where *filename* is the file name of the source part, *aaaaa* is an APAR number, and X/ZZ identify the release of the product. The value of X/ZZ is taken from the :SLVI tag in the product parameter file. (See Chapter 21, “Product Parameter File Syntax,” on page 621 for more information).

The serviceable parts are then processed by a build function to generate the usable form. Serviceable parts are named *filename XXXppppp*, where *filename* is the name of the source/replacement part, XXX is a unique 3-character abbreviation that describes the part type, and *ppppp* is the PTF number. The 3-character file type abbreviation is defined in the file type abbreviation table. See Chapter 15,

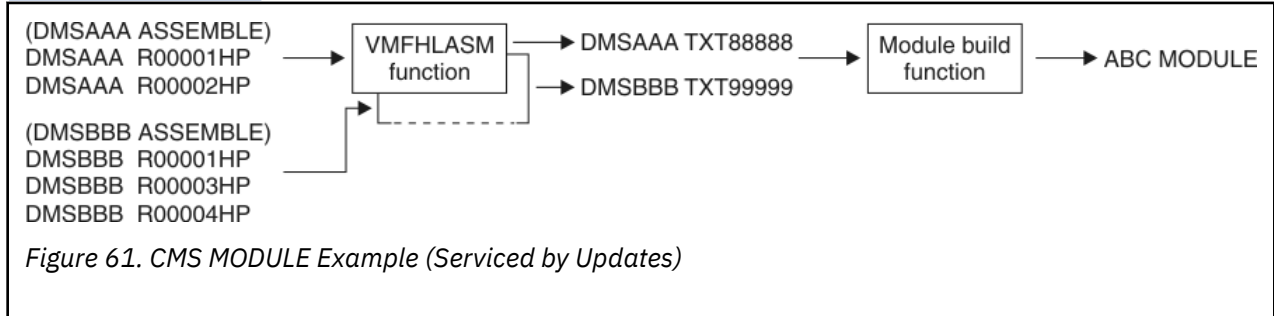


“Introduction to the Software Inventory,” on page 163 and Chapter 22, “Software Inventory Syntax,” on page 659 for more information.

**Note:** You do not have to execute the update/compile function if the serviceable parts are shipped with the PTF.

## Usable Forms Serviced by Updates

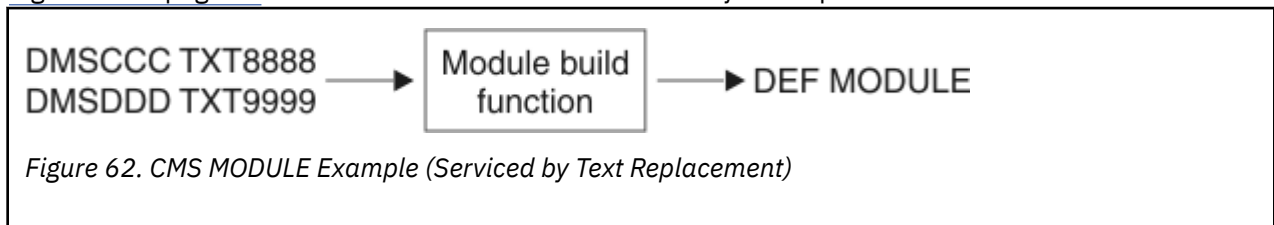
Figure 61 on page 91 shows a usable form that is serviced by updates.



The usable form, ABC MODULE, is created using a VMSES/E build function that uses the CMS LOAD and GENMOD functions. Serviceable parts (text decks with PTF numbered file types, for example DMSAAA TXT88888) are the inputs to the MODULE build function. To conserve space, only the highest version of these serviceable parts is shipped as part of a service deliverable along with the source update files. Source updates allow you to examine, modify, and create different versions of the serviceable part using the VMFHLASM function.

## Usable Forms Serviced by Text Replacement

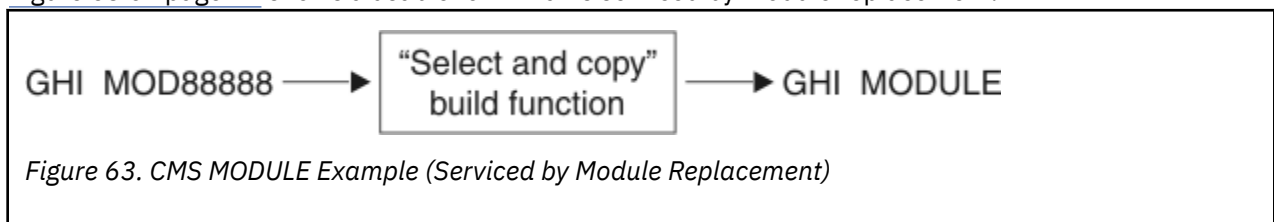
Figure 62 on page 91 shows a usable form that is serviced by text replacement.



The usable form, DEF MODULE, is created using a VMSES/E build function that uses the CMS LOAD and GENMOD functions. Serviceable parts (text decks with PTF numbered file types, for example DMSCCC TXT88888) are the inputs to the MODULE build function. These serviceable parts are shipped as part of the PTFs.

## Usable Forms Serviced by Module Replacement

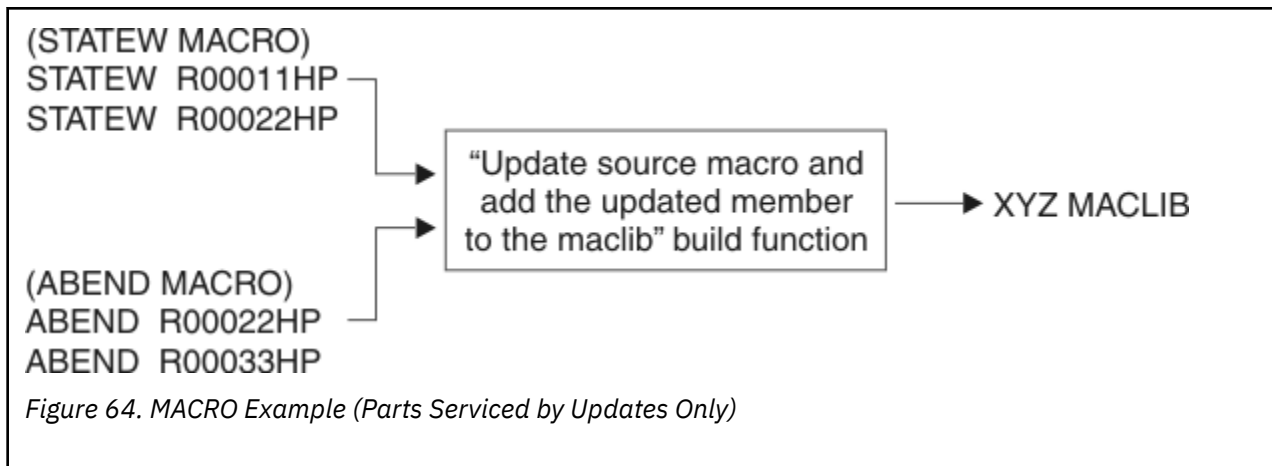
Figure 63 on page 91 shows a usable form that is serviced by module replacement.



The usable form, GHI MODULE, is created using a VMSES/E build function that copies the highest version serviceable part to a target minidisk with a file type of MODULE.

## Usable Forms Serviced by Parts that are Serviced by Updates Only

Figure 64 on page 92 shows a usable form that is serviced by a part that is serviced by updates only.



The usable form, XYZ MACLIB, is created using the VMSES/E build function which uses the CMS UPDATE and MACLIB functions. In this case, the serviceable parts are not shipped with the service. They are dynamically generated at build time. This is a trade-off of space versus time. Service media and DASD space is conserved at the expense of generation time (MACROS and COPY files tend to be large files). For information on the CMS UPDATE and MACLIB commands, see [z/VM: CMS Commands and Utilities Reference](#).

## Types of Service Supported by VMSES/E

VMSES/E provides automated functions to process service provided in the following packages:

- **Product Service Upgrade (PSU)**
- **Corrective service (COR)**
- **Expanded Service Option (ESO)**

### Product Service Upgrade (PSU)

The PSU is a nondestructive procedure in which a preventive service vehicle, such as a Recommended Service Upgrade (RSU) or a refreshed product deliverable, is used as the source of new service. It contains preapplied service, which includes service level information, PTFs, serviceable parts, and usable forms. The following steps are required and use the automated VMSES/E.

1. Load the service (service information and PTFs and usable forms)
2. Reapply any reach-ahead service
3. Rebuild usable forms affected by the reach-ahead service

Given that you meet the criteria of the intended environment, there should be very little processing required for steps 2 and 3. Because a significant amount of the service processing time is for the generation of usable forms, this option can save considerable time.

### Corrective Service (COR)

The corrective service (COR) deliverable contains PTFs that you request. You control the contents of this deliverable. By default, IBM delivers service media that contains the requested PTFs and all requisite PTFs that are not on the most recent service level. However, you can specify:

- The requisites supplied. If "no requisites" is specified, you receive only the specified PTFs.
- The service level to be delivered. If you have not installed the latest service level, you can specify the last service level, or equivalent set of PTFs installed, and receive the requisite PTFs back to this service level.

For more information, see [Chapter 9, "Installing Corrective Service,"](#) on page 95.

## Expanded Service Option (ESO)

The Expanded Service Option (ESO) is a defined collection of PTFs delivered in VMSES/E corrective service format. ESO allows you to choose the starting and ending service levels. You can also select to receive the ESO with:

- Requested products or customized to your profile
- PEs resolved
- Look ahead service
- IFREQs
- Supercede screen-out
- VMSES/E products only
- Service-on-Request only



---

## Chapter 9. Installing Corrective Service

Corrective service is shipped on a corrective (COR) service tape or in an electronic envelope, and you install it by component.

You install corrective (COR) service by:

- Receiving the service documentation
- Preparing the system for service
- Servicing the component
- Placing the newly-serviced components into production

Figure 65 on page 96 shows the usual steps in applying service to a product.

To service a product, you need to:

- Receive the service for the product.
  1. Read the *Memo-to-Users* for the product.
  2. Use the VMFREC command to receive the service.
  3. Review the receive message log and correct any errors.
- Apply the service to the component parts.
  1. Verify the component's alternate apply disk is ready to apply service for the component and determine whether merge processing is required.
  2. Use the VMFAPPLY command to apply the service.
  3. Review the apply message log and correct any errors.
- Rebuild the component.
  1. Perform any tasks that are required to rebuild the component.
  2. Review the build message log and correct any messages.
  3. Test the new level of the component and correct any errors.

To put serviced components into production, you:

- Build any saved segments.
- Merge the intermediate apply disk, with the tested service, to the production disk.

For more detailed examples, see the documentation for the product. A good reference is *z/VM: Service Guide*.

# Installing Corrective Service

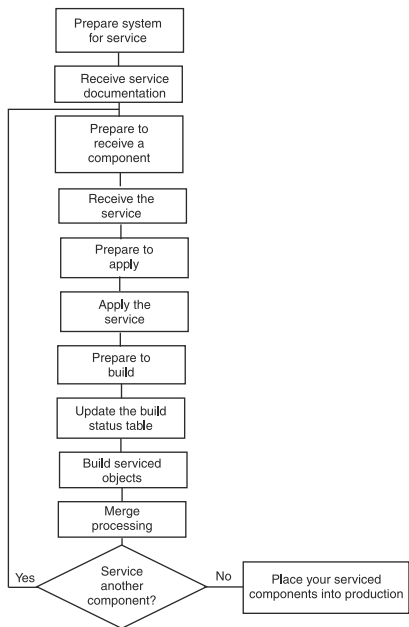


Figure 65. Installing Corrective Service

## Chapter 10. Using the Product Service Upgrade (PSU)

You can upgrade your existing licensed programs using the service files from the IBM Product Recommended Service Upgrade (RSU). The procedure for performing a Product Service Upgrade from the Recommended Service Upgrade media is outlined in this chapter.

**Note:** This procedure is not intended for use by customers migrating from a previous release. It should only be used to upgrade from a previous service level of a licensed program.

The Recommended Service Upgrade media that is processed using the PSU procedure has the following logical structure for each product. This example shows the RSU as a tape, but it could instead be on another media such as a CDROM.

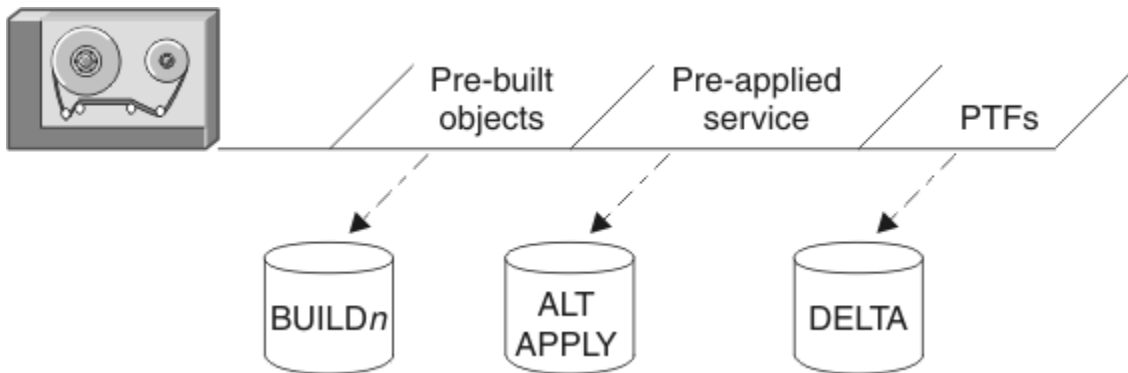


Figure 66. Recommended Service Upgrade, Files

The PSU procedure installs all PTFs included on the RSU plus the tape files containing the preapplied service and prebuilt objects. All PTF-related files are loaded to the delta disk. The file containing the preapplied service (the results of VMFAPPLY) is loaded to the alternate apply disk, and the contents of the files containing prebuilt objects are loaded to the appropriate build disks.

Points to consider about using the Product Service Upgrade procedure are:

- This process will not alter any of your tailored files in any way.
- It only pertains to z/VM and licensed programs using VMSES/E.
- Planning must be done (such as determining disk sizes, and determining what service, if any, on your existing system is not contained on the RSU) prior to actually loading the service from the RSU.

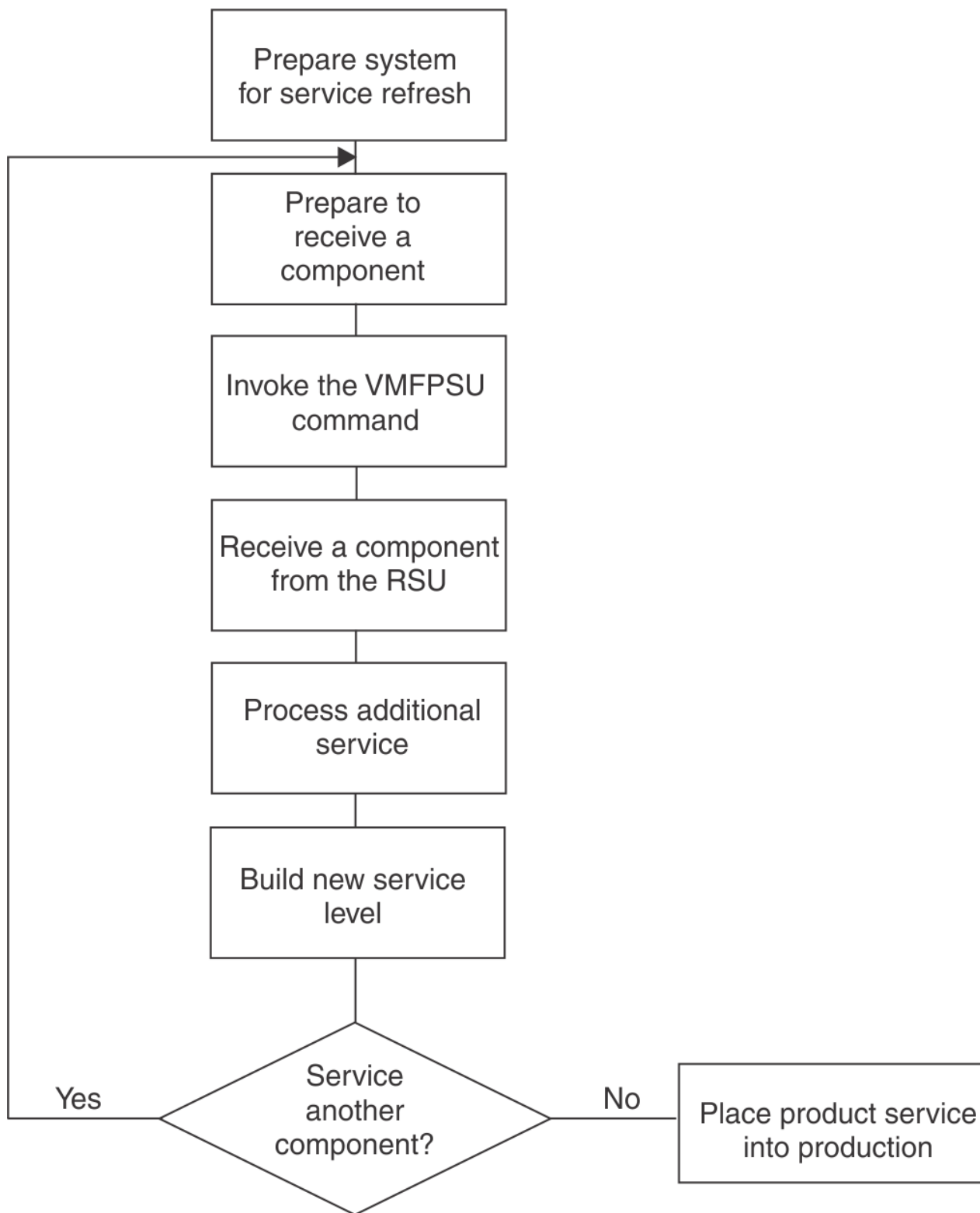


Figure 67. Service Application Flowchart using PSU

The following outline is an overview of the tasks you need to perform during the PSU procedure.

1. Receive Documentation

In this task you receive the documentation contained on the RSU and determine the DASD required to install the RSU.

2. Prepare to Receive a Product



Use the VMFPSU EXEC to obtain information to help you plan. For more information, see [“VMFPSU EXEC”](#) on page 462.

### 3. Process Preapplied, Prebuilt Service

a. Receive Preapplied, Prebuilt Service. Load the contents of the RSU to the service disks.

#### b. Process Additional Service for a Product

Reapply additional service for the product that is not contained on the RSU, including local modifications.

#### c. Rebuild Product

Build all objects that were affected by reach-ahead service that was reapplied or by local modifications. After you complete the build steps for the product, continue with the next product to be processed.

### 4. Service Another Product?

If you have another product to service, repeat the steps in this outline, beginning with step 2.

### 5. Test and Put All Products into Production

Place the new service into production. This is the last step in the PSU Procedure.



---

## Chapter 11. Installing Local Service and Modifications

This chapter contains an overview of the step-by-step procedures for applying local service to z/VM components. Local service and local modifications are defined as any service that is applied to your z/VM system that was not supplied by IBM as a COR or as part of an RSU. For step-by-step instructions, see *z/VM: Service Guide*.

This chapter also contains an overview of the step-by-step procedures for reworking local service. Existing local service can be affected by new IBM service which creates the need to rework the local service.

Examples for applying local modifications to the CP load list and CMSINST can be found in *z/VM: Service Guide*.

The local modification procedure has been automated by the LOCALMOD EXEC. For step-by-step instructions on using the LOCALMOD EXEC, see *z/VM: Service Guide*.

---

### Introduction

Local service and local modifications are defined as any service or software change that is applied to your z/VM system that was not supplied by IBM as a COR or as part of an RSU.

#### Attention

The application of local service can be a complicated and error-prone procedure because of the many variables that are involved. IBM strongly advises its customers to order service media through the IBM Support Center whenever possible.

When it is absolutely necessary to apply service from IBM before it is available as a COR, or when you need a local modification to tailor your system environment, you must apply the service locally. This includes updates supplied to you by other vendors.

The local Version Vector Table must be updated in order for VMSES/E to process your local modifications.

Local service can be placed in two categories: IBM local service and customer local modifications.

**IBM local service** includes any service that you receive from IBM that is not part of a COR or RSU deliverable.

When a severe problem arises and you cannot wait for a COR or an RSU, you can get emergency service from the change team. This service can be sent to you electronically or on a tape, or read to you over the phone. In some cases, a member of the change team can place the fix in the library that is accessed by the APARFIX command on ServiceLink. In any case, the service is not in the format required by the VMFREC and VMFAPPLY EXECs. To receive and apply this service, you must do it manually, using the instructions in this chapter.

**Customer local modifications** include any software changes that a customer makes to tailor their z/VM system. These updates can be supplied by IBM licensed products or by other vendor products. For example, if you have RACF® Security Server for z/VM, you have a customer local modification because RACF has a "mod" to CP.

**There are three ways that parts can be serviced.** Source-maintained parts are serviced by changing the source, with an update file, an AUX file, and a CNTRL file. Replacement-maintained parts are serviced by replacing the part with an updated version. Thirdly, when necessary, some parts (such as text files) can be serviced by directly changing the object code they contain. Local service for source-maintained and replacement-maintained parts is described in this chapter. The procedure for changing object code is described in the section "Apply Changes Directly to Object Code" in the *z/VM: Service Guide*.

**A rework of local service** might be necessary if the service you receive from IBM affects the existing local service. You will be notified of this possibility when you install a COR or an RSU. If this happens, re-evaluate the local service and then, if needed, rework and rebuild the affected parts.

## Overview for Local Service Procedure

---

The following is an overview of the steps in the local service procedure.

For each modified part in a component:

1. Prepare for Local Service or Modifications.

Access the component's service disks and the VMSES/E BUILD disk (5E5 by default). If necessary, modify the CNTRL file.

2. Receive Local Service or Modifications.

If you have IBM local service, load the service to the LOCALMOD disk for your product.

3. Apply Local Service to Source-Maintained parts.

a. Add an Update Record to the AUX file.

b. Create an Update File if it is not shipped.

c. Special Processing for MACROs with ASSEMBLE files.

i) Update the local version vector table with the VMFSIM CHKLVL command and the LOGMOD option.

ii) Determine what MACLIBs need to be rebuilt.

iii) For each ASSEMBLE file which uses the updated macro and does not have another change in this local service, create a dummy update for the ASSEMBLE file.

4. Apply Local Service to Replacement-Maintained parts.

a. Create or Copy the Replacement Part.

b. Special Processing for MACROs with ASSEMBLE files.

i) Determine what MACLIBs need to be rebuilt.

ii) For each ASSEMBLE file which uses the updated macro and does not have another change in this local service, create a dummy update for the ASSEMBLE file.

For each modified part in this component, repeat step [“3” on page 102](#) and or step [“4” on page 102](#).

5. Rebuild Objects.

a. Rebuild Source files.

b. Create compiled REXX™ parts.

c. Create a Replacement Part from \$Source files. Create a replacement part with the VMFEXUPD command. This command will place the output on the LOCALMOD disk, update the local Version Vector Table (VVT) and add an entry to the \$SELECT file.

d. Rebuild MACLIBs with the VMFBLD command.

e. Create a replacement part from updated ASSEMBLE files. Rebuild ASSEMBLE files with the VMFHLASM command. This command will place the output on the LOCALMOD disk, update the local Version Vector Table (VVT) and add an entry to the \$SELECT file.

f. Create a replacement part from replaced ASSEMBLE files.

g. Create a replacement part from modified National Language files with the VMFNLS command. This command will place the output on the LOCALMOD disk, update the local Version Vector Table (VVT) and add an entry to the \$SELECT file.

h. Rebuild any remaining objects.

## Overview for Rework Local Service Procedure

---

The following is an overview of the steps for the rework local service procedure. For the instructions see section "Reworking Local Service and Modifications" in [z/VM: Service Guide](#).

1. Prepare to rework local service or modifications.
2. Rework Local Service to Source-Maintained parts.
3. Rework Local Service to Replacement-Maintained parts.
4. Rebuild the Objects.

## Obtaining File Type Abbreviations

To obtain a file type abbreviation, enter the following VMFSIM command:

```
vmfsim query vm sysabrvt tdata :realft ft
```

The abbreviation on the :ABBRFT tag is returned. If more than one abbreviation is returned, you must use the one that is used by the part that you are modifying. To determine if an abbreviation is the correct one for a particular part, enter the following command:

```
vmfqobj ppfname compname tdata :part fn ftabbrev
```



## Chapter 12. Using VMSES/E for Service

VMSES/E provides EXECs to perform the primary service functions, files to direct the operation of these execs and save the status of their execution, and a database structure that isolates executable code from the control structure used to manage it.

When you service a product, you use a number of EXECs and execute a series of commands to accomplish the primary service tasks:

- **Receiving service for the product**

When you receive service, the receive function (“VMFREC EXEC” on page 477) reads the raw materials from the delivery media and places them into the VMSES/E database.

- **Applying service to the product's serviceable parts**

When you apply service, the apply function (“VMFAPPLY EXEC” on page 291) defines new maintenance levels based on the contents of the VMSES/E database.

- **Building the serviced usable forms.**

When you build the serviced parts, the build function (“VMFBLD EXEC” on page 305) uses the defined maintenance levels to select the correct level of the raw materials database to build the running product.

The Software Inventories are updated automatically after receiving, applying, and building service.

All of the above steps are automated by the SERVICE EXEC.

### The VMSES/E Database

The VMSES/E database is made up of minidisks and Shared File System (SFS) directories to separate the various types of files for each product into **logical strings**. The default minidisks and SFS directories are shown in Table 7 on page 105.

Table 7. The VMSES/E Database Defaults

Disk/Directory	Contents
A-disk (191)	VMSES/E work disk
B-disk (5E5)	VMSES/E build disk
C-disk	Reserved for user
D-disk (51D)	System-level Software Inventory
TASK	Minidisks accessed before the database
LOCAL	Customized files <ul style="list-style-type: none"> <li>• Local modifications</li> <li>• Circumventive service</li> </ul>
DELTA	PTFs (raw materials) <ul style="list-style-type: none"> <li>• PTF parts</li> <li>• PTF part lists</li> <li>• Receive status table</li> <li>• Requisite table</li> <li>• Description table</li> </ul>

Table 7. The VMSES/E Database Defaults (continued)

Disk/Directory	Contents
APPLY	Defines maintenance level <ul style="list-style-type: none"> <li>• AUX files</li> <li>• Version vector tables</li> <li>• Apply status table</li> <li>• Select data file</li> <li>• Build status table</li> </ul>
BUILD	The final usable system <ul style="list-style-type: none"> <li>• Usable forms</li> </ul>
BASE	Product (raw materials) <ul style="list-style-type: none"> <li>• Source files</li> <li>• Base object files</li> </ul>
SYSTEM	Running system disks accessed after the database

The A-disk (usually 191) is the VMSES/E work disk. The B-disk (5E5) contains the production level VMSES/E tools. The C-disk is reserved for the user. The D-disk contains the system-level Software Inventory and product parameter files.

The TASK string contains the minidisks that are accessed before the database, for example, local tools disks.

The LOCAL string contains customized files, local modifications, and circumventive service.

The DELTA string is the PTF raw materials repository. All PTF parts reside on this string. All parts must, therefore, contain a PTF or APAR number in their file name or file type. (For example, replacement parts contain a PTF number in their file type, update files contain an APAR number in their file type, and PTF parts lists contain a PTF number in their file name). The DELTA string also contains the Software Inventory files that describe the PTFs (receive status table, requisite table, and description table).

The APPLY string describes maintenance levels. Only files that are used to define maintenance levels (the apply status table, AUX files, version vector tables, the select data file, and the build status table) are on this string.

The BUILD disks contain the running code for the product being serviced, BASE disks contain the original product code, and SYSTEM disks contain the running code for other products that are required during service.

Given this database, it is easy to generate multiple systems from a single database. Because the parts on the DELTA string are all numbered, they can be used repeatedly and in different combinations. Using different APPLY strings, multiple service levels can be defined; and using different BUILD disks, multiple images of the product can be built.

## Servicing a Product with VMSES/E

Figure 68 on page 107 shows how products are serviced using the three primary service commands: VMFREC, VMFAPPLY, and VMFBLD, which are all used by the SERVICE EXEC. For a complete, step-by-step example of how to service a product, see *z/VM: Service Guide* or the documentation for the product you are servicing. For information on the automated service process, see *z/VM: Installation Guide*.



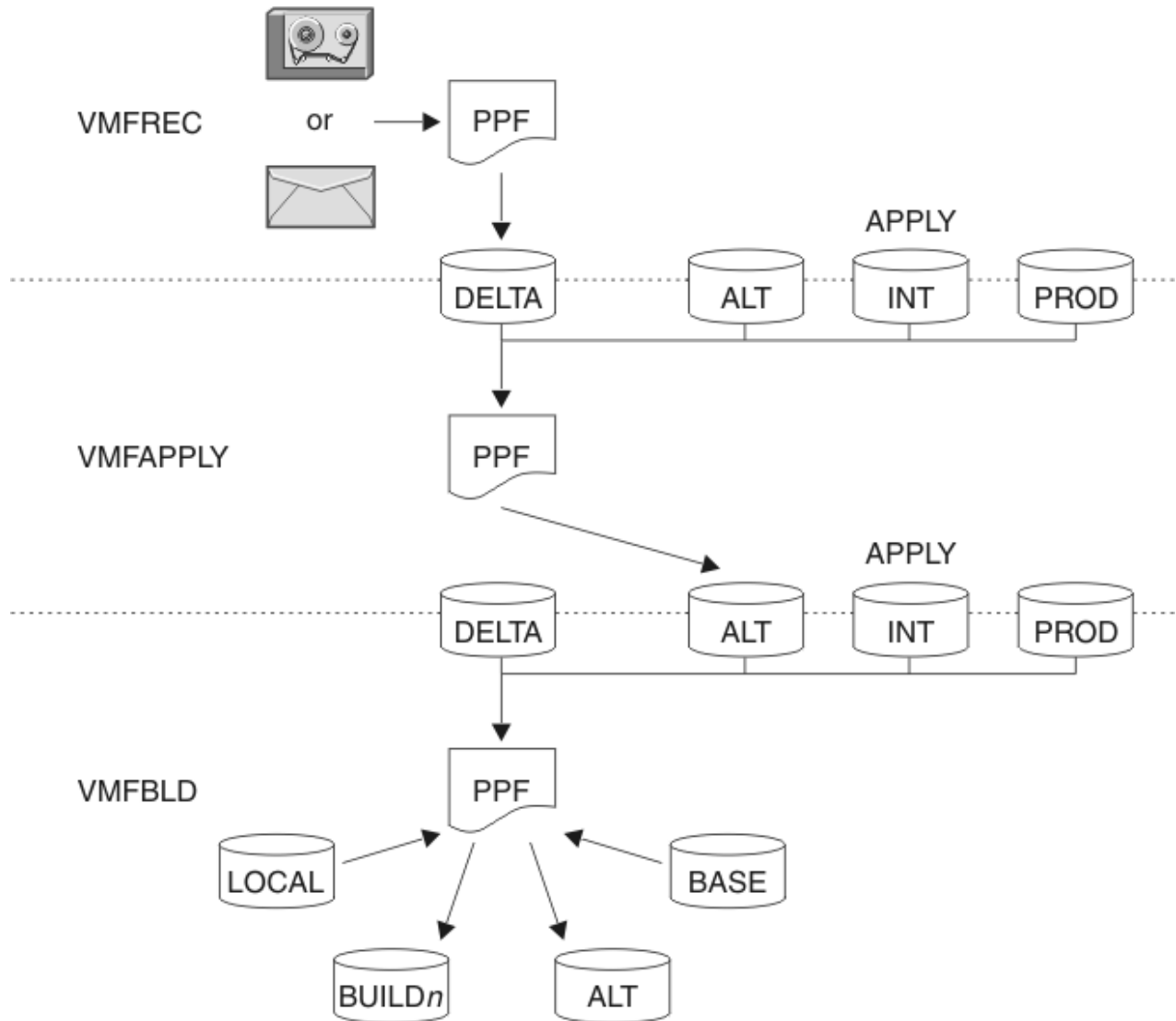


Figure 68. Servicing a Product

The number of apply disks may vary from product to product.

## Receiving Service

The VMFREC command receives service from the delivery media and places it on the DELTA disk. Before you receive new service, however, you can use the VMFMRDSK command to clear the alternate APPLY disk. This lets you easily remove the new service if you find a serious problem. For more information on these commands, see [“VMFMRDSK EXEC”](#) on page 444 and [“A Closer Look at the VMFREC EXEC”](#) on page 108.

## Applying Service

The VMFAPPLY command updates the version vector table (VVT), which identifies the service level of all serviced parts. In addition, AUX files are generated from the version vector table for parts that require them. VMFAPPLY only modifies the alternate APPLY disk. All lower level APPLY disks are unaffected. For more information, see [“A Closer Look at the VMFAPPLY EXEC”](#) on page 109.

## Reapplying Local Service

To allow VMSES/E to track the changes and build them into the system, you must enter all local service into the Software Inventory. If you do not add a local modification to the Software Inventory,

VMSES/E does not build it into the system. To add a local modification to the Software Inventory, use the LOCALMOD EXEC, use the “VMFSIM LOGMOD” on page 556 command, or specify the LOGMOD option on the VMFASM, VMFHASM, VMFHLASM, VMFNLS, VMFREPL, or VMFEXUPD command.

For more information on applying local modifications, see [Chapter 11, “Installing Local Service and Modifications,”](#) on page 101.

## Building New Levels

To generate a serviced level of an object on a build list, you need to use the appropriate command or set of commands. When you run any build command, you must access all local and service disks to ensure that the highest level parts are used to generate an object. The generated object is placed on a BUILD disk. For more information, see [“A Closer Look at the VMFBLD EXEC”](#) on page 112.

## Placing the Serviced Components into Production

Once you have tested the new service and are satisfied with the results for all serviced components, you can put them into production.

## A Closer Look at the VMFREC EXEC

---

The VMFREC EXEC reads PTFs from service tapes or envelopes and deposits them on the DELTA string. VMFREC only loads PTF parts that are numbered with the APAR or PTF number.

In addition to PTFs, VMSES/E-formatted service tapes contain header information which is used by VMFREC to position the tape. Once positioned to the beginning of the product, VMFREC calls part handlers to load the tape files that contain the PTFs. The part handlers use the Software Inventory and the existing parts to determine which PTFs and parts should be loaded and which ones should not. Once all of the new PTFs and parts are loaded, the Software Inventory is updated with the results.

## Processing Multiple Service Tapes at One Time

Frequently, multiple service tapes need to be processed at the same time. The VMFREC EXEC automates the process of receiving multiple service tapes by appending the apply and exclude lists (the lists that tell VMFAPPLY which PTFs to process) from each tape. This works with multiple COR tapes, PUTs, and even combinations of the two.

## Part Handlers

The “VMFREC EXEC” on page 477 uses a number of part handlers:

- VMFRCALL load parts unconditionally. It is primarily used during product installation when selectivity is not necessary.
- VMFRCAXL is used to load apply and exclude lists. This is the part handler that will append the contents of the apply and exclude lists when multiple tapes are being processed.
- VMFRCOM loads parts of PTFs. It is very selective about which parts it loads. It does not load usable form parts. It also does not load parts of committed PTFs unless specifically asked to.

**Note:** When a PTF ages and its parts are no longer needed because other PTFs have been added to the same parts, the PTF can be flagged as committed. This means that the obsolete parts can be discarded. The VMFREC EXEC does not receive the discarded parts from future tapes when a PTF is designated as committed.

- VMFRCPTF loads PTF parts lists. The contents of these files are the building blocks for all of the service-level Software Inventory.
- VMFRCUPP loads a tape file unconditionally to a target disk and changes each file loaded to upper case.

## Software Inventory Files Used by the VMFREC EXEC

The VMFREC EXEC uses the following files in the Software Inventory:

- PTF part file
- Receive status table
- Requisite table
- Description table

### PTF Part File

The PTF part file is the source for all PTF information. It is this file that makes it possible to avoid the excessive I/O involved in reading all of the serviceable parts and update files required to obtain the PTF/APAR history and requisite information for parts and PTFs. The file consists of three sections: a header section, a requisite section, and a parts section:

- The header section includes the PTF number and any associated APAR numbers. It also includes the APAR descriptions and APAR abstracts. Finally, it includes a user memo section which contains any special installation instructions for the PTF.
- The requisite section contains all requisite information. It includes both hard and soft pre-requisites (:HARDREQ and :PREREQ), corequisites (:COREQ) and out-of-component requisites (:IFREQ). It also includes a list of superseded PTFs (:SUP).
- The parts section contains the list of parts of the PTF. Rather than a simple list of parts, the parts section contains a great deal of information on how each part is to be processed.

### Receive Status Table

The receive status table is a list of PTFs that have been received and are available to be applied for a product. A PTF can have two different states in this table: RECEIVED or COMMITTED. When a PTF is flagged as committed, it tells VMFREC that it should not re-receive any of its parts because they are obsolete.

In addition to the status of the PTF, the table includes the date and time at which it was received or committed and the user ID from which the operation was performed.

### Requisite Table

The requisite table is the part of the Software Inventory that consolidates all of the PTF requisite information. This table includes all of the relationships that are identified in all of the PTF parts files for a product. It also includes the APAR numbers associated with each PTF, so this table can be used to translate back and forth between APAR and PTF numbers.

### Description Table

The description table contains the abstracts of each APAR included in each PTF that has been received or committed. This table can be used to do keyword searches for APAR descriptions.

The Software Inventory files used by the VMFREC EXEC are described in [Chapter 15, “Introduction to the Software Inventory,”](#) on page 163 and [Chapter 22, “Software Inventory Syntax,”](#) on page 659.

## A Closer Look at the VMFAPPLY EXEC

The VMFAPPLY EXEC defines new service levels by applying PTFs. The files that define the maintenance levels are stored on the APPLY string. VMFAPPLY performs a great deal of checking to make sure that each PTF is valid before it is added to a maintenance level.

VMFAPPLY uses the existing maintenance level, the requisite relationships, and the apply and exclude lists to determine which PTFs should be applied. (PTFs listed in the exclude list are not applied, even if they are requisites of PTFs in the apply list). It then validates each of these PTFs and (if they are valid),

it applies them. As it applies them, it updates the apply status table, version vector tables, and, when necessary, the AUX files. When the TEST option is used, this last step is bypassed. The TEST option can be used to do a *dry run* on the apply process.

Finally, the VMFAPPLY EXEC generates a list of all of the parts affected by service. This file is called the select data file. It is used by VMFBLD to determine exactly which objects need to be built as a result of new service.

## Applying a PTF

Figure 69 on page 110 shows the algorithm that is used to apply PTFs.

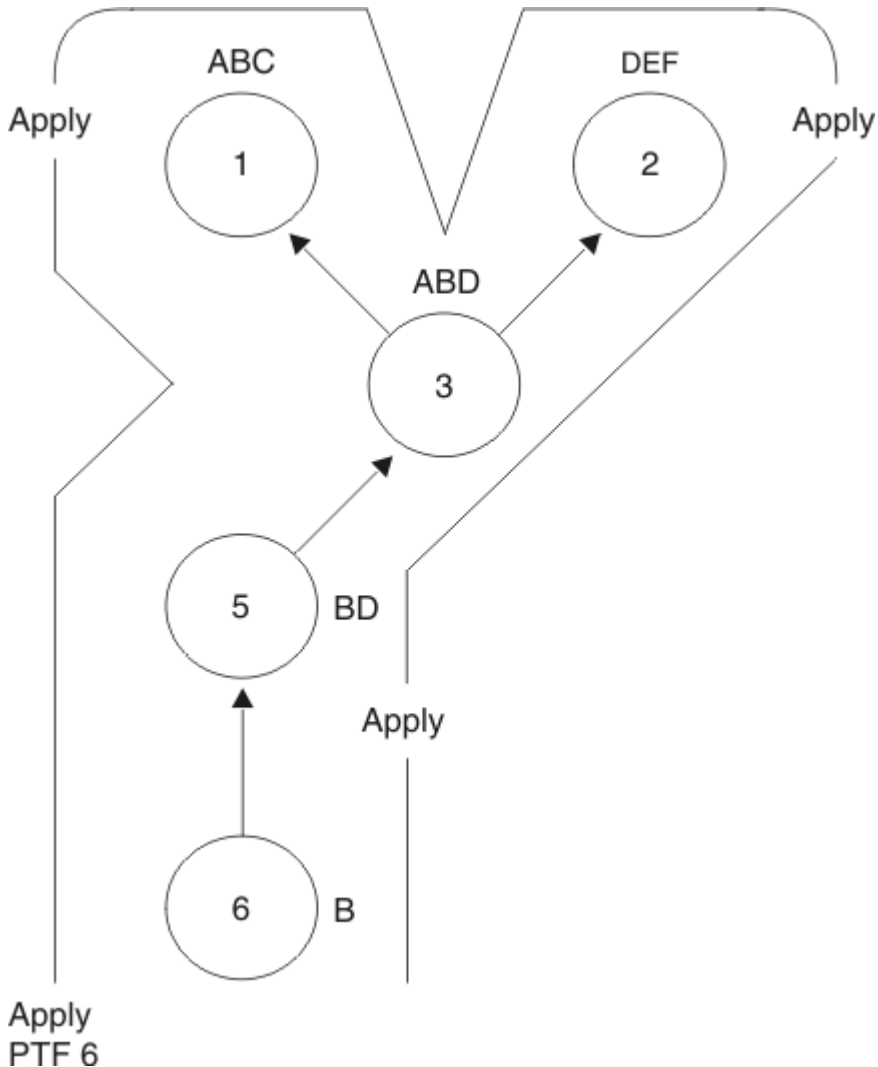


Figure 69. The Apply Algorithm

Each circle in the diagram represents a program temporary fix (PTF). The characters above the circles represent the parts affected by the PTF. Each arrow represents a requisite relationship (for example, PTF 6 requires PTF 5). The arrow encircling the diagram represents the apply algorithm, which is a post order traversal of the graph.

In the lower left corner of Figure 69 on page 110, you can see we want to apply PTF 6. Before a PTF can be applied, however, two questions must be answered:

- Has this PTF already been applied?
- Does it have any requisites?

If the PTF has already been applied, it does not need to be applied again. If the PTF has requisites, they must be applied before the PTF can be applied.

In [Figure 69](#) on [page 110](#), if PTF 6 is to be applied, its requisite, PTF 5, must be processed first. Again, the same two questions must be answered for PTF 5. Has it already been applied? Does it have any requisites? PTF 5 has not been applied, and PTF 5 requires PTF 3.

This same process continues until you reach a PTF that has no arrows leaving it, which indicates there are no requisites or all of its requisites are already applied. In this example, PTF 1 is the first terminal node. It has no requisites, so it can be applied. To apply it, each of the parts affected by it must be applied (parts A, B, and C). When PTF 1 has been applied, PTF 2 can then be applied. Following the post order traversal, PTFs 3, 5, and, finally, 6 are then applied.

Version Vector Table - PTF					Apply Status Table	
A			3	1		1
B	6	5	3	1		2
C						3
D		5	3	2	1	4
E				2		5
F				2		6

*Figure 70. Software Inventory Tables Updated During VMFAPPLY Processing*

[Figure 70](#) on [page 111](#) shows a high-level view of the version vector table and apply status table for the example shown in [Figure 69](#) on [page 110](#). The apply status table identifies the fixes that have been successfully processed. The apply status table starts out empty, but entries are added as PTFs are successfully applied. The version vector table identifies the individual parts that have changed and the PTFs that define the changes.

The VMFAPPLY EXEC is described in detail in [“VMFAPPLY EXEC”](#) on [page 291](#). For more detailed examples of the contents of these Software Inventory tables, see [Chapter 15, “Introduction to the Software Inventory,”](#) on [page 163](#).

## Software Inventory Files Used by the VMFAPPLY EXEC

### Apply Status Table

The **apply status table** is a list of PTFs that have been applied for a product. A PTF can have two different states in this table: APPLIED or SUPED. When a PTF is flagged as SUPED, it means that the PTF has been superseded by another PTF. A superseding PTF includes all of the fixes and all of the requisite relationships from all of the PTFs that it supersedes.

In addition to the status of the PTF, the table includes the date and time at which it was applied or superseded and the user ID from which the operation was performed.

### Version Vector Table

Version vector tables contain the service history for parts. They resemble AUX files in many respects. In fact, version vector tables are pointed to by a control file in the same way that AUX files are. Both PTF and APAR numbers are included in this table.

### Select Data File

The select data file is not really part of the Software Inventory, but it is important enough to warrant mention here. VMFAPPLY updates this file with a list of the parts affected by service, separated by time and date stamps to indicate when they were serviced.

The Software Inventory files used by the VMFAPPLY EXEC are described in [Chapter 15, “Introduction to the Software Inventory,”](#) on page 163 and [Chapter 22, “Software Inventory Syntax,”](#) on page 659.

## A Closer Look at the VMFBLD EXEC

---

The VMFBLD EXEC builds usable objects at the latest service level from the raw materials in the service database. First, the VMFBLD EXEC reads select data files and the build lists to determine which objects have been serviced since its last invocation. After it has identified all of the serviced objects, it updates the build status table with the results. This function is known as the VMFBLD status function. The status function is always the first function performed by VMFBLD.

Second, VMFBLD calls part handlers to build the objects. The part handlers select the correct level of parts and generate the usable form. Each time an object is built, VMFBLD changes the status of the object to BUILT in the service-level build status table.

The VMFBLD EXEC also lets you build a list of objects or a specific object by using command line parameters. When you need to build a single object or build list, include it on the command line and only that object will be built.

### Requisite Processing

VMFBLD uses object requisites information to determine extended build requirements. The user specifies the primary requirements on the VMFBLD command by entering the names of build lists and objects. VMFBLD adds all of that object's requisites that do not have a status of BUILT to the processing list. VMFBLD then searches the requisites for these newly added objects and adds additional objects.

VMFBLD status updating is influenced by object dependents. As parts are updated, information is stored in the select data file. During VMFBLD status processing, this information is used to match parts to objects and update object status to show that build processing is required. The status for both the immediate and extended dependents of these objects are similarly updated.

Updating the object status does not mean that these objects are built during this invocation of VMFBLD. Only objects specifically requested to be built and their non-BUILT primary and extended requisites are submitted to VMFBLD build processing.

The VMFBLD EXEC is described in more detail in [“VMFBLD EXEC”](#) on page 305. For more information on the part handlers, see [“Creating Objects with VMFBLD”](#) on page 325. For more information on build lists, see [“Build Lists”](#) on page 112.

## Software Inventory Files Used by the VMFBLD EXEC

### Build Status Table

The **build status table** is a list of objects that have been affected by service for a product. The objects are identified by both the build list name and the object name because the same object name can appear in multiple build lists. An object can have different states in this table: MANUAL, SERVICED, BUILDALL, BUILT, BYPASSED, DELETE, and DELETED.

In addition to the status of the object, the table includes the date and time at which it was serviced or built and the user ID from which the operation was performed. An error qualifier may also appear after the date, time, and user ID. When .ERROR appears in the build status table, it indicates an error was encountered when the object was being built. For a complete description of the build status table, see [“The Service-Level Build Status Table \(bldid SRVBLDS\)”](#) on page 717.

### Build Lists

**Build lists** contain the definitions of the objects that make up a product. There are three build list formats.

Format 1 build lists are in EXEC or EXEC2 format. Nucleus load lists are examples of format 1 build lists. A format 1 build list can only define *one object*.

Format 2 build lists are tagged files. They can contain *multiple objects*, as well as a wider variety of parameters and options that may be required to build the objects.

Format 3 build lists support *libraries*. Each library member is defined as an object in a format 3 build list.

**Note:** When VMFBLD processes build lists, it uses their latest PTF-numbered levels. Therefore, if you modify the usable form version of the build list (the one with a file type of EXEC), the modification is not picked up. You must add any local modifications to build lists to the Software Inventory.

For more information on build lists, see [“Build Lists” on page 141](#).

## Saved Segment Data File

The saved segment data file contains customized information for building the saved segments defined in a system saved segment build list. The saved segment data file is updated or created by the VMFSGMAP EXEC.

## Select Data File

Although not part of the Software Inventory, this file contains a list of all the parts affected by service, identified with time and date stamps. VMFBLD uses this file to determine if any new service has been applied by keeping track of the time and date stamps. VMFBLD also uses this file to determine which objects must be rebuilt as a result of service.

The Software Inventory files used by the VMFBLD EXEC are described in Chapter 15, [“Introduction to the Software Inventory,” on page 163](#) and Chapter 22, [“Software Inventory Syntax,” on page 659](#).

## System-Level Product Inventory Table

The system-level Product Inventory table specifies which products are installed on which systems or members. It also identifies any products that are superseded by a newer level of the product installed on a system or member. For more information, see [“The System-Level Product Inventory Table \(VM SYSPINV\)” on page 696](#).

## System-Level Base APAR Table

The system-level Base APAR table contains a list of all APARs included in the base of all supported z/VM products/components. For more information, see [“The System-Level Base APAR Table \(VM SYSAPARS\)” on page 701](#).

## Other VMSES/E EXECs

---

In addition to the primary service functions, VMSES/E provides tools to:

- Build additional raw materials, which include local modifications (the [“VMFHASM EXEC” on page 387](#), [“VMFHLASM EXEC” on page 394](#), [“VMFNLS EXEC” on page 448](#), VMFREPL EXEC, and [“VMFEXUPD EXEC” on page 381](#)).
- Consolidate levels of the database (the [“VMFMRDSK EXEC” on page 444](#)).
- Compile the parameter file, which controls the operation of the other functions (the [“VMFOVER EXEC” on page 456](#) and [“VMFPPF EXEC” on page 458](#)).
- Manage the minidisks and SFS directories that make up the service database (the [“VMFSETUP EXEC” on page 500](#) and [“VMFQMDA EXEC” on page 469](#)).
- Manage objects (the [“VMFQOBJ EXEC” on page 472](#)).
- Manage saved segments (the [“VMFSGMAP EXEC” on page 506](#)).
- Verify the primary functions complete properly (the [“VMFVIEW EXEC” on page 614](#)).

## Consolidating Levels of the Database

The VMFMRDSK EXEC consolidates raw materials on the DELTA string and service levels on the APPLY string. The VMFMRDSK EXEC checks the Software Inventory for inconsistent states before it performs the merge. The VMFMRDSK EXEC also checks that all target disks have enough room on them to accept the files that are being moved to them.

Because the APPLY string contains the files that define service levels, a merge of the APPLY string has the effect of consolidating service levels. Because the DELTA string is merely a repository for PTFs, the merge function can be used as a stager. New PTFs can be staged on the alternate disk and then moved onto the production disk where they will be stored permanently.

Figure 71 on page 114 illustrates a typical merge of the APPLY string.

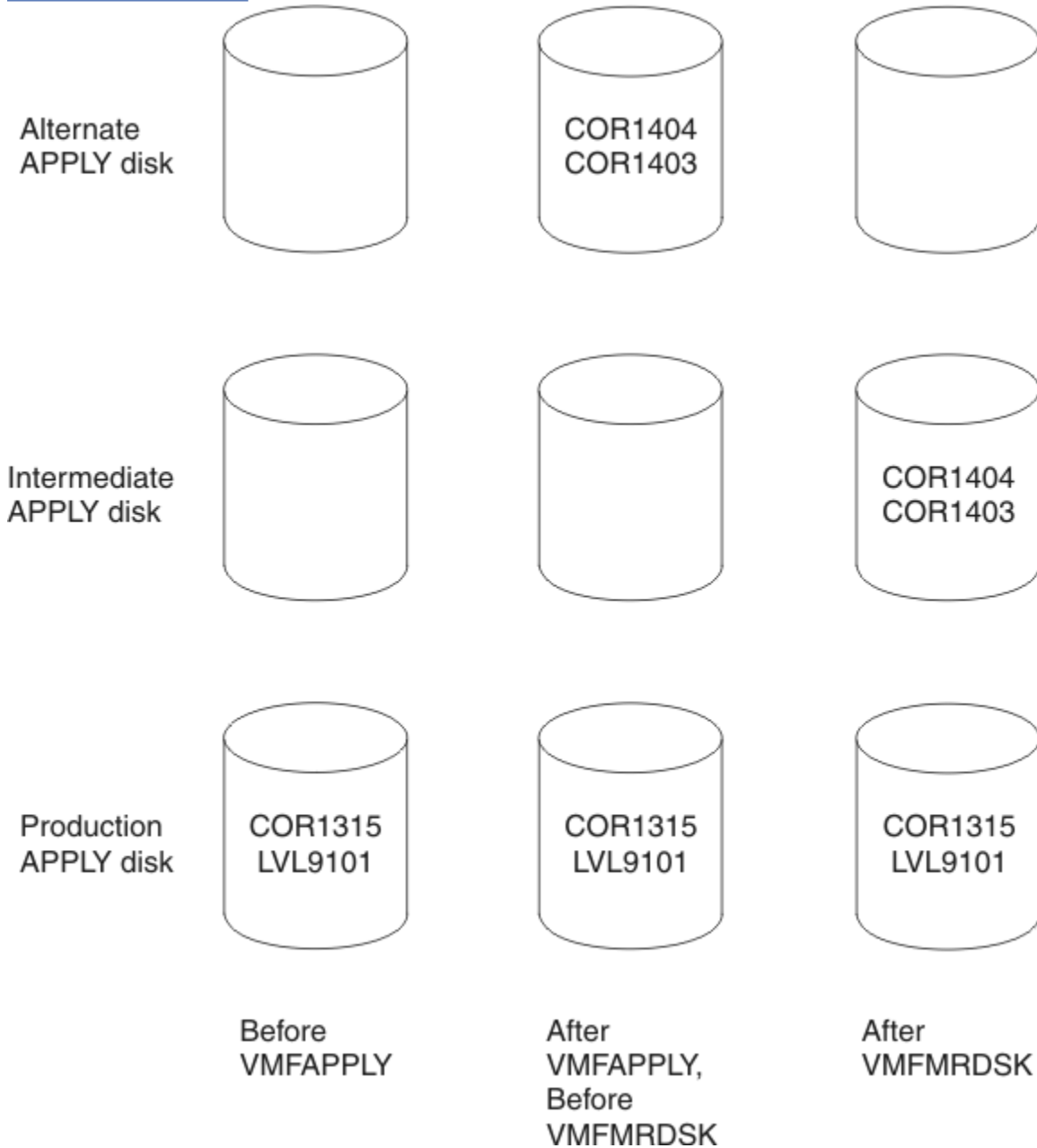


Figure 71. VMFMRDSK EXEC Example

At some point in time, the production service level on the APPLY string is made up of a service level (9101) and a COR tape (1315). When two new COR tapes arrive (1403 and 1404), they are applied together; and the resulting maintenance level is stored on the alternate APPLY disk. Once this new



maintenance level has passed some level of acceptance testing, it can be merged onto the intermediate APPLY disk, freeing up the alternate APPLY disk for the next batch of service.

## Managing Product Parameter Files

VMSES/E uses the VMFOVER and VMFPPF execs to compile the product parameter files.

### The VMFOVER EXEC

The override function (VMFOVER EXEC) applies overrides (context oriented updates) to source product parameter files (\$PPFs). The resulting temporary PPF retains the file name which was entered on its command line (that is, the override that is farthest from the source PPF in a chain of multiple overrides).

### The VMFPPF EXEC

The VMFPPF function is analogous to the assembly functions. It calls an update facility (in this case the VMFOVER EXEC) and generates the usable form PPF. In general, the term PPF refers to this usable form product parameter file.

Figure 72 on page 115 illustrates the function of the VMFPPF EXEC.

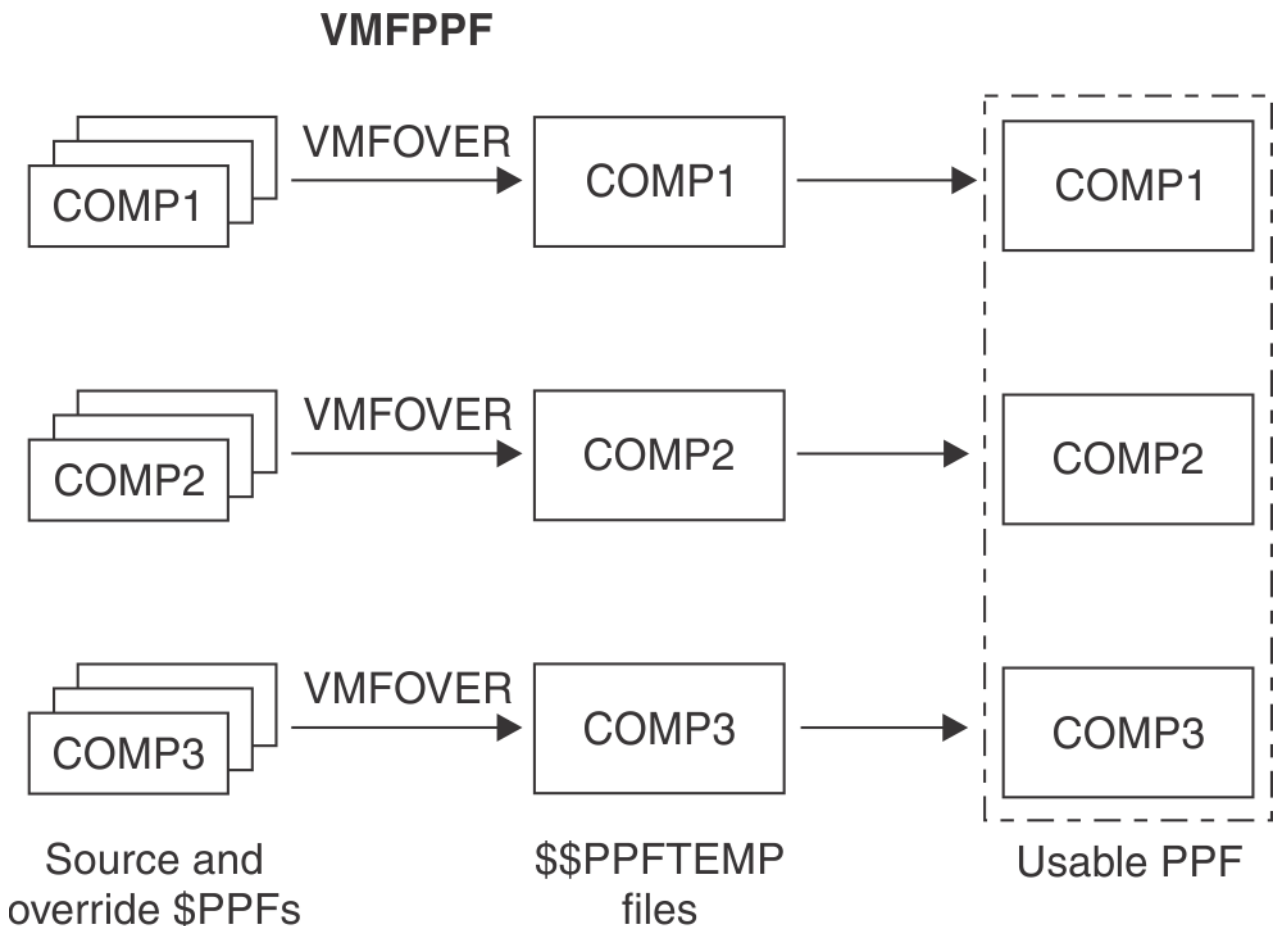


Figure 72. VMFPPF EXEC Examples

In addition to its generation function, the VMFPPF EXEC also validates the syntax of the PPF and performs some variable substitution for variables defined in the :DCL section of the PPF and used in the :MDA section of the PPF. The usable form PPF is really a collection (or library) of usable form PPFs. The correct usable form PPF is addressed by using the file name of the PPF and the component name of the product or component.

## Managing Disks for the Service Database

---

VMSES/E uses the VMFSETUP and VMFQMDA execs to manage minidisks and Shared File System directories for the service database.

### The VMFSETUP EXEC

The VMFSETUP EXEC is the tool that accesses the minidisks and SFS directories necessary to install or service a product. The VMFSETUP EXEC uses the same access order for all VMSES/E functions for a given product. You can enter the VMFSETUP EXEC once at the beginning of a session and not have to incur the overhead of constantly changing the access order.

The VMFSETUP EXEC performs links if they are necessary and desired. The links are defined in the :DCL section of the PPF. This function can also detach the disks that are linked when they are no longer needed.

### The VMFQMDA EXEC

VMFQMDA displays the current access order as it relates to a given PPF. In other words, the VMFQMDA EXEC reads the :MDA section of the specified PPF and compares it to the current access order. The output is formatted to mimic that of the CMS QUERY ACCESSED command, except that it reorders the disks by disk string and includes the names of the disk strings in the output.

## Managing Objects

---

VMSES/E uses the VMFQOBJ EXEC to manage objects.

### The VMFQOBJ EXEC

VMFQOBJ provides information on objects that are defined in build lists. For example, you can obtain the following information for objects:

- Status
- Library name
- Build requisites
- Build dependencies
- Global libraries
- Part handlers
- Target
- Build list options
- Parameters
- Serviceable parts included in an object
- Part options

See [“VMFQOBJ EXEC”](#) on page 472 for more information on this command.

## Managing Saved Segments

---

VMSES/E uses the VMFSGMAP EXEC to manage the saved segment definitions associated with a system saved segment build list.

### The VMFSGMAP EXEC

VMFSGMAP processes and displays the saved segment definitions contained in the saved segment data file associated with a system saved segment build list. VMFSGMAP also displays information about

saved segments defined on the system that are not defined in the saved segment data file. The primary VMFSGMAP display is a segment map that shows information such as:

- All the segment spaces in which a particular member resides
- All the members contained in each segment space
- Overlapping members in segment spaces
- Gaps in segment spaces
- Saved segment storage ranges that are not valid

Using VMFSGMAP functions, you can change, add, and delete saved segment definitions and display the results in the map before you build the saved segments.

For more information about the VMFSGMAP command, see [“The Source Product Parameter File” on page 13](#). For information about defining, building, and managing saved segments, see [z/VM: Saved Segments Planning and Administration](#).

## Other VMSES/E Functions

---

VMSES/E provides additional functions to help you during the service process.

### Regenerating Parts Locally

To allow for local regeneration of parts that are supported by updates, VMSES/E provides a number of tools: the VMFASM, VMFHASM, VMFHLASM, VMFNLS, and VMFEXUPD execs. Each generates output, which is named properly for VMSES/E and includes self-documenting information that lists the service history for the output.

Each of the functions dynamically unpacks the source files when required. The unpacked source file is then discarded in order to reclaim the disk space, leaving the original source file in place. These functions also support all of the options of the primitive assembly and compilation commands (ASSEMBLE, HASM, HLASM, GENCMD, GENMSG, UPDATE, and EXECUPDT). The VMFASM EXEC can call the F, H, or HL assembler based on an option. The default is the F assembler.

### Viewing Message Logs

The primary VMSES/E functions log their messages in message logs. The VMFVIEW EXEC helps you read and interpret the message logs created by the primary VMSES/E functions. VMFVIEW supports message logs for the LOCALMOD, PUT2PROD, SERVICE, SERVMGR, VMFAPPLY, VMFBLD, VMFINS, VMFMRDSK, and VMFREC commands.

The VMFVIEW EXEC lets you locate the messages from a specific run of a VMSES/E function by including the product parameter file (PPF) name and component name on the command line. This lets VMFVIEW locate just the runs for that product. This can be very useful when processing service to multiple products concurrently. The logs from one product can be viewed after already having processed a second product.

For more information, see [“The Message Log” on page 137](#).

## How VMSES/E Uses Control Files

---

VMSES/E uses control files to ensure information is correctly selected during the service process.

### Control Files

A control file is a master file that ensures information is selected in a specific order for a set of serviceable parts. The information for serviceable parts can be grouped into levels by two types of secondary files:

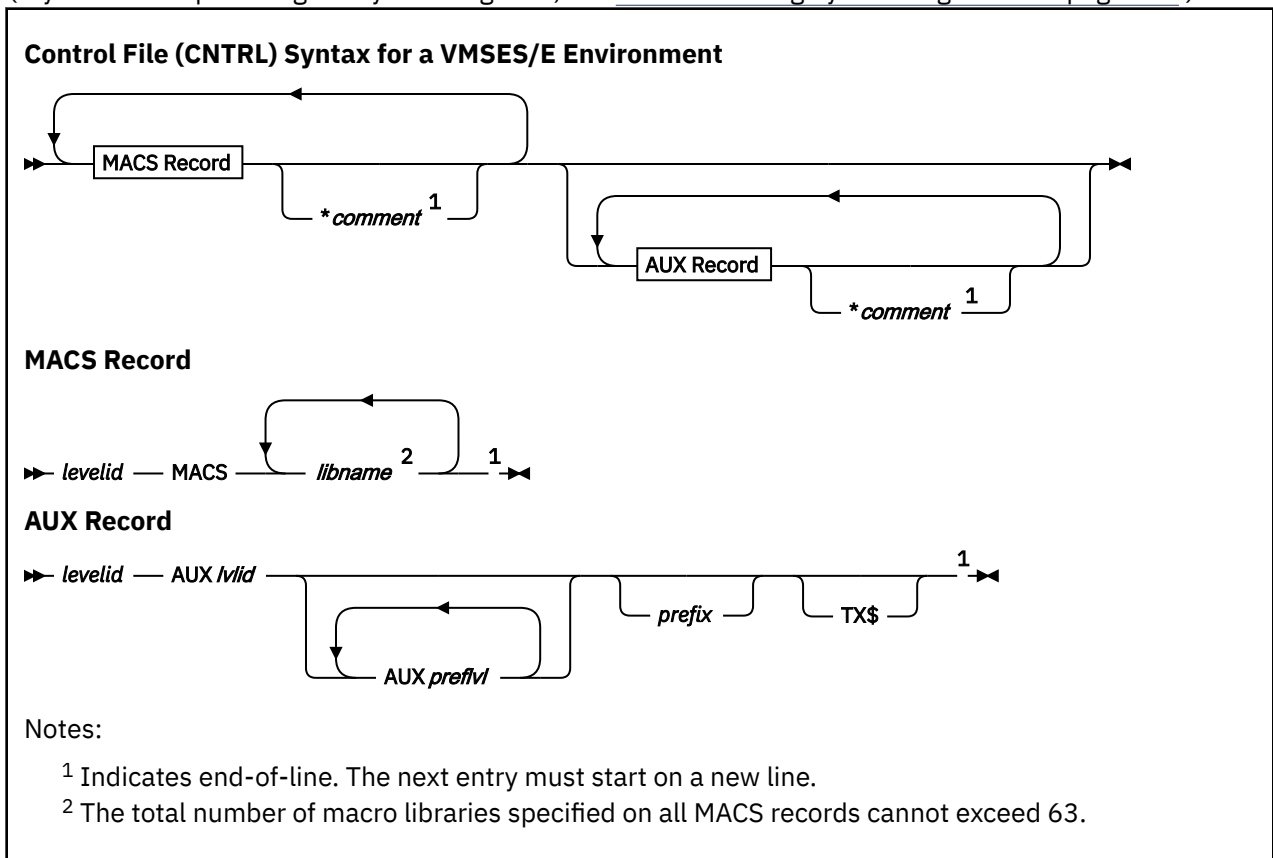
- Auxiliary Control Files (AUX)
- Version Vector Tables (VVT)

These secondary files identify update files, PTF numbers, or local modification identifiers. The file type of these secondary files are identified by entries in the control file.

The control file is used by VMSES/E when:

- Source files are updated using the CMS UPDATE facility. The UPDATE function is used by XEDIT when new updates are created and by compile functions, such as VMFASM, VMFHASM, VMFHLASM, VMFNLS, and VMFEXUPD, when serviceable parts are generated.
- Output (produced by VMFASM, VMFHASM, VMFHLASM, VMFNLS, and VMFEXUPD) is named. The file type of the serviceable parts produced by these functions depends on the information in the control file and the version vector table. If a version vector table does not exist, AUX files are used.
- Selecting serviceable parts when generating new usable forms during build processing.
- Required MACLIBs are identified for text deck assembly.
- Level information is updated while applying new service using VMFAPPLY.
- Patches are applied to text decks.

“Control File (CNTRL) Syntax for a VMSES/E Environment” on page 118 shows the syntax of a control file. (If you need help reading the syntax diagrams, see “Understanding Syntax Diagrams” on page 227.)



## MACS Record

Macro library (MACS) records contain the following:

### levelid

is an update level identifier. In a VMSES/E environment, this value, combined with a file type abbreviation, is used as the key into the control file extension table (see “Control File Extensions” on page 119).

### MACS

indicates this record contains the names of MACLIBs that are used by the assemble functions (VMFASM, VMFHASM, VMFHLASM, and VMFNLS). MACS is a keyword.

**libname**

is the name of a MACLIB.

**AUX Record**

AUX records contain the following:

**levelid**

is an update level identifier. In a VMSES/E environment, this value, combined with a file type abbreviation, is used as the key into the control file extension table.

**AUXlvlid**

identifies the file type of an AUX file or a version vector table. The file type of a version vector table has a VVT instead of the characters AUX. *AUXlvlid* is also the file type of an AUX file for parts supported with source updates.

**AUXprefvl**

identifies the file type of a preferred AUX file or version vector table. The file type of a version vector table has a VVT instead of the characters AUX. *AUXprefvl* is also the file type of a preferred AUX file for parts supported with source updates.

Preferred version vector tables and preferred AUX files are used to conditionally select information from the version vector table that is identified by *AUXlvlid*. If there is an entry in the preferred version vector table for a particular part, the information in the corresponding *AUXlvlid* version vector table is ignored.

**prefix**

is the file type abbreviation used for a part. You can use this as an alternative to using a control file extension table (CONTRLEXT) for TEXT parts.

**TX\$**

indicates that this *AUXlvlid* points to text patch files instead of source update files. The text deck prefix, *prefix*, and the patch indicator field, TX\$, can appear in either order in the command.

**Comments**

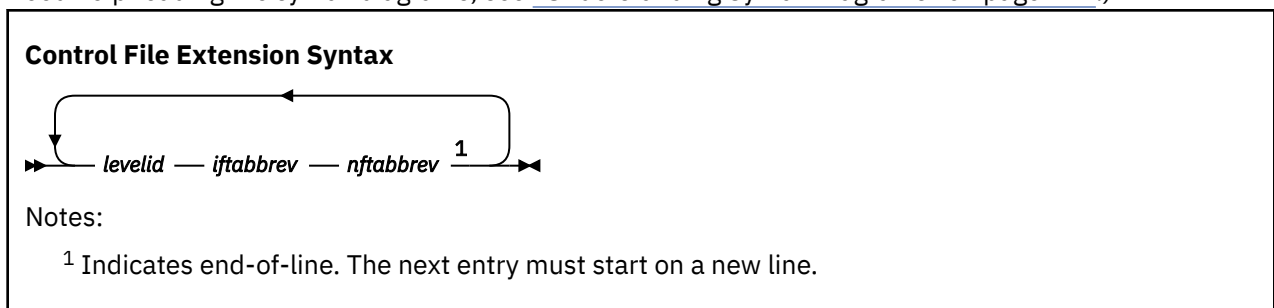
Comments are allowed on MACS records and AUX records. Comments begin with an asterisk (\*).

**Control File Extensions**

The control file extension (*cntrlfn* CONTRLEXT) is an optional file that is used to change the file type abbreviation for a part when a particular *AUXlvlid* is selected. The control file extension table is used when multiple types of systems can be generated or when additional functions are added to a product by another product. (In z/VM, several security products add functions to the CP component.) You must use this table when parts exist in each type of system, and the parts have the same file name.

The file name of the control file extension table must match the file name of the control file.

“Control File Extension Syntax” on page 119 shows the syntax for the control file extension table. (If you need help reading the syntax diagrams, see “Understanding Syntax Diagrams” on page 227.)



The control file extension table contains the following:

**levelid**

is an update level identifier. In a VMSES/E environment, this value, combined with a file type abbreviation, is used as the key into the control file extension table.

**iftabbrev**

is the file type abbreviation for the input file. In a VMSES/E environment, this value, combined with a *levelid*, is used as the key into the control file extension table.

**nftabbrev**

is the new file type abbreviation for the specified *levelid* and *iftabbrev* key.

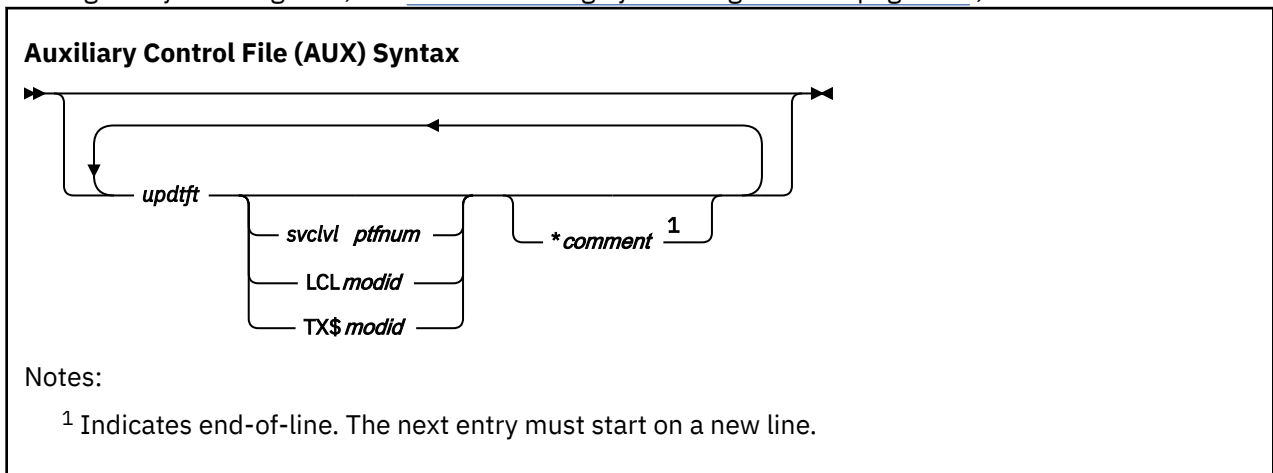
The file type abbreviation for a part is changed if the following are true:

- There is an entry for the part in the version vector table that is identified by *AUXlvlid*.
- There is an entry for the part in the control file extension table that uses a key composed of the *levelid* and file type abbreviation for the part.

## Auxiliary Control Files

Auxiliary control files are used as pointers to update files.

“Auxiliary Control File (AUX) Syntax” on page 120 shows the auxiliary control file syntax. (If you need help reading the syntax diagrams, see “Understanding Syntax Diagrams” on page 227.)



The auxiliary control file contains the following:

**updtft**

is the file type of the source update. The *updtft* must be in uppercase.

**svclvl**

is the service level that contains the program temporary fix (PTF). The *svclvl* must be in uppercase.

**ptfnum**

is the program temporary fix (PTF) that contains the source update. The *ptfnum* must be in uppercase.

**LCL**

indicates the source update is a local modification.

**TX\$**

indicates the update file contains TEXT patches.

**modid**

is a 7-character local tracking number assigned to this modification. The first two characters indicate a local tracking number follows. It is recommended that the first two characters of the local tracking number be LC. Characters 3-7 are a 5-character identifier for your local modification, which you can create according to your own tracking scheme. It is recommended that the first character be an L. Characters 3-7 of *modid* are concatenated to the *ftabbrev* to form the file type of the serviceable part that is associated with this modification level of the part. For example, LCL1234 is a local modification tracking number. In this example, L1234 would be concatenated to the *ftabbrev* to form the file type of the serviceable part. The *modid* must be in uppercase.

**We recommend you start the local tracking number with LCL to ensure it does not interfere with service delivered by IBM. If you use characters other than LCL, make sure they are unique for your product.**

**\***

indicates the beginning of a comment. You must use an asterisk to delimit the beginning of a comment, unless the comment follows a three-token update record. Even in this case, the asterisk is strongly recommended.

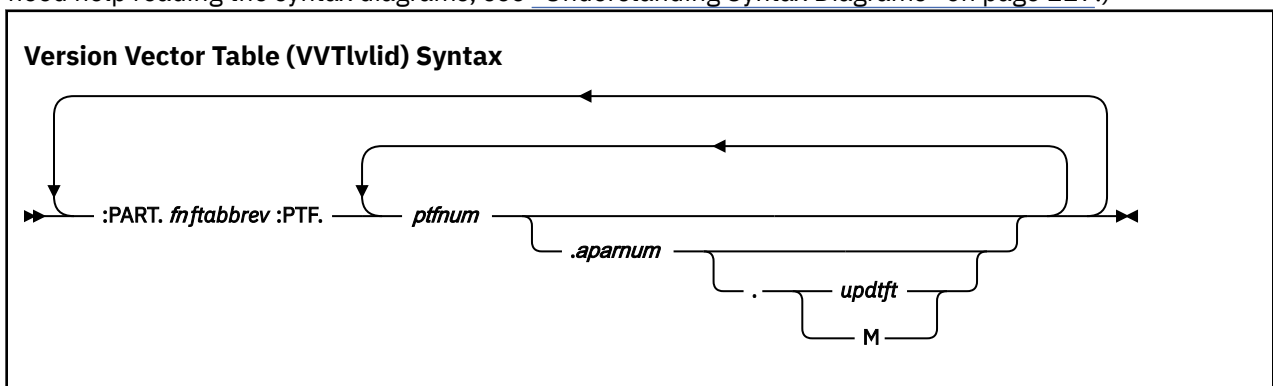
**comment**

is the comment you want to enter.

## Version Vector Tables

The version vector table contains a history of all PTFs that have been applied to a product at a specific maintenance level.

“Version Vector Table (VVTlvlid) Syntax” on page 121 shows the syntax of the version vector table. (If you need help reading the syntax diagrams, see “Understanding Syntax Diagrams” on page 227.)



For a complete description of the syntax of this file, see “The Version Vector Table (appid VVTlvlid)” on page 721.

## Patch Update Files

A patch update file contains an emergency circumventive fix for text decks which are used to build a nucleus or a module. The patch function is included in the VMFBLD EXEC and is called prior to the loader invocation. The patch function uses the information in the patch update file to modify the specified text deck. The update file could be shipped to you on a special tape, or you might have to create the file yourself under the direction of IBM.

The file name of a patch update file must match the file name of the text deck to be patched by the loader. The file type is the full APAR number, VMmmmmm. [Figure 73 on page 121](#) shows an example of a patch update file.

```
./ * * PREREQ: VM23546 VM24484
./ * * CO-REQ: NONE
./ * * IF-REQ: NONE
./ * * comment
./ * NAME csectname
./ * VER disp hexdata
./ * REP disp hexdata
./ * REP disp hexdata
```

Figure 73. Example of a Patch Update File, DMKMNT VM12345

**comment**

is any comment that you want to add. For example, you may want to explain the reason for the patch.

**NAME csectname**

identifies the control section that is to be patched. If specified, this name must match the SD name from the ESD for the text being patched. If this statement is omitted, the control section with the same name as the patch update file is patched.

**VER**

identifies a verify control statement. A VER statement must precede the first REP statement.

**REP**

identifies a replace control statement, which allows instructions and constants to be changed and additions made. The first REP statement must be preceded by a VER statement.

**disp**

is the four-digit hexadecimal starting address of the area to be verified or replaced as assigned by the assembler. Unused leading columns must be filled with zeros.

**hexdata**

is up to 11 four-digit hexadecimal fields separated by commas, each verifying or replacing (VER or REP) a previously loaded halfword (2 bytes). Unused leading columns in each field must be filled with zeros. The last field must not be followed by a comma.

### Creating Updated Source Files

The CMS UPDATE command is used to modify source files. The UPDATE command accepts a source input file and, optionally, files containing UPDATE control statements, updated source records, and comments. Using this input, it creates an updated source file. When using VMSES/E, AUX files are used to point to update files. Figure 74 on page 122 shows how we create serviceable parts, the control file structure, and examples of these items.

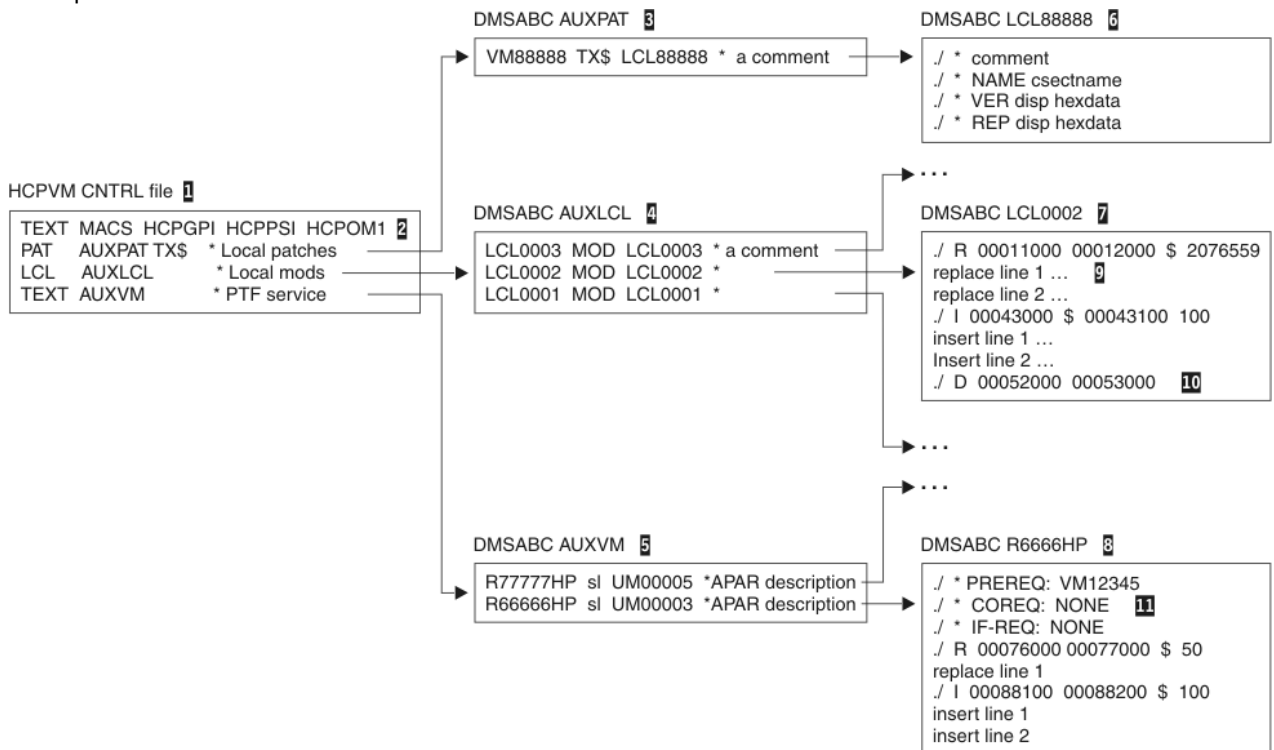


Figure 74. Control File Structure

- Item 1 is a control file. Its primary purpose is to list the levels of auxiliary control files (AUX files). It also contains a MACS record (item 2) that lists the macro libraries that must be made available using the CMS GLOBAL command when you assemble text decks. In the control file, the patch indicator (TX\$) indicates that the AUX file points to patch control statements.
- Items 3, 4, and 5 are AUX files. Item 3 is a local patch AUX file, item 4 is for local service, and item 5 is for PTF service.



- Item **6** is a local patch update file. It contains patch control statements that are input to the loader function. Because they are considered comments by the UPDATE command, they are ignored.
- Items **7** and **8** are regular update files. They contain update source records (**9**), update control statements (**10**), and comments (**11**). The comments in item **11** contain requisite information that is included in some text decks when they are assembled. This information is referred to as self-documenting information and is used to validate the level of the text deck when the CKSDI option is specified for VMFAPPLY.

## Selecting the Latest Version of the Serviceable Part

The build list defines the serviceable parts that are contained in a usable form, and the build function generates the usable form from the serviceable parts. The control file and version vector table files define the version of the serviceable parts that are to be used when the usable form is generated. (AUX files are used, if a version vector table does not exist.) The control file points to the allowable version vector tables. These tables are searched, beginning at the bottom entry, by VMFSIM GETLVL. When VMFSIM GETLVL finds a version vector table entry for the serviceable part, it determines the file type based on the highest version of the serviceable part.

For example, in [Figure 75 on page 123](#):

- If there is a record for DMSDEF in 1VMVMC23 VVTVM (**1**) and this is the only version vector table that has an entry for DMSDEF, the file type for the resulting serviceable part is EXC00005. EXC is the file type of the serviceable part DMSDEF, and it is used as input to the GETLVL function. 00005 is from characters 3-8 of the first :PTF entry.
- If a record exists for DMSDEF in 1VMVMC23 VVTLCL (**2**), the resulting serviceable part file type is EXCL0003. EXC is the file type of the serviceable part DMSDEF. L0003 is from characters 3-8 of the first :MOD entry.

If no entry can be found in any version vector table pointed to by the control file for a serviceable part, the base file type is used. This file type is obtained from the VM SYSABRVT table and the :DABBV section of the product parameter file. The base file type for EXC is EXC00000.

HCPVM CNTRL file

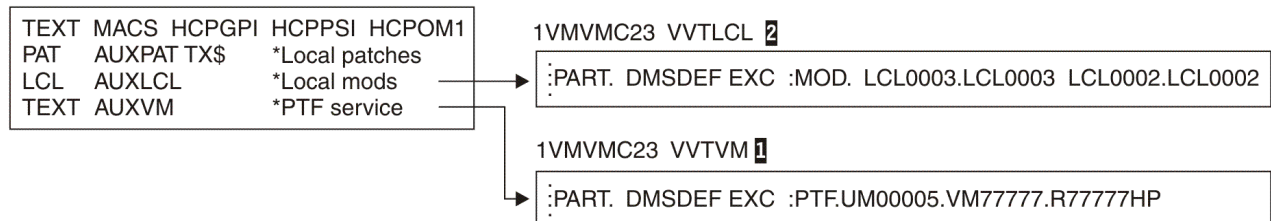


Figure 75. Control File Structure Using Version Vector Tables

## When No Version Vector Table Exists

When a version vector table does not exist, the file type of Text parts is determined in another way. A version vector table might not exist because a local modification was applied, and it was not logged in the Software Inventory.

### Important Note:

IBM strongly recommends you that log all local modifications in your local version vector table. For more information, see [Chapter 11, “Installing Local Service and Modifications,” on page 101](#).

VMSES/E uses information from the AUX files when version vector tables do not exist. Within the control file, AUX files are processed from the bottom of the file. Update records within the AUX files are also processed from the bottom of the file. The highest update is the last update processed.

For example, in [Figure 74 on page 122](#):

- If AUXVM is the only AUX file for DMSABC, the file type of the resulting text deck is TXT00005. TXT is the default prefix. 00005 is from characters 3-8 of the third token of the last applied update record (5).
- If AUXLCL exists and it is the highest AUX file for DMSABC, the file type of the resulting text deck is TXTL0003 (4). TXT is the default prefix. L0003 is from characters 3-8 of the third token of the last applied local modification.
- If the third token is missing, the resulting file type is TXT`levelid`. TXT is the default prefix, and `levelid` is from the first token of the AUX file record of the control file. When `levelid` is TEXT, the file type of the resulting text deck is TEXT.
- If AUXPAT exists and it is the highest AUX file for DMSABC, the file type of the resulting text deck is TXTPAT. The third token is not used for patches.
- If no AUX files exist for DMSABC, the file type of the resulting text deck uses the first token on the MACS record of the control file and appends it to TXT. Again, if this token is TEXT, the file type is TEXT.

## Creating Text Decks

When text decks are assembled, the name of the resulting text deck is based on the version of the serviceable part. The file type is determined using information provided by the VMFSIM GETLVL function.

The rules for naming text decks, which have just been described in the previous section, are used when you specify the PPF option for the VMFASM, VMFHASM, VMFHLASM, and VMFNLS commands.

VMFASM, VMFHASM, VMFHLASM, and VMFNLS are assemble functions that:

- Update a source file
- Assemble the updated source file
- Name the resulting text deck
- Log local modifications in the local version vector table

When you create text decks for a VMSES/E environment, you must use the PPF option. Use the LOGMOD option to automatically log local modifications in the local version vector table.

## Identifying MACLIBs

The VMFASM, VMFHASM, VMFHLASM, and VMFNLS execs call an assemble function that uses the information in MACLIBs. This information is made available to the assemble function by the CMS GLOBAL command. The names of the required MACLIBs, which are the input to the GLOBAL command, are identified on the MACS record in the control file (see item 2 in Figure 74 on page 122).

For more information on the MACS record, see [“Control File \(CNTRL\) Syntax for a VMSES/E Environment” on page 118.](#)

## Determining Local Modifications Requiring Rework

The VMFAPPLY EXEC updates the PTF level information for serviceable parts affected by PTFs in the specified apply list. The level information is stored in the version vector table specified by the :UPDTID tag in the product parameter file. If the serviceable part is supported with AUX files, the AUX file is created from the updated information in the corresponding version vector table.

Any version vector table or AUX levels that are higher than the level specified on the :UPDTID tag are considered to be local modifications that may need to be reworked as a result of PTF service. The VMFAPPLY EXEC detects this situation and informs you of any parts requiring rework.

---

## Chapter 13. Other Files Used in the Service Process

In addition to the product parameter file, the files used to maintain the control structure, and the Software Inventory, the product service process uses many other files, including the:

- Tape document
- Tape descriptor file
- Product contents directory
- Memo-to-Users
- Program level file
- Select data file
- VMSESE PROFILE
- Apply list
- Exclude list
- Place into production files
- Message log
- VMFINS DEFAULTS file
- Build lists
- National language support table

Some of these files, which are described in the following sections, are supplied with the base product, some are shipped as part of service, and others are generated by VMSES/E. In certain situations, you might have to create some files locally.

The product parameter file is described in [“The Source Product Parameter File” on page 13](#) and [Chapter 21, “Product Parameter File Syntax,” on page 621](#).

Files used to maintain the control structure are described in [Chapter 8, “z/VM Service Concepts,” on page 89](#) and [Chapter 12, “Using VMSES/E for Service,” on page 105](#).

The Software Inventory is described in [Chapter 15, “Introduction to the Software Inventory,” on page 163](#) and [Chapter 22, “Software Inventory Syntax,” on page 659](#).

If you need help reading the syntax diagrams in any of these sections, see [“Understanding Syntax Diagrams” on page 227](#).

---

### The Tape Document

The tape document describes the service procedure for COR service. It lists the steps involved in applying service to the system. The document is named COR DOCUMENT. Read the tape document before you begin to apply service.

---

### The Tape Descriptor File

The tape descriptor file contains a directory of the products for which service is contained on the service tape. There is a tape descriptor file for installation tapes, RSU tapes, and COR service.

### For Installation, RSU, and COR Descriptor File

The first record of the tape descriptor file states what type of tape it applies to: INS or COR. The second record tells how many volumes the tape has and which volume this is. If there is more than one volume, the tape descriptor file appears on each volume, but still describes the whole logical tape.

## Other Files Used in the Service Process

The other records are the multi-volume directory. A :VOL $nn$  record indicates the beginning of each volume. Each of the 3-word records following a :VOL $nn$  record describes a group of tape files. (A tape file is the data between two tape marks. It may contain more than one CMS file.)

The first record following the :VOL $nn$  tag lists the tape header files. This record is identified by the word  $XXX$  where  $XXX$  is INS or COR. The header files are the two standard tape files that begin every volume of the COR, RSU, or installation tape. One contains the tape descriptor file and the other contains informational files about all the products for which service is supplied on the tape.

In the other records that follow the :VOL $nn$  record, the first word is the *prodid*, the second word is either HDR (meaning a product header file) or the name of the component to which the files apply, and the third word is always the number of files in the group.

The *prodid* matches the value of the :RECID tag, and the component name matches the value of the :BCOMPNAME tag, both of which appear in the product parameter file. The VMFREC EXEC uses these two values as indexes to position the tape.

A product may be entirely on a single volume, it may cross volumes at a product break, or a single product may span multiple volumes. When a product crosses a volume boundary, the break is indicated in the multi-volume directory by a :VOL $nn$  record.  $nn$  indicates the continuation volume.

## For Installation and RSU Descriptor File

The installation descriptor file is a directory of the products on the installation tape. The Recommended Service Upgrade (RSU) descriptor file is a directory of the products for which service is contained on the RSU tape. The installation and RSU descriptor files are called INS *yynn*. *yynn* identifies the level of the tape. The following table shows an example of the installation and RSU descriptor files.

The VMFINS EXEC receives these files as part of its processing. These files are not erased by VMSES/E, but you can erase them if you want to.

## For COR Descriptor File

The COR descriptor file is a directory of the products for which service is contained on the corrective service tape. It shows how many files are available for each product and where they are. The COR descriptor file is called COR *yymm*.  $y$  is the last digit of the year,  $m$  is the month (numbered 1–C in hexadecimal), and  $dd$  is the day. For example, 1A01 is October 1, 2021.

The VMFREC EXEC receives these files as part of its processing. These files are not erased by VMSES/E, but you can erase them if you want to.

Install/RSU Descriptor File	COR Descriptor File
<pre> INS <i>yynn</i>  VOL01 of 02 :VOL01. INS          FILES    02 <i>prodid1</i> HDR      02 <i>prodid1</i> CMS      03 <i>prodid2</i> HDR      02 <i>prodid2</i> CP       02 :VOL02. INS          FILES    02 <i>prodid2</i> HDR      02 <i>prodid2</i> CP       02 <i>prodid3</i> HDR      02 <i>prodid3</i> DV       03           </pre>	<pre> COR <i>yymm</i>  VM Corrective Service Tape <i>nnnnn nnnnn</i>   CSD <i>nnnnnnn yy/mm/dd hh:mm:ss.xxxxxx</i> VOL01 of 01 :VOL01. COR          FILES    02 <i>prodid1</i> HDR      01 <i>prodid1</i> CMS      03           </pre>

## The Product Contents Directory

There is a product contents directory for each VMSES/E formatted product serviced by COR, Install, or RSU. It lists the tape files supplied for the product. If the product crosses a tape boundary, the product contents directory is repeated on the next tape. The product contents directory can be considered as an expansion of the tape descriptor file. The descriptor file tells you how many files you have and where they are. The product contents directory also tells you what the files are. [Table 9 on page 127](#) shows an example of a product contents directory.

The file name of the product contents directory is the product identifier (*prodid*). On a COR tape, the file type of the product contents directory is \$CORymdd. *y* is the last digit of the year, *m* is the month (numbered 1–C in hexadecimal), and *dd* is the day. For an install or RSU tape, the file type of the product contents directory is \$INSyynn. *yynn* identifies the level of the tape.

**Note:** A product may be entirely on a single volume, it may cross volumes at a product break, or a single product may span multiple volumes. When a product crosses a volume boundary, the break is indicated in the product contents directory by a :VOLnn record. *nn* indicates the continuation volume.

Install/RSU Tape	COR Tape
<pre>prodid1 \$INSyynn  :VOL01. :CMS. AXLIST PARTLST UPDT  :VOL02. TEXT MODULE</pre>	<pre>prodid1 \$CORymdd  :VOL01. :CMS. AXLIST PARTLST DELTA</pre>

## The Memo-to-Users

You will receive a *Memo-to-Users* for each product that needs service. The memo contains instructions for servicing the product. If the product is not fully supported by VMSES/E, the memo may describe any special installation instructions, such as how to use a service exec. You should read the memo for each product before you begin to apply service.

The *Memo-to-Users* file is called *prodid* MEMO.

## The Program Level File

Program level files are included in both the second tape file and the first product header tape file on VMSES/E tapes. The file name of the program level file is the product identifier (*prodid*). The file type is defined as *Ovrmmns*, where:

**O**

is constant.

**v**

is the version number of the *prodid*.

**r**

is the release number of the *prodid*.

**m**

is the modification level of the *prodid*.

**nn**

represents the number of tape files. The number of tape files (*nn*) includes one of the tape header files, all of the product header files, and all of the product tape files.

**s**

is a Boolean flag that indicates the *prodid* is supported by VMSES/E with a product parameter file (1 - will indicate that it is supported by VMSES/E).

For example, if a COR tape contained service for the MYCOMP component and the service consisted of one *prodid* header file and seven *prodid* tape files, the program level file would have a file name of 1VMVMC23 and a file type of 0123081. 1VMVMC23 is the *prodid* for the MYCOMP component. The first zero is a constant. The next three digits are the version, release, and modification identifiers. The digits 08 indicate there are a total of eight *prodid* tape files. The final digit (1) indicates the *prodid* is supported by VMSES/E with a product parameter file.

The program level file contains a one-line description of the *prodid* and a copyright statement. [Figure 76](#) on [page 128](#) is an example of a program level file.

```
1VMVMC23 -- MYCOMP V1R2.3 Component
* CONTAINS IBM COPYRIGHTED MATERIALS *
```

*Figure 76. Example of a Program Level File - 1VMVMC23 0123081*

## The Select Data File

The select data file contains a list of the parts to be processed by the VMFBLD status function. This file is updated with a timestamp and a list of the parts that were serviced when the apply step was run. This file can also be updated when using VMFSGMAP, VMFASM, re-working local modifications and rebuilding segments.

There is also a special select data file that is updated by the VMFBLD EXEC to list system objects that contain parts that have been serviced. See [“Select Data File Used for System Objects”](#) on [page 130](#).

The select data file is called *appid* \$SELECT. The value of *appid* is specified on the :APPID tag in the product parameter file. The select data file is stored on a product’s APPLY string.

If there is more than one *appid* specified on the :APPID tag in the product parameter file, the first *appid* is the primary *appid* - the identifier for the primary select data file. Any additional *appid*s identify secondary select data files.

Secondary select data files allow you to get build requirements from an associated source. For example, some REXX parts are included in the CMS nucleus. When these REXX parts are serviced, the CMS nucleus must be rebuilt. To make sure they are rebuilt, the primary *appid* in the product parameter file is specified as 7VMCMS10 (CMS) and the secondary *appid* is specified as 7VMREX10 (REXX).

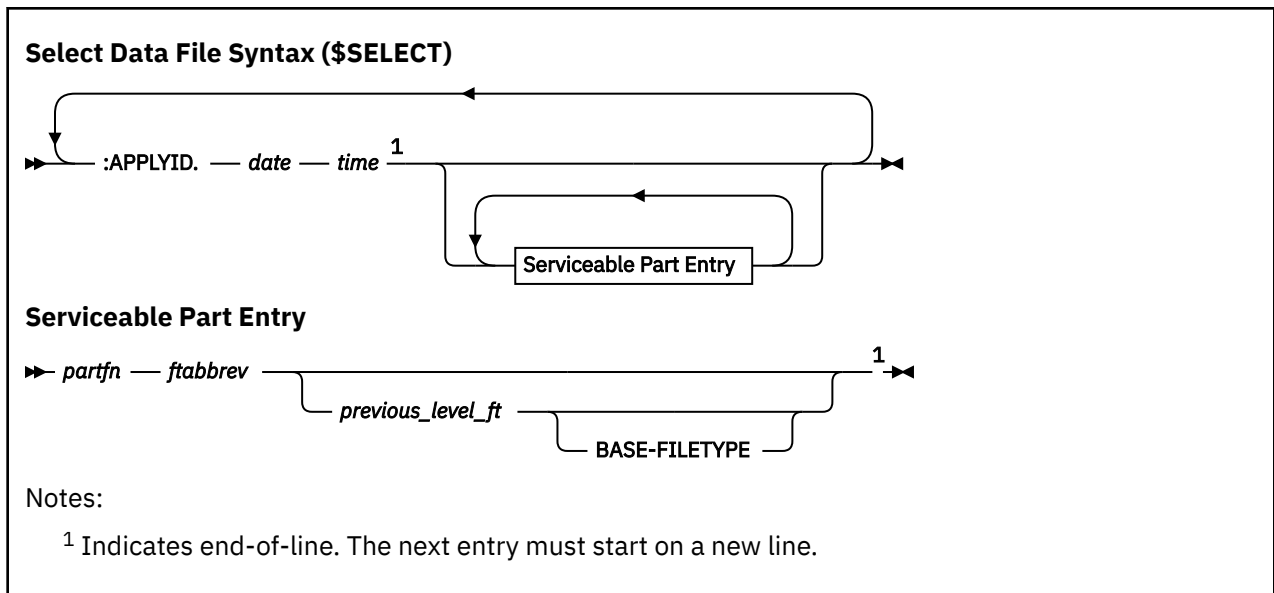
When processing the primary select data file, the VMFBLD EXEC checks the select data file to see when it was last processed and determines any new entries in the file. Objects that have been affected by a serviced part are updated to a status of SERVICED in the service-level build status table. If a serviced part does not exist in any build list for this component, the corresponding entry from the select data file is added to the :PARTID tag of the special build list named UNKNOWN with a status of MANUAL.

When processing each secondary select data file, VMFBLD checks the service-level build status table to see when each *appid* \$SELECT file was last processed and determines any new entries in the file. VMFBLD then checks the build lists of this component for any objects using the part. If VMFBLD cannot find any objects using the part specified in the secondary *appid* \$SELECT file, it ignores the part. When VMFBLD finds an object using the part, it updates the service-level build status table entry for that object with a status of SERVICED.

Finally, VMFBLD updates the :LASTAPP tag in the service-level build status table to show the level of the *appid* \$SELECT files processed.

## File Syntax

[“Select Data File Syntax \(\\$SELECT\)”](#) on [page 129](#) shows the syntax of the select data file. (If you need help reading the syntax diagrams, see [“Understanding Syntax Diagrams”](#) on [page 227](#).)

**:APPLYID.**

is a tag that precedes the information from a VMFAPPLY run.

**date**

is the date when VMFAPPLY was run. The date is in the format *mm/dd/yy*.

**time**

is the time when VMFAPPLY was run. The time is in the format *hh:mm:ss*.

**partfn**

is the file name of the part that was serviced.

**ftabbrev**

is the file type abbreviation for the part that was serviced. The *ftabbrev* must be the 3-character PTF abbreviation or the real CMS file type for parts that are not serviced by replacement.

**previous\_level\_ft**

is the file type of the previous level of the part that was serviced, for example EXC00001. The *previous\_level\_ft* is only provided when the part identified by *partfn* *ftabbrev* is a build list.

This information is used by VMFBLD to compare a new level of a build list to its previous level to determine if any object definitions have been added, changed, or deleted.

**BASE-FILETYPE**

is an identifier that is added to the select data file entry when the *previous\_level\_ft* is also the base level. See [Figure 77 on page 130](#) for an example of this type of entry.

## Example

[Figure 77 on page 130](#) is an example of the select data file.

```
:APPLYID. 06/04/22 23:25:00
DMSYLI TXT
DMSSETM TXA
DMSHMA EXC
CMSLOAD EXC EXC000000 BASE-FILETYPE
DMSMVS COPY
DMSFBO MACRO
:
:APPLYID. 06/03/22 13:25:00
DMSYLI TXT
DMSSETM EXC1VMVMC23
$PF
:
```

Figure 77. A Sample Select Data File

---

**Note:** The most recent entries in the select data file are added to the beginning of the file.

## Select Data File Used for System Objects

VMSBR \$SELECT is a special select data file that is used to identify system objects that need to be built because they contain parts that have been serviced. System objects are objects with shared build requirements, such as saved segments. They may contain objects or parts from more than one product.

When a part or object contained in a system object is built, the VMFBLD part handler VMFBDSBR updates VMSBR \$SELECT with the name of the product build list required to build the system object and issues a message that the build list contains objects that must be built. The minidisk or SFS directory location of VMSBR \$SELECT is indicated in the VMSESE PROFILE.

System objects that need to be built because their definitions on the system have been changed are identified in another select data file, SEGBLD \$SELECT. When the VMFSGMAP EXEC is used to tailor the definition of a saved segment (to change its storage range, for example), VMFSGMAP updates the SEGBLD \$SELECT file with a record that contains the name of the saved segment (*segname* DMY). The name of the select data file used by VMFSGMAP is identified on the :APPID tag in the saved segment product parameter file specified on the VMFSGMAP command. SEGBLD is the default.

Figure 78 on page 130 shows an example of the select data file (VMSBR \$SELECT) that results from service to parts contained in segments.

---

```
:APPLYID. 06/16/22 11:05:40
DMSSBPIP EXC
DMSSBSFS EXC
```

Figure 78. Example of the VMSBR \$SELECT File

---

Figure 79 on page 130 shows an example of the select data file (SEGBLD \$SELECT) containing entries that result from VMFSGMAP tailoring.

---

```
:APPLYID. 06/23/22 08:26:50
CMSBAM DMY
CMSDOS DMY
CMSVMLIB DMY
DOSINST DMY
:APPLYID. 06/01/22 08:33:34
CMSAMS DMY
CMSVSAM DMY
CMSPIPES DMY
CMSFILES DMY
CMSVMLIB DMY
```

Figure 79. Example of the SEGBLD \$SELECT File

---



## The VMSESE PROFILE

The VMSESE PROFILE identifies system data that is used by VMSES/E. It is shipped as part of VMSES/E and is installed on the 5E5 disk.

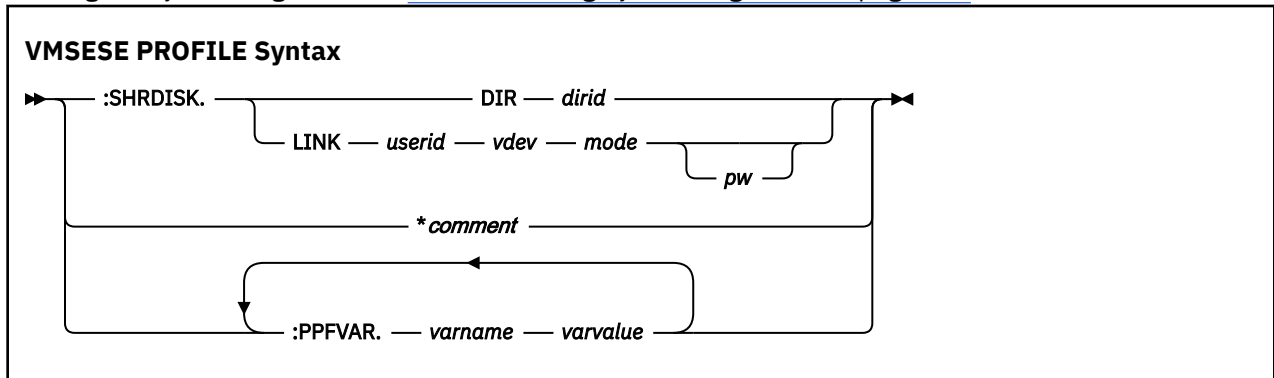
The SHRDISK record identifies the minidisk or SFS directory that is used to store information that is required by multiple products, such as the VMSBR \$SELECT file.

If you are changing the Software Inventory from a minidisk to an SFS directory, see [Chapter 18, “Changing the Software Inventory to an SFS Directory,”](#) on page 221.

The PPFVAR record identifies variables and their values that are used by the VMFPPF command when compiling product parameter files.

### File Syntax

“VMSESE PROFILE Syntax” on page 131 shows the syntax of the VMSESE PROFILE. (If you need help reading the syntax diagrams, see “Understanding Syntax Diagrams” on page 227.)



The fields in the VMSESE PROFILE are defined as follows:

#### **:SHRDISK.**

indicates a record that identifies a minidisk or SFS directory for shared information.

#### **DIR**

indicates that an SFS directory is being used.

#### **dirid**

is a fully-qualified SFS directory ID.

#### **LINK**

indicates a minidisk is being used.

#### **userid**

is the user ID of the minidisk owner. You can enter an asterisk (\*) to indicate the current user.

#### **vdev**

is the address of the minidisk.

#### **mode**

is the access mode. Write access is required.

#### **pw**

is the access mode password.

**Note:** The minidisk or directory specified on the :SHRDISK tag must match the first disk in the APPLY string defined in the saved segment product parameter file specified on the VMFBLD command. If the disk is changed in one file, it must be changed in the other.

#### **\***

indicates a comment record.

#### **comment**

is a comment line.

**:PPFVAR.**

indicates a record that identifies a PPF variable.

**varname**

is a variable used in a source product parameter file (\$PPF).

**varvalue**

is the value assigned by VMFPPF when a product parameter file is compiled.

**Note:** The chosen variable name must be unique in all PPFs that may be compiled.

## Example

Here is a sample VMSESE PROFILE:

```

* =====
* VMSESE PROFILE
* =====
* Default disk used for updating the VMSBR $SELECT file for saved segment builds
* -----
:SHRDISK. LINK MAINT730 51D MR

* -----
* PPF file user ID definitions.
* -----
:PPFVAR. $MAINT$ MAINT
:PPFVAR. $MNTVVRM$ MAINT730
:PPFVAR. $MNTCSM$ MAINTCSM
:PPFVAR. $CSMWRK$ CSMWORK

* -----
* PPF file pool definitions.
* -----
:PPFVAR. $VMSYS$ VMSYS
:PPFVAR. $VMPSFSS$ VMPSFSS

* -----
* CSM-specific variable definitions -- these should -not- be modified directly
* within this file.
* -----
:PPFVAR. $CSMVRM$ CSM730
:PPFVAR. $CSMLVL$ BASE

* -----
* SFS directory variable definitions for future use.
*
* !START! DO NOT MOVE THIS SECTION ANYWHERE ELSE IN THIS FILE.
* -----
*
* TSAFSFS / AVSSFS
:PPFVAR. $TSAFAVSLMODZ$ $VMPSFSS:$CSMVRM$. $CSMLVL$.TSAFAVS.LOCALMOD * Local mods
:PPFVAR. $TSAFAVSSAMPZ$ $VMPSFSS:$CSMVRM$. $CSMLVL$.TSAFAVS.SAMPLE * Sample files
:PPFVAR. $TSAFAVSDELTAZ$ $VMPSFSS:$CSMVRM$. $CSMLVL$.TSAFAVS.DELTAPROD * AVS service
:PPFVAR. $TSAFAVSAPPLX$ $VMPSFSS:$CSMVRM$. $CSMLVL$.TSAFAVS.APPLYALT * Aux/software
:PPFVAR. $TSAFAVSAPPLY$ $VMPSFSS:$CSMVRM$. $CSMLVL$.TSAFAVS.APPLYINT * Aux/software
:PPFVAR. $TSAFAVSAPPLZ$ $VMPSFSS:$CSMVRM$. $CSMLVL$.TSAFAVS.APPLYPROD * Aux/software
:PPFVAR. $TSAFAVSBAS2Z$ $VMPSFSS:$CSMVRM$. $CSMLVL$.TSAFAVS.OBJECT * AVS object
:PPFVAR. $TSAFAVSBLD4Z$ $VMPSFSS:$CSMVRM$. $CSMLVL$.CMSREXX.49D * Test Help disk
:PPFVAR. $TSAFAVSBLD7Z$ $VMPSFSS:$CSMVRM$. $CSMLVL$.CMSREXX.493 * Test CMS sys tool
:PPFVAR. $TSAFAVSBLD9Z$ $VMPSFSS:$CSMVRM$. $CSMLVL$.CMSREXX.402 * UCENG Help disk
*
* CMSSFS / REXXSFS
:PPFVAR. $CMSREXXLMODZ$ $VMPSFSS:$CSMVRM$. $CSMLVL$.CMSREXX.LOCALMOD * Local mods
:PPFVAR. $CMSREXXSAMPZ$ $VMPSFSS:$CSMVRM$. $CSMLVL$.CMSREXX.SAMPLE * Sample files
:PPFVAR. $CMSREXXDELTAZ$ $VMPSFSS:$CSMVRM$. $CSMLVL$.CMSREXX.DELTAPROD * CMS service
*
* PERFTKSFS
:PPFVAR. $PTKBAS1Z$ $VMPSFSS:$CSMVRM$. $CSMLVL$.PERFTK.BASE * PERFTK BASE
:PPFVAR. $PTKLMODZ$ $VMPSFSS:$CSMVRM$. $CSMLVL$.PERFTK.LOCALMOD * LOCAL MODIFICATIONS
DISK
:PPFVAR. $PTKSAMPZ$ $VMPSFSS:$CSMVRM$. $CSMLVL$.PERFTK.SAMPLE * SAMPLES DISK
:PPFVAR. $PTKDELTAZ$ $VMPSFSS:$CSMVRM$. $CSMLVL$.PERFTK.DELTA * PERFTK SERVICE
:PPFVAR. $PTKAPPLX$ $VMPSFSS:$CSMVRM$. $CSMLVL$.PERFTK.APPLYALT * AUX AND SW INVEN FILES
:PPFVAR. $PTKAPPLZ$ $VMPSFSS:$CSMVRM$. $CSMLVL$.PERFTK.APPLYPROD * AUX AND SW INVEN FILES
:PPFVAR. $PTKBLD0Z$ $VMPSFSS:$CSMVRM$. $CSMLVL$.PERFTK.TBUILD * TEST DISK
:PPFVAR. $PTKBLD2Z$ $VMPSFSS:$CSMVRM$. $CSMLVL$.PERFTK.1CC * CONFIGURATION CODE DISK
:PPFVAR. $PTKBLD4Z$ $VMPSFSS:$CSMVRM$. $CSMLVL$.PERFTK.PERFTKHELP * HELP FILES

```

```

*
* -----
* !END! DO NOT MOVE THE ABOVE SECTION ANYWHERE ELSE IN THIS FILE.
* -----

```

## The Apply List

The apply list contains a list of the PTFs to be applied to a product. The order of the PTFs in the list is not significant. The PTFs listed in the apply lists shipped by IBM with service are in age order, with the oldest PTFs listed first.

The value specified on the :AXLIST tag in the product parameter file is the file name of the apply list supplied by IBM on the service tape.

- For COR service, the file type is:
  - \$APCymdd for the apply list associated with a single COR tape. *ymdd* is the date (*y* is the last digit of the year, *m* is the month numbered in hexadecimal, and *dd* is the day).
  - \$APCALL for the cumulative apply list.

Because COR tapes are cumulative, this apply list is a duplicate of the previous one.

- For an install or RSU tape, the file type is:
  - \$APIyynn for the apply list associated with a single tape. *yynn* is the level identifier.
  - \$APIALL for the cumulative apply list.

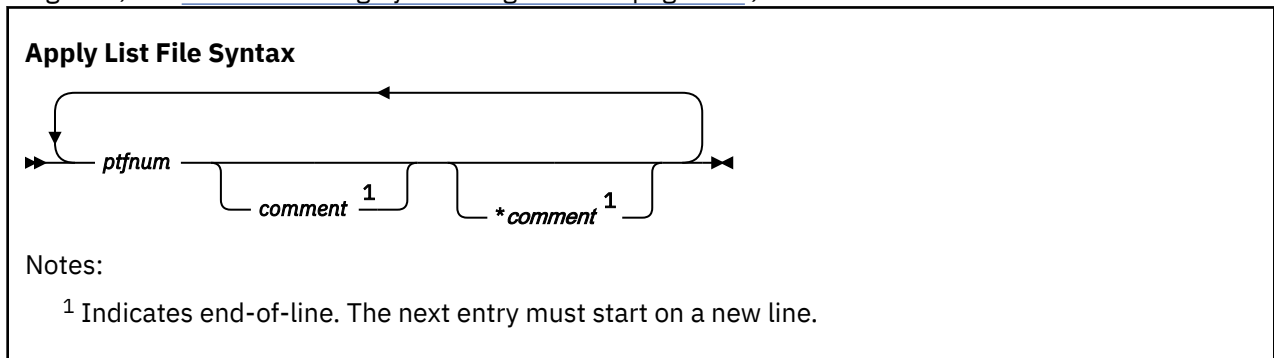
The VMFREC EXEC loads the numbered apply lists unmodified. The cumulative apply lists are renamed to have a file type of \$APPLIST, which is the file type required by the VMFAPPLY EXEC. When this is done, the new lists from the tape either overlay any existing lists with the same names or are appended to them, depending on the option selected on the VMFREC EXEC command invocation. Comments identifying the product, tape type, and tape number are at the top of the apply list that is shipped from IBM.

A one-line entry for each PTF follows this information. The first word on the line is the PTF number. The second word on the line, if present, is a comment containing the corresponding APAR number.

An asterisk (\*) indicates a comment.

## File Syntax

“Apply List File Syntax” on page 133 shows the apply list syntax. (If you need help reading the syntax diagrams, see “Understanding Syntax Diagrams” on page 227.)



## Example

Figure 80 on page 134 is an example of an apply list.

```
* APPLY LIST FOR PRODUCT vVMCMSm COR nnnn
UM14144 VM40611
UM14428 VM41021
```

Figure 80. Example of an Apply List—DMSVM \$APPLIST

## The Exclude List

The exclude list contains a list of the PTFs that are not to be applied to a product, even if they are listed in the apply list. Excluded PTFs will not be applied, nor will any PTFs that list an excluded PTF as a requisite. The first line in the exclude list, as shipped from IBM, is a comment identifying the product and the tape type and number. Before you apply service to a product with the VMFAPPLY EXEC, you can create an exclude list to define PTFs that you want to exclude.

The value specified on the :AXLIST tag in the product parameter file is the file name of the exclude list supplied by IBM on the service tape. The value specified on the :EXCLIST tag in the product parameter file is the file name of the optional exclude list supplied by you. The PTFs from both of these lists are used by the VMFAPPLY process.

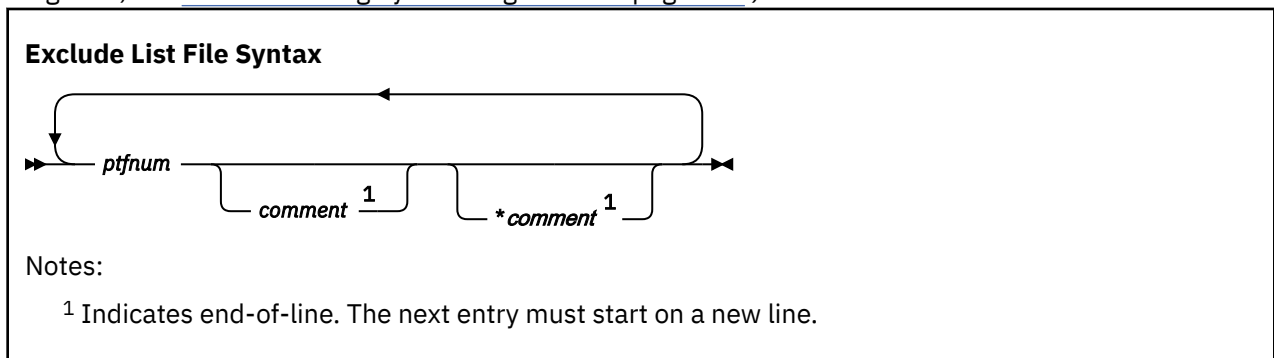
- The file type for COR service is:
  - \$EXCymdd for the exclude list associated with a single COR tape. *y* is the last digit of the year, *m* is the month numbered in hexadecimal, and *dd* is the day).
  - \$EXCALL for the cumulative exclude list.

Because COR tapes are cumulative, this exclude list is a duplicate of the previous one.
- The file type for the install or RSU tape is:
  - \$EXIyynn for the exclude list associated with a single install or RSU tape. *yynn* is a level identifier.
  - \$EXIALL for the cumulative exclude list.

The VMFREC EXEC loads the numbered exclude lists unmodified. The cumulative exclude lists are renamed to have a file type of \$EXCLIST, which is the file type required by the VMFAPPLY EXEC. When this is done, the new lists from the tape either overlay any existing lists with the same names or are appended to them, depending on the option selected on the VMFREC EXEC command invocation.

## File Syntax

“Exclude List File Syntax” on page 134 shows the exclude list syntax. (If you need help reading the syntax diagrams, see “Understanding Syntax Diagrams” on page 227.)



## Example

Figure 81 on page 135 is an example of an exclude list with two PTF entries.

```
* EXCLUDE LIST FOR PRODUCT vVMCMSim COR nnnn
UM03601
UM90033
```

Figure 81. Example of an Exclude List—DMSVM \$EXCLIST

## Place Into Production Files

The place into production files are the SERVICE \$PRODS file and the *systemid* \$PRODS files. These files contain the products and associated objects that were serviced. The VMFBLD part handlers and the SERVICE command update the SERVICE \$PRODS file. The *systemid* \$PRODS file is created from a SERVICE \$PRODS file.

### The SERVICE \$PRODS File

The SERVICE \$PRODS output file contains the following types of records:

- COPYPART, ERASEPART, COPYHELP, and ERASEHELP record syntax:

► *recid* — *compname* — COPYPART — *ppffn* — *compname* — *&fromvarn* — *&tovarn* —►

ERASEPART

COPYHELP

ERASEHELP

► NONE — *prodid* — *filename filetype* —►

UPCASE

- BFS (Byte File System) record syntax:

► *recid* — *compname* — BFS — *buildlist* — *objectname* —►

- SEGMENTS record syntax:

► *recid* — *compname* — SEGMENTS — *buildlist* —►

- NUCLEUS record syntax:

► *recid* — *compname* — NUCLEUS — *buildlist* —►

- COMPONENT record syntax:

► *recid* — *compname* — COMPONENT — *ppffn* —►

- PRODLEV (production level) record syntax:

► *recid* — *compname* — PRODLEV — *servlvl* —►

- SYNCLEV (synchronize service level) record syntax:

► *recid* — *compname* — SYNCLEV — *servlvl* —►

- SAVECMS record syntax:

## Other Files Used in the Service Process

► *recid* — *compname* — SAVECMS ◄

- ENABLE record syntax:

► *recid* — *compname* — ENABLE — *systemid* — *prodrecord* ◄

If there are multiple ENABLE records for the same *recid*, the last one in the file is used.

- DDRCMS record syntax:

► *recid* — *compname* — DDRCMS — *ppffn* — *compname* — *&tovarn* — *filename filetype* ◄

A diagram showing a feedback loop. A horizontal line contains the text: ► *recid* — *compname* — DDRCMS — *ppffn* — *compname* — *&tovarn* — *filename filetype* ◄. A curved arrow starts from the right side of *filename filetype* and points back to the *compname* field.

- DDRREST record syntax:

► *recid* — *compname* — DDRREST — *ppffn* — *compname* — *&fromvarn* — *&tovarn* — NONE →

◄ *prodid* — *filename filetype* ◄

- WARNMSG record syntax:

► *recid* — *compname* — WARNMSG — *msgnum.msgfmt* ◄

Where:

### ***recid***

is the 1- through 8-character alphanumeric product identifier. This is the same value that is on the :RECID. tag in the product's PPF.

### ***compname***

is the 1- through 16-character base component name.

### ***ppffn***

is the 1- through 8-character alphanumeric Product Parameter File (PPF) file name.

### ***&fromvarn***

is the variable name defining the test build disk from the :DCL section in the PPF.

### ***&tovarn***

is the variable name defining the production build disk from the :DCL section in the PPF.

### **NONE | UPCASE**

defines an option for processing parts.

### ***prodid***

is the 1- through 8-character alphanumeric identifier assigned to the product, concatenated with a percent sign (%) to the 1- through 16-character alphanumeric component name identifier.

### ***filename filetype***

is the file name and file type of all of the serviced objects to be copied or erased.

### ***buildlist***

is the build list associated with the part handler in the :BLD. section of the product's PPF file. For NUCLEUS records, *buildlist* can be optional, depending on the type of nucleus being built.

### ***objectname***

is the name of the object that was serviced.

### ***systemid***

is the system where the SET PRODUCT command was issued.

### ***prodrecord***

is the operand(s) used on the SET PRODUCT command.

### ***servlvl***

is the service level of the product.

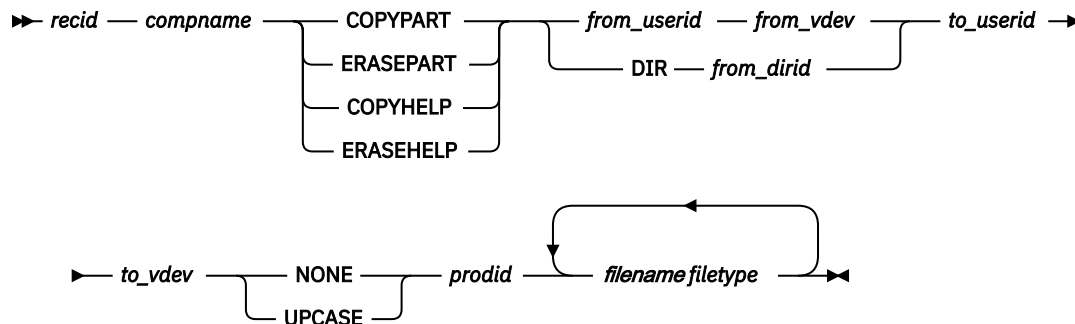
***msgnum.msgfmt***

is the message number and the message format number, concatenated with a period (.) character.

**The *systemid* \$PRODS file**

The *systemid* \$PRODS file contains the same records and syntax as the SERVICE \$PRODS file with the exception of the following:

- The DCL variable value *&fromvarn* is replaced, for specific records, with minidisk-specific values or SFS-specific values. Similarly, the DCL variable value *&tovarn* is replaced with minidisk-only specific values.
- COPYPART, ERASEPART, COPYHELP, and ERASEHELP record syntax:



- DDRCMS record syntax:



- The *systemid* \$PRODS file can contain an ERROR record. The format is:

► *recid* — *compname* — ERROR — *to\_userid* — *to\_vdev* ►

The changed variables and values that pertain to these records are:

***from\_userid***

is the owning user ID of the test build disk.

***from\_vdev***

is the address of the test build disk.

***from\_dirid***

is the fully-qualified SFS directory of the test build directory.

***to\_userid***

is the owning user ID of the production build disk.

***to\_vdev***

is the address of the production build disk.

All other variables are described under [“The SERVICE \\$PRODS File”](#) on page 135.

**The Message Log**

The message log contains any error or informational messages issued by its corresponding EXEC. The message log is written to the A-disk. If the message log already exists when the exec is invoked, a date and time stamp are inserted to separate the earlier log entries from the new entries. You should browse the message log with the VMFVIEW EXEC after the LOCALMOD, PUT2PROD, SERVICE, SERVMGR, VMFAPPLY, VMFBLD, VMFINS, VMFMRDSK, or VMFREC EXEC finishes processing.

The message logs related to z/VM Centralized Service Management (z/VM CSM) are \$CSMCMG \$MSGLOG (produced by the SERVMGR EXEC) and \$CSMAGT \$MSGLOG (produced by the CSMAGENT EXEC).

## Other Files Used in the Service Process

For the LOCALMOD, PUT2PROD, SERVICE, VMFAPPLY, VMFBLD, VMFINS, VMFMRDSK, and VMFREC EXECs, the message log is called \$VMFxxx \$MSGLOG, where xxx is:

- LMD for the local modification log created by the LOCALMOD EXEC
- P2P for the put into production log created by the PUT2PROD EXEC
- SRV for the service log created by the SERVICE EXEC
- APP for the apply message log created by the VMFAPPLY EXEC
- BLD for the build message log created by the VMFBLD EXEC
- INS for the install message log created by the VMFINS EXEC
- MRD for the merge message log created by the VMFMRDSK EXEC
- REC for the receive message log created by the VMFREC EXEC.

Table 10 on page 138 shows the different types of messages that appear in these message logs.

Table 10. VMSES \$MSGLOG Message Codes	
Code	Explanation
ST:	A status message pertaining to a major function of the current process
BD:	Informational messages indicating build requirements
CK:	A condition that must be checked
WN:	A warning message
RO:	Messages pertaining to requisites outside the component
RQ:	Messages pertaining to requisites
MS:	Mismatched parts, such as AUX files and AUX entries in the front of text decks
SV:	A severe problem encountered

**Note:** All messages except status messages *must* be investigated. A warning message does not necessarily mean that anything is wrong, but *you cannot be sure until you check*.

Message logs are cumulative. The most recent entries are at the *top*.

Figure 82 on page 139 shows a sample message log.



```

*****
**** PPFNAME: SERVP2P  COMPNAME: VMSES          RECID: 7VMSES30  ****
*****
****          Date: 2022-09-16          Time: 09:09:55          ****
*****
ST:VMFREC2195I VMFREC PPF SERVP2P VMSES ( LOG SRV ENV VMF0009 APPEND
ST:
NOSETUP NORECVALL
ST:VMFREC2760I VMFREC processing started
ST:VMFUTL2205I Minidisk|Directory Assignments:
ST:          String      Mode  Stat  Vdev  Label  (OwnerID Odev : Cyl/%Used)
ST:          -or-          SFS Directory Name
ST:VMFUTL2205I LOCALMOD  E    R/W  5C4  MNT5C4 (MAINT730 05C4 : 5/02)
ST:VMFUTL2205I LOCALSAM  F    R/W  5C2  MNT5C2 (MAINT730 05C2 : 5/01)
ST:VMFUTL2205I APPLY     G    R/W  5A6  MNT5A6 (MAINT730 05A6 : 6/01)
ST:VMFUTL2205I          H    R/W  5A4  MNT5A4 (MAINT730 05A4 : 6/01)
ST:VMFUTL2205I          I    R/W  5A2  MNT5A2 (MAINT730 05A2 : 6/00)
ST:VMFUTL2205I DELTA     J    R/W  5D2  MNT5D2 (MAINT730 05D2 : 30/00)
ST:VMFUTL2205I BUILD8    B    R/W  5E6  MNT5E6 (MAINT730 05E6 : 9/81)
ST:VMFUTL2205I BUILD7    K    R/W  493  MNT493 (MAINT730 0493 : 250/53)
ST:VMFUTL2205I BUILD6    L    R/W  490  MNT490 (MAINT730 0490 : 207/41)
ST:VMFUTL2205I BUILD4    M    R/W  49D  MNT49D (MAINT730 049D : 146/65)
ST:VMFUTL2205I BASE2     N    R/W  5B2  MNT5B2 (MAINT730 05B2 : 38/88)
ST:VMFUTL2205I -----  A    R/W  191  MNT191 (MAINT730 0191 : 175/19)
ST:VMFUTL2205I -----  C    R/W  500  MNT500 (MAINT730 0500 : 900/60)
ST:VMFUTL2205I -----  D    R/W  51D  MNT51D (MAINT730 051D : 26/45)
ST:VMFUTL2205I -----  S    R/O  190  MNT190 (MAINT 0190 : 207/41)
ST:VMFUTL2205I -----  Y/S  R/O  19E  MNT19E (MAINT 019E : 500/33)
ST:VMFREC1852I Volume 1 of 1 of COR ENVELOPE created on 16 September 20
ST:VMFREC1851I (1 of 3) VMFRCAXL processing AXLIST
ST:VMFRCX2159I Loading 4 part(s) to DELTA 5D2 (J)
ST:VMFRCX2193I Appending new Apply list VMFVM $APPLIST to the existing
ST:              list on DELTA 5D2 (J)
ST:VMFRCX2193I Appending new Exclude list VMFVM $EXCLIST to the existing
ST:              list on DELTA 5D2 (J)
ST:VMFREC1851I (2 of 3) VMFRCPTF processing PARTLST
ST:VMFRCP2159I Loading 1 part(s) to DELTA 5D2 (J)
ST:VMFREC1851I (3 of 3) VMFRCOM processing DELTA
ST:VMFRC2159I Loading 2 part(s) to DELTA 5D2 (J)
ST:VMFREC2189I Updating Requisite table 7VMSES30 SRVREQT, Description
ST:              table 7VMSES30 SRVDESCT and Receive Status table 7VMSES30
ST:              SRVRECS with 1 new PTFs from COR 0009
ST:VMFREC2760I VMFREC processing completed successfully

```

Figure 82. A Sample Message Log: \$VMFREC \$MSGLOG

See “[Viewing Message Logs](#)” on page 117 for more information.

## The VMFINS DEFAULTS File

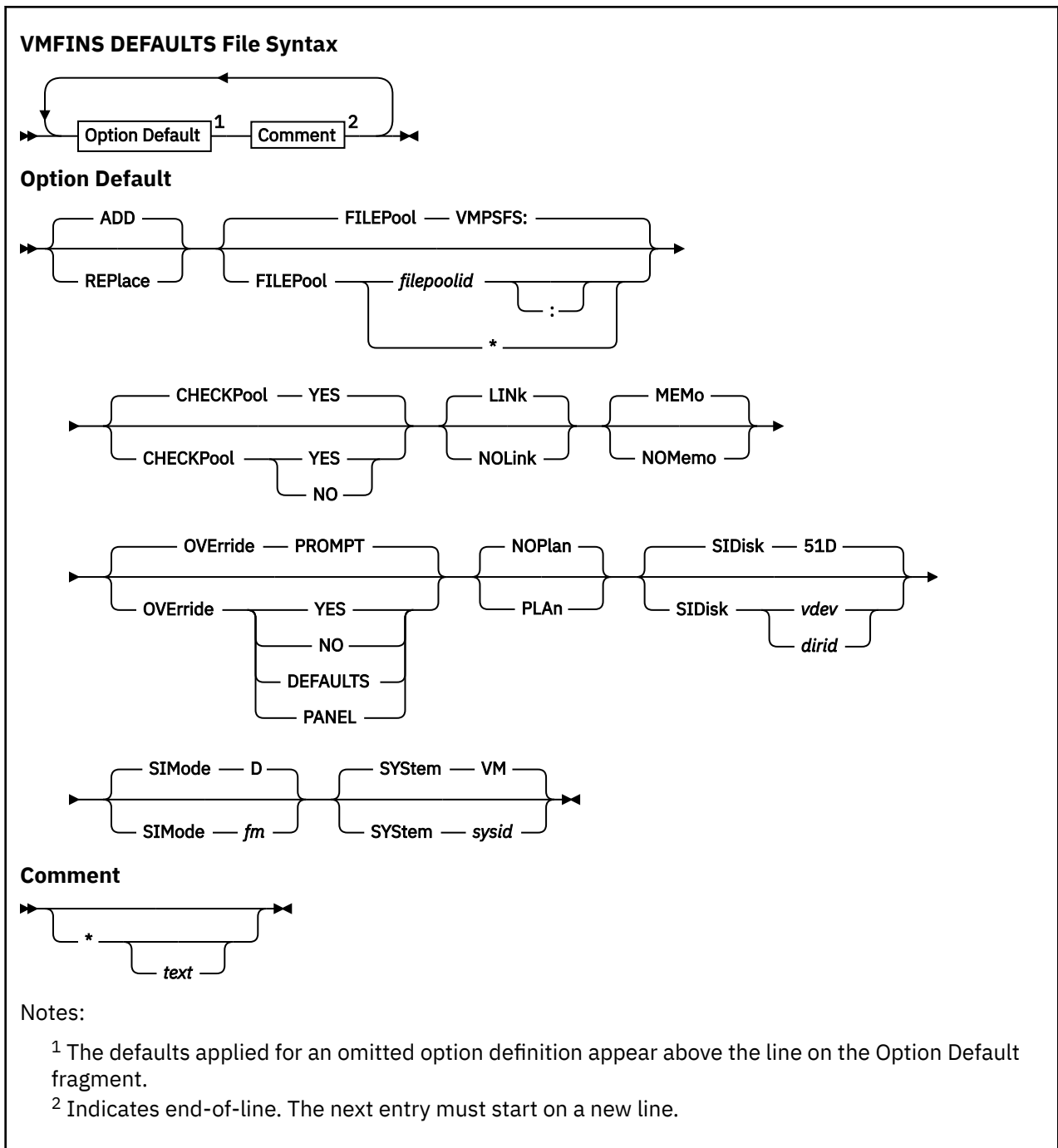
The VMFINS DEFAULTS file establishes command option defaults for the VMFINS EXEC and for various other VMSES/E commands (such as SERVICE and PUT2PROD). These command defaults are obtained from the first VMFINS DEFAULTS file that exists in the CMS search order when an associated VMSES/E command is issued.

The option defaults in the VMFINS DEFAULTS file can be customized for your specific environment by:

- Directly modifying the VMFINS DEFAULTS file that resides on the VMSES/E build minidisk (by default, MAINTvrm 5E5).
- Creating a private copy of this file (for example on the MAINTvrm 191 minidisk) and then customizing this copy.

### Syntax

“[VMFINS DEFAULTS File Syntax](#)” on page 140 shows the syntax of the VMFINS DEFAULTS file. (If you need help reading the syntax diagrams, see “[Understanding Syntax Diagrams](#)” on page 227.)



For information on the VMFINS options, see “VMFINS EXEC” on page 404.

## Example

Figure 83 on page 141 shows sample definitions for the VMFINS DEFAULTS file.

```

ADD                * default is ADD
NOPLAN            * default is NOPLAN
MEMO              * default is MEMO
LINK              * default is LINK
*
SIDISK            51D        * default is 51D
SIMODE            D          * default is D
SYSTEM            VM        * default is system ID is VM
FILEPOOL          VMPSFS:    * default is VMPSFS:
CHECKPOOL         YES       * Default is YES
OVERRIDE         PROMPT     * default is PROMPT

```

Figure 83. The VMFINS DEFAULTS File

## Usage Notes

- The defaults specified in the VMFINS DEFAULTS file are used by respective VMSES/E commands on an individual basis. Therefore, only a subset of these definitions might be applied when a specific command is used.
- For some VMSES/E commands there are no command line options that correspond to definitions within the VMFINS DEFAULTS file. For these commands, the definitions and values within the VMFINS DEFAULTS file are the only means of control for the subject parameters. See the description of a specific command to determine what options and controls can be applied.
- Case is not significant for definitions in the VMFINS DEFAULTS file. All information is used in upper case form.
- If multiple instances of an option definition exist in the VMFINS DEFAULTS file, only the last instance is used.

## Build Lists

While the build status table identifies usable forms that have had service applied, the build list indicates where a serviceable part is used in building a usable form (nucleus, MACLIB, and so forth). All serviceable parts are listed in build lists. If a serviceable part is not listed in a build list, VMSES/E cannot identify or build the usable forms automatically. The file type of a build list is either EXEC or EXCnnnnn.

VMSES/E uses build lists in three different formats. The older format, format 1, is retained for compatibility with non-VMSES/E execs like HCPLDR and VMFMAC, which also use build lists. (In connection with these execs, build lists are sometimes called loadlists or MACLIB execs.) Format 1 build lists define only a *single object*.

Format 2 is more flexible than format 1. Format 2 build lists allow *multiple objects* (usable forms) within a build list. And also allows variations determined by the part handler that will process the build list.

Format 3 supports *libraries*. Each library member is defined as an object in a format 3 build list.

Table 11 on page 141 shows the build list format used by each part handler. For detailed build list specifications for the various part handlers, see “Creating Objects with VMFBLD” on page 325.

Table 11. Build List Formats Used by the VMFBLD Part Handlers

Part Handler	Format	Function
VMFDBBFS	2	Moves files into the Byte File System
VMFBDCLB	3	Builds callable services libraries
VMFBDCOM	2	Builds replacement objects
VMFBDCPY	1	Builds replacement text only
VMFBDDDR	2	Restores DDR image files to minidisks

Table 11. Build List Formats Used by the VMFBLD Part Handlers (continued)

Part Handler	Format	Function
VMFBDDL	3	Builds CMS/DOS phase libraries
VMFBGEN	2	Builds generated objects, such as text decks
VMFBLLB	3	Builds LOADLIBs
VMFBMLB	3	Builds MACLIBs
VMFBMOD	2	Builds modules
VMFBNUC	1	Builds nuclei
VMFBPMD	2	Builds modules using the C89 or CPLINK command.
VMFBDSBR	2	Identifies system objects (saved segments) to be built
VMFBSEGE	2	Builds saved segments
VMFBTLB	3	Builds TXTLIBs

**Note:** Use care when creating or adding your own build lists. Records are processed differently by the various VMFBLD part handlers.

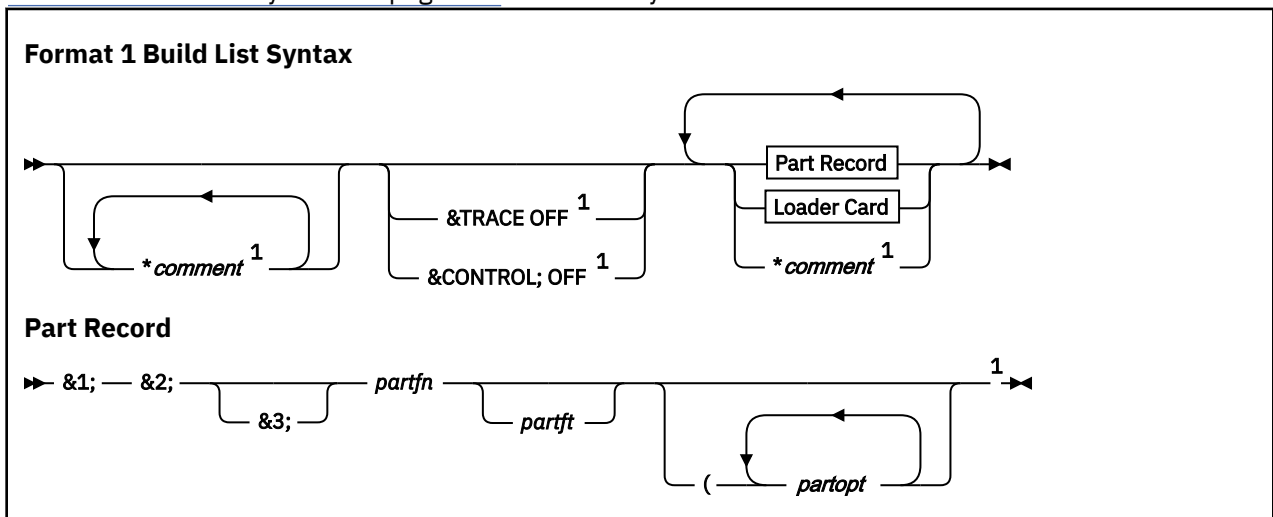
## Syntax Notation

If you need help reading the syntax diagrams for the different build list formats, see “Understanding Syntax Diagrams” on page 227. In addition, the following syntax is used while describing the build list parameters:

- [ ] Brackets enclose optional items that may be displayed.
- { } Braces enclose alternative items that may be displayed.
- | The vertical bar separates items within braces or brackets.
- ... The ellipsis indicates that the preceding item may be repeated.

## Format 1 Build List Syntax

“Format 1 Build List Syntax” on page 142 shows the syntax of a format 1 build list.



**Loader Card**

► x'02' — loader card <sup>1</sup> —►

Notes:

<sup>1</sup> Indicates end-of-line. The next entry must start on a new line.

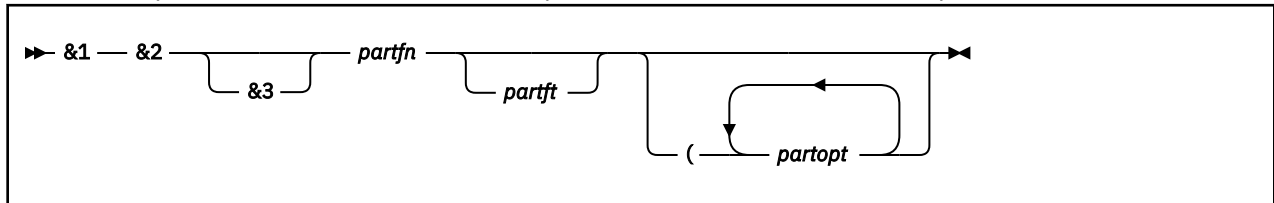
Format 1 build lists have four types of records: EXEC or EXEC2 statements, part records for serviceable parts, X'02' loader cards, and comments.

**EXEC or EXEC2 Statements**

One EXEC statement, &CONTROL; OFF, and one EXEC2 statement, &TRACE; OFF, are permitted as build list records. EXEC or EXEC2 statements must be placed before any serviceable part entries or loader cards.

**Serviceable Part Records**

Serviceable part records name a serviceable part. The format of a serviceable part record is:

**&1 &2 &3**

are variables to be used by the exec that processes the build list.

**partfn**

is the file name of a serviceable part.

**partft**

is the file type of a serviceable part.

The part handler selects the correct level of each serviceable part. If you do not specify *partft*, VMFSIM GETLVL selects the level based on the information in the version vector table.

**partopt**

are part options used by the part handler when processing the serviceable part. In format 1 build lists, part options are specified after the parentheses on each serviceable part record. See [Figure 84 on page 144 \(6\)](#). See “Creating Objects with VMFBLD” on page 325 for a description of the part options for each part handler.

Serviceable part entries may not appear before the EXEC or EXEC2 statements, but they may be mixed in any order with loader cards and comments.

**Note:** For format 1 build lists, the object name is BLDLIST, and it appears only as output from the VMFQOBJ EXEC and the service-level build status table.

**Loader Cards**

Loader cards contain X'02' control characters. Loader cards may not appear before the EXEC or EXEC2 statements, but they may be mixed in any order with entries and comments.

**Comments**

Comments begin with an asterisk (\*) in column 1. They may appear anywhere in the build list. They are the only records that may be placed before the EXEC or EXEC2 statements.

## Format 1 Build List Example

Figure 84 on page 144 shows a sample format 1 build list.

```

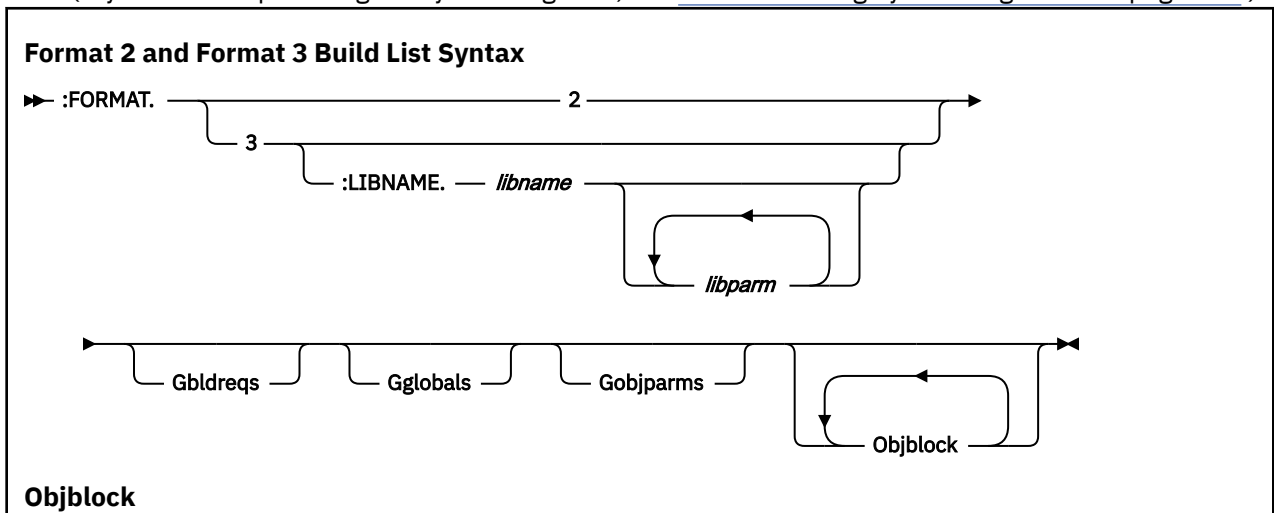
* Begin by setting &TRACE; OFF
&TRACE; OFF           1
&1 &2 &3 HCPLDR  LOADER
&1 &2 &3 DMSNUC     2
&1 &2 &3 DMSZNR
&1 &2 &3 SLC L00E000  3
:
***** DMSINN must be after DMSIND and not part of NUC
&1 &2 &3 DMSZINN
"LDT                   4
* The " on the prior line is a Hex '02'  5
&1 &2 &3 DMSZINV
* DMSTRT needs the LANG option
&1 &2 &3 DMSTRT (LANG  6
&1 &2 &3 LDT DMSINIW
    
```

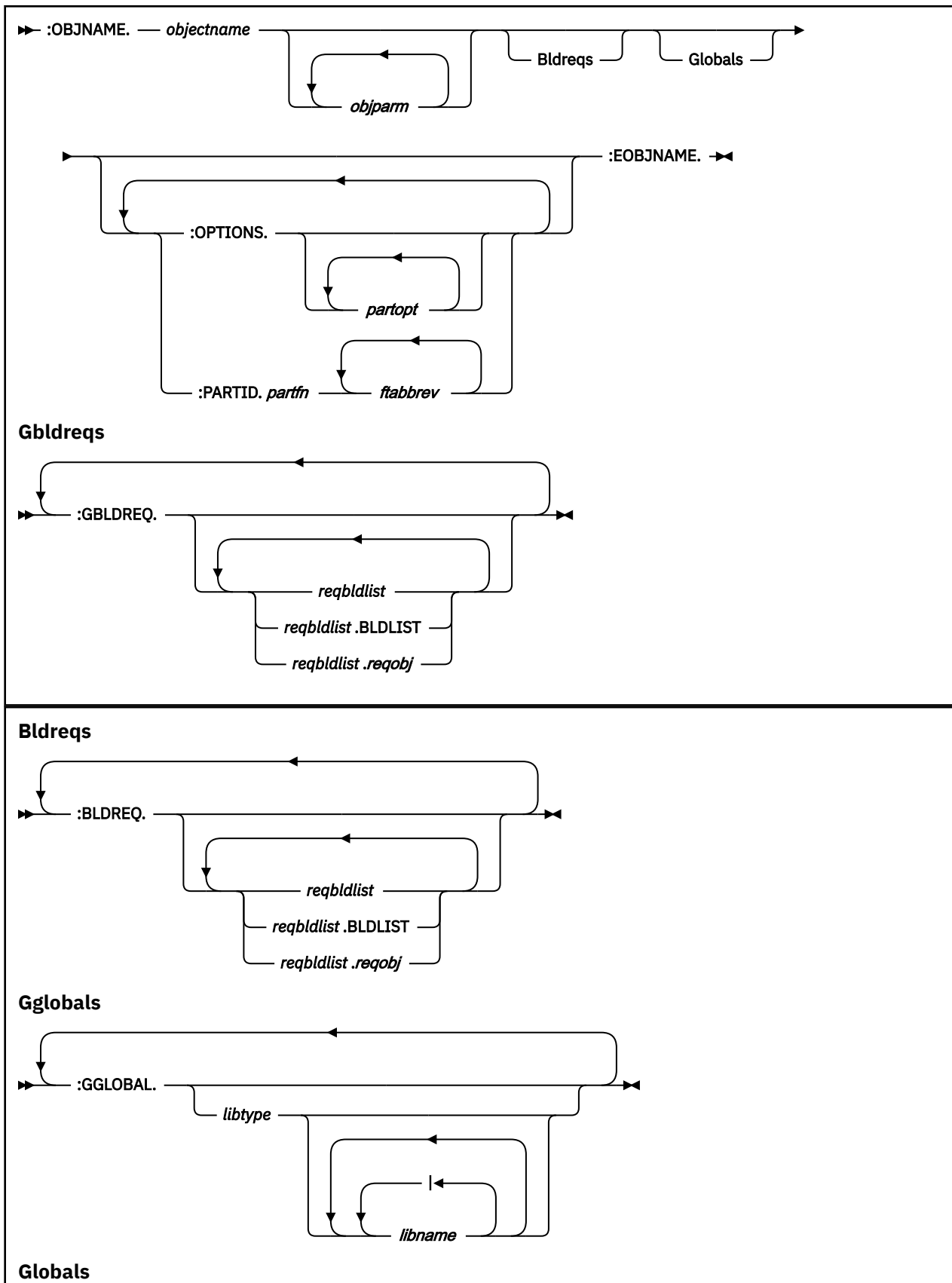
Figure 84. Example of a Format 1 Build list

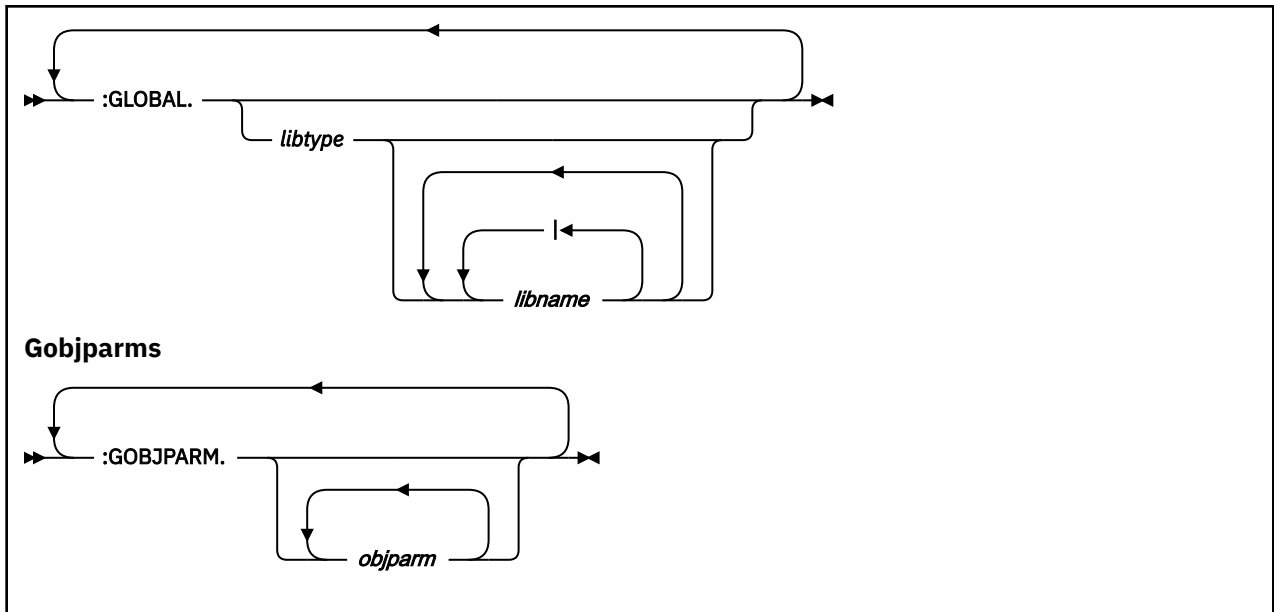
It contains one EXEC2 statement (1), several serviceable part records with no file type specified (2), a serviceable part record with the file type specified (3), a loader card (4), and a comment (5). The other records are all entry records or comments. Some of the entry records specify file names and file types and the rest specify only file names. One entry record specifies an option (6).

## Format 2 and 3 Build List Syntax

“Format 2 and Format 3 Build List Syntax” on page 144 shows the syntax of format 2 and format 3 build lists. (If you need help reading the syntax diagrams, see “Understanding Syntax Diagrams” on page 227.)







### General Information

The following guidelines apply to format 2 and 3 build lists:

- All tags must begin on a new line.
- The data on a tag can span multiple lines.
- All tagged records can have zero or more spaces after the tag.
- All records, except comments, must be in uppercase English.
- Comment lines are permitted anywhere in the file.
- A comment line begins with an asterisk (\*).

### Deleting Objects from Build Lists

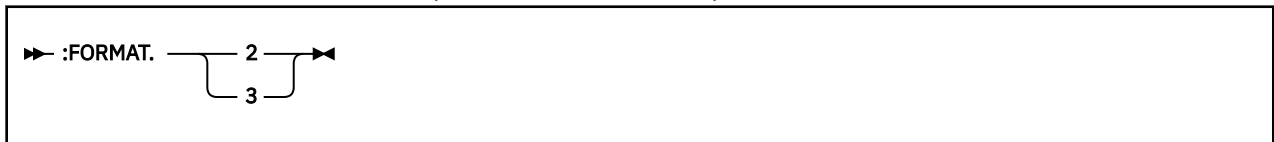
Entries in the select data file, service-level build status table, and build lists are used to delete objects.

VMFBLD determines the objects to be deleted by reading the select data file to determine which build lists have been serviced. (When a build list has been serviced, a third token is added to the select data file entry.) VMFBLD then compares the previous level of the build list to the new level of the build list. VMFBLD gives the status of DELETE to any object found in the previous level of the build list that is not found in the new serviced level.

The following sections describe the records in format 2 and 3 build lists. Any special considerations for certain build list formats are also provided.

### :FORMAT Record

A :FORMAT record is the first record, other than a comment, in a build list. Its format is:



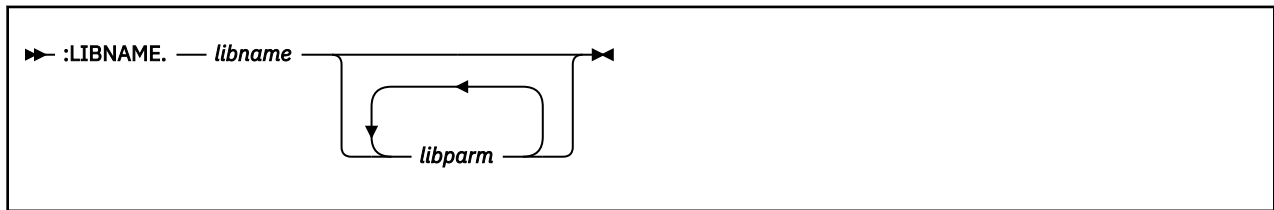
- 2
- 3

identifies the build list format.



## :LIBNAME Records

A :LIBNAME record identifies a library. :LIBNAME records are only valid in format 3 build lists. Its format is:



### **libname**

is the file name of the library to be built. The default file name is the file name of the build list. Each member of the library must be defined as an object.

### **libparm**

are parameters to use when processing the :LIBNAME record. The parameters can be:

### **C370LIB**

C370LIB indicates to VMFBDTLB that the C370LIB family of commands should be used for TXTLIB manipulation instead of the TXTLIB family of commands.

### **NOERASE**

NOERASE indicates to the part handler that the library is not to be erased during the build. The NOERASE parameter will have no effect on the CSLLIB part handler since a CSLLIB can only be updated by being totally rebuilt.

If NOERASE is omitted, all library part handlers erase and rebuild a library if all of its objects are to be built. In addition, the MACLIB part handler erases and rebuilds whenever an object is deleted, and the LOADLIB and DOSLIB part handlers erase and rebuild if the history file is not found.

The NOERASE parameter is useful if a library is to be updated from multiple products, each of which has its own build list for the library. Without NOERASE, if a product erases and rebuilds the library, any objects (members) that have been added by other products are lost.

## Part Handlers

- VMFBDCLB - tolerates NOERASE
- VMFBDDLb - full support for NOERASE
- VMFBDLLB - full support for NOERASE
- VMFBDMLB - full support for NOERASE
- VMFBDTLB - full support for NOERASE and C370LIB

## Example

The C370LIB parameter will be used by Licensed Products that wish to have their TXTLIBs built using the C370LIB family of commands. By not specifying C370LIB as a library parameter on the :LIBNAME tag, VMFBDTLB will default to the use of TXTLIB commands.

The following is an example of a format 3 build list, which contains the C370LIB parameter, used by VMFBDTLB to create a TXTLIB.

```
:FORMAT. 3
:LIBNAME. MYTXTLIB C370LIB
:OBJNAME. MEMBER1
:OPTIONS. ENTRY CSECT1
:PARTID. PART1 TXT
:EObjNAME.
```

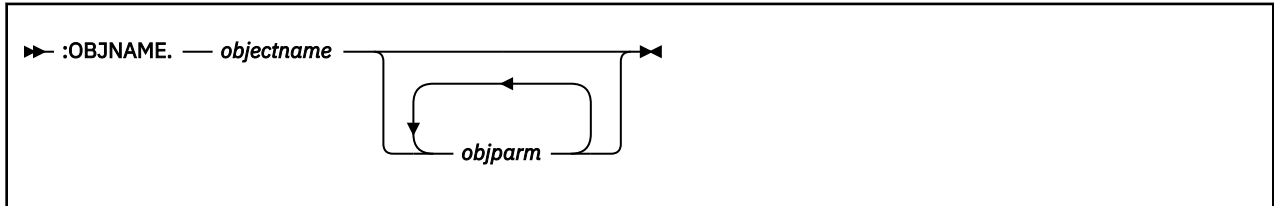
Figure 85. Example of Format 3 Build List for TXTLIB

## :OBJNAME Records

An object (:OBJNAME) record identifies the beginning of an object block, a group of records containing information about a specific object or usable form (which may consist of one or more serviceable parts).

The :OBJNAME record must be the first noncomment record in an object block. Multiple object blocks are permitted within a build list. VMFBDMOD uses the :OBJNAME record for the file name of the module being generated. VMFBDSEG uses the :OBJNAME record for the name of the saved segment being generated.

The format of the :OBJNAME record is:



### **objectname**

is the name of the object being defined. Object names must always be unique within a build list.

### **Object Names in Format 2 Build Lists**

In format 2 build lists, the object name is a combination of the object file name and the object file type, joined by a period (for example, RECEIVE.EXEC). See [Figure 86 on page 153](#) for an example of a valid object name in a format 2 build list.

The object file name can be any valid CMS file name, except BLDLIST, or an equal sign (=) for a wildcard object (see [“Using Wildcard Objects in Format 2 Build Lists”](#) on page 148). Object names must be unique within a build list.

The object file type is the type of object being generated (MODULE for a module, EXEC for an exec, SEGMENT for a saved segment, and so on).

### **Using Wildcard Objects in Format 2 Build Lists**

Wildcard objects are used to define a large group of parts that are similar in nature and have the same part and object file types. In effect, a wildcard object is really many objects. Wildcard objects are commonly used for help files, because large sets of help files frequently have the same file type, and help files are not considered as critical to the system as other objects.

To specify a wildcard object in a format 2 build list, the object file name must be an equal sign (=); and the part file name must be an asterisk (\*). These characters are used for compatibility with the CMS COPYFILE command. All files that match the specified *ftabbrev* on the :PARTID tag are included in the object. The file name of each part becomes the file name of the corresponding object.

### **Object Names in Format 3 Build Lists**

In format 3 build lists, the object name is the member name in a library. The object name can be any valid CMS file name, except BLDLIST.

### **objparm**

are parameters to use while processing the object listed on the :OBJNAME record. This field varies greatly from build list to build list. It generally contains parameters that are passed directly to the command that is used to build the object that is being processed.

### **Specifying Object Parameters**

Object parameters are specified after the object name on the :OBJNAME tag in format 2 and 3 build lists. [“Creating Objects with VMFBLD”](#) on page 325 shows the object parameters for each part handler.

## :OPTIONS Records

Part options (:OPTIONS) records are part of the object block. The :OPTIONS record indicates the part options to be used in processing the parts listed on the :PARTID records within the same object block. Its format is:



### *partopt*

are part options. This field varies greatly from build list to build list. It generally contains options that are passed directly to the command that is building the defined object. These options apply to all :PARTID tags that follow them until the next :OPTIONS tag is encountered. To specify the same options for the next object block, you must specify the options on the :OPTIONS tag for that block of :PARTID records.

When parts are language sensitive, the LANGFUNC option is included here. The LANGFUNC option is a keyword-value pair, and the value is the name of the language function processor.

For more information on part options, see [“Specifying Part Options”](#) on page 149.

:OPTIONS records must appear within an object block. Multiple :OPTIONS records are allowed. They can be mixed with :PARTID records.

The values on an :OPTIONS record remain in effect until the next options tag is encountered or the object block ends.

## Specifying Part Options

In format 2 and 3 build lists, part options are specified on :OPTIONS tags within object blocks. (In format 1 build lists, part options are specified after the parentheses on each serviceable part record). With part options, you can also specify language specific parts. [“Creating Objects with VMFBLD”](#) on page 325 shows the part options for each part handler.

## :PARTID Records

A part (:PARTID) record gives the file name and, optionally, one or more valid file type abbreviations for a serviceable part. Its format is:



### *partfn*

is the name of a serviceable part that is included in the object being defined. A *partfn* must be either a valid CMS file name or an asterisk (\*). An asterisk indicates a wildcard part (see [“Using Wildcard Objects in Format 2 Build Lists”](#) on page 148).

In a system saved segment build list, *partfn* is the name of a product saved segment build list or the name of a saved segment.

### *ftabbrev*

are the file type abbreviations for the file type of the part. At least one *ftabbrev* must be defined. More than one is allowed when multiple types of systems can be generated. Only the first abbreviation is passed to the VMFSIM GETLVL function. The others are used by the VMFBLD STATUS function. For the VMFBDBMLB part handler, *ftabbrev* can be the real file type or the file type abbreviation, see [“MACLIBS”](#) on page 344.

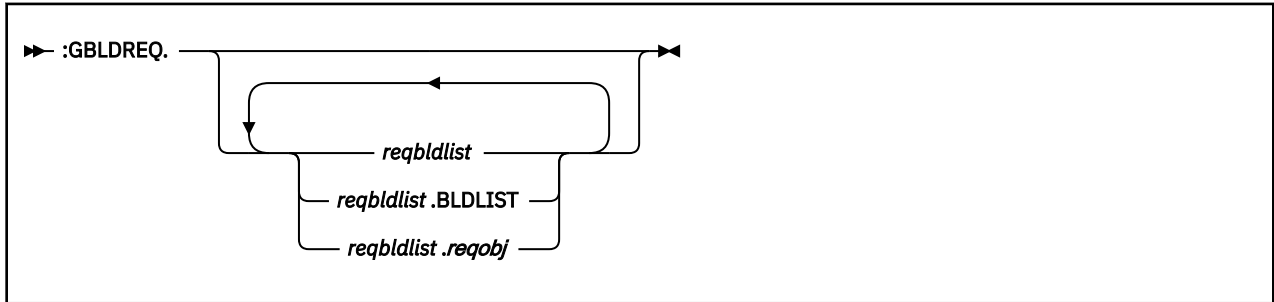
:PARTID records must appear within an object block.

### :GBLDREQ Records

Global build requisite records begin with the :GBLDREQ tag. The :GBLDREQ tag specifies the requisites for all objects in the build list that do not contain a :BLDREQ tag. Use the :GBLDREQ record when many or all of the objects in a build list have the same build requirements.

When a requisite object is built, the status of the object specifying the requisite is changed to the same status as the requisite in the service-level build status table. Before this object is built, VMFBLD builds any of its requisites that have been serviced.

The format of a :GBLDREQ record is:



***reqbldlist***

is a build list that contains a requisite object. It must be a valid CMS file name.

***reqobj***

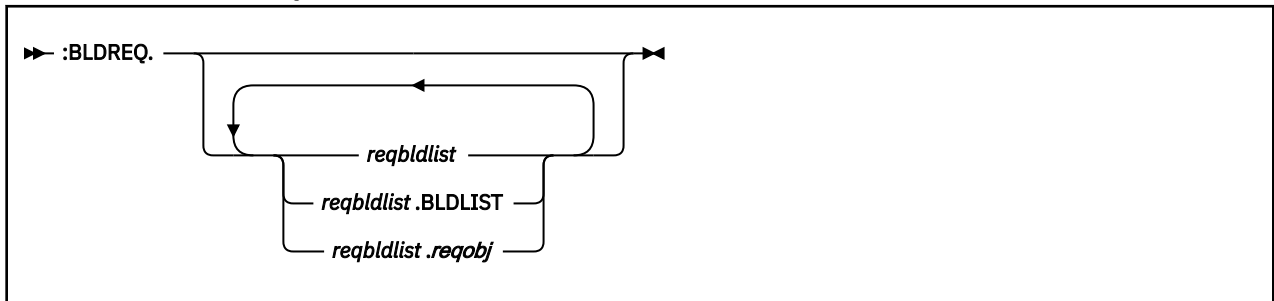
is a requisite object. It must be a valid object name. All requisite objects must be built before the object being defined is built. If all objects in a build list are requisites, BLDLIST is used as the object name because it represents the entire build list. For format 1 build lists, the object name is BLDLIST. See “Object Names in Format 2 Build Lists” on page 148 and “Object Names in Format 3 Build Lists” on page 148 for more information on valid object names.

### :BLDREQ Records

Build requisite records begin with a :BLDREQ tag. The :BLDREQ tag is used in object blocks, and it specifies the requisites for the object defined by the object block. Object requisites defined on :BLDREQ tags override any requisite objects defined on :GBLDREQ tags.

When a requisite object is built, the status of the object specifying the requisite is changed to the same status as the requisite in the service-level build status table. Before this object is built, VMFBLD builds any of its requisites that have been serviced.

The format of a :BLDREQ record is:



***reqbldlist***

is a build list that contains a requisite object. It must be a valid CMS file name.

***reqobj***

is a requisite object. It must be a valid object name. All requisite objects must be built before the object being defined is built. If all objects in a build list are requisites, BLDLIST is used as the object name because it represents the entire build list. For format 1 build lists, the object name is BLDLIST.





```

* This build list has wild cards for file names
* and specifies the LANGFUNC and VMFLANG options.
:FORMAT.2
:OBJNAME.RECEIVE.EXEC      1
:PARTID.RECEIVE EXC
:EOBJNAME.
:OBJNAME.RECEIVE.XEDIT
:PARTID.RECEIVEX XED
:EOBJNAME.
:OBJNAME.=.HELPCMS
:OPTIONS.LANGFUNC VMFLANG 2
:PARTID.* HCM
:EOBJNAME.
:

```

Figure 86. Format 2 Build List Example

In this example, **1** shows the file name and file type are joined together by a period to form the object name. **2** shows the language function, VMFLANG, has been specified on the :OPTIONS tag.

Figure 87 on page 153 shows an example of a system saved segment build list.

```

:FORMAT. 2
:OBJNAME. CMSAMS.SEGMENT
:PARTID. CMSAMS DMY      1
:EOBJNAME.
:OBJNAME. CMSBAM.SEGMENT
:PARTID. DMSSBBAM EXC   2
:PARTID. CMSBAM DMY
:EOBJNAME.
:OBJNAME. CMSDOS.SEGMENT
:BLDREQ. SEGBLIST.DOSINST.SEGMENT 3
:PARTID. DMSSBDOS EXC
:PARTID. CMSDOS DMY
:EOBJNAME.
:
:OBJNAME. DOSINST.SEGMENT 4
:PARTID. DMSSBDOS EXC
:PARTID. DOSINST DMY
:EOBJNAME.

```

Figure 87. System Saved Segment Build List Example

In this example, **1** shows the part ID for a saved segment that does not have a product build list. The file name of the record is the name of the saved segment, and the file type (DMY) is a place holder. **2** shows the additional part ID for a saved segment that does have a product build list. **3** shows a build requisite. The requisite saved segment, **4**, must be defined in the same build list.

Figure 88 on page 153 shows an example of a product saved segment build list.

```

:FORMAT. 2
:OBJNAME. CMSFILES.SEGMENT
:BLDREQ. SERVLOAD.DMSDAC.MODULE 1
          SERVLOAD.DMSSAC.MODULE 2
          DMSBL493.DMSDAC.LSEG    3
          DMSBL493.DMSSAC.LSEG    4
:OPTIONS. LOADFUNC(LSEG DMSDAC) 5
          LOADFUNC(LSEG DMSSAC) 6
:EOBJNAME.

```

Figure 88. Product Saved Segment Build List Example

This example shows a physical saved segment that contains two logical saved segments, each of which contains a module. The two modules, **1** and **2**, and the two logical saved segments, **3** and **4**, are all build requisites for the physical saved segment. The example also shows two part options, **5** and **6**, specifying the load functions for the two logical saved segments.

## Format 3 Build List Example

Figure 89 on page 154 shows a sample format 3 build list.

```

:FORMAT.3
:LIBNAME.LOADLIBA 1
:GBLDREQ.TXTLIBA.BLDLIST 2
:OBJNAME.LOADMODA LEPARMS NCAL LIST XREF LET RENT SIZE 256K 32K 3
:PARTID.LOADMODA TXT 4
:OPTIONS.INCLUDE TXTLIBA(MEMBERA) 5
          INCLUDE TXTLIBA(MEMBERB)
          INCLUDE TXTLIBA(MEMBERC)
          INCLUDE TXTLIBA(MEMBERD)
          INCLUDE TXTLIBA(MEMBERE)
:PARTID.USEREXIT TXT 4
:OPTIONS.ENTRY LOADMODA
:EOBJNAME.
:

```

Figure 89. Format 3 Build List Example

In this example, **1** shows the name of the library to be created.

**2** shows the requisite objects that apply to all objects in the build list. The requisite is an entire TXTLIB. It is a requisite because it must be built to the latest level of the TXTLIB before the LOADLIB can be built.

**3** shows the name of the object (or member of the LOADLIB) and the object parameters.

**4** shows TXT decks are also being included from CMS minidisks.

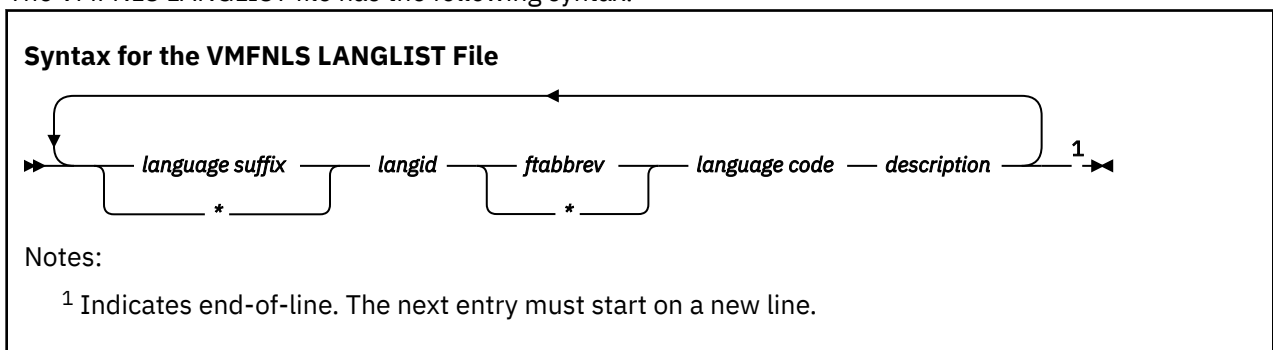
**5** shows five members of a TXTLIB are being included.

## The National Language Support Table (VMFNLS LANGLIST)

The VMFNLS LANGLIST file contains a list of language IDs (*langid*), language names, language codes (*language code*), and language descriptions, as well as other information for compatibility with previous releases of VMSES/E. Information in this file supports national language processing.

### File Syntax

The VMFNLS LANGLIST file has the following syntax.



**Note:** In the VMFNLS LANGLIST file, each record begins on a new line.

#### **language suffix**

is 1- to 2-character suffix that is appended to the file name. An asterisk (\*) indicates there is no language suffix.

#### **langid**

is a 5-character identifier for a language. *langid* is supported for compatibility with products that exploit VMSES/E 1.1.



***ftabbrev***

is an abbreviation for a language part. This field is maintained for compatibility with releases of VMSES prior to 1.1.

**Note:** When you add new languages, enter an asterisk (\*) in this field.

***language code***

is the 3-character language code from the *National Language Support Reference Manual Volume 2*, SE09-8002.

***description***

is the description for the language.

## Example

Figure 90 on page 155 shows an example VMFNLS LANGLIST file.

```
* AMENG TAM ENU American English
B UCENG TUC UPP Uppercase English
:
```

Figure 90. The VMFNLS LANGLIST File

The first field contains the language suffix. An asterisk (\*) indicates a blank. In Figure 90 on page 155, B is the language suffix for Uppercase English.

The second field is the language ID (*langid*). For example, in Figure 90 on page 155, UCENG is the *langid* for Uppercase English.

The third field contains the file type abbreviation for the language part (*ftabbrev*). In Figure 90 on page 155, TUC is the file type abbreviation for Uppercase English.

The fourth field contains the language code (*language code*). In Figure 90 on page 155, UPP is the language code for Uppercase English.

The last field contains a description for the language. In Figure 90 on page 155, you can see the description, 'Uppercase English'.



---

## Part 4. Planning and Managing Your Software Inventories

This part of the book describes:

- The structure of the product parameter file and the Software Inventory
- The files that make up the Software Inventory
- How to use the VMFSIM and VMFINFO commands to manage your Software Inventory
- How to change the Software Inventory to an SFS Directory.

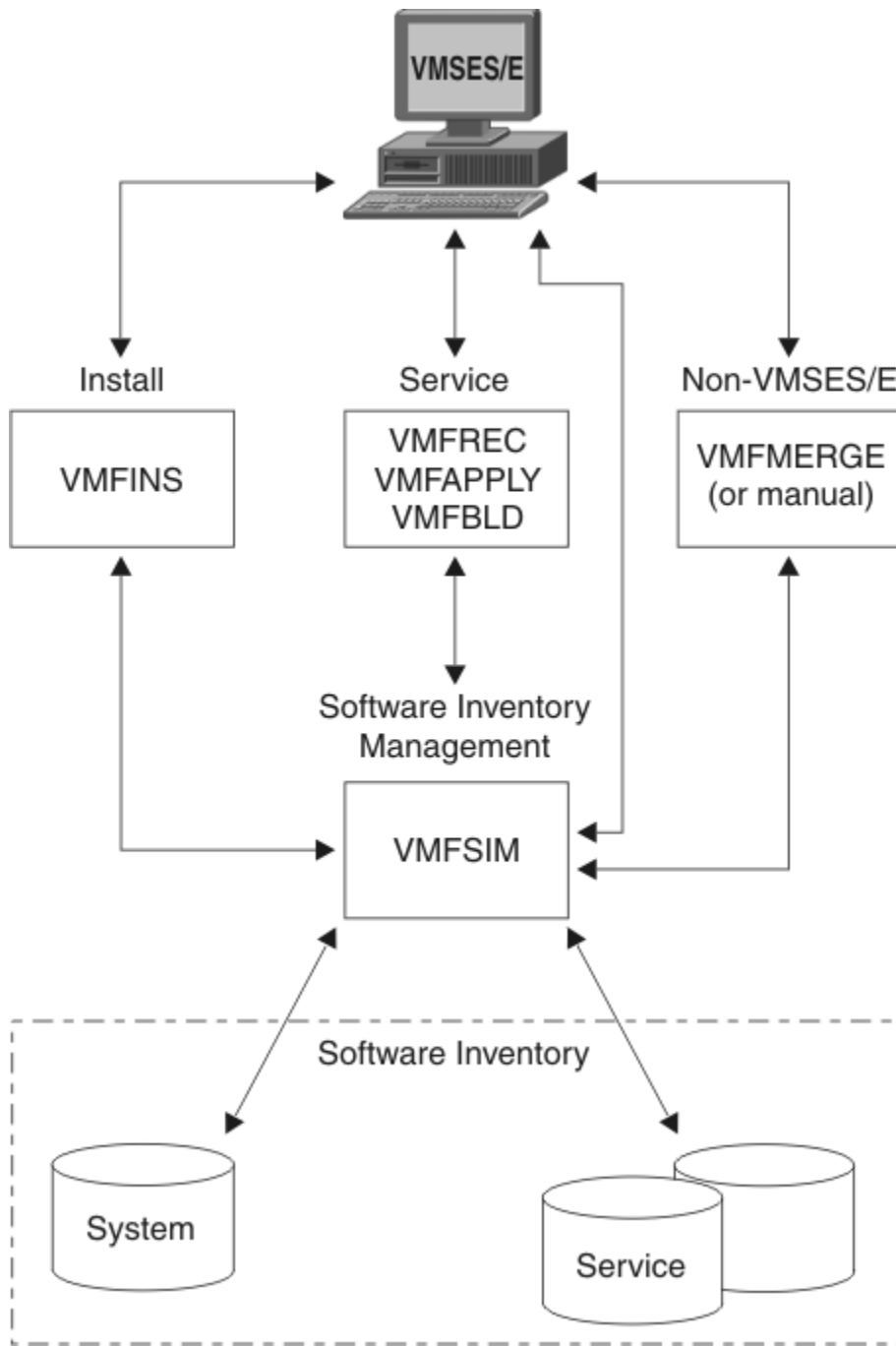


Figure 91. VMSES/E - Software Inventory Management

## Chapter 14. Introduction to the Product Parameter File

The product parameter file (also called the PPF) is the customization file for the installation and service of a product. It contains values that define and control the VMSES/E EXECs for a product, resources defined for a product (including user IDs, minidisks, and SFS directories), layouts of installation media and service tapes for a product, and the build lists necessary to build a product.

The defaults or recommended values for all of this information are supplied by the product in a source PPF. This file should not be modified directly. Instead, you should use override PPFs to update the information in the source PPF. When the source PPF is combined with all of the overrides, a usable form PPF is created. The usable form PPF is often referred to as just the PPF.

The relationship between source, override, and usable form PPFs is analogous to the relationship between a source ASSEMBLE file, update files, and a text deck. With an ASSEMBLE file, you use the CMS UPDATE facility to apply updates to the source file. With VMSES/E, you use the VMFOVER EXEC to apply override PPFs to a source PPF. The key difference between the UPDATE command and the VMFOVER EXEC is that the VMFOVER EXEC does not use line or sequence numbers to apply the changes. VMFOVER is a context-oriented facility that identifies changes in the source file by indicating in which part of which section the change is.

For example, if you want to add a user exit to the :USEREXIT tag, you do not have to know the line number of the :USEREXIT tag. You just have to remember that the :USEREXIT tag is in the :CNTRLOP section.

Just as the VMFASM, VMFHASM, and VMFHLASM execs call the UPDATE command to generate an updated source ASSEMBLE file before assembling the text deck, the VMFPPF EXEC calls the VMFOVER EXEC to generate the overridden source PPF (or temporary PPF) before compiling the usable form PPF. In addition to generating the usable form PPF, the VMFPPF EXEC also validates the syntax of the PPF.

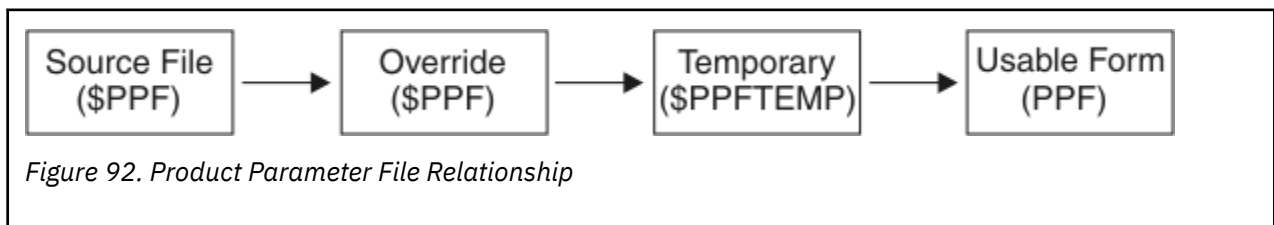
**Note:** If you are using the VMFINS EXEC, you would use the Make Override Panel to create and apply override PPFs to a source PPF. See [“Using the Make Override Panel”](#) on page 33 for an example of creating a PPF override.

### Types of Product Parameter Files

There are four types of product parameter files (PPFs) used by VMSES/E:

- Source product parameter files
- Override product parameter files
- Temporary product parameter files
- Usable form product parameter files

Each type of product parameter file has a distinct function in the VMSES/E environment. [Figure 92](#) on page 159 shows the order in which product parameter files are created.



### Source Product Parameter Files

Source PPFs are supplied with the product. The file name of the source PPF matches the *prodid* of the product, and the file type is \$PPF. Source PPFs contain the defaults and recommendations for the

## Introduction to the Product Parameter File

product. Source PPFs may also contain override areas, which are equivalent to override PPFs. Override areas in source PPFs are supplied when it is anticipated that many users will want or need certain changes to the defaults for the product.

To find out more about the content and syntax of the source product parameter file, see [“Source Product Parameter File Syntax”](#) on page 623.

## Override Product Parameter Files

Override PPFs can be supplied with the product or created by the user. They can have any file name, but they must have a file type of \$PPF. Override PPFs can contain pointers to source PPFs, and they can contain pointers to other override PPFs, which in turn point to source PPFs. In other words, you can create chains of overrides to a source PPF. Override PPFs can contain multiple override areas that may be part of multiple override chains.

The usable form PPF is named using the file name of the last override PPF in the chain. (The first override in the chain is the one that points directly to the source PPF). It is a good strategy, therefore, to group the ends of your override chains into override PPFs that identify the groupings by their file names.

To find out more about the content and syntax of the override product parameter file, see [“Override Product Parameter File Syntax”](#) on page 653.

## Temporary Product Parameter Files

Temporary PPFs are the output of the VMFOVER EXEC. The file name of a temporary PPF matches either the file name of the last override PPF in the chain of overrides or the file name of the source PPF, if there are no overrides. The file type is \$PPFTEMP. You can run the VMFOVER EXEC while you are designing your overrides to validate the resulting temporary PPFs before you include them in a usable form PPF.

To find out more about the content and syntax of the temporary product parameter file, see [“Temporary Product Parameter File Syntax”](#) on page 654.

## Usable Form Product Parameter Files

Usable form PPFs are used by the majority of VMSES/E execs. The file name of the usable form PPF matches the file name of either the last override PPF in the chain of overrides or the file name of the source PPF, if there are no overrides. The file type is PPF.

A usable form PPF can contain multiple component areas. Each component area contains a complete set of parameters for a product. When you see *ppfname* and *compname* included as operands on VMSES/E execs, *ppfname* is referring to the file name of the usable form PPF; and *compname* is referring to a specific component area in the usable form PPF.

To find out more about the content and syntax of the usable form product parameter file, see [“Usable Form Product Parameter File Syntax”](#) on page 654.

## Sections of the Product Parameter File

---

The product parameter file is divided into sections that contain related information. These sections are:

- The control options section
- The variable declarations section
- The minidisk/directory assignments section
- The receive installation tape definition section
- The receive service media definition section
- The build product definitions section
- The file type abbreviations extensions section

## The Control Options Section

The control options section identifies parameters that are used to control the operation of VMSES/E and is delimited by the :CNTRLOP and :ECNTRLOP tags. For more information, see [“Control Options Section” on page 625](#).

## The Variable Declarations Section

The variable declarations section identifies variables and the values assigned to them in the PPF. In source and override PPFs, the variables defined in this section are used in the minidisk/directory assignment section. In the usable form PPF, this section is used for resource allocation and minidisk linking. The VMFPPF EXEC substitutes the values of these variables into the minidisk/assignment section when it creates the usable form PPF. The variables can also be used as parameters on product processing exits in the receive installation tape definition and receive service media definition sections of the PPF. For more information, see [“Variable Declarations Section” on page 631](#).

## The Minidisk/Directory Assignments Section

The minidisk/directory assignments section identifies the symbolic strings of minidisks or SFS directories that make up the service database of the product. In source and override PPFs, this section may contain variables that are defined in the variable declarations section. In the usable form PPF, all of these variables have been resolved by the VMFPPF EXEC. For more information, see [“Minidisk/Directory Assignments Section” on page 633](#).

## The Receive Installation Tape Definition Section

The receive installation tape definition section defines the layout of the installation tape for the product. It also identifies how each tape file is processed and where it is loaded. For more information, see [“Receive Installation Tape Definition Section” on page 637](#).

## The Receive Service Media Definition Section

The receive service media definition section defines the layout of service tapes and envelopes for the product. It also identifies how each tape file is processed and where it is loaded. For more information, see [“Receive Service Media Definition Section” on page 639](#).

## The Build Product Definitions Section

The build product definition section defines build lists and objects for the product. It also identifies how each object is processed and where it is built. For more information, see [“Build Product Definition Section” on page 641](#).

## The File Type Abbreviations Extensions Section

The file type abbreviations extensions section defines file type abbreviations for the product that override entries in the file type abbreviation table (VM SYSABRVT). For more information, see [“File Type Abbreviations Extensions Section” on page 644](#).

## Syntax of the Product Parameter File

---

See Chapter 21, “Product Parameter File Syntax,” on page 621 for a complete description of the syntax for the different types of product parameter files.





---

## Chapter 15. Introduction to the Software Inventory

VMSES/E maintains a software inventory. The software inventory consists of files that contain control and status information, which are used during product installation and service. The software inventory has two levels, the system-level software inventory and the service-level software inventory.

Each level of the software inventory contains the following files:

- A description table, describing the product or PTF
- A requisite table, describing the requisites defined for the product or PTF
- A receive status table, describing when the product or PTF was received
- An apply status table, describing the apply status of the product or PTF
- A build status table, describing the build status of the product or PTF

The nature of the information in each table differs depending on whether the table is in the system-level software inventory or the service-level software inventory.

The system-level software inventory contains information on all products installed on a system.

The service-level software inventory contains information about all service applied to each product in a system. VMSES/E maintains service-level Software Inventories only for products that use VMSES/E for service.

[Figure 93 on page 164](#) shows the two levels of the software inventory.

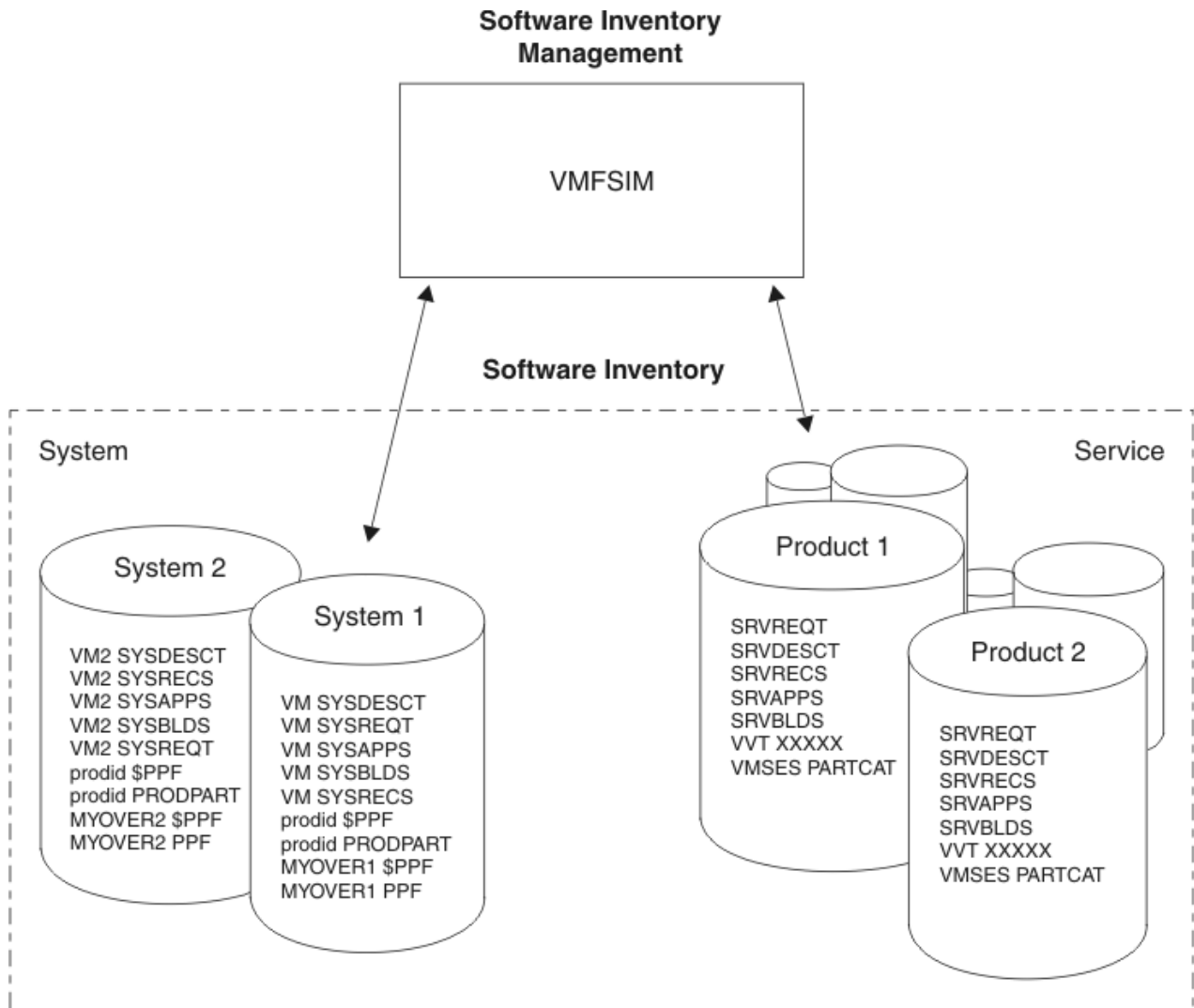


Figure 93. Software Inventory Files in the VMSES/E Database

## Overview of the System-Level Software Inventory

The files in the system-level software inventory are:

- [“The Product Parts \(PRODPART\) File” on page 166](#)
- [“The Saved Segment Data \(SEGDATA\) File” on page 166](#)
- [“The System-Level Description Table \(VM SYSDESCT\)” on page 166](#)
- [“The System-Level Requisite Table \(VM SYSREQT\)” on page 167](#)
- [“The System-Level Receive Status Table \(VM SYSRECS\)” on page 167](#)
- [“The System-Level Apply Status Table \(VM SYSAPPS\)” on page 168](#)
- [“The System-Level Build Status Table \(VM SYSBLDS\)” on page 168](#)
- [“The System-Level Service Update Facility Table \(VM SYSSUF\)” on page 169](#)
- [“The System-Level Restart Table \(VM SYSREST\)” on page 170](#)
- [“The File Type Abbreviation Table \(VM SYSABRVT\)” on page 170](#)
- [“The Parts Catalog \(VMSES PARTCAT\)” on page 170](#)

All files in the system-level software inventory, except the parts catalog and the file type abbreviation table, reside on the software inventory minidisk or directory, which is defined to be MAINT<sub>vm</sub>'s 51D minidisk by default. Product parameter files also reside on the software inventory minidisk or directory.

If you are maintaining multiple systems, you will have a software inventory minidisk or directory for each system.

**Note:** The default file name for all system-level software inventory files (except the product parts file, the saved segment data file, and the parts catalog) is VM. You can modify this name, if you are supporting multiple systems, using the SYSTEM option of the VMFINS command. For more information, see [“Changing the Software Inventory Defaults”](#) on page 660.

## Types of Information Provided

The files in the system-level software inventory contain four types of information:

- System information
- Product information
- Build status information
- Supporting information

### System Information

System information is contained in the:

- Saved segment data file

The saved segment data file contains information about all the saved segments defined by all the products on the system.

### Product Information

Product information is contained in the:

- Product parts file
- System-level description table
- System-level requisite table
- System-level receive status table
- System-level apply status table
- System-level service update facility table

Each time a product is loaded to a system, VMFINS EXEC uses information from the PRODPART file for the product to update the system-level software inventory with product-specific information. VMFSUFTB EXEC updates the system-level service update facility table from product information found in the other tables. Product information includes:

- The product identifier
- The file name of the product parameter file (PPF) that was used to process the product
- The component name in the PPF that was used to load the product
- The description of the product
- The requisite relationships defined for the product
- The apply status of the product
- The date and time the product was received and the user ID used for the receive operation
- The date and time the product was applied and the user ID used for the apply operation

### Build Status Information

Build status information is contained in the:

- System-level build status table.

## The Contents of the System-Level Software Inventory

Each time a product is built on a system, the VMFINS BUILD command updates the system-level build status table with build status information. Build status information is also updated when a product is deleted from the system. Build status information includes:

- The product identifier
- The file name of the PPF that was used to process the product
- The component name in the PPF that was used to build the product
- The build status of the product
- The date and time the product was built
- The user ID used for the build operation

### Supporting Information

Supporting information is contained in the:

- File type abbreviation table

The file type abbreviation table is loaded as part of the VMSES/E component. It is used by VMSES/E to process PTF-numbered parts. The information in the file type abbreviation table includes:

- The 3-character abbreviated file type used in renaming serviceable parts
- The 1-8 character real CMS file type corresponding to the abbreviation
- The 1-8 character identifier used for the associated base-level part

- System-level restart table

The system-level restart table stores data needed to restart the VMFSUFIN EXEC.

## The Contents of the System-Level Software Inventory

Each of the files in the system-level software inventory is described below. For detailed information about the syntax of the entries in each file, see [Chapter 22, “Software Inventory Syntax,” on page 659](#).

**Note:** The data in the examples of the system-level software inventory files may not appear exactly as shown in the files on your own system.

### The Product Parts (PRODPART) File

VMSES/E uses information in the product parts file, which is included on a product's installation media, to update entries in the system-level software inventory each time a product is loaded onto your system. The product parts file also contains the default definitions for the saved segments used by the product. To find out more about the content and the syntax of the product parts file, see [“The Product Parts \(PRODPART\) File” on page 660](#).

### The Saved Segment Data (SEGDATA) File

The saved segment data file contains customized information for building the set of saved segments on a z/VM system, which includes saved segments defined by the product and saved segments defined by application products that run on z/VM. The set of saved segments on a z/VM system is identified in a system saved segment build list. To find out more about the content and syntax of the saved segment data file, see [“The Saved Segment Data \(SEGDATA\) File” on page 677](#).

### The System-Level Description Table (VM SYSDESC)

The system-level description table contains the description of a product that has been received on the system.

The system-level description table resides on the software inventory disk and is updated by the VMFINS EXEC during receive processing for installation media. Information from the PRODPART files is used to update this table.

### Example

Figure 94 on page 167 shows an example of the system-level description table.

```
:PPF. ESAINS MYCOMP :PRODID. 1VMVMC23 :DESC. Component of z/VM
```

Figure 94. System-Level Description Table Example

### Contents

Each entry in the table contains the file name of the product parameter file used to process the product, the component name in the PPF used to load the product, the product identifier, and a text description of the product.

To find out more about the syntax and content of the system-level description table, see [“The System-Level Description Table \(VM SYSDESCT\)”](#) on page 684.

### The System-Level Requisite Table (VM SYSREQT)

The system-level requisite table contains the relationships between products.

The system-level requisite table resides on the software inventory disk and is updated during receive processing for installation media.. Information from the PRODPART files is used to update this table.

### Example

Figure 95 on page 167 shows an example of the system-level requisite table.

```
:1VMVMC23%MYCOMP :PRREQ.1VMVMP11 << 1VMVMS10 | 1VMVMS20 >>
                  :COREQ. 1VMVMF10%PRODUCT
                  :SUP. 1VMVMC22
                  :IFREQ. 1VMVME10
                  :NPRES. 1VMVMG10
```

Figure 95. System-Level Requisite Table Example

### Contents

Each entry in the table contains a product identifier and the requisites defined for the product.

To find out more about the syntax and content of the system-level requisite table, see [“The System-Level Requisite Table \(VM SYSREQT\)”](#) on page 686.

### The System-Level Receive Status Table (VM SYSRECS)

The system-level receive status table contains a list of all products that have been received on the system.

The system-level receive status table resides on the software inventory disk and is updated by the VMFINS EXEC during receive processing for installation media. Information from the PRODPART files is used to update this table.

### Example

Figure 96 on page 168 shows an example of the system-level receive status table.

```
:PPF.ESAINS MYCOMP :PRODID.1VMVMC23%MYCOMP :STAT.RECEIVED.06/01/22.10:10:10.MAINT.9101-911  
:PPF.ESATEST MYCOMP2 :PRODID.1VMVMS10%MYCOMP2 :STAT.DELETED.06/02/22.11:11:11.JONES
```

Figure 96. System-Level Receive Status Table Example

### Contents

Each entry in the table contains the:

- File name of the product parameter file used to process the product
- Component name in the PPF used to load the product
- Product identifier
- Receive status of the product
- Date and time the product was received
- User ID that was used for the receive operation
- Service level of the product when it was received

To find out more about the syntax and content of the system-level receive status table, see [“The System-Level Receive Status Table \(VM SYSRECS\)”](#) on page 688.

### The System-Level Apply Status Table (VM SYSAPPS)

The system-level apply status table contains a list of all products that have been applied on the system. It also identifies the file name of the PPF used to process the product and the component name in the PPF that was used to load the product.

The system-level apply status table resides on the software inventory disk and is updated during processing for installation media.

### Example

Figure 97 on page 168 shows an example of the system-level apply status table.

```
:PPF.ESAINS MYCOMP :PRODID.1VMVMC23%MYCOMP :STAT.APPLIED.06/01/22.11:11:11.MAINT  
:PPF.ESATEST MYCOMP2 :PRODID.1VMVMS10%MYCOMP2 :STAT.DELETED.06/02/22.10:10:10.JONES
```

Figure 97. System-Level Apply Status Table Example

### Contents

Each entry in the table contains the file name of the PPF used to process the product, the component name in the PPF used to load the product, the product identifier, and the apply status of the product. The apply status includes the date the product was received, the time the product was applied, and the user ID that was used for the receive operation.

To find out more about the syntax and content of the system-level apply status table, see [“The System-Level Apply Status Table \(VM SYSAPPS\)”](#) on page 690.

### The System-Level Build Status Table (VM SYSBLDS)

The system-level build status table contains a list of all products that have been built on the system. It also identifies the file name of the PPF and the component name in the PPF that was used to build the product.

The system-level build status table resides on the software inventory disk and is updated during build processing for installation media.

### Example

Figure 98 on page 169 shows an example of the system-level build status table.

```
:PPF.ESAINS MYCOMP :PRODID.1VMVMC23%MYCOMP :STAT.BUILT.06/01/22.10:10:10.MAINT
:PPF.ESATEST MYCOMP2 :PRODID.1VMVMS10%MYCOMP2 :STAT.SUPED.06/10/22.11:11:11.MAINT
:PPF.ESATEST MYCOMP3 :PRODID.1VMVMP11%MYCOMP3 :STAT.DELETED.06/10/22.12:12:12.MAINT
```

Figure 98. System-Level Build Status Table Example

### Contents

Each entry in the table contains the file name of the PPF used to process the product, the component name in the PPF used to build the product, the product identifier, and the build status of the product. The build status includes the date and time the product was built and the user ID that was used for the build operation.

To find out more about the syntax and content of the system-level build status table, see [“The System-Level Build Status Table \(VM SYSBLDS\)”](#) on page 692.

### The System-Level Service Update Facility Table (VM SYSSUF)

The system-level service update facility table contains a list of all products that are installed on the system and related data needed by the automated service commands.

The system-level service update facility table resides on the software inventory disk and is updated by VMFSUFTB EXEC.

### Example

Figure 99 on page 169 shows an example of the system-level service update facility table.

```
:PRODID.1VMVMC23%MYCOMP :SERVLEV.RSU-0701 :DESC.Component of z/VM
:INCLUDE.YES :INSTALL.YES :INSPPF.SERV2P MYCOMP :BUILD.YES :BLDPPF.SERV2P MYCOMP
:P2PPPF.SERV2P MYCOMPP2P :PRODLEV.RSU-0701
```

Figure 99. System-Level Service Update Facility Table

### Contents

Each entry in the table might contain:

- Product identifier
- Service level of the product
- Text description of the product
- Tag to indicate if preventive service for the product is to be automatically installed
- Product parameter file name and component name to be used to install service
- Tag to indicate if installed service is to be automatically built
- Product parameter file name and component name to be used to build serviced files
- Product parameter file name and component name to be used to put the component into production (optional)
- Production service level of product (optional)

To find out more about the syntax and content of the system-level service update facility table, see [“The System-Level Service Update Facility Table \(VM SYSSUF\)”](#) on page 693.

## The System-Level Restart Table (VM SYSREST)

The system-level restart table contains records used to restart the VMFSUFIN EXEC. This table resides on the software inventory disk and is updated by the VMFSUFIN EXEC.

### Example

Figure 100 on page 170 shows an example of the system-level restart table.

```
:PACKAGE.BUILD.7VMTCP30%TCPIP :PRODID.7VMTCP30%TCPIP :STAT.INIT.09/12/22.14:28:57.MAINT730
```

Figure 100. System-Level Restart Table Example

### Contents

Each entry contains the data needed to restart a service install invocation. This data includes:

- Names of the RSU service envelopes or COR bucket service envelopes or both
- Product identifiers
- Restart data

To find out more about the syntax and content of the system-level restart table, see [“The System-Level Restart Table \(VM SYSREST\)”](#) on page 696.

## The File Type Abbreviation Table (VM SYSABRVT)

The file type abbreviation table contains a map of the 3-character abbreviations for file types to their corresponding real CMS file types and their base file types. (A base file type is the file type used when there is no service). This translation is required by VMSES/E when processing PTF-numbered parts. The file type abbreviation table is shipped as part of the VMSES/E component.

### Example

Figure 101 on page 170 shows an example of the file type abbreviation table.

```
:ABBRFT.CPY      :REALFT.COPY      :BASEFT.CPY000000  
:ABBRFT.EXC      :REALFT.EXEC      :BASEFT.EXC000000  
:ABBRFT.HCM      :REALFT.HELPCMS   :BASEFT.HCM000000  
:ABBRFT.MAC      :REALFT.MACRO     :BASEFT.MAC000000  
:ABBRFT.TXT      :REALFT.TEXT      :BASEFT.TXT000000  
:ABBRFT.XED      :REALFT.XEDIT     :BASEFT.XED000000  
:ABBRFT.$EX      :REALFT.$EXEC     :BASEFT.$EX000000  
:ABBRFT.$XE      :REALFT.$XEDIT    :BASEFT.$XE000000
```

Figure 101. File Type Abbreviation Table Example

### Contents

Each entry in the table contains a 3-character abbreviated file type, the 1-8 character real CMS file type corresponding to the abbreviation, and a 1-8 character identifier used for the associated base-level part.

To find out more about the syntax and content of the file type abbreviation table, see [“The File Type Abbreviation Table \(VM SYSABRVT\)”](#) on page 702.

## The Parts Catalog (VMSES PARTCAT)

The parts catalog is a set of software inventory files that catalogs all parts of a product on a VMSES/E target minidisk or SFS directory. All product parts are cataloged when they are loaded onto the system,



when they are generated, and when they are moved. The VMFINS DELETE function reads the catalogs to determine which product parts may be deleted when removing a product from the system.

A parts catalog, called VMSES PARTCAT, resides on each minidisk or SFS directory that is the target of a merge disk, receive, apply, or build operation.

The parts catalog is updated each time VMSES/E loads, modifies or creates a part on any minidisk or SFS directory identified in the :MDA section of the PPF. This minidisk or SFS directory is called the target of the operation.

**Note:** VMSES/E may create temporary work files on targets. Temporary work files that are used only by a single operation are not cataloged. VMSES/E does catalog temporary files created by an operation that are to be used by a subsequent operation.

### Example

Figure 102 on page 171 shows an example of the parts catalog.

```
:PARTID.DMSABC COPY :PRODID.1VMVMC23 :STAT.VMFREC.06/11/22.10:10:10.MAINT
:PARTID.DMSXXX TEXT :PRODID.1VMVMC23 :STAT.VMFREC.06/11/22.10:10:10.MAINT
:PARTID.RECEIVE EXEC :PRODID.1VMVMC23 :STAT.VMFREC.06/11/22.10:10:10.MAINT
```

Figure 102. Parts Catalog Table Example

### Contents

Each entry in the table contains a part name and type, the product identifier of the product that owns the part, the VMSES/E command that placed the part on the target, date and time the part was placed on the target, and the user ID that was used for the operation.

To find out more about the syntax and content of the parts catalog, see [“The Parts Catalog \(VMSES PARTCAT\)”](#) on page 703.

### How Receive Processing Affects the Parts Catalog

Parts are cataloged by the receive operation before they are loaded. If a recoverable error, such as a tape drive failure, occurs during the load, the receive operation deletes the entries in the parts catalog. If an irrecoverable error, such as a system ABEND, occurs during the load, the parts catalog may contain entries for more parts than were actually loaded to the targets. Restarting the receive operation corrects this discrepancy.

### How Apply Processing Affects the Parts Catalog

Parts are cataloged by the apply operation before they are generated. If a recoverable error occurs during the load, the apply operation deletes the entries in the parts catalog. If an irrecoverable error, such as a system ABEND, occurs while applying service, the parts catalog may contain more entries for more parts than were actually created on the targets. Restarting the apply operation corrects this discrepancy.

### How Build Processing Affects the Parts Catalog

Parts are cataloged by the build operation before they are generated. If a recoverable error occurs while a part is being built, the build operation deletes the entries in the parts catalog. If an irrecoverable error, such as a system ABEND, occurs while building a part, the parts catalog may contain more entries for more parts than were actually created on the targets. Restarting the build operation corrects this discrepancy.

## Overview of the Service-Level Software Inventory

The files in the service-level software inventory are:

- [“The \\$PTFPART File”](#) on page 173

## The Contents of the System-Level Software Inventory

- [“The Service-Level Description Table \(VM SRVDESCT\)” on page 173](#)
- [“The Service-Level Requisite Table \(VM SRVREQT\)” on page 174](#)
- [“The Service-Level Receive Status Table \(VM SRVRECS\)” on page 174](#)
- [“The Service-Level Apply Status Table \(appid SRVAPPS\)” on page 175](#)
- [“The Version Vector Table \(appid VVTlvld\)” on page 176](#)
- [“The Service-Level Build Status Table \(bldid SRVBlds\)” on page 175](#)

prodid, appid, and bldid are the values assigned to the :RECID, :APPID, and :BLDID tags in the product parameter file. lvld identifies the maintenance level and is obtained from the AUX record of the control file that has been identified for the product. (See [“How VMSES/E Uses Control Files” on page 117.](#)) The default control file for the product is identified by the value assigned to the :CNTRL tag in the product parameter file.

**Note:** If you are maintaining one system, the values of *prodid*, *appid*, and *bldid* are usually all the same. If you are maintaining multiple systems, you vary the values of *appid* and *bldid*.

All files in the service-level software inventory reside on each product’s APPLY and DELTA strings. (Remember, VMSES/E maintains service-level Software Inventories only for products that use VMSES/E for service).

## Types of Information Provided

The files in the service-level software inventory contain three types of information:

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

### PTF Information

PTF information resides on the DELTA string and is contained in the following files:

- The \$PTFPART file
- The service-level receive status table
- The service-level requisite table
- The service-level description table

Each time service is loaded to a system, the VMFREC EXEC uses the \$PTFPART file, which is received on the service package, to update the service-level software inventory with PTF-specific information. PTF-specific information includes:

- The PTF number
- The receive status of the PTF
- The date and time the PTF was received and the user ID used for the receive operation
- The APAR number
- The APAR description
- A mapping of PTF number to APAR number
- The requisite relationships defined for the PTF

### Maintenance-Level Information

Maintenance-level information resides on the APPLY string and is contained in the following files:

- The service-level apply status table
- The version vector table

Each time service is applied to a product, the VMFAPPLY EXEC updates the service-level software inventory with maintenance-level information. Maintenance-level information includes:

- The PTF number
- The apply status of the PTF
- The date and time the PTF was processed and the user ID that was used for the apply operation
- The PTFs (and their associated APARs) that have been processed for each part
- The service history of all parts serviced

## Build Status Information

Build status information also resides on the APPLY string. Build status information is contained in the following file:

- The service-level build status table

Each time a serviced product is re-built, the VMFBLD EXEC updates the service-level software inventory with build status information. Build status information includes:

- A build list name
- An object name in the specified build list
- The build status of the object and, if processing was unsuccessful, an error qualifier
- The date and time the object was processed and the user ID used for the build operation

## The Contents of the Service-Level Software Inventory

Each of the files in the service-level software inventory is described below. For detailed information about the syntax of the entries in each file, see [Chapter 22, “Software Inventory Syntax,”](#) on page 659.

**Note:** The data in the examples of the service-level software inventory files may not appear exactly as shown in the files on your own system.

### The \$PTFPART File

VMSES/E uses a file called the \$PTFPART file, which is included on a product's service media, to update entries in the service-level software inventory each time a PTF is loaded onto your system. To find out more about the content and the syntax of the \$PTFPART file, see [“The Source Product Parameter File”](#) on page 13.

### The Service-Level Description Table (VM SRVDESCT)

The service-level description table contains the abstract information for an APAR that has been received on the system.

The service-level description table resides on the product's DELTA disk string and is updated by the VMFREC EXEC during receive processing for service tapes. Information from the \$PTFPART file is used to update this table. The VMFAPPLY EXEC uses this information to add comments to the AUX file it builds for a serviced part.

#### Example

Figure 103 on page 173 shows an example of the service-level description table.

```
:APARNUM.VM23456 :ABSTRACT.Fix problem with CMS IPL  
:APARNUM.VM22222 :ABSTRACT.DMSABC branches to location FFFFFFFF
```

Figure 103. Service-Level Description Table Example

### Contents

Each entry in the table contains an APAR number and the text of the associated APAR abstract.

To find out more about the syntax and content of the service-level description table, see [“The Service-Level Description Table \(recid SRVDESCT\)”](#) on page 712.

### The Service-Level Requisite Table (VM SRVREQT)

The service-level requisite table contains the relationships between PTFs and a mapping of PTFs to APARs.

The service-level requisite table resides on the product’s DELTA disk string and is updated by the VMFREC EXEC during receive processing for service media. The VMFAPPLY EXEC uses this information to determine PTF requisites. Information from the \$PTFPART file is used to update this table.

### Example

Figure 104 on page 174 shows an example of the service-level requisite table.

```
:PTF.UV12345 :APARNUM.VM12345
:PREREQ. UV23456
:COREQ. UV45678 UV56789
:SUP. UV77777 UV88888
:IFREQ. UV66666.1VMVMC23
:HARDREQ.VM00001
```

Figure 104. Service-Level Requisite Table Example

### Contents

Each entry in the table contains a PTF number, the associated APAR numbers, and the requisites defined for the PTF.

To find out more about the syntax and content of the service-level requisite table, see [“The Service-Level Requisite Table \(recid SRVREQT\)”](#) on page 713.

### The Service-Level Receive Status Table (VM SRVRECS)

The service-level receive status table contains a list of all PTFs that have been received for the product.

The service-level receive status table resides on the product’s DELTA disk string and is updated by the VMFREC EXEC as PTFs are processed during receive processing for service media.

### Example

Figure 105 on page 174 shows an example of the service-level receive status table.

```
:PTF.UV12345 :STAT.RECEIVED.03/03/22.11:11:11.SMITH
:PTF.UV23456 :STAT.RECEIVED.02/03/22.12:12:12.JONES
:PTF.UV01234 :STAT.COMMITTED.04/05/22.22:22:12.JONES
RECEIVED.01/10/22.06:06:06.MIKED
```

Figure 105. Service-Level Receive Status Table Example

### Contents

Each entry in the table contains a PTF number, the status of the PTF, the date and time the PTF was received, and the user ID that was used for the receive operation.

To find out more about the syntax and content of the service-level receive status table, see [“The Service-Level Receive Status Table \(recid SRVRECS\)”](#) on page 715.

## The Service-Level Apply Status Table (appid SRVAPPS)

The service-level apply status table contains a list of all PTFs that have been applied to the product.

The service-level apply status table resides on the product's APPLY disk string and is updated by the VMFAPPLY EXEC during apply processing for service media.

### Example

Figure 106 on page 175 shows an example of the service-level apply status table.

```
:PTF.UV12345 :STAT.APPLIED.03/03/22.22:22:22.JONES
:PTF.UV23456 :STAT.APPLIED.02/03/22.11:11:11.SMITH
:PTF.UV01234 :STAT.SUPED.11/10/22.12:12:12.SMITH
              APPLIED.06/06/22.02:02:02.JONES
```

Figure 106. Service-Level Apply Status Table Example

### Contents

Each entry in the table contains the PTF number and the apply status of the product. The apply status includes the date the PTF was applied, the time the PTF was applied, and the user ID that was used for the apply operation.

To find out more about the syntax and content of the service-level apply status table, see [“The Service-Level Apply Status Table \(appid SRVAPPS\)”](#) on page 716.

## The Service-Level Build Status Table (bldid SRVBLDS)

The service-level build status table contains a list of all objects that have been serviced for the product.

The service-level build status table resides on the product's APPLY string and is updated by the VMFBLD EXEC as objects are generated during build processing for service media.

### Example

Figure 107 on page 175 shows an example of the service-level build status table.

```
:LASTAPP.06/05/22 12:01:21 1VMVMC23
:BLDLIST.UNKNOWN :OBJECT.BDLIST :STAT.MANUAL.06/05/22.12:01:21.MAINT :PARTID.HCPXYZ TXT
:BLDLIST.HCPEXC :OBJECT.ABC.EXEC :STAT.BUILT.06/05/22.10:24:13.MAINT
:BLDLIST.HCPEXC :OBJECT.BDLIST :STAT.BUILT.06/05/22.10:24:13.MAINT
:BLDLIST.HCPMODS :OBJECT.HCPABC.MODULE :STAT.BUILT.06/05/22.10:35:21.MAINT
:BLDLIST.HCPMODS :OBJECT.HCPDEF.MODULE :STAT.BUILDALL.06/05/22.11:16:10.MAINT.ERROR
:BLDLIST.HCPMODS :OBJECT.BDLIST :STAT.BUILDALL.06/05/22.11:16:10.MAINT.ERROR
:BLDLIST.CPLOAD :OBJECT.BDLIST :STAT.BUILT.06/05/22.10:11:40.BILL
```

Figure 107. Service-Level Build Status Table Example

### Contents

The first line in the table shows the date and time when the table was last updated with new build requirements from the select data file. The file name of the select data file is also provided. Each entry in the table contains a build list name, the name of an object generated from the build list, the build status of that object, and possibly an error qualifier, if processing could not be completed successfully. The build status includes the date and time the object was built and the user ID that was used for the build operation.

To find out more about the syntax and content of the service-level build status table, see [“The Service-Level Build Status Table \(bldid SRVBLDS\)”](#) on page 717.

## The Version Vector Table (appid VVTlvld)

The version vector table contains a history of all parts that have been serviced. A product may have more than one version vector table associated with it. Products may have one version vector table for each AUX level identified in the product's control file.

The version vector table resides on the product's APPLY string and is updated by the VMFAPPLY EXEC during apply processing as PTFs are processed.

### Example

Figure 108 on page 176 shows an example of the version vector table.

```
:PART.DMSABC  TXT  :PTF.  UV12345.VM00001  UV23456.VM00002
:PART.RECEIVE EXC  :PTF.  UV12345.VM00001  UV34567.VM00003
:PART.FILELIST EXC  :PTF.  UV12345.VM00001.P00001DS  UV34567.VM00003.M
:PART.FSOPEN  MACRO :PTF.  UV12345.VM00001.P00001DS
```

Figure 108. Version Vector Table Example

### Contents

Each entry in the table contains a part name and type, a PTF number, an APAR number associated with the PTF number, and the 8-character file type of the source update file that contains the changes for the specified APAR. An "M" following a PTF/APAR number pair indicates the PTF and APAR have been merged into a new or refreshed source file for the part.

**Note:** The APAR number and the file type of the source update file are optional values and may not appear in each entry in the version vector table.

To find out more about the syntax and content of the version vector table, see [“The Version Vector Table \(appid VVTlvld\)”](#) on page 721.

---

## Chapter 16. Introduction to the VMFSIM EXEC

The VMFSIM EXEC is the interface to the software inventory. You can use the VMFSIM EXEC to:

- Query the software inventory (“VMFSIM QUERY” on page 567)
- Update the software inventory (“VMFSIM MODIFY” on page 562)
- Compare the version vector tables to the AUX file structure (“VMFSIM CHKLVL” on page 531)
- Identify the latest version of a part (“VMFSIM GETLVL” on page 543)
- Compare two Software Inventory tables (“VMFSIM COMPTBL” on page 538)
- Build apply and exclude lists (“VMFSIM SRVREQ” on page 579 and “VMFSIM SRVDEP” on page 573)
- List the requisite PTFs for a given PTF (“VMFSIM SRVREQ” on page 579)
- List the requisite products for a given product (“VMFSIM SYSREQ” on page 591)
- List the dependent PTFs for a given PTF (“VMFSIM SRVDEP” on page 573)
- List the dependent products for a given product (“VMFSIM SYSDEP” on page 585)
- Support your local modification structure (“VMFSIM LOGMOD” on page 556)
- Initialize and recover the software inventory with products or PTFs (“VMFSIM INIT” on page 550)

You see examples of how to use the VMFSIM EXEC in the following sections. The VMFSIM EXEC and its command syntax are described in [“The Source Product Parameter File” on page 13](#).

The VMFINFO EXEC provides a user-friendly, panel interface to the VMFSIM EXEC and the Software Inventory. For more information on the VMFINFO EXEC and the VMFINFO panels, see [Chapter 17, “Using the VMFINFO Panels,” on page 199](#) and [“VMFINFO EXEC” on page 402](#).

---

### Providing Input to VMFSIM

VMFSIM uses tagged data as input. There are three ways to enter tagged data for VMFSIM commands:

- From the command line
- In a REXX stem
- From a file

For more information on using tagged data as input, see [“VMFSIM: Tagged Data \(TDATA\)” on page 527](#).

---

### Receiving Output from VMFSIM

Output TDATA statements are returned from VMFSIM to the terminal display, a file, or a REXX stem.

---

### Querying the Software Inventory

The VMFSIM QUERY function is the main interface to the software inventory. You can either use the VMFINFO panels to run VMFSIM queries, or you can enter VMFSIM QUERY commands manually. To find out how to use the VMFINFO panels, see [Chapter 17, “Using the VMFINFO Panels,” on page 199](#).

To illustrate the power and flexibility of the VMFSIM QUERY function, the following sections show a number of sample queries. For the command syntax, see [“VMFSIM QUERY” on page 567](#).

---

### Querying the System-Level Software Inventory after Receive Processing

This section shows examples of how to use the VMFSIM command to access the data available after products are received on the system by VMSES/E. You can also use the VMFINFO panel interface to perform these tasks.

### Determining the Status of Products Received

The VM SYSRECS table contains a list of all products that are received on the system. To query the status of all installed products, enter:

```
VMFSIM QUERY VM SYSRECS TDATA :PPF :STAT
```

The data returned by this query is:

```
VMFSIP2480I RESULTS FOR
          TDATA :PPF :STAT
:PPF ESA MYCOMP
  :STAT RECEIVED.06/28/22.09:16:01.MAINT.240-9901
:PPF ESA MYCOMP2
  :STAT RECEIVED.06/28/22.10:50:34.MAINT.240-9901
```

### Finding the Description of Products Received

The VM SYDESCT table contains a descriptive name for each product that has been received on the system. To find the description for all products received on the system, enter:

```
VMFSIM QUERY VM SYDESCT TDATA :PPF
```

The data returned by this query is:

```
VMFSIP2480I RESULTS FOR
          TDATA :PPF
:PPF ESA MYCOMP
  :PRODID 1VMVMC23%MYCOMP
  :DESC MYCOMP component for z/VM
:PPF ESA MYCOMP2
  :PRODID 1VMVMS10%MYCOMP2
  :DESC MYCOMP2 component for z/VM
```

## Querying the System-Level Software Inventory after Apply Processing

This section shows examples of how to use the VMFSIM command to access the data that is available after products have been applied to the system by VMSES/E.

### Querying the Status of Products Applied

The VM SYSAPPS table contains the status of all the products that are applied on the system. To determine the apply status of all the products installed on the system, enter:

```
VMFSIM QUERY VM SYSAPPS * TDATA :PPF :STAT
```

The data returned by this query is:

```
VMFSIP2480I RESULTS FOR
          TDATA :PPF :STAT
:PPF ESA MYCOMP
  :STAT APPLIED.06/28/22.09:16:03.MAINT
:PPF ESA MYCOMP2
  :STAT APPLIED.06/28/22.10:50:36.MAINT
```

You can also use the VMFINFO panel interface to perform this query.

## Querying the System-Level Software Inventory after Build Processing

This section shows examples of how to use the VMFSIM command to access the data that is available after products have been built by VMSES/E.



## Querying the Status of Products Built

The VM SYSBLDS table contains the status of all the products that are built on the system. To determine the build status of all the products installed on the system, enter:

```
VMFSIM QUERY VM SYSBLDS * TDATA :PPF :STAT
```

The data returned by this query is:

```
VMFSIP2480I RESULTS FOR
      TDATA :PPF :STAT
:PPF ESA MYCOMP
      :STAT BUILT.06/28/22.11:36:29.MAINT
:PPF ESA MYCOMP2
      :STAT BUILT.06/28/22.12:02:29.MAINT
```

You can also use the VMFINFO panel interface to perform this task.

## Performing Additional Queries on the System-Level Software Inventory

This section shows additional queries you can perform on the system-level software inventory tables.

### What Do I Have Installed that is Related to a Specific Product?

To determine what you have installed that is related to a specific product, for example RSCS, enter:

```
VMFSIM QUERY VM SYSDESC TDATA :DESC PVM :PPF :PRODID
```

The data returned by the query is:

```
:PPF 5684100E PVMINS
      :PRODID 5684100E%PVMINS
      :DESC Installing PVM 2.1.1
:PPF 5684100E PVMUCENG
      :PRODID 5684100E%PVMUCENG
      :DESC Upper Case English help
:PPF 5684100E PVMSRC
      :PRODID 5684100E%PVMSRC
      :DESC Installing PVM 2.1.1 Optional source
:PPF 5684100E PVMISFS
      :PRODID 5684100E%PVMISFS
      :DESC Installing PVM 2.1.1 using SFS directories
:PPF 5684100E PVMUSFS
      :PRODID 5684100E%PVMUSFS
      :DESC Servicing PVM 2.1.1 Upper Case English help using SFS directories
:PPF 5684100E PVMSSFS
      :PRODID 5684100E%PVMSSFS
      :DESC Installing PVM 2.1.1 Optional source using SFS directories
```

### What Options are Coded in the Product Parameter File (PPF)?

To find out which options are coded in the product parameter file for the MYCOMP component, enter:

```
VMFSIM QUERY ESA PPF TDATA :COMPNAME MYCOMP :CNTRLOP
```

The data returned by the query is:

```
VMFSIP2480I Results for
      TDATA :COMPNAME MYCOMP :CNTRLOP
:COMPNAME MYCOMP
      :CNTRLOP
      :BCOMPNAME MYCOMP
      :PRODDDESC MYCOMP for z/VM
      :RECID 1VMVMC23
      :APPID 1VMVMC23
      :BLDID 1VMVMC23
      :AXLIST AGWVM
      :EXCLIST
      :LOG YES
```

```
:RECVALL NO
:VERSION z/VM v.r.m
:SETUP NO
:CNTRL AGWVM
:SLVI B/AW
:NLS AMENG
:RETAIN
:CKAUX YES
:CKSDI NO
:CKVV NO
:CKGEN
:UPDTID AUXVM
:PTFPFX UM
:APARPFX VM
:USEREXIT
:ECNTRLOP
```

You can use the VMFINFO PPF Fileid - Help Panel to find this same information. For more information, see [“PPF Fileid - Help Panel”](#) on page 202.

## Querying the Service-Level Software Inventory after Receive Processing

This section shows examples of how to use the VMFSIM command to access the data that is available after PTFs are received for a product by VMSES/E. You can also use the VMFINFO panel interface to perform these tasks.

### Querying the Status of PTFs Received

The *prodid* SRVRECS table contains a list of all PTFs received for a product. To query the status of all PTFs received for a product, enter:

```
VMFSIM QUERY tblfn SRVRECS * TDATA :PTF
```

The data returned by this query is:

```
VMFSIP2480I RESULTS FOR
          TDATA :PTF
:PTF UM18082
          :STAT RECEIVED.06/28/22.14:54:22.MAINT
:PTF UM18109
          :STAT RECEIVED.06/28/22.14:54:22.MAINT
```

### Determining the Status of APARs Received

To determine the receive status of the APARs, you must first query the *prodid* SRVREQT table to obtain a list of PTFs that contain the APARs in question. Then, use the results of that query to obtain the status of the PTF. Enter the following commands:

```
VMFSIM QUERY tblfn SRVREQT * TDATA :APARNUM (FILE SRVAPARS
VMFSIM QUERY tblfn SRVRECS * FILE SRVAPARS
```

The data returned by these queries is:

```
VMFSIP2480I RESULTS FOR
          TDATA :PTF UM18082 :APARNUM VM47103
:PTF UM18082
          :STAT RECEIVED.06/28/22.14:54:22.MAINT
VMFSIP2480I RESULTS FOR
          TDATA :PTF UM18109 :APARNUM VM47104
:PTF UM18109
          :STAT RECEIVED.06/28/22.14:54:22.MAINT
VMFSIP2480I RESULTS FOR
          TDATA :PTF UM18135 :APARNUM VM47105
:PTF UM18135
          :STAT RECEIVED.06/28/22.14:54:22.MAINT
```

The first query of the APAR from the *prodid* SRVREQT table captures the PTF numbers associated with the APAR and saves the results in the SRVAPARS SIMDATA file. (SRVAPARS is the file name specified in the first query.) The second query uses the SRVAPARS SIMDATA file as the input data and returns the status

of the PTFs from the *prodid* SRVRECS table. The result of these two queries is the status of the associated PTF for APARs VM47103, VM47104, and VM47105.

## Querying the Description of APARs Received

To obtain the description of an APAR, enter the following command against the *prodid* SRVDESCT table.

```
VMFSIM QUERY tblfn SRVDESCT * TDATA :APARNUM
```

The data returned by this query is:

```
VMFSIP2480I RESULTS FOR
      TDATA :APARNUM
:APARNUM VM47103
      :ABSTRACT FIX LOAD PROBLEM WITH DMSABC
:APARNUM VM47104
      :ABSTRACT SENDFILE EXITS WITH RC=28
:APARNUM VM47105
      :ABSTRACT MESSAGE xyz ISSUED WHEN FILE NOT FOUND
```

## Determining APARs Contained in PTFs

To determine the APARs contained in all PTFs currently received, enter the following command against the *prodid* SRVREQT table.

```
VMFSIM QUERY tblfn SRVREQT * TDATA :APARNUM
```

The data returned by this query is:

```
VMFSIP2480I RESULTS FOR
      TDATA :APARNUM
:PTF UM18082
      :APARNUM VM47103
:PTF UM18109
      :APARNUM VM47104
:PTF UM18135
      :APARNUM VM47105
```

## Determining the Receive Status of All PTFs Containing an APAR

To determine the receive status of all PTFs containing a specific APAR, you must first query the *prodid* SRVREQT table to obtain a list of PTFs that contain the APAR in question. Then, use the results of that query to obtain the status of the PTF. Enter these VMFSIM QUERY commands:

```
VMFSIM QUERY tblfn SRVREQT * TDATA :APARNUM VM29997 (FILE PTFRECS
VMFSIM QUERY tblfn SRVRECS * FILE PTFRECS
```

The results of these queries are:

```
VMFSIP2480I RESULTS FOR
      TDATA :PTF UM18082 :APARNUM VM47103
:PTF UM18082
      :STAT RECEIVED.06/28/22.14:54:22.MAINT
```

The first query of the APAR from the *prodid* SRVREQT table captures the PTF numbers associated with the APAR and saves the results in the PTFRECS SIMDATA file. (PTFRECS is the file name specified in the first query.) The second query uses the PTFRECS SIMDATA file as the input data and returns the status of the PTFs from the *prodid* SRVRECS table. The result of these two queries is the status of all PTFs that are contained in the APARs specified.

## Querying the Service-Level Software Inventory after Apply Processing

This section shows examples of how to use the VMFSIM command to access data that is available after PTFs have been applied for a product by VMFAPPLY. You can also use the VMFINFO panel interface to perform some of these tasks.

## Querying the Status of PTFs Applied

The *prodid* SRVAPPS table consists of the list of PTFs with their appropriate statuses: APPLIED, SUPED, or REMOVED. To get a list of all PTFs in the *prodid* SRVAPPS table with a status of APPLIED, enter:

```
VMFSIM QUERY tblfn SRVAPPS * TDATA :STAT APPLIED
```

The data returned by this query is:

```
VMFSIP2480I RESULTS FOR
          TDATA :STAT APPLIED
:PTF UM18082
  :STAT APPLIED.06/28/22.14:59:04.MAINT
:PTF UM18109
  :STAT APPLIED.06/28/22.14:59:04.MAINT
```

## Querying the Status of APARs Applied

To determine the apply status of an APAR, you must first query the *prodid* SRVREQT table to obtain a list of PTFs that contain the APAR in question. Then, use the results of that query to obtain the status of the PTF. Enter the following VMFSIM QUERY commands:

```
VMFSIM QUERY tblfn SRVREQT * TDATA :APARNUM VM30514 (FILE SRVAPARS
VMFSIM QUERY tblfn SRVAPPS * FILE SRVAPARS
```

The data returned by these queries is:

```
VMFSIP2480I RESULTS FOR
          TDATA :PTF UM18082 :APARNUM VM30514
:PTF UM18082
  :STAT APPLIED.06/28/22.14:59:04.MAINT
```

The first query of the APAR from the *prodid* SRVREQT table captures the PTF numbers associated with the APAR and saves the results in the SRVAPARS SIMDATA file. (SRVAPARS is the file name specified in the first query.) The second query uses the SRVAPARS SIMDATA file as the input data and returns the status of the PTFs from the SRVAPPS table. The result of these two queries is the status of the APARs applied.

## Determining All Parts Serviced by a Specific PTF

The version vector table consists of the list of PTFs, APARs, and, optionally, updates that are applied to the part of a product at a specific level. To determine all parts affected by a PTF, for example UM18135, enter:

```
VMFSIM QUERY tblfn VVTVM * TDATA :PTF UM18135
```

You can enter the VMFSIM QUERY this way, because the :PTF tag in the version vector table contains the PTF number and the APAR number.

The data returned by this query is:

```
VMFSIP2480I RESULTS FOR
          TDATA :PTF UM18135
:PART HCPXLF TXT
  :PTF UM18135.VM47105.S47105HP
:PART HCPXLE TXT
  :PTF UM18135.VM47105.S47105HP
:PART HCPVDI TXT
  :PTF UM18135.VM47105.S47105HP
:PART HCPREP TXT
  :PTF UM18135.VM47105.S47105HP
:PART HCPPTU TXT
  :PTF UM18135.VM47105.S47105HP
:PART HCPMPS TXT
  :PTF UM18135.VM47105.S47105HP
```

## Determining All Parts Serviced by a Specific APAR

The version vector table consists of the list of PTFs, APARs, and, optionally, updates that are applied to the part of a component at a specific level. To determine all parts affected by an APAR, for example VM47103, enter:

```
VMFSIM QUERY tblfn VVTVM * TDATA :PTF VM47103
```

The data returned by this query is:

```
VMFSIP2480I RESULTS FOR
      TDATA :PTF VM47103
:PART HCPTDD TXT
      :PTF UM18082.VM47103.S47103HP
:PART HCPRDA TXT
      :PTF UM18109.VM47104.S47104HP UM18082.VM47103.S47103HP
:PART HCPPTF TXT
      :PTF UM18082.VM47103.S47103HP
:PART HCPPTF TXT
      :PTF UM18082.VM47103.S47103HP
```

## Determining All Service Applied to a Specific Part

The version vector table consists of the list of PTFs, APARs, and, optionally, updates that are applied to every part of the product that has been serviced. To determine all service applied to a specific part, for example HCPRDA, enter:

```
VMFSIM QUERY tblfn VVTVM * TDATA :PART HCPRDA
```

The data returned by this query is:

```
VMFSIP2480I RESULTS FOR
      TDATA :PART HCPRDA
:PART HCPRDA TXT
      :MOD
      :PTF UM18109.VM47104.S47104HP UM18082.VM47103.S47103HP
```

## Querying the Service-Level Software Inventory after Build Processing

This section shows examples of how to use the VMFSIM command to access data that is available after the VMFBLD command has been run. You can also use the VMFINFO panel interface to perform some of these tasks.

### Querying the Status of an Object

The service-level build status table (*appid* SRVBLDS) has a list of build lists and objects that have been serviced or built for the product. To determine the build status of a specific object (for example, ddr), issue the following VMFSIM QUERY command against the *appid* SRVBLDS table:

```
VMFSIM QUERY tblfn SRVBLDS * TDATA :OBJECT DDR
```

The data returned by this query is:

```
VMFSIP2480I RESULTS FOR
      TDATA :OBJECT DDR.MODULE
:BLDLIST HCPMLoad
      :OBJECT DDR.MODULE
      :STAT BUILT.06/28/22.13:09:59.MAINT
```

### Determining Which Objects to Build

The service-level build status table has a list of objects that have been serviced for the component. To determine the specific objects that have been serviced and need to be built, issue the following VMFSIM QUERY command against the *appid* SRVBlds table searching for objects with a status of SERVICED.

```
VMFSIM QUERY tblfn SRVBlds * TDATA :STAT SERVICED
```

The data returned by this query is:

```
VMFSIP2480I RESULTS FOR
      TDATA :STAT SERVICED
:BLDLIST HCPMLoad
  :OBJECT DIRECTXA.MODULE
  :STAT SERVICED.06/28/22.13:09:59.MAINT
:BLDLIST HCPMLoad
  :OBJECT CPEREPXA.MODULE
  :STAT SERVICED.06/28/22.13:09:59.MAINT
:BLDLIST HCPMLoad
  :OBJECT MONWRITE.MODULE
  :STAT SERVICED.06/28/22.13:09:59.MAINT
:BLDLIST HCPMLoad
  :OBJECT OVERRIDE.MODULE
  :STAT SERVICED.06/28/22.13:09:59.MAINT
:BLDLIST HCPMLoad
  :OBJECT HCPCCU.MODULE
  :STAT SERVICED.06/28/22.13:09:59.MAINT
```

### Determining Objects Requiring Manual Build Processing

The service-level build status table has a list of objects that have been serviced for the component. If VMFBLD cannot locate a part that has been serviced, it updates the *appid* SRVBlds table entry for that part with a status of MANUAL.

To determine the parts requiring manual processing after service has been applied, issue the following VMFSIM QUERY command against the *appid* SRVBlds table searching for objects with a status of MANUAL.

```
VMFSIM QUERY tblfn SRVBlds * TDATA :STAT MANUAL :PARTID
```

The data returned by this query is:

```
VMFSIP2480I RESULTS FOR
      TDATA :STAT MANUAL :PARTID
:BLDLIST UNKNOWN
  :OBJECT BLDLIST
  :STAT MANUAL.06/28/22.12:27:53.MAINT
  :PARTID HCPABC TXT
```

In this example, the :BLDLIST field contains the keyword UNKNOWN. This indicates the parts identified on the :PARTID tag were not found in any build lists identified in the PPF file. The :PARTID field contains a list of the file names and file types of the parts that require manual build processing.

## Performing Additional Queries on the Service-Level Software Inventory

This section shows examples of additional types of queries you can perform on the service-level software inventory tables.

### Which PTFs Have a Direct Prerequisite of a Specific PTF?

To determine which PTFs have a direct prerequisite of a specific PTF, for example UM15059, enter:

```
VMFSIM QUERY 1VMVMC23 SRVREQ TDATA :PTF :PREREQ UM15059
```

The data returned by the query is:

```
:PTF UM15066
      :PREREQ UM15059
:PTF UM15067
      :PREREQ UM15059
:PTF UM15068
      :PREREQ UM15059
```

The same query can be run using the VMFINFO panel shown in [Figure 118 on page 206](#). See “PTF Dependencies/Superseding Query Output” on [page 209](#) for an example of the information provided by the VMFINFO panel query.

## What Are the Direct Requisites for a Specific PTF?

To determine the direct requisites for a specific PTF, for example UM15066, enter:

```
VMFSIM QUERY 1VMVMC23 SRVREQT TDATA :PTF UM15066
```

The data returned by the query is:

```
:PTF UM15066
      :APARNUM VM42215
      :PREREQ UM15059
      :HARDREQ VM42300
```

The same query can be run using the VMFINFO panel shown in [Figure 118 on page 206](#). See “PTF Requisites/Supersedes Query Output” on [page 208](#) for an example of the information provided by the VMFINFO panel query.

## Which PTFs Are Applied?

To determine which PTFs are applied, enter:

```
VMFSIM QUERY 1VMVMC23 SRVAPPS TDATA :PTF
```

The data returned by the query is:

```
:PTF UM15042
      :STAT APPLIED.06/28/22.14:10:32.MAINT
:PTF UM15043
      :STAT APPLIED.06/28/22.14:10:32.MAINT
:PTF UM15044
      :STAT APPLIED.06/28/22.14:10:32.MAINT
:PTF UM15045
      :STAT APPLIED.06/28/22.14:10:32.MAINT
:PTF UM15046
      :STAT APPLIED.06/28/22.14:10:32.MAINT
:PTF UM15047
      :STAT APPLIED.06/28/22.14:10:32.MAINT
```

The same query can be run using the VMFINFO panel shown in [Figure 118 on page 206](#).

## Is a Specific APAR Applied?

To determine if a specific APAR is applied, for example VM41612, enter:

```
VMFSIM QUERY 1VMVMC23 SRVREQT TDATA :APARNUM VM41612 ( FILE TTVMF
SIM QUERY 1VMVMC23 SRVAPPS FILE TT
```

The data returned by the second query is:

```
:PTF UM14894
      :STAT APPLIED.06/28/22.14:10:32.MAINT
```

The same query can be run using the VMFINFO panel shown in [Figure 118 on page 206](#). See “PTF Status Query Output” on [page 207](#) for an example of the information provided by the VMFINFO panel query.

### What Is the Service History for a Part?

To determine the service history for a part, for example HCPPAG, enter:

```
VMFSIM QUERY 1VMVMC23 VVTVM TDATA :PART HCPPAG
```

The data returned by the query is:

```
:PART HCPPAG TXT
:MOD
:PTF UM14894.VM41612.H41612HP UM14580.VM41042.H41042HP
UM11765.VM39201.H39201HP UM09344.VM38072.H38072HP
UM09135.VM37393.H37393HP UM08863.VM35968.H35968HP
UM08012.VM36349.H36349HP UM90043.VM34943.H34943HP
:
```

The same query can be run using the VMFINFO panel shown in [Figure 125 on page 211](#). See “[Part Service History Query Output](#)” on [page 216](#) for an example of the information provided by the VMFINFO panel query.

### What Parts are Affected by a Specific PTF?

To determine what parts are affected by a specific PTF, for example UM14894, enter:

```
VMFSIM QUERY 1VMVMC23 VVTVM TDATA :PTF UM14894
```

The data returned by the query is:

```
:PART HCPPAG TXT
:PTF UM14894.VM41612.H41612HP UM14580.VM41042.H41042HP
UM11765.VM39201.H39201HP UM09344.VM38072.H38072HP
UM09135.VM37393.H37393HP UM08863.VM35968.H35968HP
UM08012.VM36349.H36349HP UM07833.VM36082.H36082HP
:
:PART HCPPAH TXT
:PTF UM14894.VM41612.H41612HP UM14580.VM41042.H41042HP
UM11765.VM39201.H39201HP UM09344.VM38072.H38072HP
UM09135.VM37393.H37393HP UM08863.VM35968.H35968HP
UM08012.VM36349.H36349HP UM90043.VM34943.H34943HP
:
```

The same query can be run using the VMFINFO panel shown in [Figure 118 on page 206](#). See “[PTF Serviceable Parts Query Output](#)” on [page 210](#) for an example of the information provided by the VMFINFO panel query.

## Updating the Software Inventory

The VMFSIM MODIFY function updates the Software Inventory. To illustrate the power and flexibility of the VMFSIM MODIFY function, sample commands for the following tasks are shown:

- Adding a product to the system-level Software Inventory
- Updating the SRVBLDS table after manual build processing

### Adding A Product to the System-Level Software Inventory

If you install a product that is not supported by VMSES/E, you can update the system-level Software Inventories to identify the product. Follow this procedure to add the product to the system-level Software Inventory. You will enter VMFSIM MODIFY four times.

- Step 1: Update the SYSRECS table.



The SYSRECS table contains the status of all the products or components that are received on the system. New products and status are added to this table by using the VMFSIM MODIFY function. In this example, we use the *prodid* 5684100E.

```
VMFSIM MODIFY VM SYSRECS * TDATA :PPF 5684100E NONE :PRODID 5684100E :STAT RECEIVED
```

The VM SYSRECS table will be updated to reflect that product 5684100E has been received on the system.

- Step 2: Update the SYSDESCT table.

The SYSDESCT table contains a short description of all the products or components that are received on the system. The new product description is added to this table by using the MODIFY function of VMFSIM.

```
VMFSIM MODIFY VM SYSDESCT * TDATA :PPF 5684100E NONE :PRODID 5684100E :DESC NEW PRODUCT
```

The VM SYSDESCT table will be updated to reflect the description of the product that has been installed.

- Step 3: Update the SYSREQT table.

The SYSREQT table contains the relationships between the products and components that are received on the system. The new products are added to this table using the VMFSIM MODIFY function.

```
VMFSIM MODIFY VM SYSREQT * TDATA :PRODID 5684100E
```

- Step 4: Update the SYSAPPS table.

The SYSAPPS table contains the status of all the products or components that are applied on the system. The new product's apply status is added to this table by using the MODIFY function of VMFSIM.

```
VMFSIM MODIFY VM SYSAPPS * TDATA :PPF 5684100E NONE :PRODID 5684100E :STAT APPLIED
```

The VM SYSAPPS table will be updated to reflect that product 5684100E has been applied on the system.

- Step 5: Update the SYSBLDS table.

The SYSBLDS table contains the status of all the products or components that are built on the system. The new product's build status is added to this table by using the VMFSIM MODIFY function.

```
VMFSIM MODIFY VM SYSBLDS * TDATA :PPF 5684100E NONE :PRODID 5684100E :STAT BUILT
```

The VM SYSBLDS table will be updated to reflect that product 5684100E has been built on the system.

## Updating the SRVBLDS Table after Manual Build Processing

The VMFSIM QUERY function retrieves data from the SRVBLDS file, and the VMFSIM MODIFY function uses this data to update the service-level build status table.

To update the status of manually processed parts, enter the following VMFSIM commands:

```
VMFSIM QUERY tblfn SRVBLDS TDATA :STAT MANUAL (FILE UPSTAT
VMFSIM MODIFY tblfn SRVBLDS * FILE UPSTAT TDATA :STAT BUILT :PARTID (REPLACE
VMFSIM QUERY tblfn SRVBLDS TDATA :BLDLIST UNKNOWN :STAT BUILT
```

The data returned to the second query is:

```
VMFSPC2480I Results for TDATA :STAT BUILT
:BLDLIST UNKNOWN
:OBJECT BLDLIST
:STAT BUILT.02/08/22.12:11:14.MAINT
```

## Comparing the VVT and AUX File Structure

The first query returns all entries in the SRVBLDS table that have a status of MANUAL and saves the results in a file UPSTAT SIMDATA. This file is then used as input to the modify function.

The VMFSIM MODIFY function is then entered with the file input UPSTAT SIMDATA and a TDATA statement that contains a status of BUILT. The modify function processes the input specified in the UPSTAT SIMDATA file and adds a status of BUILT to each entry in the file. The net result is that all entries in the SRVBLDS table that had a status of MANUAL now have a status of BUILT. If you look at the data returned to the second query, you see the modified entry in the SRVBLDS table.

For more information on this VMFSIM command, see [“VMFSIM MODIFY”](#) on page 562.

## Comparing the Version Vector Table to the AUX File Structure

The VMFSIM CHKLVL function compares the control/AUX file structure to the version vector table. It is used to validate:

- All service applied to a part is contained in the version vector table and AUX files for that part.
- Any local modification contained in AUX files are also identified in the corresponding version vector table.
- Any local version vector table entries are also identified in local AUX files for the part.

To illustrate the VMFSIM CHKLVL function, sample commands for the following tasks are shown:

- Checking the AUX files and version vector table match for the HCP RDA module.
- Using the LOGMOD option to update the local version vector table.

## Checking that the AUX Files and Version Vector Table Match

To use the VMFSIM CHKLVL function to check that the AUX files and version vector table match for DMSABC, enter:

```
VMFSIM CHKLVL ESA MYCOMP TDATA :PART DMSABC
```

This command checks the levels of the AUX files for DMSABC against the version vector table entries. In the output returned, the "X" shows the first mismatch detected.

```
=====DMSABC TXT=====1VMVMC23
VVTVM1 E.....|DMSABC AUXVM1 E
      PTF-----|
UM00004.VM40000.H40000DS...|H40000DS PTF UM00004 * Fix to problem 4
UM00003.VM30000.H30000DS...|H30000DS PTF UM00003 * Fix to problem 3
UM00002.VM20000.H20000DS...|X|HXXXXXDS PTF UM00002 * Fix to problem 2
UM00001.VM10000.H10000DS...|H10000DS PTF UM00001 * Fix to problem 1
=====
```

## Adding Local AUX File Entries to the Service-Level Software Inventories

To use VMFSIM CHKLVL to update the local version vector table, enter:

```
VMFSIM CHKLVL ppfname compname TDATA :PART DMSNGP TXT (LOGMOD
```

The LOGMOD option automatically logs information in the version vector table based on information in the AUX files. VMFSIM CHKLVL compares the AUX files for all AUX file levels above the level specified on the :UPDTID tag in the product parameter file to the corresponding version vector tables. If a mismatch is detected, the information from the AUX file is used to replace the existing version vector table entry. The first token in the AUX file is the update file type, the second token is ignored (however, LCL is recommended), and the third token is *modid*.

To see the results of the VMFSIM CHKLVL command, enter:

```
VMFSIM QUERY tblfn VVTLCL E TDATA :PART DMSNGP TXT
```

The query returns the following:

```
VMFSPC2480I Results for TDATA :PART DMSNGP TXT
:PART DMSNGP TXT
:MOD LCL2222.UPDTMOD2
:PTF
```

The VMFSIM CHKLVL function added the local modification LCL2222 to the local version vector table for the part DMSNGP TXT. This local modification is identified in the version vector table on the :MOD field. The data on this field has the format *lclmodid.updtft*.

For more information on this VMFSIM command, see [“VMFSIM CHKLVL” on page 531](#).

## Identifying the Latest Version of a Part

The VMFSIM GETLVL function uses a control file, the version vector tables, and, optionally, AUX files to find the current level of a part. To illustrate the VMFSIM GETLVL function, a sample command for the following task is shown:

- Determining the current service level of a part

### Determining the Current Service Level of a Part

To determine the current service level of a part, for example HCPRDA, enter the VMFSIM GETLVL command and specify the file name of the usable form product parameter file (*ppfname*) and the component name (*compname*) for the product:

```
VMFSIM GETLVL ESA MYCOMP TDATA :PART HCPRDA
```

The query returns the following data:

```
HCPRDA TXT18109
```

The TXT18109 token is the file type of the part at the highest service level found in the version vector table.

For more information on this VMFSIM command, see [“VMFSIM GETLVL” on page 543](#).

## Comparing Two Software Inventory Tables

The VMFSIM COMPTBL function compares two Software Inventory tables. To illustrate the VMFSIM COMPTBL function, sample commands for the following tasks are shown:

- Building an APPLY list of all PTFs received and not applied
- Creating an APPLY list from two SRVAPPS tables

### Building an APPLY List of All PTFs Received and Not Applied

The SRVRECS table contains a list of PTFs that have been received, and the SRVAPPS table contains a list of all PTFs already applied. When you compare these two tables, you get a file (NEW \$APPLIST) that contains an apply list of all PTFs received and not applied.

You can use this function when you want to create your own apply list containing only the new service you have received and not use the apply list shipped on the service tape, which may contain PTFs that you already have applied.

For example, when you enter:

```
VMFSIM COMPTBL 1VMVMC23 SRVRECS * 1VMVMC23 SRVAPPS * TDATA :PTF (APPLIST NEW
```

The file, NEW \$APPLIST, looks like this:

```
* APPLIST FROM VMFSIM COMPTBL 1VMVMC23 SRVRECS K 1VMVMC23 SRVAPPS A
UM18537
UM18538
UM18583
UM18598
```

The VMFSIM COMPTBL function compares the PTF entries in the SRVRECS table to the PTF entries in the SRVAPPS table. Any PTF that is found in the SRVRECS table and not in the SRVAPPS table is added to the NEW \$APPLIST file. The NEW \$APPLIST file can now be used as input to the VMFAPPLY command to install all new PTFs just received.

## Creating an APPLY List from Two SRVAPPS Tables

The SRVAPPS table on the first APPLY disk identified in the :MDA section of the PPF contains the most current service level of all PTFs applied. The SRVAPPS table on subsequent APPLY disks identified in the :MDA section of the PPF identifies previous service levels of the product or component. You can use this type of comparison to:

- Validate that all service that was previously applied is applied in the current service level.
- Create an apply list that contains all PTFs applied to one copy of a product or component that are not applied to a second copy of a product or component.
- Create an apply list that contains all reach-ahead service that is currently applied to a product or component and not contained on a service refresh tape, for example, a Product Service Upgrade (PSU).

To perform this type of comparison, enter:

```
VMFSIM COMPTBL 1VMVMC23 SRVAPPS K 1VMVMC23 SRVAPPS J TDATA :PTF (APPLIST NEW
```

The file, NEW \$APPLIST, looks like this:

```
* APPLIST FROM VMFSIM COMPTBL 1VMVMC23 SRVAPPS K 1VMVMC23 SRVAPPS J
UM18537
UM18538
UM18583
UM18598
```

The VMFSIM COMPTBL function compares the PTF entries in the first SRVAPPS table to the PTF entries in the second SRVAPPS table. Any PTF that is found in the first SRVAPPS table and not in the second SRVAPPS table is added to the NEW \$APPLIST file. The NEW \$APPLIST file can now be used as input to the VMFAPPLY command to install all PTFs that were applied in the first table and not in the second. This would make both levels contain the same service level.

For more information on this VMFSIM command, see [“VMFSIM COMPTBL” on page 538](#).

## Building Apply and Exclude Lists after Receive Processing

The VMFSIM SRVREQ function uses the service-level requisite table and the service-level apply status table and returns the requisites for the specified PTFs. The VMFSIM SRVDEP function uses the service-level requisite table and the service-level apply status table and returns the dependent PTFs. To illustrate the VMFSIM SRVREQ function and VMFSIM SRVDEP functions, sample commands for the following tasks are shown:

- Building an apply list containing requisites of a PTF
- Building an exclude list containing dependents of a PTF

## Building an APPLY List Containing All Requisites of a PTF

To build an APPLY list containing all requisites required to install a specific PTF, use the VMFSIM SRVREQ function. This apply list is useful in determining how much service will be applied before actually applying it.

Use this function to create your own apply list containing only the requisites required to apply a specific PTF you have received, as shown in the following example:

```
vmfsim srvreq tblfn srvreqt a tblfn srvapps a tdata :ptf um15010 (applist ptfreqs
```

The data returned is:

```
==== * * * TOP OF FILE * * *
==== UM15010
==== UM18538
==== UM18583
==== UM18598
==== * * * END OF FILE * * *
```

The VMFSIM SRVREQ function adds an entry to the PTFREQS \$APPLIST for every PTF that is not applied and is a requisite of the PTF specified on the command. The PTFREQS \$APPLIST can then be used as input to the VMFAPPLY command to install a specific PTF and all its requisites.

For more information on this VMFSIM command, see [“VMFSIM SRVREQ” on page 579](#).

## Building an EXCLUDE List Containing All Dependents of a PTF

Use the VMFSIM SRVDEP function to create an EXCLUDE list containing all PTFs that are dependent on a specific PTF. This exclude list is useful when you are identifying PTFs that depend on a specific PTF you do not want to apply.

```
vmfsim srvdep tblfn srvreqt * tblfn srvapps * tdata :ptf um15010 (applist ptfdeps
rename ptfdeps $applist a = $exclist a
```

The data returned is:

```
==== * * * TOP OF FILE * * *
==== UM15010
==== UM18531
==== UM18582
==== UM18593
==== * * * END OF FILE * * *
```

The VMFSIM SRVDEP function adds an entry to the PTFDEPS \$APPLIST for every PTF that is applied and is dependent on the PTF specified on the command. The PTFDEPS \$EXCLIST can then be used as input to the VMFAPPLY command to exclude a specific PTF and all its dependents.

**Note:** It is not necessary to have all dependent PTFs in the exclude list. VMFAPPLY will automatically exclude all PTFs dependent on a PTF specified in the exclude list.

For more information on this VMFSIM command, see [“VMFSIM SRVDEP” on page 573](#).

## Listing the Requisites for PTFs

The VMFSIM SRVREQ function uses the service-level requisite table and service-level apply status table to return the requisites for the PTFs specified. Use this function to determine if you have applied or received all the PTFs that are required by a PTF you want to install. To illustrate the VMFSIM SRVREQ function, a sample command for the following task is shown:

- Determining the prerequisites for a PTF

### Determining the Prerequisites for a PTF

You can use the VMFSIM SRVREQ command to determine the prerequisites for a PTF. For example, to determine the prerequisites for PTF UM18135, enter:

```
VMFSIM SRVREQ tblfn SRVREQT * = SRVAPPS * TDATA :PTF UM18135
```

The response is:

```
VMFSIP2480I RESULTS FOR
                TDATA :PTF UM18135
:PTF UM18135
:PREREQ UM18109
:HARDREQ VM47104
:SUBREQ UM18082
:SUBIF *NONE*
:SUBHARDREQ *NONE*
```

UM18109 must be installed before UM18135. UM18135 has no corequisites or if-requisites and does not supersede another PTF. APAR VM47104 is required for UM18135 to function. A requisite PTF (which must be UM18109, the only requisite) requires UM18082, but UM18109 has no if-requisites or required APARS.

For more information about this command, see [“VMFSIM SRVREQ” on page 579](#).

## Listing the Requisites for a Product

The VMFSIM SYSREQ function uses the system-level requisite table and system-level apply status table to return the requisites for the *prodid* specified. (*prodid* is the 7- or 8-alphanumeric identifier assigned to the product.) Use this function to determine if you have all the products or components installed on your system that are required by the product you want to install.

To illustrate the VMFSIM SYSREQ function, a sample command for the following task is shown:

- Determining the prerequisites for a product

## Determining the Prerequisites for a Product

To determine the prerequisites for a specific product, for example 1VMVMC23, enter:

```
VMFSIM SYSREQ VM SYSREQT * = SYSAPPS * TDATA :PRODID 1VMVMC23%MYCOMP
```

The response is:

```
VMFSIP2480I RESULTS FOR
                TDATA :PRODID 1VMVMC23%MYCOMP
:PRODID 1VMVMC23%MYCOMP 1
:PREREQ 1VMVMP11 2
:REQ *NONE*
:DREQ 1VMVMC22 3
:SUP *NONE*
:IFREQ *NONE*
:NPRE *NONE*
:SUBREQ 1VMVME10 4
:SUBIF *NONE*
:PTFREQS *NONE*
```

As you can see in the above example, product 1VMVMP11 (**2**) is a pre-requisite and must be installed before you install product 1VMVMC23 (**1**). There are no other pre-requisites.

1VMVMC23 is a dependent of 1VMVMC22 (**3**). 1VMVMC23 does not supersede any other product. It has no if-requisites and does not preclude installing any other products.

A prerequisite product, 1VMVMP11 (**2**), has 1VMVME10 (**4**) as a requisite, but it has no if-requisites. No PTFs are required.

For more information on this VMFSIM command, see [“VMFSIM SYSREQ” on page 591](#).

## Listing the Dependent PTFs for Another PTF

The VMFSIM SRVDEP function uses the system-level requisite table and system-level apply status table to return the dependents for the PTFs specified. Use this function to determine which PTFs are dependent on a specific PTF that you want to remove from the product or component. To illustrate the VMFSIM SRVDEP function, a sample command for the following task is shown:

- Determining the PTFs dependent on a PTF

## Determining the PTFs Dependent on a PTF

To determine the PTFs dependent on a PTF, for example UM15010, enter:

```
VMFSIM SRVDEP tblfn SRVREQT * = SRVAPPS * TDATA :PTF UM15010
```

The response is:

```
VMFSPC2480I Results for TDATA :PTF UM15010
:PTF UM15010
:DEPS UM15020
:SUPBY *NONE*
:OUTREQS *NONE*
```

One PTF, UM15020, depends on UM15010. No PTFs supersede UM15010, and there are no PTFs in other products or components that depend on UM15010.

For more information on this VMFSIM command, see [“VMFSIM SRVDEP” on page 573](#).

## Listing the Dependent Products for Another Product

The VMFSIM SYSDEP function uses the service-level requisite table and service-level apply status table to return the dependents for the *prodid* specified. Use this function to determine which products or components are dependent on a specific product or component that you want to remove from the system. To illustrate the VMFSIM SYSDEP function, a sample command for the following task is shown:

- Determining the dependents of a product

## Determining the Dependents of a Product

To determine the dependents of a product, for example 1VMVMC23, enter:

```
VMFSIM SYSDEP VM SYSREQT * = SYSAPPS * TDATA :PRODID 1VMVMC23%MYCOMP
```

The response is:

```
VMFSIP2480I RESULTS FOR
          TDATA :PRODID 1VMVMC23%MYCOMP
:PRODID 1VMVMC23%MYCOMP
:DEPS 1VMVMS20 1VMVME10 1VMVMG10 1
:DREQDEPS 1VMVMC22 2
:SUPBY *NONE*
:OUTREQS *NONE*
```

As you can see in this example, products or components 1VMVMS20, 1VMVME10, and 1VMVMG10 are all dependent on 1VMVMC23 (**1**). In other words, if you remove , 1VMVMC23, these products may not function. 1VMVMC22 is a dependent feature of 1VMVMC23 (**2**). 1VMVMC23 is not superseded by any other product, and no PTFs in other products or components depend on it.

For more information on this VMFSIM command, see [“VMFSIM SYSDEP” on page 585](#)

## Adding Local Modifications to the Software Inventory

The VMFSIM LOGMOD function supports your local modifications structure.

If you are adding a local modification for an ASSEMBLE, NLS, or \$Source file, you can use the LOGMOD option on the VMFASM, VMFHASM, VMFHLASM, VMFNLS, or VMFEXUPD command to update the local version vector table. For other source updated parts, use the VMFSIM CHKLVL command with the LOGMOD option to update the local version vector table. You can also update \$SELECT files and specify the LOCALMOD disk for output using the \$SELECT and OUTMODE options. For replacement only parts, you must use the VMFSIM LOGMOD command.

To illustrate the VMFSIM LOGMOD function, sample commands for the following tasks are shown:

- Add local source update modifications to service-level Software Inventories

- Adding local replacement files to service-level Software Inventories

### Adding Local Source Update Modifications to Service-Level Software Inventories

The local version vector table (VVTLCL) is the default table that is used to track local modifications to a product or component. Each time an entry is added to a local AUX file (AUXLCL), a corresponding entry should be made in the local version vector table to ensure that all local modifications are identified by VMFAPPLY when applying service and built into the objects when VMFBLD is run. To add a local modification to the version vector table, enter the following VMFSIM LOGMOD command:

```
VMFSIM LOGMOD tblfn VVTLCL E TDATA :PART HCPRIO TXT :MOD LCL2222.UPDTMOD2
```

Entering this VMFSIM QUERY:

```
VMFSIM QUERY tblfn VVTLCL E TDATA :PART HCPRIO TXT
```

shows the results of the VMFSIM LOGMOD command:

```
VMFSPC2480I Results for TDATA :PART HCPRIO TXT
:PART HCPRIO TXT
:MOD LCL2222.UPDTMOD2
:PTF
```

The VMFSIM LOGMOD function added the local modification LCL2222 to the VVTLCL table for the part HCPRIO TXT. This local modification is identified on the :MOD field in the version vector table. The data in this field has the format *modid.updtft*.

LCL2222 is the local tracking number for this modification. It is recommended that the first two characters of the local tracking number be LC. The modification identifier is 5 characters, and it is recommended that the first character be an L. The 5 characters of the modification identifier identify the file type of the text deck (HCPRIO in this example) that is used by VMFBLD. The file type that is processed is a composite of the 3 character file type abbreviation and the 5 characters from the local tracking number (TXTL2222 in this example). This local tracking number must also be identified in the AUX file that contains the source update for this part (the AUX file would have a file type of AUXLCL in this example).

#### Important Note

We recommend you start the local tracking number with LCL to ensure it does not interfere with service delivered by IBM. If you use characters other than LCL, make sure they are unique for your product.

UPDTMOD2 is the file type of the source update file containing the changes to HCPRIO for this modification.

### Adding Local Replacement Files to Service-Level Software Inventories

The local version vector table (VVTLCL) is the default table that is used to track local modifications to a product or component. If a part of the product that is serviced by part replacement is modified, the modification must be identified in the local version vector table (VVTLCL). This ensures that all local modifications are identified by VMFAPPLY when applying service and built into objects when VMFBLD is run. If the modification is not added to the VVTLCL file, VMFAPPLY is unable to identify the change; and VMFBLD does not pick up the modification when building objects.

To add a local modification (without source updates) to the version vector table, enter the following VMFSIM LOGMOD command:

```
VMFSIM LOGMOD tblfn VVTLCL E TDATA :PART DMSABS EXC :MOD LCL2222
```

Entering this VMFSIM QUERY:



```
VMFSIM QUERY tblfn VVTLCL E TDATA :PART DMSABC EXC
```

shows the results of the VMFSIM LOGMOD command:

```
VMFSPC2480I Results for TDATA :PART DMSABC EXC
:PART DMSABC EXC
:MOD LCL2222
:PTF
```

The VMFSIM LOGMOD function added the local modification LCL2222 to the VVTLCL table for the part DMSABC EXC. This local modification is identified on the :MOD field in the version vector table. The data in this field has the format *modid*.

LCL2222 is the local tracking number for this modification. It is recommended that the first two characters of the local tracking number be LC. The modification identifier is 5 characters, and it is recommended that the first character be an L. The 5 characters of the modification identifier are used to identify the file type of the replacement part (DMSABC in this example) that would be used by VMFBLD. The file type that is processed is a composite of the 3 character file type abbreviation and the 5 characters from the local tracking number (EXCL2222 in this example). Because there are no source updates for this part, no AUX file needs to be built containing this local tracking number. Also, the version vector table entry would not identify any source update files.

### Important Note

We recommend you start the local tracking number with LCL to ensure it does not interfere with service delivered by IBM. If you use characters other than LCL, make sure they are unique for your product.

For more information on this VMFSIM command, see [“VMFSIM LOGMOD” on page 556](#).

## Initializing and Recovering the Software Inventory Tables

The VMFSIM INIT function initializes or recovers the product or PTF information in the software inventory . To illustrate the VMFSIM INIT function, sample commands for the following tasks are shown:

- Recovering the system-level Software Inventory tables
- Recovering the service-level Software Inventory tables

These examples are more detailed than the other examples in this chapter. The commands you need to enter before and after VMFSIM INIT are shown in order to provide an example of using VMFSIM INIT to do a realistic task.

### Recovering the System-Level Software Inventories

If you need to recover the system-level inventory, follow this procedure:

1. Recover the SYSREQT and SYSDESCT tables.

The SYSREQT and SYSDESCT tables are created from the *fn* PRODPART file using the VMFSIM INIT function. To recover these system-level software inventory tables, you need to create a list of all PRODPART files on the system by entering this command:

```
listfile * prodpart * (exec
```

Then you need to recreate the tables by entering:

```
vmfsim init vm * file cms exec
```

The SYSREQT table is updated to reflect all requisite data contained in the PRODPART files processed.

The SYSDESCT table is updated to reflect all default PPFs and the descriptions associated with them in the PRODPART files processed.

2. Recover the system-level software inventory status tables.

## Initializing and Recovering Software Inventory Tables

You can create the system-level software inventory status tables by using the list of products contained in the SYSDESCT table as a base. You can modify it to add any additional products or delete products. To recover the status tables, use the following procedure:

- a. Create a file that lists all the products contained in the SYSDESCT table by using this VMFSIM QUERY command:

```
vmfsim query vm sysdesct tdata :ppf (file proddata
```

The PRODDATA SIMDATA file contains a list of all products and their descriptions. This data is used as input to the VMFSIM MODIFY function.

- b. Edit the PRODDATA SIMDATA file. Add any additional products that have been installed on the system, and delete any product that has not been installed. Add any additional \$PPF file overrides that you may have created for the products you have installed.
- c. Update the VM SYSRECS, VM SYSAPPS, and VM SYSBLDS tables using the PRODDATA SIMDATA file created above. Enter:

```
vmfsim modify vm sysrecs d file proddata tdata :stat received
vmfsim modify vm sysapps d file proddata tdata :stat applied
vmfsim modify vm sysblds d file proddata tdata :stat built
```

The PRODDATA SIMDATA file contains a list of all products and their descriptions. The TDATA statement adds, to each product identified in the VMFSIM PRODDATA file:

- A status of RECEIVED in the VM SYSRECS table
- A status of APPLIED in the VM SYSAPPS table
- A status of BUILT in the VM SYSBLDS table

## Recovering Service-Level Software Inventories

If you need to recover the service-level Software Inventories, follow this procedure:

1. Recover the SRVREQT, SRVDESCT, and SRVRECS tables.

The SRVREQT, SRVRECS and SRVDESCT service-level Software Inventory tables are created from the *fn* \$PTFPART files using the VMFSIM INIT function. To do this, you need to create a list of all \$PTFPART files for the product. Enter:

```
vmfsetup esa compname
listfile * $ptfpart * (exec
vmfsim init prodid * fm file cms exec
```

The *prodid* SRVREQT table is updated to reflect all requisite data contained in the \$PTFPART files processed.

The *prodid* SRVDESCT table is updated to reflect all APARs, and the descriptions associated with them, in the \$PTFPART files processed.

The *prodid* SRVRECS table is updated with a status of RECEIVED for each \$PTFPART file processed.

2. Recover the SRVAPPS and version vector tables.

The SRVAPPS and VVTVM service-level Software Inventory tables are created by the VMFAPPLY command. In the following example, you can see an example of the commands you need to enter to rebuild the SRVAPPS and version vector tables.

```
vmfsetup esa compname
listfile ctlfile $ap* * (exec args
erase reapp $applist a
cms cmdcall copyfile reapp $applist a (append
vmfapply ppf ppfname compname (applist reapp
```

The *prodid* SRVAPPS table is updated with a status of APPLIED or SUPED for each PTF processed.

The *prodid* VVTVM table is updated with the levels of all parts processed.

AUXVM files are created for all parts in the *prodid* VVTVM table that were updated and require AUX files.

3. Recover the SRVBLDS table and rebuild objects. Run VMFBLD to regenerate or update the SRVBLDS table and rebuild all objects serviced.

For more information on this VMFSIM command, see [“VMFSIM INIT” on page 550](#).



## Chapter 17. Using the VMFINFO Panels

You can use the VMFINFO command to query the software inventory tables. VMFINFO provides easy-to-use panels and a variety of predefined queries for both product information and service information.

Let's take a few minutes to see how to use the VMFINFO panels to query the Software Inventory.

### Where Does the Information Come From?

When you use the VMFINFO command, queries are issued against the Software Inventory tables, just as if you were entering a VMFSIM query.

### Understanding the VMFINFO Panel Information

Figure 109 on page 199 is the main VMFINFO panel. This panel lists all the product queries, as well as the topics for the service queries.

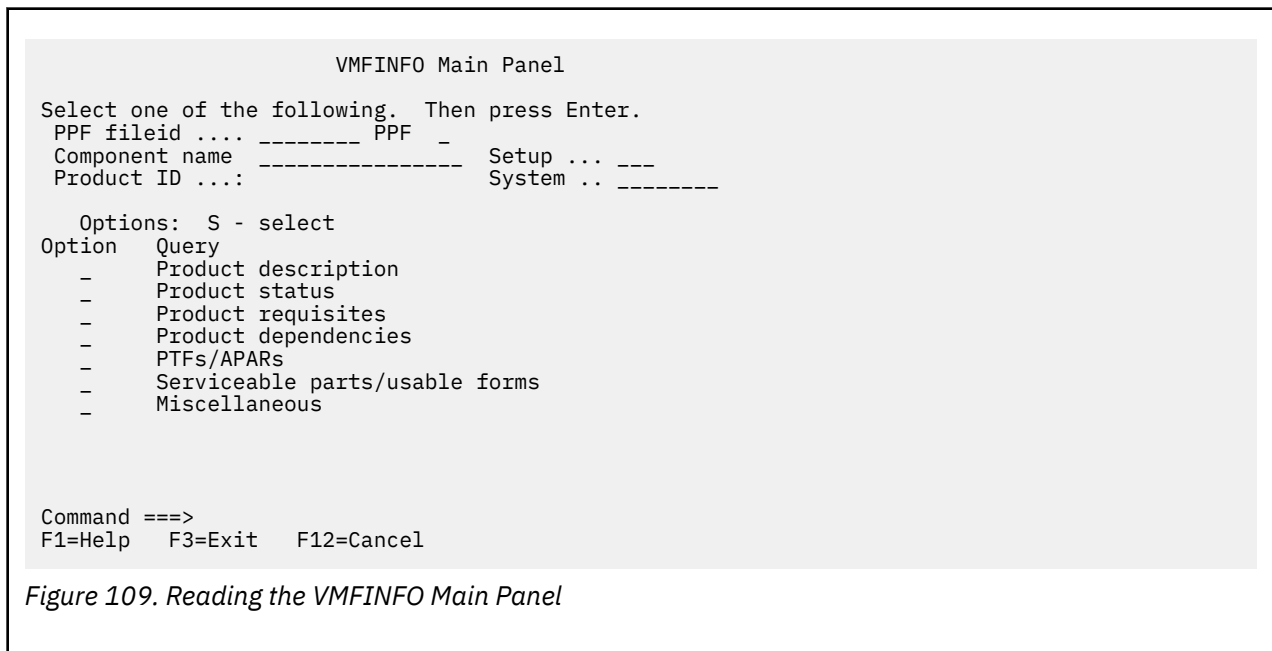


Figure 109. Reading the VMFINFO Main Panel

#### PPF fileid

is the file identifier of the usable form product parameter file that is to be used for the queries. You can enter the file name and the file mode. The file type is always PPF. You can change the PPF fileid on any VMFINFO query panel.

When you enter a valid product parameter file and component name and you enter an asterisk (\*) for file mode, VMFINFO uses the first file that is found with a matching file identifier on the first accessed disk.

#### Component name

is the name of the component that is to be used for the queries. You can change the component name on any VMFINFO query panel.

#### Product ID

is the 8-character product identifier. The product identifier is supplied for you from the selected product parameter file. You cannot enter it manually.

### Setup

indicates whether a new minidisk/directory access order should be established each time you change the name of the product parameter file or the component. The access order is determined by the information in the product parameter file.

You can enter YES or NO, and you can change the value from any VMFINFO query panel.

**Note:** If you change either the product parameter file name or the component name and Setup has been specified as No, you receive a message and, possibly, inaccurate output.

### System

is the file name of the system-level software inventory tables for your system. The system-level software inventory tables contain information on products. You can change this value from any VMFINFO query panel. The default file name for the system-level software inventory is VM.

**Note:** An asterisk (\*) can only be entered in the file mode field. For a list of valid selections for other input fields, use the F1=Help key.

## General Information

Keep the following information in mind when you use the VMFINFO panels:

- When you see **More: +** in the upper right corner of a panel, there is more information available. You may scroll through it with the F8=Forward key.
- When you see **More: -** in the upper right corner of a panel, there is more information available. You may scroll through it with the F7=Backward key.
- If you are using F1=Help on the VMFINFO panel:
  - And the cursor is on an input field, F1=Help gives you help for that input field.
  - And the cursor is on an area other than an input field, F1=Help provides general help on the panel.
- The F12=Cancel key returns you to the previous panel.
- Any valid CP or CMS command can be entered on the command line.

## Using the Function Keys

Figure 110 on page 200 shows the Function key assignments. Table 12 on page 200 explains the function each key provides.

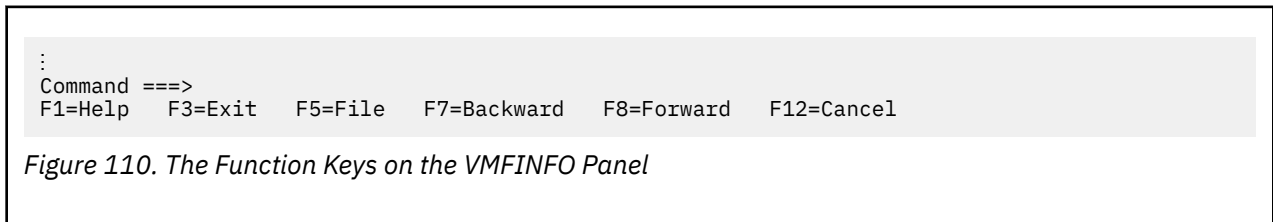


Table 12. Function Key Assignments for the VMFINFO Panels

Function Key	Function Provided
F1=Help	Provides a list of valid selections when the cursor is on an input field. When the cursor is on a query topic on a query panel F1=Help gives you a description of the query. When the cursor is on the command line or in an area on the panel that has not been previously described, F1=Help provides general help on the VMFINFO panel. When you are on a query output panel F1=Help gives you general help on the query that has been selected.
F3=Exit	Exits from the VMFINFO session.

Table 12. Function Key Assignments for the VMFINFO Panels (continued)

Function Key	Function Provided
F5=File	Allows you to save the output from the VMFINFO sessions in a file with the file identifier VMFINFO <i>mmddhhtt</i> . Within a single VMFINFO session each consecutive query is appended to the bottom of this file.
F7=Backward	Moves backward through the information. More: - in the upper right corner of the panel means there is more information, and you may scroll backward through it.
F8=Forward	Moves forward through the information. More: + in the upper right corner of the panel means there is more information to display, and you may scroll forward through it.
F12=Cancel	Exits from the VMFINFO panels, one at a time.

## Getting Online Help for the VMFINFO Panels

To get HELP online, enter `help vmses vmfinfop` on the VMFINFO panel command line or the CMS command line.

## Getting Started

To perform VMFSIM queries using the VMFINFO panels, begin by entering either:

- `vmfinfo (setup`

When you enter this command, a setup of the component's minidisks or shared file directories will be performed. The minidisks or directories need to be available in order for VMFINFO to access the component's service software inventory files. VMFINFO displays a list of all product parameter files (PPFs) found on all accessed disks. You can view or select any PPF in the list. See [“PPF Fileid - Help Panel”](#) on page 202 for an example of this panel.

When you select a PPF, the Component Name - Help panel is displayed. This panel shows you a list of the components within the PPF. See [“Component Name - Help Panel”](#) on page 202 for an example of this panel.

- `vmfinfo ppfname compname (setup`

When you enter this command (with the product parameter file name, *ppfname*, and component name, *compname*), a setup of the component's minidisks or shared file directories will be performed. The minidisks or directories need to be available in order for VMFINFO to access the component's service software inventory files. VMFINFO displays the Main Panel. From the main panel, you can select product queries or select the topics for the service queries.

For the complete command syntax, see [“The Source Product Parameter File”](#) on page 13.

## Selection Panels

To use the VMFINFO queries, you need to enter the name of a product parameter file and a component name. You can get a list of product parameter file names and component names by entering a question mark (?) in the appropriate input field.

If you enter a question mark (?) or request help for the PPF fileid field on any panel, the PPF Fileid - Help panel is displayed. The PPF Fileid - Help panel lists all the product parameter files found on all accessed disks. You also receive the PPF Fileid - Help panel when you leave the PPF fileid field blank.

If you enter a question mark (?) or request help for the component name field after a PPF has been specified, the Component Name - Help panel is displayed. The Component Name - Help panel lists the component names found within the selected product parameter file. You also receive the Component Name - Help panel when you leave the component name field blank.

## PPF Fileid - Help Panel

The PPF Fileid - Help panel displays a list of all product parameter files found on all accessed disks. The list of PPFs will be alphabetized. [Figure 111 on page 202](#) shows an example of the PPF Fileid - Help panel.

```

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
S      ESA          PPF  D
-      1VMVMC23    PPF  D

Command ==>
F1=Help  F3=Exit  F12=Cancel

```

*Figure 111. PPF Fileid - Help Panel*

You can select or view a product parameter file from this panel. When you specify the view option, you are placed in an XEDIT session so you can check the settings in the product parameter file.

Once you select a PPF, the Component Name - Help panel is displayed.

## Component Name - Help Panel

The Component Name - Help panel displays a list of all components within the product parameter file. The following screen shows an example of the Component Name - Help panel.

```

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
S      MYCOMP   - VMSES for z/VM
-      MYCOMP2  - REXX Programming Language

Command ==>
F1=Help  F3=Exit  F12=Cancel

```

*Figure 112. Component Name - Help Panel*

On this panel, you can select one of the components for the specified product parameter file. Once you select a component name, you either:

- Go to the VMFINFO Main Panel to select the type of query you want to perform.
- Return to the panel that you were on when you changed either the value in the PPF fileid field or the component name.



## VMFINFO Main Panel

The following screen shows the VMFINFO Main Panel. It lists all of the product queries and topics for the service queries.

```

VMFINFO Main Panel

Select one of the following. Then press Enter.
PPF fileid ... ESA      PPF D
Component name MYCOMP   Setup ... YES
Product ID ...: 1VMVMC23      System .. VM

Options: S - select
Option  Query
-      Product description
-      Product status
-      Product requisites
-      Product dependencies
-      PTFs/APARs
-      Serviceable parts/usable forms
-      Miscellaneous

Command ==>
F1=Help  F3=Exit  F12=Cancel

```

*Figure 113. VMFINFO Main Panel*

Once you have either entered or selected the product parameter file and component names, specified either yes or no for the Setup option, and indicated the name of the system-level software inventory tables for your system, you can run a query on the software inventory tables.

## Product Queries

From the VMFINFO Main Panel, you can select the following queries for a product:

### Description

provides the descriptive name for the product, such as PVM 2.1.1.

### Status

provides the status of the product, such as received, applied, and built.

### Requisites

provides a list of products that are required to be installed before the product you are processing can be installed, built, and run.

### Dependencies

provides a list of products that depend on the product you are processing.

After you select a product query, a Query Output panel is displayed. You can save the information from the Query Output panel by pressing the F5=File key.

## Service Queries

From the VMFINFO Main Panel, you can also select queries on:

- PTFs and APARs
- Serviceable parts and usable forms
- Miscellaneous information

Additional panels are displayed based on the selections you make.

An example of each VMFINFO query panel, as well as an example of each type of query output panel, is shown in the following sections.

## Product Description Query

You can use the product description query to find the description of a product. To get the description for a product, select the product description query from the VMFINFO Main Panel.

Figure 114 on page 204 shows the output from a product description query. For this example, we entered ESA PPF D for the PPF fileid and MYCOMP for the component name.

```

                                Query Output - Product Description
PPF fileid ...: ESA           PPF D
Component name: MYCOMP           Setup ..: YES
Product ID ...: 1VMVMC23         System .: VM
-----
Product description: MYCOMP component for z/VM

Command ==>
F1=Help  F3=Exit  F5=File  F12=Cancel

```

*Figure 114. Product Description Query Output*

For information on the output provided, see [“The System-Level Description Table \(VM SYSDESC\)”](#) on page 684.

## Product Status Query

You can use the product status query to find the receive, apply, and build status for the product. To determine the status of a product, select the product status query from the VMFINFO Main Panel.

Figure 115 on page 204 shows the output from a product status query. In this example, we entered ESA PPF D for the PPF fileid and MYCOMP for the component name.

```

                                Query Output - Product Status
PPF fileid ...: ESA           PPF D
Component name: MYCOMP           Setup ..: YES
Product ID ...: 1VMVMC23         System .: VM
-----
Receive status: RECEIVED.06/18/22.09:16:01.MAINT
Apply status:   APPLIED.06/18/22.10:08:04.MAINT
Build status:   BUILT.06/19/22.09:10:06.MAINT

Command ==>
F1=Help  F3=Exit  F5=File  F12=Cancel

```

*Figure 115. Product Status Query Output*

For information on the output provided, see [“The System-Level Receive Status Table \(VM SYSRECS\)”](#) on page 688, [“The System-Level Apply Status Table \(VM SYSAPPS\)”](#) on page 690, and [“The System-Level Build Status Table \(VM SYSBLDS\)”](#) on page 692.

## Product Requisites Query

You can use the product requisites query to check the requisites for a product. To determine the requisites for a product, select the product requisites query from the VMFINFO Main Panel.

Figure 116 on page 205 shows the output from a product requisite query. In this example, we entered ESA PPF D for the PPF fileid and MYCOMP for the component name.

```

Query Output - Product Requisites

PPF fileid ...: ESA          PPF  D
Component name: MYCOMP      Setup ..: YES
Product ID ...: 1VMVMC23    System .: VM
-----
PRODIG:  1VMVMC23%MYCOMP
PREREQ:  1VMVMP11
REQ:     *NONE*
DREQ:    1VMVMC22
SUP:     *NONE*
IFREQ:   *NONE*
NPRE:    *NONE*
SUBREQ:  1VMVME10
SUBIF:   *NONE*
PTFREQS: *NONE*

Command ==>
F1=Help  F3=Exit  F5=File  F12=Cancel

```

*Figure 116. Product Requisite Query Output*

For information on the output provided, see [“The System-Level Requisite Table \(VM SYSREQT\)”](#) on page 686.

## Product Dependencies Query

You can use the product dependencies query to find the dependencies for a product. To determine the dependencies for a product, select the product dependencies query from the VMFINFO Main Panel.

Figure 117 on page 206 shows the output from a product dependencies query. In this example, we entered ESA PPF D for the PPF fileid and MYCOMP for the component name.

```

                                Query Output - Product Dependencies
PPF fileid ...: ESA          PPF D
Component name: MYCOMP          Setup ...: YES
Product ID ...: 1VMVMC23        System ..: VM
-----
PRODID: 1VMVMC23%MYCOMP
DEPS: 1VMVMS20 1VMVME10 1VMVMG10
DREQDEPS: 1VMVMC22
SUPBY: *NONE*
OUTREQS: *NONE*

Command ==>
F1=Help  F3=Exit  F5=File  F12=Cancel
    
```

Figure 117. Product Dependencies Query Output

For information on the output provided, see [“The System-Level Requisite Table \(VM SYSREQT\)”](#) on page 686.

## PTF/APAR Query Panel

You can use the PTF/APAR Query Panel to obtain information on the PTFs and APARs installed on your system.

When you select the PTF/APAR option from the VMFINFO Main Panel, you see the following panel:

```

                                PTF/APAR Queries
Enter a PTF or APAR number and type an option code.  Then press Enter.
PPF fileid .... ESA          PPF D
Component name MYCOMP          Setup ... YES
Product ID ...: 1VMVMC23        System .. VM
PTF number .... -----
APAR number ... -----

Options: S - select
Option  Query
-      Status of PTF
-      Requisites/supersedes of PTF
-      Dependencies/superseding of PTF
-      User memo of PTF
-      Serviceable parts included by PTF
-
-      Abstract of APAR(s)

Command ==>
F1=Help  F3=Exit  F12=Cancel
    
```

Figure 118. PTF/APAR Query Panel

You need to enter any combination of PTF or APAR numbers, select an option, and press enter.

You can get a list of valid PPF file identifiers, component names, PTF numbers, and APAR numbers by putting the cursor on the field and pressing the F1 key for help. For a description of each type of query listed on this panel, put the cursor on the query topic and press the F1 key. If the cursor is not on an input field or a query topic and you press F1=Help, you receive general help on the query panel.

Table 13 on page 207 shows an example of the possible combinations of inputs and the outputs received for each combination.

<i>Table 13. Example PTF/APAR Query Inputs and Results</i>			
<b>When You Enter PTF Number</b>	<b>When You Enter APAR Number</b>	<b>And You Select This Query</b>	<b>You Receive</b>
X	None	<ul style="list-style-type: none"> <li>• Status of PTF</li> <li>• Requisites/supersedes of PTF</li> <li>• Dependencies/superseding of PTF</li> <li>• User memo of PTF</li> <li>• Serviceable parts included by PTF</li> </ul>	Information for just PTF X
None	Y	<ul style="list-style-type: none"> <li>• Status of PTF</li> <li>• Requisites/supersedes of PTF</li> <li>• Dependencies/superseding of PTF</li> <li>• User memo of PTF</li> <li>• Serviceable parts included by PTF</li> </ul>	All PTFs with APAR number Y
X	Y	<ul style="list-style-type: none"> <li>• Status of PTF</li> <li>• Requisites/supersedes of PTF</li> <li>• Dependencies/superseding of PTF</li> <li>• User memo of PTF</li> <li>• Serviceable parts included by PTF</li> </ul>	Information for just PTF X
X	None	<ul style="list-style-type: none"> <li>• Abstract of APAR(s)</li> </ul>	All abstracts for all APARs in PTF X
None	Y	<ul style="list-style-type: none"> <li>• Abstract of APAR(s)</li> </ul>	Information for just APAR Y
X	Y	<ul style="list-style-type: none"> <li>• Abstract of APAR(s)</li> </ul>	The abstract for the PTF X and APAR Y combination

The following examples show the output for each type of query.

### **PTF Status Query Output**

Figure 119 on page 208 shows the output from a PTF status query. You can use the PTF status query to get the receive and apply status for a PTF.

In this example, we entered ESA PPF D for the PPF fileid, MYCOMP for the component name, and UM18082 for the PTF number.

```

Query Output - PTF Status

PPF fileid ...: ESA      PPF D
Component name: MYCOMP      Setup ...: YES
Product ID ...: 1VMVMC23    System ..: VM
-----
PTF: UM18082
-----
Receive status: RECEIVED.07/18/22.09:16:01.MAINT
Apply status:   APPLIED.07/18/22.10:08:04.MAINT

Command ==>
F1=Help  F3=Exit  F5=File  F12=Cancel

```

*Figure 119. PTF Status Query Output*

For information on the output provided, see [“The Service-Level Receive Status Table \(recid SRVRECS\)”](#) on page 715.

## PTF Requisites/Supersedes Query Output

Figure 120 on page 208 shows the output from a PTF requisites/supersedes query. You can use the PTF requisites/supersedes query to find the requisites for the PTF and any PTFs replaced by the PTF number you enter.

In this example, we entered ESA PPF D for the PPF fileid, MYCOMP for the component name, and UM18135 for the PTF number.

```

Query Output - PTF Requisites/Supersedes

PPF fileid ...: ESA      PPF D
Component name: MYCOMP      Setup ...: YES
Product ID ...: 1VMVMC23    System ..: VM
-----
PTF: UM18135
-----
PREREQ:      UM18109
HARDREQ:     VM47104
SUBREQ:      UM18082
SUBIF:       *NONE*
SUBHARDREQ:  *NONE*

Command ==>
F1=Help  F3=Exit  F5=File  F12=Cancel

```

*Figure 120. PTF Requisites/Supersedes Query Output*

For information on the output provided, see [“The Service-Level Requisite Table \(recid SRVREQT\)”](#) on page 713 and [“The Service-Level Apply Status Table \(appid SRVAPPS\)”](#) on page 716.

## PTF Dependencies/Superseding Query Output

Figure 121 on page 209 shows the output from a PTF dependencies/superseding query. You can use the PTF dependencies/superseding query to find the PTFs that depend on the PTF number you enter, as well as any PTFs that replace (supersede) that PTF number.

In this example, we entered ESA PPF D for the PPF fileid, MYCOMP for the component name, and UM15010 for the PTF number.

```

                                Query Output - PTF Dependencies/Superseding
PPF fileid ...: ESA           PPF D
Component name: MYCOMP           Setup ...: YES
Product ID ...: 1VMVMC23         System ..: VM
-----
PTF: UM15010
-----
DEPS:      UM15020
SUPBY:     *NONE*
OUTREQS:   *NONE*

Command ==>
F1=Help   F3=Exit   F5=File   F12=Cancel

```

*Figure 121. PTF Dependencies/Superseding Query Output*

For information on the output provided, see “The Service-Level Requisite Table (recid SRVREQT)” on page 713 and “The Service-Level Apply Status Table (appid SRVAPPS)” on page 716.

## PTF User Memo Query Output

Figure 122 on page 209 shows the output from a PTF user memo query. You can use the user memo query to get additional information for a PTF that is provided in the user memo.

In this example, we entered ESA PPF D for the PPF fileid, MYCOMP for the component name, and UM18082 for the PTF number.

```

                                Query Output - PTF User Memo
PPF fileid ...: ESA           PPF D
Component name: MYCOMP           Setup ...: YES
Product ID ...: 1VMVMC23         System ..: VM
-----
PTF: UM18082
-----
This field contains special instructions.

Command ==>
F1=Help   F3=Exit   F5=File   F12=Cancel

```

*Figure 122. PTF User Memo Query Output*

For information on the output provided, see [“The PTF Parts \(\\$PTFPART\) File”](#) on page 705.

### PTF Serviceable Parts Query Output

Figure 123 on page 210 shows the output from a PTF serviceable parts query. In this example, we entered ESA PPF D for the PPF fileid, MYCOMP for the component name, and UM18135 for the PTF number.

```
Query Output - PTF Serviceable Parts
PPF fileid ...: ESA      PPF D
Component name: MYCOMP      Setup ...: YES
Product ID ...: 1VMVMC23    System ..: VM
-----
PTF: UM18135
-----
VMFAPPLY EXC
VMFREC EXC
VMFBLD EXC
VMFSIM EXC

Command ==>
F1=Help  F3=Exit  F5=File  F12=Cancel
```

*Figure 123. PTF Serviceable Parts Query Output*

For information on the output provided, see [“The PTF Parts \(\\$PTFPART\) File”](#) on page 705.

### APAR Abstract Query Output

Figure 124 on page 210 shows the output from an APAR abstract query. In this example, we entered ESA PPF D for the PPF fileid, MYCOMP for the component name, and VM47103 for the APAR number.

```
Query Output - APAR Abstract(s)
PPF fileid ...: ESA      PPF D
Component name: MYCOMP      Setup ...: YES
Product ID ...: 1VMVMC23    System ..: VM
-----
APAR: VM47103
-----
This field contains the APAR abstract.

Command ==>
F1=Help  F3=Exit  F5=File  F12=Cancel
```

*Figure 124. APAR Abstract Query Output*

For information on the output provided, see [“The Service-Level Description Table \(recid SRVDESCT\)”](#) on page 712 and [“The PTF Parts \(\\$PTFPART\) File”](#) on page 705.

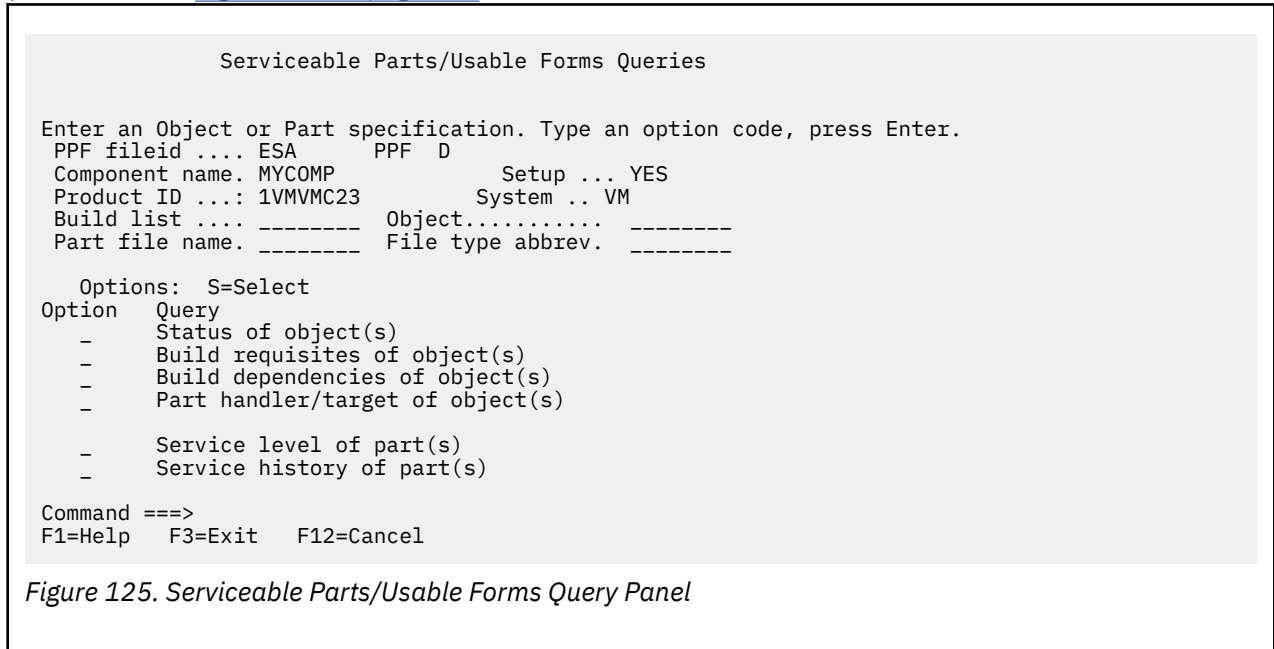


## Serviceable Parts/Usable Forms Query Panel

You can use the Serviceable Parts/Usable Forms Query Panel to obtain information on the:

- Status of objects
- Build requisites of objects
- Build dependencies of objects
- Part handler and target of objects
- Service level of parts
- Service history of parts

When you select the Serviceable Parts/Usable Forms option from the VMFINFO Main Panel, you see the panel shown in [Figure 125 on page 211](#).



*Figure 125. Serviceable Parts/Usable Forms Query Panel*

You need to enter the information requested, select an option code, and press enter.

[Table 14 on page 211](#) shows the different combinations of inputs and the type of information you receive when you select one of the following queries:

- Status of object(s)
- Build requisites of object(s)
- Build dependencies of object(s)
- Part handler/target of objects(s)

When You Enter				You Receive
Build List	Object	File Name	File Type Abbreviation	
●				Information for each object in the build list
●	●			Information for that object in that build list

*Table 14. Serviceable Parts/Usable Forms Query Inputs and Results for Objects (continued)*

When You Enter				You Receive
Build List	Object	File Name	File Type Abbreviation	
●	●	●		Information for the object with that file name in that particular build list
●	●	●	●	Information for only that object with the matching file name and file type abbreviation in that particular build list
	●			Information for all build lists containing that object
	●	●		Information for that object with that file name in all build lists
	●	●	●	Information for that object with that file name and file type abbreviation in all build lists
		●		Information for all objects with that file name in all build lists
		●	●	Information for all objects with that file name and file type abbreviation in all build lists
			●	A message. You must enter a build list, object, or part file name.

Table 15 on page 212 shows the different combinations of inputs and the type of information you receive when you select one of the following queries:

- Service level of part(s)
- Service history of part(s)

*Table 15. Serviceable Parts/Usable Forms Query Inputs and Results for Parts*

When You Enter				You Receive
Build List	Object	File Name	File Type Abbreviation	
●				Information for all parts in all objects in that build list
●	●			Information for all parts in that object in that particular build list
●	●	●		Information for the part with that file name in that object in that particular build list

When You Enter				You Receive
Build List	Object	File Name	File Type Abbreviation	
●	●	●	●	Information for only the part with that file name and file type abbreviation in that object in that particular build list
	●			Information for all parts in that object in all build lists
	●	●		Information for only the part with that file name in that object in all build lists
	●	●	●	Information for only the part with that file name and file type abbreviation in that object in all build lists
		●		Information for all parts with that file name in all combinations of the build list and object
		●	●	Information for all parts with that file name and file type abbreviation in all combinations of the build list and object
			●	A message. You must enter a build list, object, or part file name.

The following examples show the output for each type of query. For information on the output provided by these queries, see [“Build Lists”](#) on page 141.

### Object Status Query Output

Figure 126 on page 214 shows the output from an object status query. In this example, we entered ESA PPF D for the PPF fileid, MYCOMP for the component name, and VMFSBHLP for the build list.

```

Query Output - Object Status

PPF fileid ...: ESA      PPF D
Component name: MYCOMP          Setup ...: YES
Product ID ...: 1VMVMC23      System ..: VM
-----
OBJECT: VMFSBHLP.BLDLIST
-----
STATUS:      BUILT

OBJECT: VMFSBHLP.HELP.SEGMENT
-----
STATUS:      BUILT

Command ==>
F1=Help  F3=Exit  F5=File  F12=Cancel
    
```

*Figure 126. Object Status Query Output*

### Object Build Requisites Query Output

Figure 127 on page 214 shows the output from an object build requisites query. In this example, we entered ESA PPF D for the PPF fileid, MYCOMP2 for the component name, DMSSBVMT for the build list, and CMSVMLIB.SEGMENT for the object.

```

Query Output - Object Build Requisites

PPF fileid ...: ESA      PPF D
Component name: MYCOMP2          Setup ...: YES
Product ID ...: 1VMVMS10      System ..: VM
-----
OBJECT: DMSSBVMT. CMSVMLIB.SEGMENT
-----
BLDREQS:      DMSBLVMT.BLDLIST DMSBL493.VMMLIB.LSEG

Command ==>
F1=Help  F3=Exit  F5=File  F12=Cancel
    
```

*Figure 127. Object Build Requisites Query Output*

### Object Build Dependencies Query Output

Figure 128 on page 215 shows the output from an object build dependencies query. In this example, we entered ESA PPF D for the PPF fileid, MYCOMP2 for the component name, DMSBLVMT for the build list, and VMMLIB for the object.

```

Query Output - Object Build Dependencies

PPF fileid ...: ESA      PPF D
Component name: MYCOMP2      Setup ...: YES
Product ID ...: 1VMVMS10    System ..: VM
-----
OBJECT: DMSBLVMT.VMMTLIB
-----
BLDDEPS:      DMSBLVML.VMLIB CMSLOAD.BLDLIST
DMSSBVMT. CMSVMLIB.SEGMENT

Command ==>
F1=Help  F3=Exit  F5=File  F12=Cancel

```

*Figure 128. Object Build Dependencies Query Output*

## Object Part Handler/Target Query Output

Figure 129 on page 215 shows the output from an object part handler/target query. In this example, we entered ESA PPF D for the PPF fileid, MYCOMP2 for the component name, and DMSBLVMT for the build list.

```

Query Output - Object Part Handler/Target

PPF fileid ...: ESA      PPF D
Component name: MYCOMP2      Setup ...: YES
Product ID ...: 1VMVMS10    System ..: VM
-----
OBJECT: DMSBLVMT.*
-----
PART HANDLER: VMFBDCLB
TARGET:      BUILD6

Command ==>
F1=Help  F3=Exit  F5=File  F12=Cancel

```

*Figure 129. Object Part Handler/Target Query Output*

## Part Service Level Query Output

Figure 130 on page 216 shows the output from a part service level query. In this example, we entered ESA PPF D for the PPF fileid, MYCOMP3 for the component name, HCPBDUTL for the part file name, and EXC for the part file type.

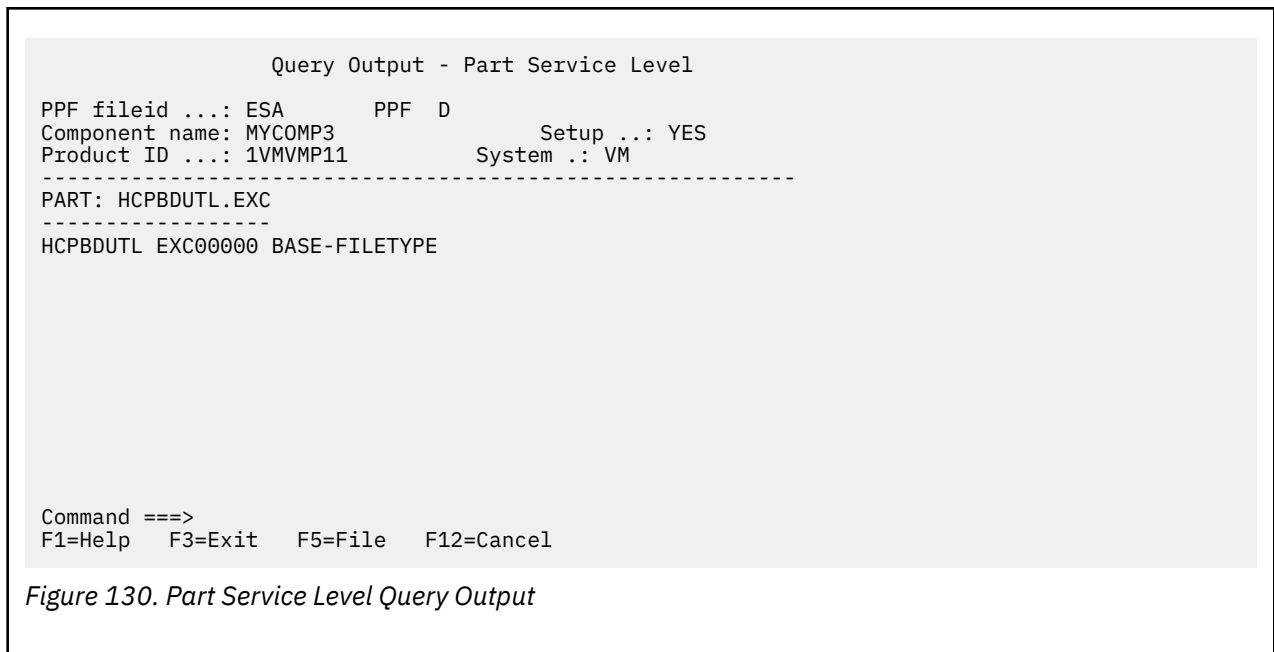


Figure 130. Part Service Level Query Output

### Part Service History Query Output

Figure 131 on page 216 shows the output from a part service history query. In this example, we entered ESA PPF D for the PPF fileid, MYCOMP3 for the component name, HCPVOT for the part file name, and TXT for the part file type.

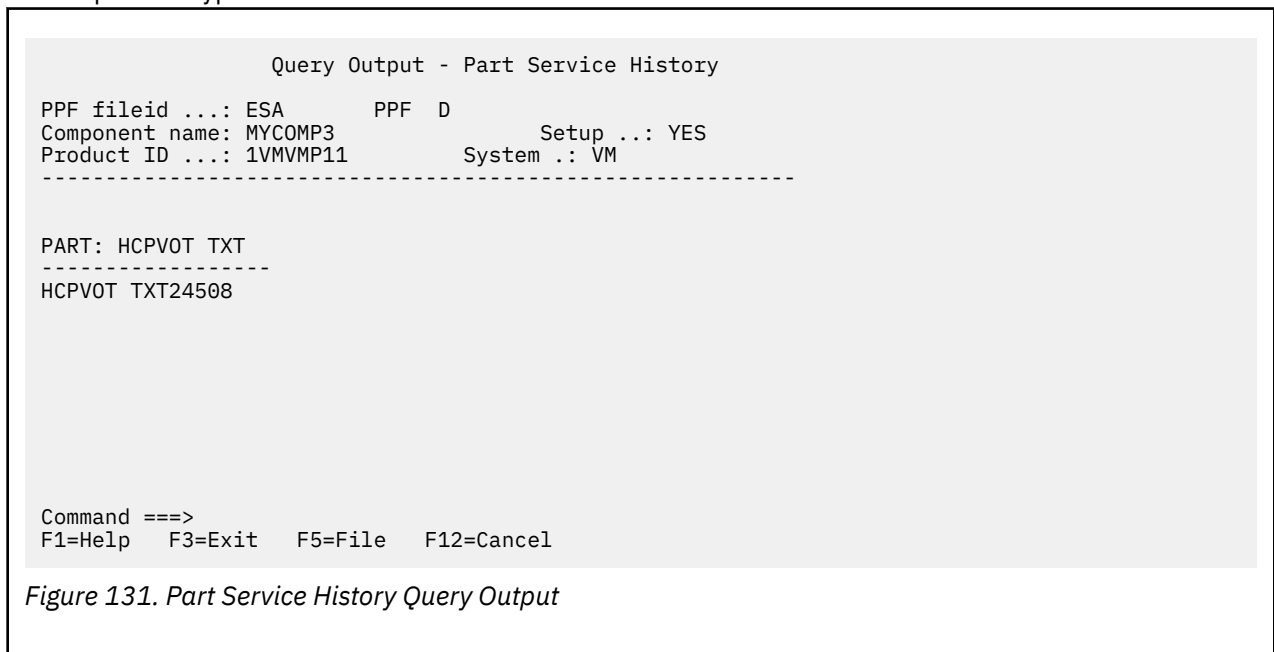


Figure 131. Part Service History Query Output

### Miscellaneous Queries Panel

Use the Miscellaneous Queries Panel to obtain information on the:

- Minidisks and Shared File System directories accessed
- The results of a comparison between tables
- Build requirements
- File type abbreviations.

When you select the Miscellaneous option from the VMFINFO Main Panel, you see the panel shown in Figure 132 on page 217.

```

Miscellaneous Queries

Type an option code. Then press Enter.
PPF fileid .... ESA      PPF D
Component name. MYCOMP      Setup ... YES
Product ID ...: 1VMVMC23    System .. VM

Options: S - Select
Option  Query
-      Minidisk/directory access
-      Compare table contents
-      Build requirements
-      File type abbreviations

Command ==>
F1=Help  F3=Exit  F12=Cancel

```

*Figure 132. Miscellaneous Queries Panel*

You need to select an option and press enter. A second panel may be displayed, depending on the option you select. Each option is discussed briefly in the following sections.

## Minidisk/Directory Access Query

When you select the minidisk/directory access query, you receive information on the minidisks and Shared File System directories that are accessed. The access order is defined in the usable form product parameter file.

Figure 133 on page 217 shows the output from a minidisk/directory access query.

```

Query Output - Minidisk/Directory Access

PPF fileid ...: ESA      PPF D
Component name: MYCOMP      Setup ...: YES
Product ID ...: 1VMVMC23    System .. VM
-----
Minidisk|Directory Assignments:
String  Mode  Stat  Vdev  Label (OwnerID Odev : Cyl/%Used)
      -or-      SFS Directory Name
LOCALMOD E    R/W   5C4   MNT5C4 (MAINT730 05C4 : 5/02)
LOCALSAM F    R/W   5C2   MNT5C2 (MAINT730 05C2 : 5/01)
APPLY    G    R/W   5A6   MNT5A6 (MAINT730 05A6 : 6/01)
DELTA    J    R/W   5D2   MNT5D2 (MAINT730 05D2 : 30/00)
BUILD8   K    R/W   5E6   MNT5E6 (MAINT730 05E6 : 9/81)
BUILD7   L    R/W   493   MNT493 (MAINT730 0493 : 250/53)
BUILD6   M    R/W   490   MNT490 (MAINT730 0490 : 207/41)

BUILD4   M    R/W   49D   MNT49D (MAINT730 049D : 146/65)
BASE3    P    R/W   5B4   MNT5B4 (MAINT730 05B4 : 70/86)
BASE2    Q    R/W   5B2   MNT5B2 (MAINT730 05B2 : 40/88)
----- A    R/W   191   MNT191 (MAINT730 0191 : 175/19)
----- B    R/W   DIR   VMPSFS:MAINT730.

Command ==>
F1=Help  F3=Exit  F5=File  F12=Cancel

```

*Figure 133. Minidisk/Directory Access Query Output*

## Compare Table Contents Panel

When you select the compare table contents option from the Miscellaneous Queries panel, the Compare Table Contents panel (Figure 134 on page 218) is displayed. You can use the Compare Table Contents panel to compare information in two Software Inventory tables.

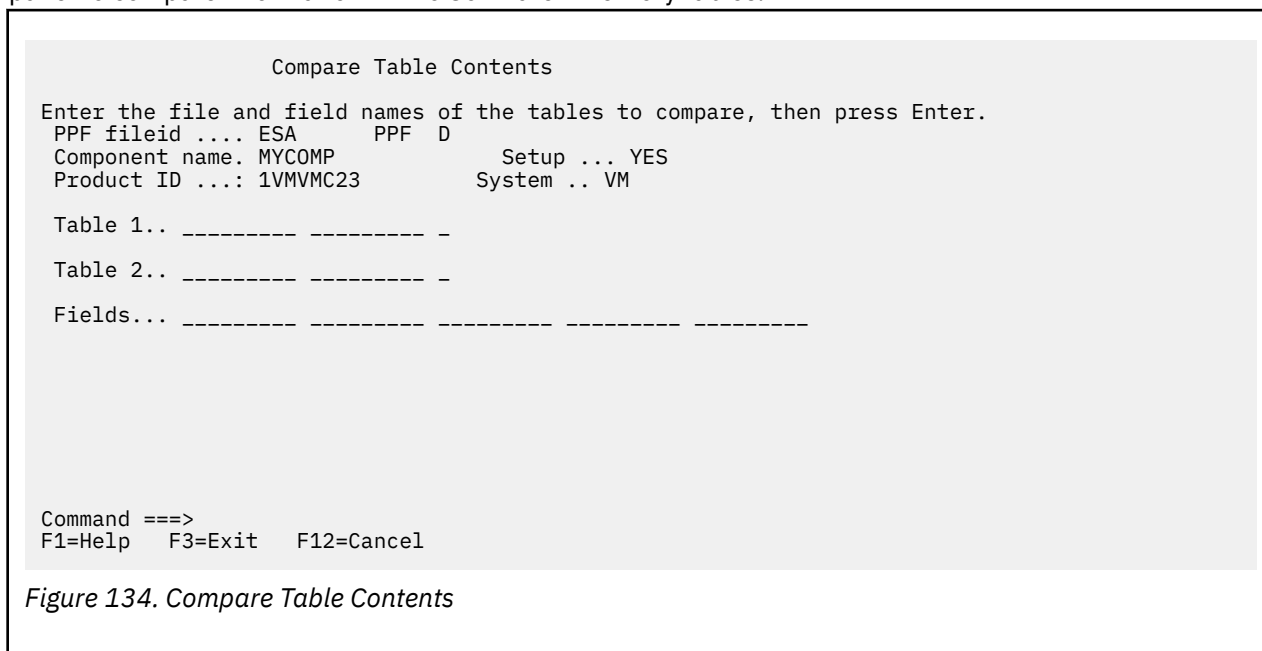


Figure 134. Compare Table Contents

You need to enter the file name, file type, and file mode for each table being used in the comparison. You also need to enter the names of the fields (tags) that you want to compare.

The field name **must** begin with a colon (:), and it can be up to 9 characters in length. For a list of valid tags, enter the file identifiers for the two tables that you want to compare, place the cursor on one of the input fields, and press F1=Help. Tags that are common to both tables are displayed.

## Compare Table Contents Query Output

Figure 135 on page 219 shows the output from a compare table contents query. In this example, we entered 1VMVMC23 SRVAPPS D for the first table identifier, 1VMVMC23 SRVAPPS E for the second table identifier, and :PTF as the field name to be used in the comparison.



```

Query Output - Compare Table Contents

PPF fileid ...: ESA      PPF D
Component name: MYCOMP      Setup ...: YES
Product ID ...: 1VMVMC23    System ..: VM
-----
===== 1VMVMC23 SRVAPPS  D2 =====
:PTF UV12345 :STAT APPLIED.01/03/22.11:11:11.JONES
----- 1VMVMC23 SRVAPPS  E2 -----
:PTF **NOT FOUND**
===== 1VMVMC23 SRVAPPS  D2 =====
:PTF UV00007 :STAT SUPED.02/04/21.12:12:12.MAINT
----- 1VMVMC23 SRVAPPS  E2 -----
:PTF UV00007 :STAT APPLIED.01/04/22.22:22:22.MAINT

Command ==>
F1=Help  F3=Exit  F5=File  F12=Cancel

```

Figure 135. Compare Table Contents Query Output

For information on the output provided, see the descriptions for the tables used for the query in [Chapter 22, “Software Inventory Syntax,”](#) on page 659.

## Build Requirements Query

When you select the build requirements option on the Miscellaneous Queries panel, you receive information on the build requirements for the component selected. [Figure 136 on page 219](#) shows the output from a build requirements query.

```

Query Output - Build Requirements

PPF fileid ...: ESA      PPF D
Component name: MYCOMP      Setup ...: YES
Product ID ...: 1VMVMC23    System ..: VM
-----
Build List  Object      Status                                     Partid
-----
VMFBLSSES  VMFSIM    SERVICED.02/05/22.14:14:14.MAINTurm
VMFBLSSES  VMFBLD    BUILDALL.01/08/22.06:09:43.MAINTurm.ERROR
VMFBLSYS   VMFHASM   SERVICED.02/05/22.14:19:16.MAINTurm

Command ==>
F1=Help  F3=Exit  F5=File  F12=Cancel

```

Figure 136. Build Requirements Query Output

For information on the output provided, see [“The Service-Level Build Status Table \(bldid SRVBLDS\)”](#) on page 717.

## File Type Abbreviations Panel

When you select the file type abbreviations option from the Miscellaneous Queries panel, the File Type Abbreviations panel ([Figure 137 on page 220](#)) is displayed.

You can use the File Type Abbreviations panel to find the real file type and base file type for a file type abbreviation.

**Note:** You can request information for only one file type at a time — abbreviation, real, or base.

```
File Type Abbreviations

Enter either the abbrev, real, or base file type. Then press Enter.
PPF fileid .... ESA      PPF  D
Component name. MYCOMP      Setup ... YES
Product ID ...: 1VMVMC23      System .. VM

Abbreviation .. ___  Real .. _____  Base .. _____

Command ==>
F1=Help  F3=Exit  F12=Cancel
```

*Figure 137. File Type Abbreviations Panel*

You need to enter the 3-character file type abbreviation, the real file type, or the base file type and press enter.

The following example shows the output for this type of query.

### File Type Abbreviations Query Output

Figure 138 on page 220 shows the output from a file type abbreviations query. In this example, the file type abbreviation EXC was entered; and the real and base file types were provided.

```
Query Output - File Type Abbreviations

PPF fileid ...: ESA      PPF  D
Component name: MYCOMP      Setup ...: YES
Product ID ...: 1VMVMC23      System ..: VM
-----
Abbrev: EXC
Real:   EXEC
Base:   EXC00000

Command ==>
F1=Help  F3=Exit  F5=File  F12=Cancel
```

*Figure 138. File Type Abbreviations Query Output*

For more information on the output provided, see [“The File Type Abbreviation Table \(VM SYSABRVT\)”](#) on page 702.

## Chapter 18. Changing the Software Inventory to an SFS Directory

When you use VMSES/E to install and service licensed programs, you need write access to the Software Inventory disk (51D). If multiple users install and maintain licensed products on your system, there may be a problem getting the necessary write access to the 51D disk.

If you find that there is contention for write access to the 51D disk, you can eliminate it by converting the Software Inventory from minidisk to the Shared File System (SFS). With SFS, multiple users can have write access at the same time. SFS also ensures the integrity of the data in any file that is updated. If you would like to use an SFS directory instead of the 51D disk for your Software Inventories, follow the procedures in this section. These procedures are examples only. The minidisk file modes and the SFS file pools in these examples may be different from those on your system.

### Create the SFS Directory

1. Log on to the MAINT $vr$ m user ID, where  $vr$ m is the version, release, and modification level of the z/VM system:

```
logon maint $vr$ m
```

MAINT $vr$ m will be the owner of the Software Inventory directory.

2. Create the Software Inventory directory:

```
create directory vm $psfs$ :maint $vr$ m.sidisk
```

### Initialize the VM $PSFS$ :MAINT $vr$ m.SIDISK Directory

**Note:** You must update the VMFINS DEFAULTS, VMSESE PROFILE, and SEGBLD PPF files immediately after performing this step. This will prevent files from being placed on the 51D minidisk instead of the VM $PSFS$ :MAINT $vr$ m.SIDISK directory.

1. Copy the 51D minidisk to the VM $PSFS$ :MAINT $vr$ m.SIDISK directory:

```
access 51d d
```

```
access vm $psfs$ :maint $vr$ m.sidisk e
```

```
copyfile * * d = = e (olddate
```

Use the COPYFILE command to initialize the VM $PSFS$ :MAINT $vr$ m.SIDISK directory.

**Note:** You should backup the VM $PSFS$ :MAINT $vr$ m.SIDISK directory periodically. The 51D disk can be used as a backup for the VM $PSFS$ :MAINT $vr$ m.SIDISK directory.

### Change the Software Inventory Default from Minidisk to SFS Directory

1. Establish the correct minidisk access order for VMSES/E:

```
access vm $psfs$ :maint $vr$ m.sidisk d
```

2. Modify the VMFINS DEFAULTS file.

- a. Run LOCALMOD and reply to the VMFLMD1301R prompt with a '1' to enter an XEDIT session.

## Changing the Software Inventory to an SFS Directory

```
localmod vmses vmfins defaults
```

df1l0001 is the file type of your replacement part for VMFINS DEFAULTS. DFL is the file type abbreviation for DEFAULTS, and L0001 is the local *modid* that was selected for this change.

- b. Update the replacement VMFINS DEFAULTS file. Enter the following commands to change the default for SIDISK from 51D to VMPSFS:MAINT*vr*m.SIDISK:

```
===> change /51D/vmpsfs:maintvrm.sidisk/ *  
===>  
file
```

### 3. Modify the VMSESE PROFILE file.

- a. Run LOCALMOD and reply to the VMFLMD1301R prompt with a '1' to enter an XEDIT session:

```
localmod vmses vmsese profile
```

prf10002 is the file type of your replacement part for VMSESE PROFILE. PRF is the file type abbreviation for PROFILE, and L0001 is the local *modid* that was selected for this change.

- b. Update the replacement VMSESE PROFILE file. Enter the following commands to change the default for SIDISK from 51D to VMPSFS:MAINT*vr*m.SIDISK:

```
===>  
locate :SHRDISK.  
===>  
delete 1  
===>  
input :SHRDISK. dir vmpsfs:maintvrm.sidisk  
===>  
file
```

### 4. Rebuild the VMFINS DEFAULTS and VMSESE PROFILE files and make your changes active.

```
service vmses build
```

**Note:** The updated files containing your change have been built to the test build disk (5E6). To test your update, you can access the test build disk in place of the production build disk (5E5) and issue some VMFINS commands.

After you are satisfied with your change, you can place it into production, by entering these commands:

```
put2prod vmses  
copyfile * * t = = b (olddate  
release t
```

**Note:** If you are running in an SSI cluster, you must run the PUT2PROD command on all members in the cluster.

### 5. Update the SEGBLD PPF.

- a. Run LOCALMOD and reply to the VMFLMD1301R prompt with a '1' to enter an XEDIT session.

```
localmod cp segbld $ppf
```

- b. Update the replacement SEGBLD \$PPF file. Enter the following commands to change the default for SIDISK from 51D to VMPSFS:MAINT*vr*m.SIDISK.

```
===> locate /&SID/  
===>  
delete 1  
===>  
up 1  
===>  
input &SID DIR $VMPSFS$: $MNTVRM$.SIDISK
```

```
===>
file
```

Note that in the input command above, the "VRM" in \$MNTVRM\$ is the characters "VRM" rather than the version, release and modification level.

- c. Build the new SEGBLD \$PPF and PPF.

```
service cp build
```

- d. Place the new SEGBLD \$PPF into production.

```
put2prod cp
```

**Note:** If you are running in an SSI cluster, you must run the PUT2PROD command on all members in the cluster.

## Enroll Users and Give Them Access Authority

You need to issue the following ENROLL and GRANT AUTHORITY commands for each current user and any future user that uses VMSES/E to install and service licensed products. You only need to issue the commands once for each user.

1. Log on to the MAINT`vr`m user ID if you are not already logged on:

```
logon maintvr
```

Because MAINT`vr`m owns the VMPSFS:MAINT`vr`m.SIDISK directory, you must issue the ENROLL and GRANT AUTHORITY commands from the MAINT`vr`m user ID.

2. Enroll users in the VMPSFS: filepool:

```
enroll user userid vmpsfs:
```

*userid* is any user ID that requires access to the VMPSFS: filepool.

3. Give users write access to the VMPSFS:MAINT`vr`m.SIDISK directory and all files that reside in it:

```
grant authority vmpsfs:maintvr.sidisk to userid (write newwrite
```

*userid* is the user ID of each VMSES/E user.

The WRITE option gives users authority to write to the VMPSFS:MAINT`vr`m.SIDISK directory.

The NEWWRITE option lets the user update any new file that is added to the directory. If you are only giving write access on an individual file basis, you will not want to give all users NEWWRITE authority to this directory.

**Note:** If you want to control write authority on individual files, you must specify actual file names and file types in place of the \* \* in this command. If you choose to grant authority for individual files, you must ensure that all users have write access to the system-level Software Inventory files.

4. Give users write access to all the files that already exist in the directory:

```
grant authority * * vmpsfs:maintvr.sidisk to userid (write
```

\* \* identifies all files in the VMPSFS:MAINT`vr`m.SIDISK directory.

The WRITE option on this command lets the user update files that were specified in this command.

5. Update the MAINT`vr`m PROFILE EXEC to change the access to the 51D disk.

```
change /51D/vmpsfs:maintvr.sidisk (forcerw / *
```

**Note:** Update any other PROFILE EXECs that access the 51D software inventory disk.



## Part 5. Reference

This part of the book provides reference information on the following topics:

- Railroad track command syntax
- Online HELP Facility
- VMSES/E exec and command formats
- Product parameter file syntax
- Software Inventory syntax

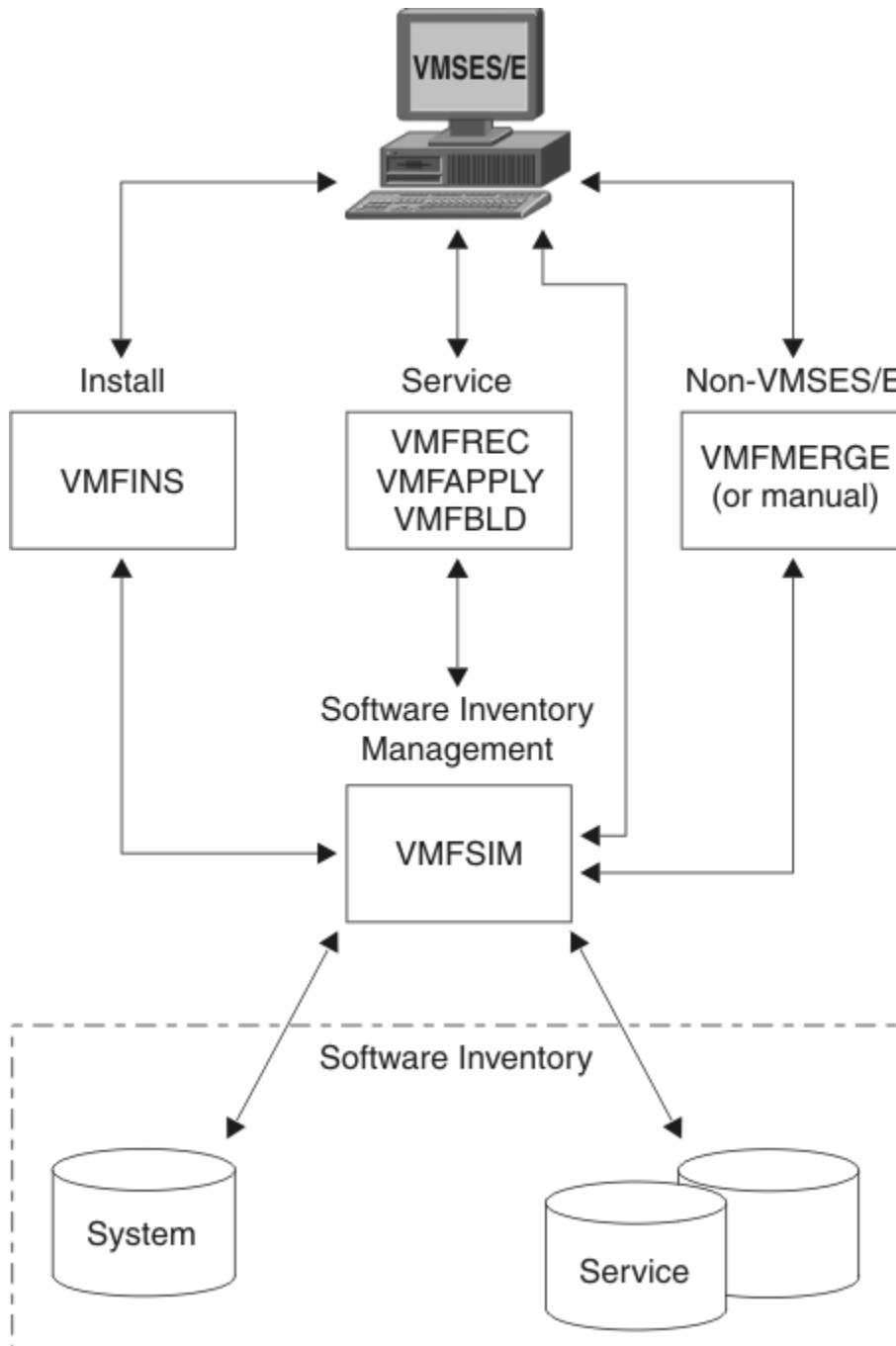


Figure 139. VMSES/E - Reference Information





---

## Chapter 19. Using the VMSES/E Reference Information

This chapter provides information to help you use the reference information in this book.

---

### Understanding Syntax Diagrams

For an explanation of the syntax diagrams used in this book, see [“Syntax, Message, and Response Conventions”](#) on page xxix.

---

### Using the Online HELP Facility

You can receive online information about VMSES/E commands using the z/VM HELP Facility. For example, to display a menu of VMSES/E commands, enter:

```
help vmses menu
```

To display information about a specific VMSES/E command (VMFREC in this example), enter:

```
help vmses vmfrec
```

For more information about using the HELP Facility, see [z/VM: CMS User's Guide](#). To display the main HELP Task Menu, enter:

```
help
```

For more information about the HELP command, see [z/VM: CMS Commands and Utilities Reference](#) or enter:

```
help cms help
```

---

### Using the VMSES/E Commands

In order to use the VMSES/E commands, you should have:

- An A-disk accessed read-write (R/W).
- The VMSES/E build disk accessed. By default, the VMSES/E build disk is the 5E5 disk, and it is accessed as B.
- The Software Inventory disk accessed. By default, the Software Inventory disk is the 51D disk, and it is accessed as D.



# Chapter 20. VMSES/E Exec and Command Format Summaries

This chapter is a general reference for VMSES/E commands that are used during product installation and service application. For information on how to read the syntax diagrams, use the online HELP Facility or see [“Understanding Syntax Diagrams”](#) on page 227. To run the VMSES/E commands, see [“Using the VMSES/E Commands”](#) on page 227.

## Using Tools for Service and System Generation

z/VM provides a number of tools to help you perform service and system generation tasks. [Table 16](#) on page 229 lists the tools, their task, and where to find more information.

<i>Table 16. z/VM Service and System Generation Tools</i>		
<b>Tool</b>	<b>Task</b>	<b>For more information, see...</b>
CHKAPARS	Evaluates system APAR data.	<a href="#">“CHKAPARS EXEC”</a> on page 234
EXPAND	Adds space to a program in object deck form.	<a href="#">“EXPAND Command”</a> on page 783
GENCPBLS	Updates the CP load list build list.	<a href="#">“GENCPBLS EXEC”</a> on page 237
INSTFPP	Installs optional products.	<a href="#">“INSTFPP EXEC”</a> on page 723
LOCALMOD	Creates a local modification for a part.	<a href="#">“LOCALMOD EXEC”</a> on page 245
PRODUTL	Selectively copies one or more groups of files for a product, component, or feature into production, or selectively compares such files, for content-change notification purposes.	<a href="#">“PRODUTL EXEC”</a> on page 249
PUT2PROD	Places the z/VM components, features, or products that are installed on the z/VM system into production.	<a href="#">“PUT2PROD EXEC”</a> on page 258
SERVICE	Installs an RSU or applies corrective service (COR).	<a href="#">“SERVICE EXEC”</a> on page 261
SERVMGR	Handles all tasks associated with z/VM Centralized Service Management (z/VM CSM).	<a href="#">“SERVMGR EXEC”</a> on page 269
SNTINFO	Gets discontinuous saved segment (DCSS) information directly from CP.	<a href="#">“SNTINFO EXEC”</a> on page 733
VMFAPPLY	Updates the maintenance level of the specified product.	<a href="#">“VMFAPPLY EXEC”</a> on page 291
VMFASM	Updates an ASSEMBLE source file according to entries in a control file, then assembles the source file to produce an object file.	<a href="#">“VMFASM EXEC”</a> on page 297
VMFBTMAP	Creates a file, for each PTF prefix, that contains a bitmap representation of all PTFs received on the z/VM system for the PTF prefix.	<a href="#">“VMFBTMAP EXEC”</a> on page 364
VMFBLD	Builds objects for the specified product.	<a href="#">“VMFBLD EXEC”</a> on page 305

Table 16. z/VM Service and System Generation Tools (continued)

<b>Tool</b>	<b>Task</b>	<b>For more information, see...</b>
VMFCNVT	Converts size and block size data into cylinders and displays the results.	<a href="#">“VMFCNVT EXEC” on page 367</a>
VMFCOPY	Copies a file to a VMSES/E target minidisk or SFS directory and updates the parts catalog table on that target.	<a href="#">“VMFCOPY EXEC” on page 369</a>
VMFERASE	Erases a file on a VMSES/E target minidisk or SFS directory and updates the parts catalog table on that target.	<a href="#">“VMFERASE EXEC” on page 377</a>
VMFENRPT	Creates a report of the products that are enabled and disabled on your system.	<a href="#">“VMFENRPT EXEC” on page 373</a>
VMFEXUPD	Calls the EXECUPDT command to apply updates to a \$Source program.	<a href="#">“VMFEXUPD EXEC” on page 381</a>
VMFHASM	Updates an ASSEMBLE source file according to entries in a control file, then uses the H assembler to produce an object file.	<a href="#">“VMFHASM EXEC” on page 387</a>
VMFHLASM	Updates an ASSEMBLE source file according to entries in a control file, then uses the HL assembler to produce an object file.	<a href="#">“VMFHLASM EXEC” on page 394</a>
VMFINFO	Queries the software inventory tables.	<a href="#">“VMFINFO EXEC” on page 402</a>
VMFINS	Installs, migrates, builds, and deletes products.	<a href="#">“VMFINS EXEC” on page 404</a>
VMFMERGE	Applies PTFs to systems network architecture (SNA) products. VMFMERGE is used only to service SNA products.	<a href="#">“VMFMERGE EXEC” on page 774</a>
VMFMRDSK	Consolidates the contents of minidisks/ directories within a string.	<a href="#">“VMFMRDSK EXEC” on page 444</a>
VMFNLS	Applies updates to national language files and compiles the updated versions.	<a href="#">“VMFNLS EXEC” on page 448</a>
VMFOVER	Creates a temporary PPF by applying overrides to a source PPF.	<a href="#">“VMFOVER EXEC” on page 456</a>
VMFPPF	Compiles a source PPF into its usable form.	<a href="#">“VMFPPF EXEC” on page 458</a>
VMFPSU	Helps you choose which method to use when you install a product service upgrade (PSU).	<a href="#">“VMFPSU EXEC” on page 462</a>
VMFQMDA	Displays the current VMSES/E access order.	<a href="#">“VMFQMDA EXEC” on page 469</a>
VMFQOBJ	Returns information about objects defined in build lists.	<a href="#">“VMFQOBJ EXEC” on page 472</a>
VMFREC	Processes installation media and service tapes.	<a href="#">“VMFREC EXEC” on page 477</a>
VMFREM	Removes PTFs received by the VMFREC EXEC and applied by the VMFAPPLY EXEC.	<a href="#">“VMFREM EXEC” on page 485</a>
VMFREPL	Supports the local modification of replacement maintained parts.	<a href="#">“VMFREPL EXEC” on page 493</a>

<b>Tool</b>	<b>Task</b>	<b>For more information, see...</b>
VMFREMOV	Removes PTFs from systems network architecture (SNA) products. VMFREMOV is used only to service SNA products.	<a href="#">“VMFREMOV EXEC” on page 777</a>
VMFSETUP	Sets up a minidisk and SFS directory access order, or detaches minidisks that were linked by previous invocations of the VMFSETUP EXEC, depending on how it is invoked.	<a href="#">“VMFSETUP EXEC” on page 500</a>
VMFSGMAP	Processes and displays the saved segment information defined in a saved segment configuration build list and save segment data file.	<a href="#">“VMFSGMAP EXEC” on page 506</a>
VMFSIM	Provides an interface to the software inventories.	<a href="#">“VMFSIM EXEC” on page 525</a>
VMFSUFIN	Installs service from RSU service envelope files, COR service envelope files, or both.	<a href="#">“VMFSUFIN EXEC” on page 597</a>
VMFSUFTB	Builds the VM SYSSUF table that contains a list of all installed products and related data needed by the automated service commands.	<a href="#">“VMFSUFTB EXEC” on page 602</a>
VMFUPDAT	Updates selected software inventory tables.	<a href="#">“VMFUPDAT EXEC” on page 604</a>
VMFVIEW	Displays message logs using XEDIT with predefined function keys.	<a href="#">“VMFVIEW EXEC” on page 614</a>
VMFZAP	Applies ZAPs to systems network architecture (SNA) products. VMFZAP is used only to service SNA products.	<a href="#">“VMFZAP EXEC” on page 779</a>
ZAPTEXT	Modifies or dumps individual text files.	<a href="#">“ZAPTEXT EXEC” on page 781</a>

z/VM provides a number of tools to help you perform service and system generation tasks. [Table 17 on page 231](#) lists z/VM service and system generation commands and where to find more information about them.

<b>Tool</b>	<b>Task</b>	<b>For more information, see...</b>
ASSEMBLE	Processes source statements in assembler language source files.	<a href="#">z/VM: CMS Commands and Utilities Reference</a>
CSLGEN	Builds a callable services library (CSL).	<a href="#">z/VM: CMS Commands and Utilities Reference</a>
DCSSGEN	Builds the CMS installation saved segment (CMSINST).	<a href="#">z/VM: CMS Commands and Utilities Reference</a>
DIRECTXA	Creates a user directory.	<a href="#">z/VM: CP Commands and Utilities Reference</a>
DISKMAP	Summarizes the MDISK statements in the user directory. The output shows gaps and overlaps between minidisk assignments.	<a href="#">z/VM: CP Commands and Utilities Reference</a>
DOSGEN	Builds the CMSDOS physical saved segment.	<a href="#">z/VM: CMS Commands and Utilities Reference</a>
EXECUPDT	Produces an updated version of a \$Source file.	<a href="#">z/VM: CMS Commands and Utilities Reference</a>

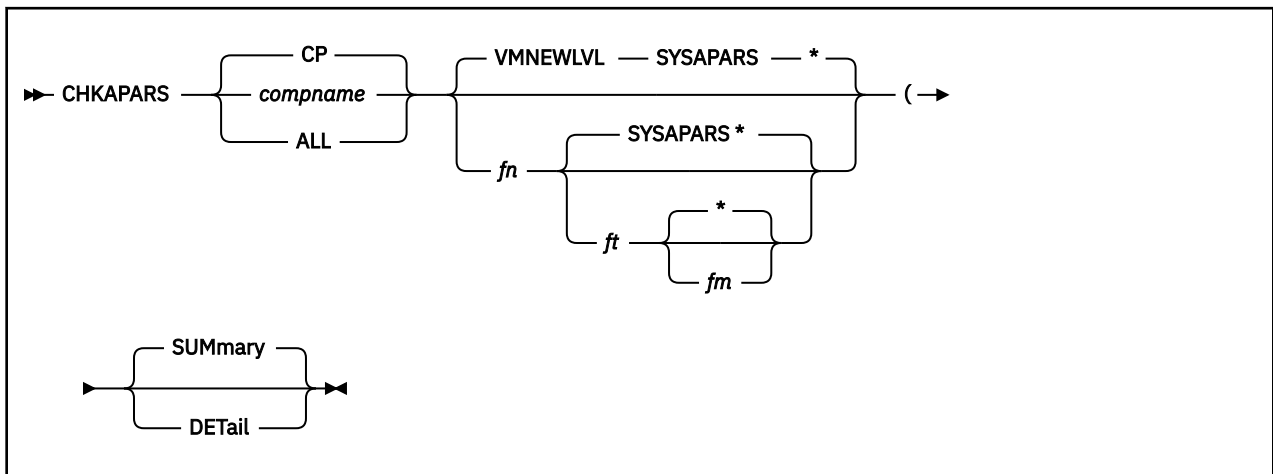
Table 17. Additional z/VM Service and System Generation Tools (continued)

<b>Tool</b>	<b>Task</b>	<b>For more information, see...</b>
GENMOD	Generates CMS module files.	<a href="#">z/VM: CMS Commands and Utilities Reference</a>
GROUP	Builds a GCS configuration file.	<a href="#">z/VM: Group Control System</a>
HCPLDR	Calls and controls the system loader.	<a href="#">z/VM: CP Commands and Utilities Reference</a>
IPWIZARD	Creates a minimal TCP/IP configuration that establishes basic connectivity to your IP network. Creates the TCP/IP SYSTEM DTCPARMS, TCPIP DATA, and PROFILE TCPIP files.	<a href="#">z/VM: CP Commands and Utilities Reference</a>
LANGGEN	Loads national language text files into a saved segment.	<a href="#">z/VM: CMS Commands and Utilities Reference</a>
LANGMERG	Combines national language files for an application into a single text file.	<a href="#">z/VM: CMS Commands and Utilities Reference</a>
LOADLIB	Lists, copies, or compresses CMS load libraries.	<a href="#">z/VM: CMS Commands and Utilities Reference</a>
MIGR51D	Updates the system software inventory files.	<a href="#">z/VM: CP Commands and Utilities Reference</a>
MOVE2SFS	Moves data from minidisks to shared file system (SFS) servers, and reclaims the unused minidisk space.	<a href="#">z/VM: CP Commands and Utilities Reference</a>
PRELOAD	Collects multiple text files and reformats them into a single text file.	<a href="#">z/VM: CMS Commands and Utilities Reference</a>
SAMGEN	Builds the CMSBAM physical saved segment.	<a href="#">z/VM: CMS Commands and Utilities Reference</a>
SAMPNSS	Defines named saved systems.	<a href="#">z/VM: CMS Commands and Utilities Reference</a>
SAVEFD	Places file directory information for a shared, extended data format (EDF) R/O minidisk into a discontinuous shared segment (DCSS).	<a href="#">z/VM: CMS Commands and Utilities Reference</a>
SEGGEN	Builds logical saved segments defined in a physical saved segment.	<a href="#">z/VM: CMS Commands and Utilities Reference</a>
SPXTAPE	Saves standard spool files and system data files on tape and restores SPXTAPE-format files from tape to the spooling system.	<a href="#">z/VM: CP Commands and Utilities Reference</a>
UTILITY	Provides occasionally-used installation functions, such as, issuing DIAGNOSE code X'24' and X'210' for a virtual device and creating a stand-alone service utility tape for either or both ICKDSF and DDRXA.	<a href="#">z/VM: CP Commands and Utilities Reference</a>
VMFLKED	Link edits modules into a load library (LOADLIB).	<a href="#">z/VM: CMS Commands and Utilities Reference</a>
VMFMAC	Builds macro libraries (MACLIBs) containing macro and copy files.	<a href="#">z/VM: CMS Commands and Utilities Reference</a>
VMFPLC	Provides a front end to routines that use VMFPLC2 when conversion to VMFPLCD or a dual path is desired.	<a href="#">z/VM: CMS Commands and Utilities Reference</a>

Table 17. Additional z/VM Service and System Generation Tools (continued)

<b>Tool</b>	<b>Task</b>	<b>For more information, see...</b>
VMFPLCD	Loads files from an envelope, dumps files to an envelope, and controls various envelope operations.	<a href="#"><u>z/VM: CMS Commands and Utilities Reference</u></a>
VMFPLC2	Loads files from tape, dumps files to tape, and controls various tape drive operations.	<a href="#"><u>z/VM: CMS Commands and Utilities Reference</u></a>
VMFTXT	Builds a text library (TXTLIB) from text decks.	<a href="#"><u>z/VM: CMS Commands and Utilities Reference</u></a>
ZAP	Modifies or dumps MODULE, LOADLIB, or TXTLIB files.	<a href="#"><u>z/VM: CMS Commands and Utilities Reference</u></a>

## CHKAPARS EXEC



### Purpose

The CHKAPARS EXEC evaluates system APAR data for a selected z/VM component or all components. APAR data for the subject z/VM system is acquired by using the VMSES/E SERVICE STATUS ALLAPARS command. The acquired data is compared to APAR reference data that is acquired from a specified CMS file. A report of the evaluation is produced in a program-created report file.

### Operands

#### *compname*

The *compname* operand specifies a z/VM component name. The operand can be a component of interest (for example, CP or TCPIP) or the keyword ALL, which specifies all z/VM base components. If you do not specify a component name, then the value defaults to CP.

#### *fn*

The *fn* operand specifies the filename of a file that contains reference APAR information. The asterisk wildcard is not allowed in the filename. If you do not specify a *fn* operand, then the exec uses the default file specification: VMNEWLVL SYSAPARS \*.

#### *ft*

The *ft* operand specifies the filetype of a file that contains reference APAR information. The asterisk wildcard is not allowed in the filetype. If you do not specify a *fn* operand, then you cannot specify a *ft* operand. The default value for the *ft* operand is SYSAPARS.

#### *fm*

The *fm* operand specifies the filemode of a file that contains reference APAR information. If you do not specify a *ft* operand, then you cannot specify a *fm* operand. The default value for the *fm* operand is \*.

If two or more files match the file specification, CHKAPARS processes only the first file in the CMS search order.

#### SUMmary

The SUMMARY keyword specifies that only summary information is included in the report. The summary report lists APARs by product ID and identifies three APAR categories:

- Matched system APARs. APARs are on the system and on the reference list of APARs.
- Unique system APARs. APARs are on the system but not on the reference list of APARs.
- Reference APARs only. APARs are on the reference list but not present on the subject system.



**DETail**

The DETAIL keyword specifies that detail information is included in the report. The detail report lists APARs by the categories in the summary report. For system APARS, the detail report also groups APARs by the following statuses:

- RECEIVED
- APPLIED
- BUILT
- PUT2PROD

**Usage Notes**

1. CHKAPARS can only display APAR information for products that have supplied previous-release APAR information for their respective new-level equivalent.

**Input and Output Files****Input Files****APAR reference file**

The APAR reference file is a textual data file that lists the z/VM APARs in a new z/VM release.

A file with filetype SYSAPARS must have the same (tagged) format as the VMSES/E System-Level Base APAR Table. The Base APAR Table file is VM SYSAPARS, and is supplied as part of the installation media for a z/VM release.

A file with filetype other than SYSAPARS must have one of the following formats:

- A plain text file in the same format as the LIST input that is required for the VMSES/E SERVICE command.
- A list of two-word records where the second word is a component name. In this case, the report data combines the data by component. The result is the same as when a file with filetype SYSAPARS file is used.

**Output Files****CHKAPARS REPORT**

The CHKAPARS REPORT file contains the evaluation results. The information is in plain text format. Commentary information is delimited by an asterisk in column 1 of a file record.

**\$VMFCHK \$MSGLOG**

The \$VMFCHK \$MSGLOG contains messages that are associated with the CHKAPARS command. The content can be viewed by using the VMFVIEW command.

**Return Codes**

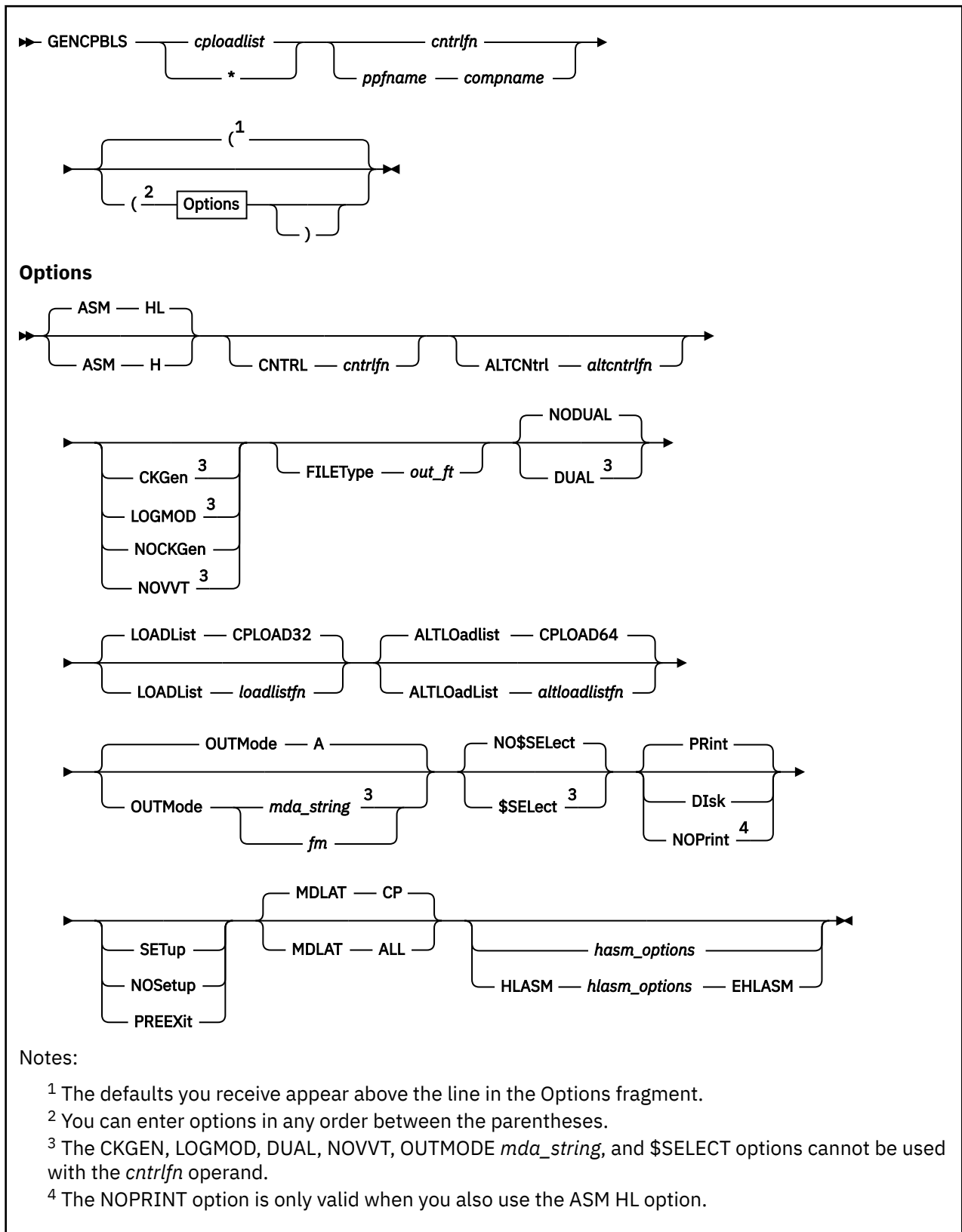
The CHKAPARS EXEC issues the following return codes:

Return Code	Explanation
0	Successful execution and normal completion. No errors were encountered.
4	A warning condition was encountered. Review of output and results is advised.
6	Program Initialization errors were encountered.
8	General-case processing errors were encountered.
100	Significant program processing errors were encountered.

## Recovery Information

The CHKAPARS EXEC can be restarted by issuing the command again.

## GENCPBLS EXEC



## Purpose

The GENCPBLS EXEC updates the CP load list build list based on a local modification to HCPMDLAT MACRO.

## Operands

### *cploadlist*

is the file name of the CP load list to be created if the NODUAL option is specified.

\*

is a place holder that can be used with the DUAL option. *cploadlist* is ignored if the DUAL option is specified.

### *cntrlfn*

is the file name of a control file. The file type must be CNTRL.

### *ppfname*

is the file name of a usable form product parameter file. The product parameter file must have a file type of PPF. The name of the control file that is to be used to update the source file is obtained from this product parameter file.

### *compname*

is the name of the component (such as CP or CMS) as it is specified on the :COMPNAME tag in the product parameter file. *compname* is a 1- to 16-character alphanumeric identifier.

## Options

### ASM H

indicates HASM is used to assemble the part.

### ASM HL

indicates HLASM is used to assemble the part.

### CNTRL

specifies a control file is used to identify the AUX file structure.

### *cntrlfn*

is the file name of the control file that is used to identify the AUX file structure. The file type of the control file is CNTRL. This value overrides the value on the :CNTRL tag in the PPF. The CNTRL option can not be used if operand *cntrlfn* is specified.

### ALTCNtrl

specifies a control file is used to identify the AUX file structure for the alternate CP load list.

### *altcntrlfn*

is the file name of the control file that is used to identify the AUX file structure for the alternate CP load list. The file type of the control file is CNTRL. This value overrides the value on the :ALTCNTRL tag in the PPF. The ALTCNTRL option cannot be used if operand *cntrlfn* is specified.

### CKGen

requests validation of the AUX files against the version vector tables and issues an error message if a mismatch is detected. The version vector tables are not updated. In addition, CKGen verifies that the current service level of the MACRO and CP load list match.

### LOGMOD

requests validation of the AUX files for HCPMDLAT against the version vector tables and automatically updates the local version vector tables when a mismatch is detected. When you specify the LOGMOD option, GENCPBLS modifies only the VVTs that are defined in the control file above the :UPDTID level defined in the PPF. All other VVT levels are only compared to the AUX files, and mismatches are displayed. All LOCAL disks must be accessed as Read/Write.

When you use the LOGMOD option:

- If a version vector table does not exist on a LOCAL disk, it is created on the first disk in the LOCAL string.

- If the AUX file for a part is not found, the :PART entry (if found) is deleted from the version vector table.
- If the AUX file for a part is empty, the :MOD data is deleted from the version vector table for that part. The :PART entry is not deleted from the version vector table.
- LOGMOD will update the local VVTs for the HCPMDLAT MACRO and the *cploadlist* EXEC. No VVT entries are created for the HCPLDL assemble file.

**NOCKGen**

requests no validation of the AUX files against the version vector tables. The AUX file structure is used to update the source file and name the output file.

**NOVVT**

requests no validation of the AUX files against the version vector tables. The AUX file structure is used to update the source file and name the output file.

**Note:** If you omit the CKGEN, LOGMOD, NOCKGEN, and NOVVT options, the GENCPBLS EXEC uses the value of the :CKGEN tag in the PPF to determine whether to validate the AUX files against the version vector table. If the :CKGEN tag does not appear in the PPF, no validation is performed (NOCKGEN is assumed).

**FILEType**

indicates the file type for the output file that is created. This option overrides any naming from the AUX, CNTRL, or VVT structures.

***out\_ft***

is the file type for the output file.

**DUAL**

indicates both the primary and alternate load lists specified by the LOADLIST and ALTLOADLIST options should be created.

**NODUAL**

indicates only the load list specified by the *cploadlist* operand should be created.

**LOADList**

specifies the filename of the primary loadlist.

**CPLOAD32**

is the default primary load list file name.

***loadlistfn***

is the primary load list file name.

**ALTLOadlist**

specifies the filename of the alternate load list.

**CPLOAD64**

is the default alternate load list filename.

***altloadlistfn***

is the alternate load list filename.

**PRint**

sends the listing output to the virtual printer. PRINT is the default.

**DIsk**

creates the listing output on your A-disk.

**NOPrint**

suppresses the writing of the listing output.

**Note:** If you specify PRINT, NOPRINT, and DISK between the HLASM and EHLASM keywords, they are ignored.

**SETup**

sets up a minidisk/directory access order for the assemble function according to entries in the :MDA section of the product parameter file. This option is valid only when using a product parameter file. If a user exit is specified in the product parameter file, setup will occur after the user exit is called.

**NOSetup**

does not set up a new minidisk/directory access order.

**PREExit**

sets up a minidisk or SFS directory access order for the assemble function according to entries in the :MDA section of the product parameter file. If a user exit is specified in the product parameter file, setup will occur before the user exit is called.

**Note:** If the SETUP, PREEXIT, and NOSETUP options are omitted, the GENCPBLS EXEC uses the value of the :SETUP tag in the product parameter file to determine whether to set up a new access order.

**MDLAT CP**

indicates that only the HCPMDLAT MACRO will be used to generate the CP load list. This is the default.

**MDLAT ALL**

indicates that all the xxxMDLAT MACROs that are members of any maclibs that are specified by the MACS statement in the selected control file will be used to build the CP load list.

**OUTMode**

indicates the file mode for the output files (for example, update files, AUX files, text decks, and load list) that are created. The file mode specified as OUTMODE must be accessed Read/Write.

**A**

indicates that file mode A is the file mode for the output files. A is the default.

***fm***

is the file mode for the output files.

***mda\_string***

is the name of the symbolic string of disks from the :MDA section of the product parameter file. The output is placed on the first disk specified in this string.

**NO\$SElect**

does not update the *appid* \$SELECT file. NO\$SELECT is the default.

**\$SElect**

updates the *appid* \$SELECT file to indicate the HCPMDLAT MACRO and CP LOAD LIST have changed. The first APPLY disk in the :MDA section of the product parameter file must be accessed Read/Write. Other APPLY disks must be accessed.

**HASM OPTIONS SUPPORTED BY GENCPBLS:**

You can only enter *hasm\_options* when you use the ASM H option.

In the following table, the left column shows the options of the HASM command. The right column shows how these options are supported by GENCPBLS when invoking the HASM command. The default values for these options are also shown. The HASM defaults are used wherever possible. Keyword-function options must be entered without the parentheses.

HASM Options	GENCPBLS Options
ALIGN NOALIGN	same
NOBATCH BATCH	same
NODBCS DBCS	same
DECK	NODECK
same	
ESD	NOESD
same	
FLAG(0) FLAG(n)	FLAG 0 FLAG n
LINECOUN(55) LINECOUN(nn)	LINECOUN 55 LINECOUN nn

HASM Options	GENCPBLS Options
LIST NOLIST	same
NUM NONUM	same
OBJECT NOOBJECT	OBJect NOOBJect
PRINT NOPRINT DISK	PRint DIsk
RENT NORENT	same
RLD NORLD	same
STMT NOSTMT	same
SYSPARM(string) SYSPARM(?) SYSPARM()	SYSPARM string SYSPARM ? SYSPARM SUP SUP SYSPARM EXP EXP
TERM NOTERM	same
TEST NOTEST	same
XREF (FULL) XREF(SHORT) NOXREF	XREF FULL XREF SHORT NOXREF
<p><b>Note:</b> Defaults are shown highlighted.</p> <p>The SYSPARM SUP option suppresses the expansion of macros. The SYSPARM EXP option activates the expansion of macros. SYSPARM SUP is the default.</p>	

### HLASM

indicates the beginning of the HLASM options, which are passed directly to the HLASM command. GENCPBLS does not parse these options; the HLASM command performs the parsing.

#### *hlasm\_options*

are the HLASM options. You can only enter *hlasm\_options* when you use the ASM HL option.

For a description of the HLASM options, see *IBM High Level Assembler/MVS & VM & VSE Programmer's Guide*, SC26-4941.

### EHLASM

indicates the end of the HLASM options.

### Usage Notes

1. You must modify the HCPMDLAT MACRO or the alternate xxxMDLAT MACRO before you run GENCPBLS.  
  
You need to add an update record to the HCPMDLAT AUX file and create a local update file for the HCPMDLAT MACRO. All xxxMDLAT MACROS used to build the CP load list must be members in maclibs that are specified in the control file used by the GENCPBLS command. The control file used is defined in the PPF or is specified as the CNTRL *cntrlfn* option of the command syntax. See [z/VM: Service Guide](#) for more information.
2. GENCPBLS **creates** a temporary MACLIB (\$TMP\$ MACLIB) on the A-disk that contains an updated level of the HCPMDLAT MACRO. This MACLIB is the first one to be made global when the HCPLDL file is assembled. The temporary MACLIB is erased after HCPLDL is assembled.
3. GENCPBLS **does not** rebuild the MACLIB that contains HCPMDLAT MACRO. You must run VMFBLD to rebuild the MACLIB containing HCPMDLAT MACRO after you run GENCPBLS.
4. GENCPBLS **does not** rebuild the CP nucleus. You must run VMFBLD to rebuild the CP nucleus after you run GENCPBLS.
5. The HASM OBJECT option is always used, rather than using the HASM default NOOBJECT.
6. GENCPBLS handles packed files.

7. If you receive warnings or errors from the UPDATE command, check the *fn* UPDLOG file for additional information.
8. When you specify the \$SELECT option, the select data file (*appid* \$SELECT) is updated with two records. The first record is the HCPMDLAT MACRO and the other record is the CP load list consisting of either:
  - *cploadlist* and EXC
  - *cploadlist* and the full file type (when the FILETYPE option is specified).
 The select data file is used by VMFBLD to determine which objects need to be built.
9. When you create local modifications, you can use the \$SELECT, LOGMOD, and OUTMODE options to eliminate some manual steps, such as updating the *appid* \$SELECT file, updating local version vector table files, and saving the results on a LOCALMOD disk.
10. If the product parameter file is not used the GENCPBLS command uses the control file structure to determine the file type of the resulting output file, unless you used the FILETYPE option. For more information, see “How VMSES/E Uses Control Files” on page 117.
11. VMFBLD uses the version vector table to determine the correct level of the part to use during build processing. If you do not specify the LOGMOD option, you must either manually update the version vector table before you run VMFBLD or you must rerun GENCPBLS and specify the LOGMOD option.
12. If the file type option is not specified the file type of the CP load list is determined from the HCPMDLAT MACRO's CNTRL and AUX structure.
13. The maclibs must be specified on the MACS statement in the control file and they must be on an accessed minidisk or SFS directory.

### Examples

- To run GENCPBLS and use the defaults:

```
GENCPBLS CLOAD SERVP2P CP
```

- To include all the xxxMDLAT MACROs when building the CP load list enter:

```
GENCPBLS CLOAD SERVP2P CP (MDLAT ALL
```

## Input and Output Files

### Input Files

#### ***cntrlfn* CNTRL**

The control file identified by the :CNTRL tag in the product parameter file (PPF).

#### ***altcntrlfn* CNTRL**

The control file identified by the :ALTCNTRL tag in the product parameter file (PPF).

#### **HCPMDLAT AUX*lvlid***

HCPMDLAT user-updated AUX file.

#### ***ppfname* PPF**

The usable form product parameter file.

#### **HCPLDL ASSEMBLE**

Assemble file for CP load list.

#### **HCPMDLAT MACRO**

The macro defining the entries for the load list.

#### ***xxxMDLAT* MACRO**

The macro defining user entries for modules to be added to the load list.

### Output Files

#### ***cploadblist* *cploadftype***

The CP load list build list.



***altcploadblist cploadftype***

The alternate CP load list build list.

***cploadblist LISTING***

The CP load list build list LISTING file.

***altcploadblist LISTING***

The alternate CP load list build list LISTING file.

***HCPLDL updtft***

Dummy update for HCPLDL ASSEMBLE file.

***appid \$SELECT***

The select data file.

**Input/Output Files*****appid VVTlvlid***

The version vector table specified by the control file which is pointed to by the :CNTRL tag in the product parameter file.

***HCPLDL AUXlvlid***

HCPLDL AUX file.

**Temporary Files*****\$TMP\$ MACLIB***

MACLIB containing updated HCPMDLAT MACRO.

***HCPLDL TXTmodid***

The assembled object deck.

**PPF Tags Used****:APPID**

The identifier of the product used to name the version vector table and select data file.

**:BLD**

Defines build processing for the product.

**:COMPNAME**

Defines the component in the product parameter file to be used.

**:CNTRL**

Identifies the file name of the control file to be used.

**:ALTCNTRL**

identifies the file name of the alternate control file to be used.

**:MDA**

Defines symbolic strings and the minidisks or SFS directories associated with them.

**:SETUP**

Controls whether the VMFSETUP EXEC is called to access minidisks/directories.

**Messages and Return Codes**

Appendix D, “Module Identifiers for VMSES/E Messages,” on page 749 shows the format of VMSES/E messages and lists the identifiers used by each VMSES/E exec.

For a complete explanation of any VMSES/E message, use the HELP Facility to view the message explanation online or see the appropriate messages documentation. To display information on a specific message, for example VMF002E, enter:

```
help msg vmf002e
```

If you are unfamiliar with the z/VM HELP Facility, enter help to display the main HELP Menu. For information on the HELP command, enter:

```
help cms help
```

**PI**

Return codes issued by the GENCPBLS EXEC may be returned to a user exit. For more information about user exits, see [:USEREXIT.](#)

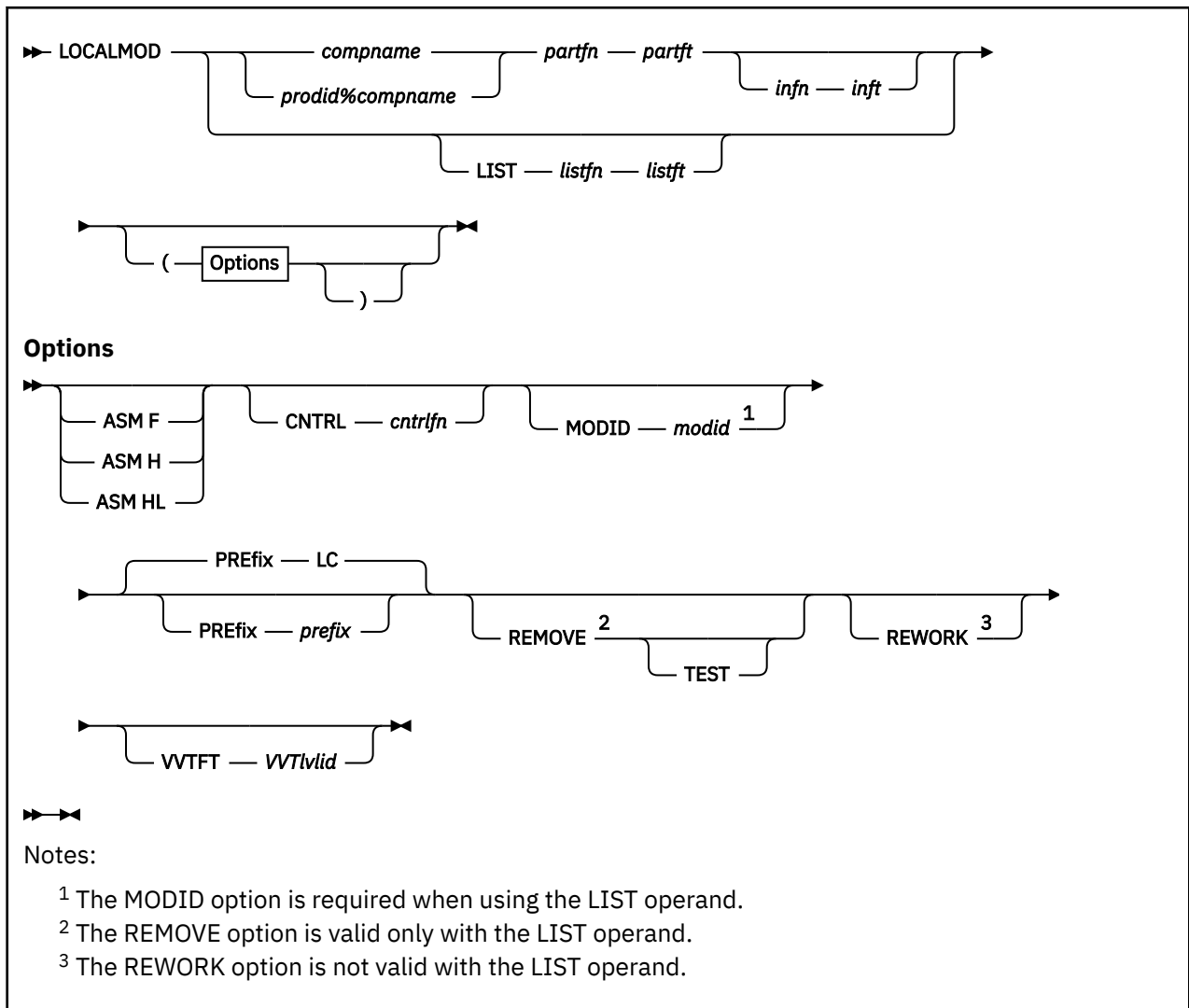
The GENCPBLS EXEC issues the following return codes:

<b>Return Code</b>	<b>Explanation</b>
0	Command completed successfully.
4	Command completed with one or more warning conditions.
8	Command completed but at least one major process failed.
12	Command failed because of an internal error.
24	Command failed because of a command line syntax error.
28	Command failed because a required file was not found.
36	Command failed because a target disk or directory was not available.
100	Command failed because of an external error.

**PI end****Recovery Information**

The GENCPBLS command can be restarted by reissuing the command.

## LOCALMOD EXEC



### Purpose

The LOCALMOD EXEC creates or reworks a local modification for a part.

### Operands

#### ***compname***

is the name of the component as it is specified on the :PRODID tag in the system-level service update facility table (VM SYSSUF). *compname* is a 1- to 16-character alphanumeric identifier.

#### ***prodid%compname***

is the *prodid%compname* as specified on the :PRODID tag in the system-level service update facility table (VM SYSSUF). If using this form of the parameter, the *prodid*, the percent sign (%), and the *compname* are all required.

#### **LIST**

specifies that a user-supplied list of local modifications will be used.

The input file format should consist of one local modification per line. Each line should follow the format: *compname partfn partft infn inft*. Data after *inft* is considered to be commentary. Any line that starts with an asterisk (\*) is a comment.

***listfn***

is the file name of the input list.

***listft***

is the file type of the input list.

***partfn***

is the file name of the part to be modified.

***partft***

is the file type of the part to be modified.

***infn***

is the file name of the local modification.

***inft***

is the file type of the local modification.

**Options****ASM F**

specifies that ASSEMBLE will be used for assemblies.

**ASM H**

specifies that HASM will be used for assemblies.

**ASM HL**

specifies that ASMAHL or HLASM will be used for assemblies.

**CNTRL**

specifies that a control file will be used.

***cntrlfn***

is the file name of the control file.

**MODID**

specifies that a user-supplied local modification identifier will be used. See usage note [“5” on page 247](#) for the default value.

***modid***

is the local modification identifier.

**PREfix**

specifies that a user-supplied local modification identification prefix will be used. When combined with the local modification identifier (*modid*), the pair forms the local modification identification value. The default prefix is LC.

***prefix***

is the two-character local modification prefix.

**REMOVE**

specifies that all local modifications with the local modification identifier (*modid*) will be removed from each component in the specified list (parts will be ignored).

**TEST**

specifies that VMFREM will be called with the TEST option when the REMOVE option is specified.

**REWORK**

specifies that the highest local modification for the part will be reworked.

**VVTFT**

specifies that a version vector table will be used.

***VVTftid***

is the file type of the version vector table.

## Usage Notes

1. One of the assembler commands (ASMAHL, HLASM or HASM) and the REXXC command need to be available.
2. PASCAL parts are supported for TCP/IP only.
3. C parts are not supported.
4. The input file type for a local modification cannot contain the local modification identifier that is specified by the MODID option.
5. If a local modification identifier (*modid*) is not supplied and the REWORK option is not specified, a unique local modification identifier will be generated. The default format begins with L and is followed by a 4–digit number starting with 0001. For example, L0001.
6. Specifying an existing local modification identifier (*modid*) implies rework.
7. Only the highest local modification can be reworked.
8. If REWORK is entered with a user-supplied file name and file type, then the highest level of the local modification is replaced with the supplied part.
9. If VVTFT is specified to create a local modification, there cannot be a local modification at a higher level.
10. To remove local modifications without using the LIST operand, see “VMFREM EXEC” on page 485.
11. Creating or reworking a local modification for a cross-system highest release level program residing on the SSI system common disk (PMAINT 551, by default) must be done from a member that has the highest release of the product installed in the SSI cluster.

## Examples

- To create a local modification for the CP load list by modifying part HCPMDLAT MACRO, enter:

```
LOCALMOD CP HCPMDLAT MACRO
```

- To create a local modification for replacement maintained part CMSINST LSEG, enter:

```
LOCALMOD CMS CMSINST LSEG
```

- To create a local modification for source maintained part SYSPROF EXEC, enter:

```
LOCALMOD 1VMVMC23%MYCOMP SYSPROF $EXEC
```

- To create a local modification for CP part HCPIOX TXT12345 received from IBM, enter:

```
LOCALMOD CP HCPIOX TEXT HCPIOX TXT12345 ( MODID L2345
```

- To rework local modification for replacement maintained part CMSINST LSEG, enter:

```
LOCALMOD CMS CMSINST LSEG ( REWORK
```

- To test removing a local modification where the parts are included in a file, enter:

```
LOCALMOD LIST IBMFIX APAR0023 ( MODID A0023 REMOVE TEST
```

## Input and Output Files

### Input Files

#### VM SYSSUF

The system-level service update facility table.

#### *ppfname* PPF

The usable form product parameter file.

## LOCALMOD EXEC

### *cntrlfn* CNTRL

The control file.

### *appid* VVTIvid

The version vector table.

### Input/Output Files

#### LOCALMOD \$RESTART A

The local modification restart file.

#### \$VMFLMD \$MSGLOG A

The LOCALMOD message log.

### Output Files

#### VM SYSLMOD

The system-level modification table.

## Messages and Return Codes

For a complete explanation of each message, use the HELP Facility to view the message explanation online or see the appropriate messages documentation. To display information on a specific message, for example VMF002E, enter:

```
help msg vmf002e
```

If you are unfamiliar with the z/VM HELP Facility, enter `help` to display the main HELP Menu. For information on the HELP command, enter:

```
help cms help
```

Appendix D, “[Module Identifiers for VMSES/E Messages](#),” on page 749 shows the format of VMSES/E messages and lists the identifiers for each VMSES/E EXEC.

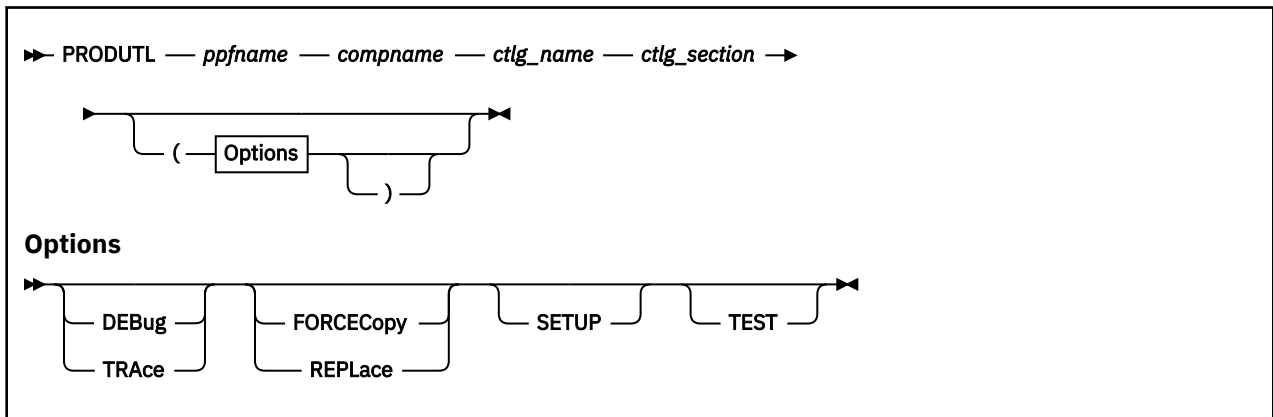
LOCALMOD issues the following return codes:

Return Code	Explanation
0	Command completed successfully.
2	Command completed successfully but extraneous data was encountered.
4	Command completed with one or more warning conditions.
12	Command failed because of an internal error.
24	Command failed because of a command line syntax error.
28	Command failed because a required file was not found.
36	Command failed because a target disk or directory was not available.
100	Command failed because of an external error.

## Recovery Information

The LOCALMOD EXEC can be restarted by issuing the command again.

## PRODUTL EXEC



### Purpose

Use the PRODUTL command to copy one or more groups of files for a product, component or feature into production, or to selectively compare such files (to obtain notification about files that have been updated by service actions). When files are placed into production, PRODUTL uses the VMSES/E VMFCOPY command to copy designated files from one resource (a minidisk or SFS directory) to another. When PRODUTL is used to identify changed files, the designated files are compared, and those that differ are reported (and by default, no copying of the subject files is performed). A catalog file, identified by a command operand, is used to identify the subject files, as well as the minidisks and SFS directories that are to be used for these processes. See [“CATALOG Files”](#) on page 252 for more information about the catalog file and its structure and content.

**Note:** This command is intended for use by a maintenance user ID, such as MAINTvrm, and should be used only during product installation or when applying service.

### Operands

#### *ppfname*

The name of the usable form product parameter file that is used for the installation and maintenance of a given z/VM product or feature. The file type must be PPF.

#### *compname*

The name of a component, as specified for a :COMPNAME . tag in the product parameter file; *compname* is a 1- to 16-character alphanumeric identifier.

The PPF Variable Declarations (:DCL.) section defined for *compname* determines the source minidisks and SFS directories from which product files are copied; likewise for the target production minidisks to which these files are copied.

#### *ctlg\_name*

The name of the product catalog file to be processed. The file type must be CATALOG.

#### *ctlg\_section*

The definition section of the catalog file to be processed. The *ctlg\_section* value is used to determine the "begin" and "end" tags that define each section of grouped entries (records) within a catalog file.

### Options

#### DEBUG

#### TRACE

Causes supplementary messages to be issued to provide information for diagnostic purposes. The DEBUG and TRACE options are synonymous.

### FORCECopy

#### REPLace

Causes files identified within a **configuration** or **sample** definition section to be copied to their configured (or targeted) name and type, regardless of whether such a counterpart target file already exists. By default, a file listed in a configuration definition section is copied to its configured name and type *only* if the production instance of that file does not already exist, whereas PRODUTL by default does not copy the files listed in a sample definition section to any target resource. See [“Conventional Catalog Definitions”](#) on page 255 for details about how entries within a configuration section are processed. The FORCECOPY and REPLACE options are synonymous. These options cannot be used when a **notify** definition section is processed.

#### SETUP

Causes a VMSES/E VMFSETUP (LINK command to be issued as part of PRODUTL processing, to establish a correct operational environment. The *ppfname* and *compname* operands supplied for the PRODUTL command are also used as operands for the VMFSETUP command.

#### TEST

Causes processing for the current invocation to be performed such that no files are placed into production. The TEST option allows you to verify that required minidisks and SFS directories can be accessed without error, and that the appropriate catalog file entries will be processed. Additional messages are issued to clarify what actions would occur if this option were not specified.

### Usage Notes

1. By default, PRODUTL does not issue minidisk LINK commands as part of its processing. Use the SETUP option to acquire the appropriate minidisk resources if the availability of such resources cannot be determined.
2. PRODUTL uses the first CATALOG file found in the CMS search order that matches that specified by the *ctlg\_name* operand.
3. Catalog file entries that are found to be unusable are bypassed, with appropriate warning or error messages issued.
4. PRODUTL requires a minidisk or SFS directory to be accessed at file mode A with read/write (R/W) status, for use as temporary work space and for message logging.

## PRODUTL File Exclusion Support

To ensure that production minidisks contain only those files necessary to provide or use TCP/IP services, the PRODUTL command incorporates support that allows various files present on a source minidisk or directory resource to be excluded as certain "wildcard" catalog entries are processed. When this "file exclusion" support is applied, files can be excluded based on specific file names, file types, or by using conventional CMS file pattern matching techniques that employ the wildcard (\*) and pattern matching (%) characters.

Files that are to be excluded in this manner must be identified within one or more catalog file exclude sections that are separately defined for this purpose. Within such a section, one or more exclusion entries are defined that identify the specific files or file groups that are to be excluded.

PRODUTL file exclusion processing is activated by the XCLUDE entry processing option, which is specified as part of a wildcard file entry. See [“Entry Processing Options”](#) on page 254 for details about the XCLUDE option.

#### Note:

1. When exclusion processing is activated for a given entry (and thus, a specific source and target resource pairing), PRODUTL automatically excludes *unchanged* files from a copy operation in addition to those files identified within an exclude section. This avoids unnecessarily processing source files that have not been modified since such files were last placed into production.

In this context, a file is considered to be unchanged when file attributes — other than file mode number and CMS data block count — for a source file and its production counterpart are identical.



- Automatic exclusion of unchanged files is performed only when a valid XCLUDE option has been specified for a wildcard entry.

## Messages and Return Codes

Appendix D, “Module Identifiers for VMSES/E Messages,” on page 749 shows the format of VMSES/E messages and lists the identifiers used by each VMSES/E exec.

For a complete explanation of any VMSES/E message, use the HELP Facility to view the message explanation online or see the appropriate messages documentation. To display information on a specific message, for example VMF002E, enter:

```
help msg vmf002e
```

If you are unfamiliar with the z/VM HELP Facility, enter help to display the main HELP Menu. For information on the HELP command, enter:

```
help cms help
```

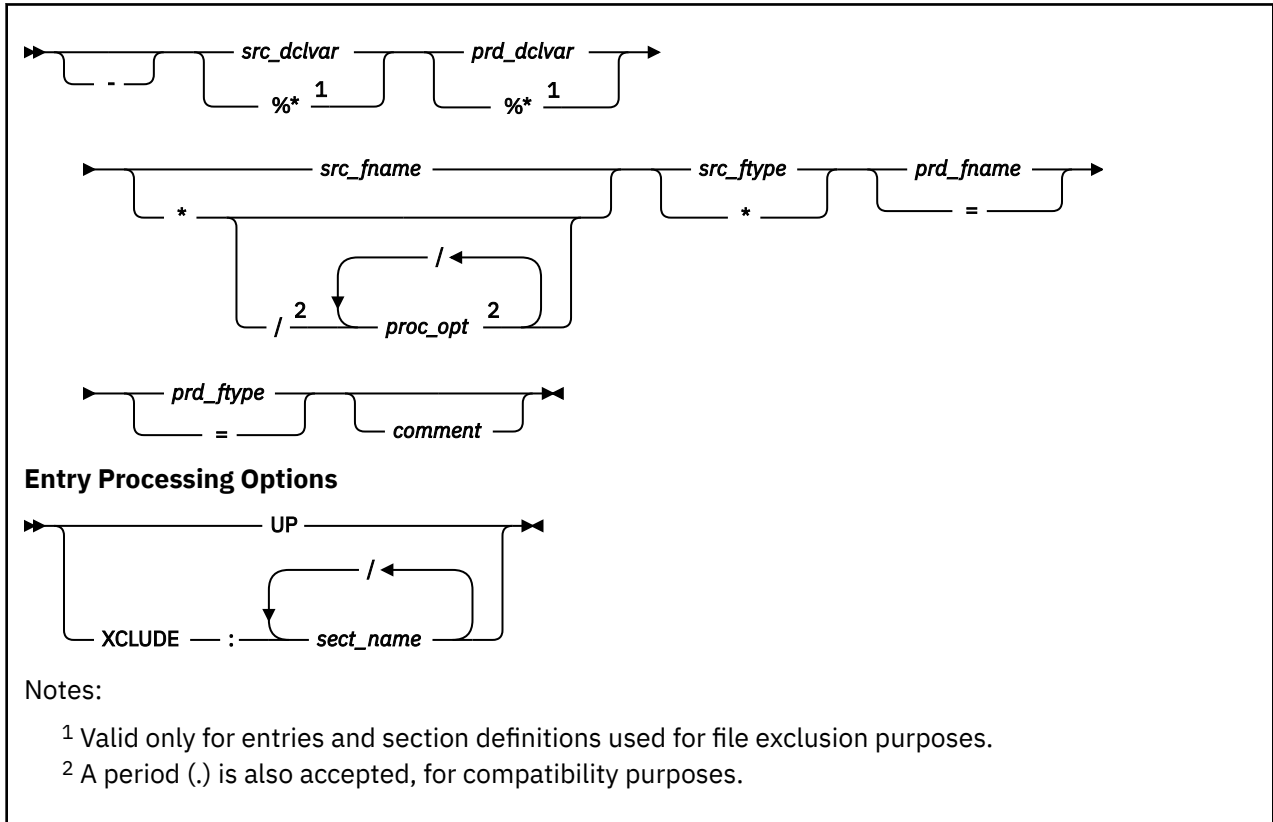
**PI**

Return Code	Explanation
0	Successful execution; no processing errors were encountered.
1	Incorrect invocation. PRODUTL was invoked with an incorrect number of operands, or a supplied operand or command option was found not to be valid. A message that identifies the problem is displayed.
2	Internal error. If this return code is produced, processing status is indeterminate. Contact the product support group for problem determination and assistance in addressing this type of error.
4	Errors encountered, with warnings issued. The errors encountered might have caused processing to complete with only partial success. Review the PRODUTL \$MSGLOG for warning messages that identify any problems that were encountered.
8	Errors encountered; processing has not completed successfully. Review the PRODUTL \$MSGLOG for messages regarding the problems encountered.

**PI end**

For more information, see “[CATALOG Files](#)” on page 252.

## CATALOG Files



### Purpose

A catalog file is used by the PRODUTL command to identify which product files are to be placed into production or compared, as well as the minidisks and SFS directories that are to be used for this process. Structure, content and customization requirements and considerations associated with the catalog file are described in the sections that follow.

### Catalog File Structure

Within the catalog file, distinct sections are defined which identify groups of related files that are to be copied from a given (source) minidisk or SFS directory resource to one or more (target) production resources, or that are to be compared across two such resources. For example, product run-time files (such MODULE and EXEC files) might be defined for processing as one group, whereas product configuration files might be defined for handling as another, separate group.

Similarly named, paired begin and end tags are used to define a given section of grouped catalog entries. For example, in the *prodid* CATALOG file supplied with TCP/IP for z/VM, the section defined for sample configuration files is delimited by the `:TCPCSAMPLE.` and `:ETCPSAMPLE.` tags.

In general, the various entries (or, records) in the catalog file provide information sufficient for PRODUTL to process the files for a given group. See [“Catalog Entry Types” on page 254](#) for more specific information.

A unique, *exclusion* entry can also be defined (within a separate catalog section) that identifies certain files or groups of files to be excluded as PRODUTL processes the conventional entries just described. For more information about these entries and their use, see [“PRODUTL File Exclusion Support” on page 250](#).

**Note:**

1. The file type of the catalog files used by the PRODUTL command must be CATALOG.

2. Section definition tags must begin with a colon (:), end with a period (.), and must be comprised of a non-blank string (intervening blanks are not permitted). Case is not significant.
3. Section tags must be present on unique lines within a catalog file — they cannot be combined with file data entries. Tags must also be properly paired (that is, no attempt is made to detect a missing end tag for a given begin tag).
4. File exclusion entries must be defined separately from conventional catalog file entries, in sections defined specifically for this purpose.

## Operands

-

The entry bypass character, a hyphen (-). The presence of this character at the beginning of a file entry signifies that PRODUTL should *not* process such an entry.

**Note:** It is suggested that *all* entries within a given section be maintained within the IBM-supplied catalog files. Doing so allows PRODUTL to report such bypassed entries when files are processed, and also allows file entries to more readily be distinguished from comments as a catalog file is modified over time.

If a given file is not required for use by your installation, its corresponding entry should be bypassed as just described, rather than deleted or changed to a comment.

### ***src\_dclvar***

A PPF :DCL. variable name for the (source) minidisk or SFS directory resource where a source file resides. For file exclusion entries, a wildcard value (%\*), that allows an entry to be matched to a given source resource when files are processed, can be specified .

### ***prd\_dclvar***

A PPF :DCL. variable name for the (target) minidisk or SFS directory resource where a copied production file is to reside. For file exclusion entries, a wildcard value (%\*) is accepted, although this operand serves only as a positional place holder (that is, the specification of a wildcard for this operand has no effect on file exclusion processing).

### ***src\_fname***

The file name for a given source file. An asterisk (\*) can be specified as wildcard value to signify that all files of the type specified by *src\_ftype* are to be processed. When a wildcard (\*) is used, the production file name remains unchanged from the source file name.

### ***proc\_opt***

An entry-specific processing option. Such options affect how the PRODUTL command processes (copies) files that are associated with the designated entry. Entry processing options are valid only for wildcard (\*) source file names, and are delimited by a slash (/), with no intervening spaces. Processing options recognized by PRODUTL are further explained in [“Entry Processing Options” on page 254](#).

### ***src\_ftype***

The file type for a given *source* file. An asterisk (\*) can be specified as wildcard value to signify that all files of the type specified by *src\_fname* are to be processed. When a wildcard (\*) is used, the production file type remains unchanged from the source file type.

### ***prd\_fname***

The file name for a given target production file. If the *source* file name is specified as a wildcard (\*), *prd\_fname* must still be specified to maintain correct entry format. For clarity, it is suggested that an equal sign (=) be specified for *prd\_fname* in such a case (though any specified value is processed as if '=' had been specified).

### ***prd\_ftype***

The file type for a given target production file. If the source file type is specified as a wildcard (\*), *prd\_ftype* must still be specified to maintain correct entry format. For clarity, it is suggested that an equal sign (=) be specified for *prd\_ftype* in such a case (though any specified value is processed as if '=' had been specified).

### ***comment***

Commentary text that is ignored by PRODUTL during processing.

**Note:**

1. All operands must be separated by at least one space.
2. Comment lines within a catalog file must begin with an asterisk (\*). Such lines are ignored during PRODUTL processing.
3. Literal resource values (such as a minidisk device address or an SFS directory name) are not accepted in place of the *src\_dclvar* or *prd\_dclvar* operands.
4. :DCL. wildcard values (%\*) are unique to the PRODUTL catalog file and are *not* supported (or present) within a VMSES/E PPF file. These values should be used only to define a file exclusion entry that can be referenced during a wildcard file copy operation.
5. For file exclusion entries, the *prd\_fname* and *prd\_ftype* operands (if specified) are treated as commentary information.

**Entry Processing Options**

Options that can be specified as part of a catalog file entry to affect PRODUTL copy processing are:

**UP**

Instructs PRODUTL to use the CMS COPYFILE UPCASE option when files are copied to production resources.

**XCLUDE:sect\_name**

Instructs PRODUTL to exclude one or more files when processing is performed for a wildcard catalog entry. The files to be excluded must be listed in a separate catalog file section (identified by *sect\_name*) that is specifically defined for this purpose.

Multiple section names can be specified for a given XCLUDE option (with each name separated by a colon), from which a cumulative exclusion list is generated. See [“PRODUTL File Exclusion Support”](#) on page 250 for more information about PRODUTL file exclusion support.

**Note:**

1. The XCLUDE option can be specified for only a wildcard entry within a general catalog section, and for only the source file name of such an entry. This restriction stems from the presumption that any files that are to be excluded from PRODUTL operations are a subset of a substantially larger file group – that is, a group that is more readily processed through file name and file type wildcard (\*) pattern matching, than on a file-by-file basis.
2. When multiple entry options are specified, do not include intervening blanks between operands or delimiters.
3. To maintain compatibility with prior option processing support, a period (.) is also accepted as an option delimiter. However, mixed use of this alternate value and the preferred delimiter (/) is not supported.

**Catalog Entry Types**

The different types of catalog entries that can be defined within a catalog file are described here:

Entry Type	Description
<p><b>Conventional</b></p>	<p>A conventional (or, general) entry is one that identifies an individual file that is to be processed without special consideration. Such an entry can be used in any catalog definition section. An example of such an entry is:</p> <pre data-bbox="506 1738 1468 1793" style="background-color: #f0f0f0; padding: 5px;"> &amp;BLD1Z  &amp;BLD0Z  PROFILE  STCPIP  =  =  *A comment...</pre>

Entry Type	Description
<p><b>Wildcard</b></p>	<p>A wildcard catalog entry is similar to a conventional entry, but is one in which the source file name or source file type (or both) is specified as an asterisk (*). For such an entry, all of the files to which this wildcard pattern is matched are processed. An example of such an entry is:</p> <pre data-bbox="508 327 1463 394"> &amp;BLD1Z  &amp;BLD0Z  *          MODULE  = =  Commentary text                     </pre>
<p><b>Exclusion</b></p>	<p>An exclusion catalog entry is somewhat different from the previous types and is used to exclude one or more files from being processed.</p>

## Conventional Catalog Definitions

While the entries described in the previous section identify specific files or file groups to be processed by PRODUTL, they do not convey how those files are to be processed. The manner in which files are processed is controlled to a large extent by the name associated with a catalog definition (specifically, by its delimiting begin tag). The connotation of a section name, and its effect on PRODUTL processing is further explained here.

### Configuration, Notify and Sample Definitions

A section defined by a begin tag that ends with the strings **config** (meaning configuration), **notify** or **sample** must contain only conventional, non-wildcard entries. This requirement exists because PRODUTL expects to process the entries within such sections (and the files they identify) on an individual basis.

The processing of such files on an individual basis is done to allow for a proper comparison of a given source file and its production counterpart. When differences between two given files are detected, PRODUTL then can provide appropriate notification and process the subject source file accordingly.

However, it is important to note that when files listed in a **notify** or **sample** definition section are processed, only file comparisons are performed for a subject file pairing, with any detected differences reported. No attempt is made to copy a given file and replace its counterpart. The copying of such files is presumed to be handled through other means (such as the PUT2PROD command), and that notification of changes to those files (resulting from the application of service) is the rationale for processing via PRODUTL.

While PRODUTL can be instructed to copy files listed in a **sample** section to designated resources (through use of a FORCECOPY or REPLACE command option), it cannot be used to copy files listed in a **notify** section. Such files are presumed to be grouped and listed for change notification purposes only, so they cannot be copied using PRODUTL regardless of whether the aforementioned command options are used.

By contrast, when file differences are detected as entries in a **config** (configuration) definition section are processed, the default action is to *not replace* the production instance of a given file. This is done to avoid the overlay of a (presumably) customized configuration file. When differences are detected, PRODUTL provides notification to this affect, which includes readily available attribute information for the subject files.

### General Catalog Definitions

A catalog section that contains conventional entries, but is neither a sample or configuration section, is considered to be a **general** catalog section. **General** catalog definitions can include a mixture of conventional or wildcard entries, for which (by default), no special actions are performed as files are copied from their respective source locations to designated production resources.

However, it is with such general definitions that PRODUTL file exclusion support can be used. The definitions and entries required to exploit this support are described in the next section.

## File Exclusion Definitions

The entries described in this section are used to exclude one or more source files from being copied to a production resource. Such entries are referred to as file exclusion entries, with the definition section that incorporates them known as a file exclude section.

Because file exclusion support is source file oriented, a production file name and file type are not required for, or applicable to, exclusion entries. This stems from the intended use of these entries, for which identification of only one or a group of source files is required. For this reason, production file name and file type operands are treated as commentary information when they're included as part of an exclusion entry.

A second attribute that distinguishes exclusion entries from their conventional counterparts is the accommodation of a ":DCL. wildcard" value that is unique to the PRODUTL catalog file. This wildcard (%\*) can be specified in the place of a PPF :DCL. variable name so an exclusion entry can be matched to the source :DCL. variable name of a conventional catalog entry. When such an entry is processed (and, an XCLUDE option is present), the files identified by the matched exclusion entry are omitted from the set of files that are copied to the designated production location. Thus, the ":DCL. wildcard" allows exclusion entries to be defined such that select files can be excluded from PRODUTL operations, regardless of the (source) resource on which they reside.

Note that the use of a :DCL. wildcard is not required for an exclusion entry. A PPF :DCL. variable name can still be specified to identify a specific source resource from which designated files are to be excluded.

The files that are to be excluded by an exclusion entry can be identified by a literal file name and file type, or by using conventional CMS file pattern matching techniques that employ the wildcard (\*) and pattern matching (%). Combinations of a literal file name and a file type "pattern" (and vice versa) can also be used.

In the example that follows, a wildcard entry and a separately defined exclude section are illustrated which, in combination, specify that all files for the given &BLD1Z source resource — except those identified in the XTEST exclude section — are to be copied to the &BLD0Z production resource.

```

...
&BLD1Z  &BLD0Z  */XCLUDE:xtest  *  =  =  *No XTEST files
...
:DCL.
%*      %*      *      SAMP*      ** Do not copy SAMP* variations
%*      %*      *      SEXEC      ** Do not copy any SEXEC files
:EXTEST.
...

```

File exclusion is performed by PRODUTL *only* when the XCLUDE processing option is specified for a "conventional" catalog entry. This option also identifies the catalog section which defines the exclusion entries that are to be applied during file processing.

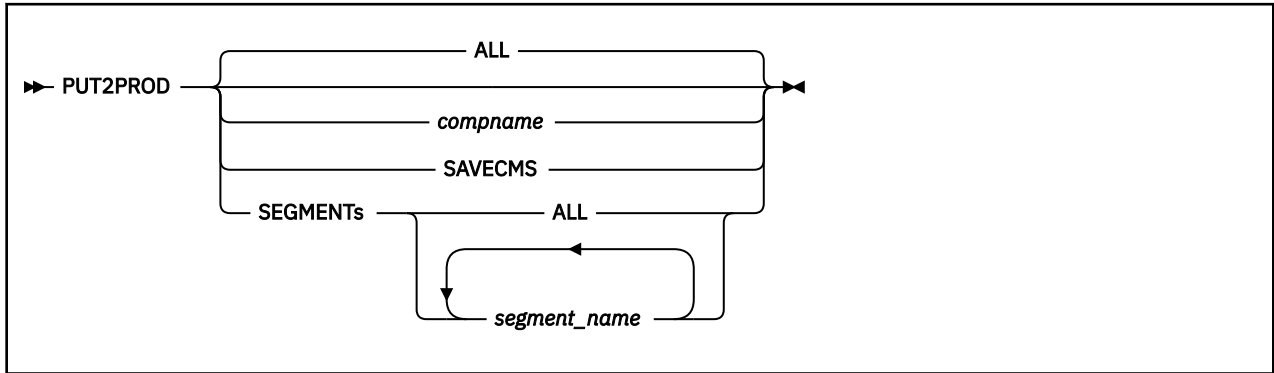
### Note:

1. The XCLUDE option can be specified for only a wildcard entry within a general catalog section, and for only the source file name of such an entry. This restriction stems from the presumption that any files that are to be excluded from PRODUTL operations are a subset of a substantially larger file group — that is, a group that is more readily processed through file name and file type wildcard (\*) pattern matching, than on a file-by-file basis.
2. To be effective, file exclusion entries must be defined using catalog definitions that are separate from conventional file processing entries. Exclusion entries that are encountered outside of a file exclude section are ignored by PRODUTL, if they can be discerned as such, as would be the case for those that employ :DCL. wildcard values. An exclusion entry that does not incorporate such values might be identified as being not valid for other reasons, or in some cases, can be construed as a conventional catalog entry.

## Customization Notes

1. It is advised that any changes to a z/VM product, component or feature CATALOG file that are required for your environment be made via a VMSES/E local modification, to allow for the reporting of service-related changes during VMSES/E processing.
2. The source and target minidisk/directory variable names used within a CATALOG file must correspond to those used within the (\$)PPF file associated with a given z/VM product, component or feature (or an override variation of that file). If any changes are made to the Variable Declarations (:DCL.) section of the subject PPF file via a PPF override, you might need to incorporate similar changes within any pertinent CATALOG files (through separate VMSES/E local modifications) to allow for the correct resolution of PPF :DCL. variable names.
3. :DCL. wildcard values (%\*) are unique to the PRODUTL catalog file and are *not* supported (or present) within a VMSES/E PPF file. These values should be used only to define a file exclusion entry that is to be referenced during a wildcard file copy operation.

## PUT2PROD EXEC



### Purpose

The PUT2PROD EXEC performs the actions listed in the *systemid* \$PRODS file for the z/VM components, features, or products that are installed on the z/VM system.

### Operands

#### ALL

specifies that products serviced using the SERVICE command will be put into production.

#### *compname*

is the 1- through 16-character alphanumeric component name identifier for the component to be put into production. Specify the component name as it is on the :PRODID tag in the VM SYSSUF table.

#### SAVECMS

specifies that the CMS saved systems will be built and put into production.

#### SEGMENTS

specifies which segment or segments will be put into production. If SEGMENTS is specified, it must be followed by ALL or by one or more segment names.

#### ALL

when specified with the SEGMENTS option, all segments defined in the system segment build list (SEGBLIST EXC00000) will be built and put into production.

#### *segment\_name*

specifies one or more segment names that will be put into production.

### Usage Notes

1. The PUT2PROD EXEC will affect your production environment. The following production environment virtual machines will be logged off in order to update minidisks or SFS directories they own:
  - DIRMAINT
  - TCPMAINT
2. The PUT2PROD EXEC must be executed from the default MAINTvrm user ID or equivalent.
3. The PUT2PROD EXEC processes components, features, and products that provide a :P2P. section in their PPF.
4. A BLDCMS user ID and a BLDSEG user ID must exist.
5. When the PUT2PROD EXEC is used to build CMS saved systems (whether overtly with the SAVECMS operand or in conjunction with product service processing), the SSI relocation domain for the MAINTvrm user ID is modified, when necessary, to that of the current member system. The updated relocation domain remains in effect upon completion of the PUT2PROD EXEC.



## Examples

- To put all components, features, and products that were serviced (have an entry in the SERVICE \$PRODS file) into production enter:

```
PUT2PROD
```

or

```
PUT2PROD ALL
```

- To put only TCPIP into production, enter:

```
PUT2PROD TCPIP
```

- To put only the CMS saved systems into production, enter:

```
PUT2PROD SAVECMS
```

- To put only the XXXX and YYYY segments into production, enter:

```
PUT2PROD SEGMENTS XXXX YYYY
```

## Input and Output Files

### Input Files

#### ***ppfname* PPF**

The usable form product parameter file.

#### **VM SYSSUF**

The system-level service update facility table.

### Input/Output Files

#### **VM SYSPINV**

The system-level Product Inventory table, which specifies which products are installed on which systems or members.

#### ***systemid* \$PRODS**

The put into production file that is a list of products and associated objects that were serviced for the system.

#### **SERVICE \$PRODS A**

The put into production file that is a list of products that were serviced using the SERVICE EXEC.

#### **PUT2PROD \$CONS A**

The console listing for the current invocation.

#### **\$VMFP2P \$MSGLOG A**

The PUT2PROD message log.

### Output Files

#### ***prodid* SRVPROD**

The service-level production status table.

#### **VM SYSPINV**

The service-level product inventory table.

## Messages and Return Codes

For a complete explanation of each message, use the HELP Facility to view the message explanation online or see the appropriate messages documentation. To display information on a specific message, for example VMF002E, enter:

```
help msg vmf002e
```

If you are unfamiliar with the z/VM HELP Facility, enter `help` to display the main HELP Menu. For information on the HELP command, enter:

```
help cms help
```

Appendix D, “[Module Identifiers for VMSES/E Messages](#),” on page 749 shows the format of VMSES/E messages and lists the identifiers for each VMSES/E exec.

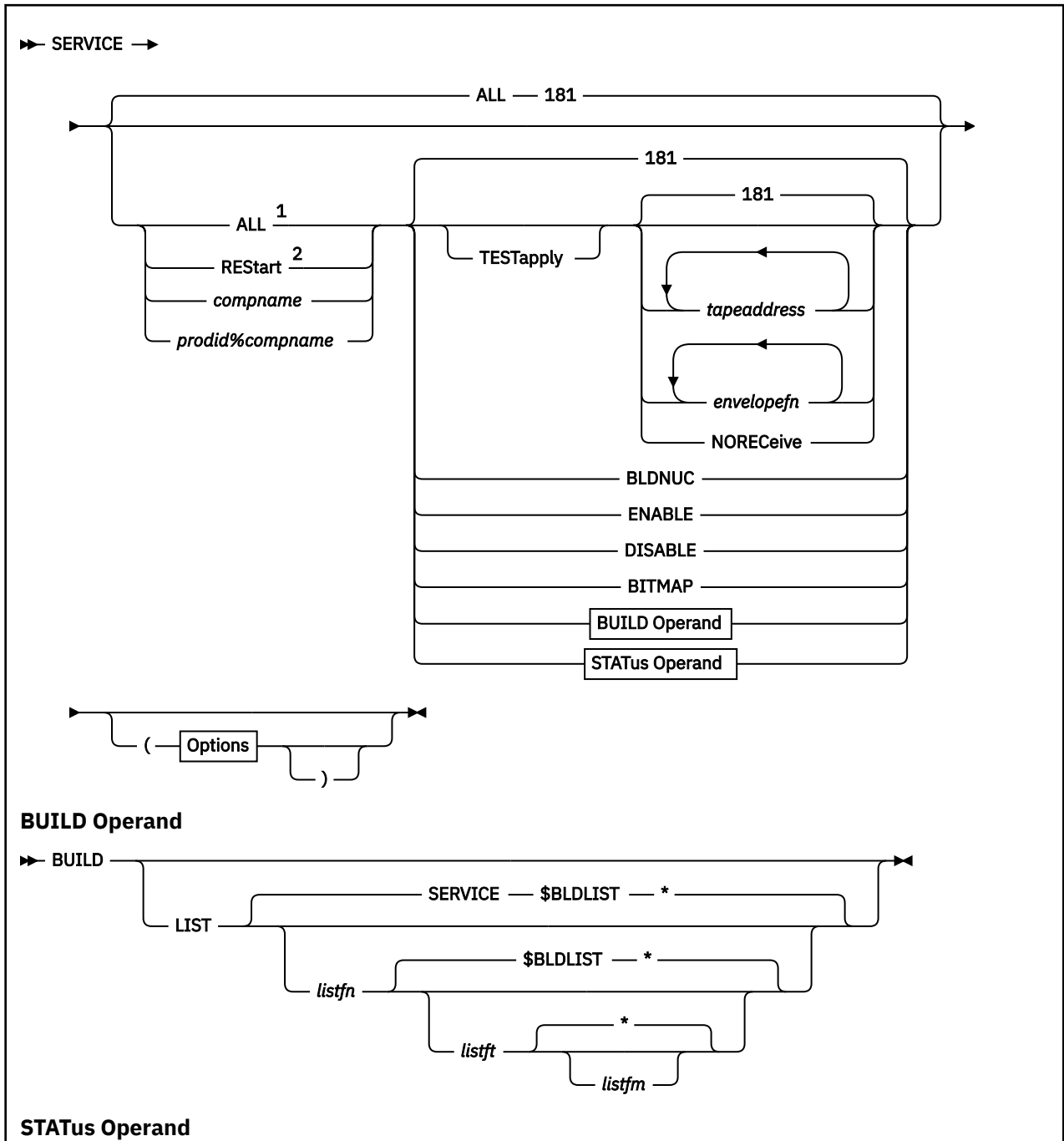
PUT2PROD issues the following return codes:

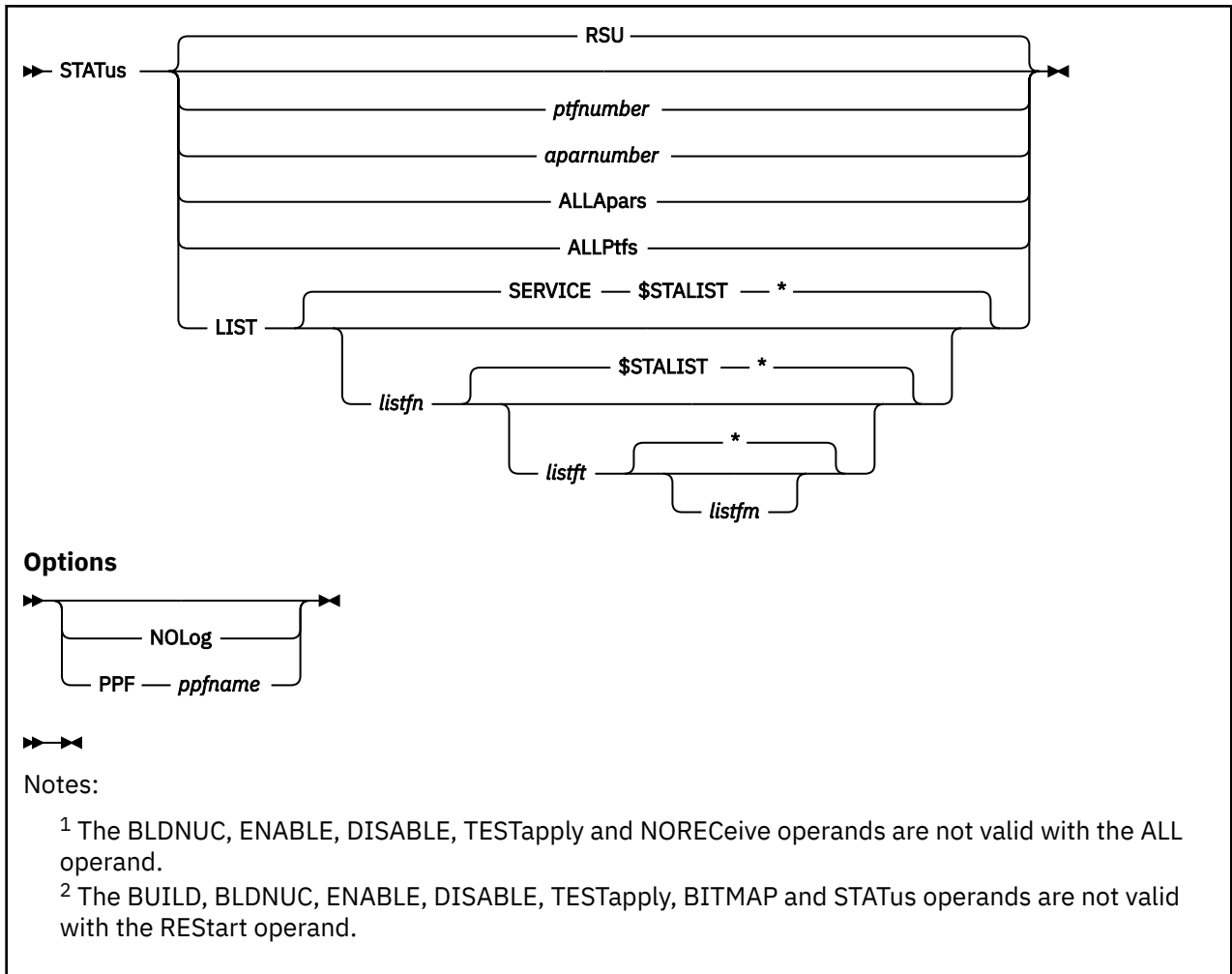
<b>Return Code</b>	<b>Explanation</b>
0	Command completed successfully.
2	Command completed successfully but extraneous data was encountered.
4	Command completed with one or more warning conditions.
12	Command failed because of an internal error.
24	Command failed because of a command line syntax error.
28	Command failed because a required file was not found.
36	Command failed because a target disk or directory was not available.
100	Command failed because of an external error.

## **Recovery Information**

The PUT2PROD EXEC can be restarted by issuing the command again.

# SERVICE EXEC





## Purpose

Use the SERVICE EXEC to:

- Install an RSU or apply corrective service (COR) for z/VM components, features, or products.
- Display the RSU level of the component specified or whether a particular PTF or APAR has been applied (when used with STATUS).
- Create PTF bitmap files (when used with BITMAP).

## Operands

### ALL

indicates that service is to be installed for all products on the selected RSU or COR. When used with STATUS or BITMAP, ALL indicates all products in the VM SYSSUF table.

### REStart

restarts a previous call to SERVICE.

### *compname*

is the component for which service will be installed or displayed or the component for which a bitmap is created. *compname* is a 1- through 16-character alphanumeric identifier.

### *prodid%compname*

is the component for which service will be installed or displayed, or the component for which a bitmap is created. Specify *prodid%compname* as it is on the :PRODID tag in the VM SYSSUF table. If using this form of the parameter, the *prodid*, the "%", and the *compname* are all required.

**TESTapply**

indicates that VMFAPPLY will be called with the TEST option and the build step will be skipped. The TESTapply operand is valid with COR service (tape or envelope) and not valid with RSU service (tape or envelope).

**tapeaddress**

is the address of the tape drive where the preventive (RSU) or corrective (COR) service tape is mounted. The default is 181. Multiple *tapeaddress* operands can be specified if the RSU or COR tape is multivolume (for example, VOL01 of 02 and VOL02 of 02).

**envelopefn**

is the file name of the preventive (RSU) or corrective (COR) service envelope to be installed. The file type must be SERVLINK. Multiple *envelopefn* operands can be specified if the RSU or COR envelope is multivolume (for example, VOL01 of 02 and VOL02 of 02).

**NOREceive**

indicates the receive step is skipped. Service is applied and built.

**BLDNUC**

indicates that if the specified component has a nucleus build step, that nucleus will be flagged to be built and all steps associated with the BUILD operand will be executed.

**ENABLE**

indicates the specified component is to be enabled.

**DISABLE**

indicates the specified component is to be disabled.

**BITMAP**

indicates PTF bitmap files are to be created.

**BUILD**

indicates service is to be built for the specified component, all components in the SYSSUF table if ALL is specified, or a list of components if ALL and LIST are specified. RSU, COR, or local service which has been applied but not yet built, will be built.

**LIST**

indicates a user-supplied list of components will be used. Each component in the list will be built.

If the ALL operand is not specified, the LIST operand will be ignored and the list not used.

The input file format should consist of one component per line. Data after the first word on a line is considered to be commentary. Any line that starts with an asterisk (\*) is a comment.

**listfn**

is the file name of the input list. The default file name is SERVICE.

**listft**

is the file type of the input list. The default file type is \$BLDLIST.

**listfm**

is the file mode of the input list. The default file mode is an asterisk (\*).

**STATus**

displays service status.

**RSU**

displays the service level specified on the :SERVLEV tag in the system-level service update facility table (VM SYSSUF). For example, RSU-0301, which indicates RSU was applied, or 303-0303, which indicates SDO installed level. This is the default.

**ptfnumber**

is the PTF number for which status is displayed. The PTF number has the format of two alphabetic characters followed by five numeric characters.

**aparnumber**

is the APAR number for which status is displayed. The APAR number has the format of two alphabetic characters followed by five numeric characters.

### **ALLApars**

displays status for all APARs that have been received and applied for the subject product.

### **ALLPtfs**

displays status for all PTFs that have been received and applied for the subject product.

### **LIST**

indicates a user-supplied list of PTFs or APARs will be used. The status will be displayed for each PTF or APAR in the list.

The input file format should consist of one PTF or APAR number per line. Data after the first word on a line is considered to be commentary. Any line that starts with an asterisk (\*) is a comment.

#### ***listfn***

is the file name of the input list. The default file name is SERVICE.

#### ***listft***

is the file type of the input list. The default file type is \$STALIST.

#### ***listfm***

is the file mode of the input list. The default file mode is an asterisk (\*).

## Options

### **NOLog**

indicates that the majority of messages produced by the SERVICE EXEC are not to be logged. This option is supported with the BITMAP and STATUS operands only. Messages that are not logged still are displayed at the console (as are messages which continue to be logged, such as VMF2760I command status messages).

### **PPF**

identifies a product parameter file that is to be referenced when certain SERVICE command operations are performed.

The PPF name tells the SERVICE command that this is a z/VM CSM environment and the SERVICE EXEC will use that PPF file instead.

#### ***ppfname***

is the file name of a usable form product parameter (or override) file. The specified PPF file will be used instead of the SERVP2P PPF file that is commonly used for SERVICE operations (and which is cited for various VM SYSSUF file PPF-related tags).

## Usage Notes

1. The SERVICE EXEC must be executed from the default MAINTvrm user ID or equivalent.
2. BLDNUC and BLDRACF user IDs must exist.
3. If a single tape address is specified, that address will be used for all volumes of a multivolume RSU.
4. The same tape address must be used for all volumes of a multivolume COR tape.
5. When the ENABLE operand is specified, the following occurs for the specified component:
  - A VMFINS ENABLE command is issued.
  - The SYSTEM CONFIG file is updated.
  - The :BUILD tag in the VM SYSSUF table is set to YES.
  - A VMFSUFIN command is issued with the BUILD option.
6. When the DISABLE operand is specified, the following occurs for the specified component:
  - A VMFINS DISABLE command is issued.
  - The SYSTEM CONFIG file is updated.
  - The :BUILD tag in the VM SYSSUF table is set to NO.

7. If the DISABLE operand is specified for the RACF component, the CP nucleus will be rebuilt with the RACF modules removed.
8. SERVICE STATUS displays the production status for all systems or members, regardless of the system or member on which you execute the command.
9. In the SERVICE STATUS display for a PTF or an APAR, for PTFs or APARs that were installed from an RSU, the RECEIVED date reflects the date the RSU was installed on your system and the APPLIED date reflects the date that the PTF/APAR was applied during IBM's RSU build processing.
10. The information produced by the SERVICE STATUS command when specified with the ALLAPARS operand or the ALLPTFS operand can be voluminous, even when a relatively small number of PTFS and APARs have been applied for a given product.

Note that by default, all such output is logged to the SERVICE command message log file. Thus, issuing repeated SERVICE STATUS commands with these options can cause the log file to substantially increase in size.

For this reason, consider specifying the NOLOG option when using the ALLAPARS operand or the ALLPTFS operand. When doing so, it is advised that one makes sure the console output is spooled properly (to the virtual reader, for example) or that command console output is directed to a file using the appropriate CMS pipeline command.

### Examples

- To install service to all of the products on the RSU mounted on tape drive 181 enter:

```
SERVICE
```

or

```
SERVICE ALL
```

- To check prerequisites before installing the corrective (COR) service mounted on tape drive 181, enter:

```
SERVICE CP TESTAPPLY
```

If all requisites are satisfied, install the service by entering:

```
SERVICE CP NORECEIVE
```

- To install service to all of the products on a three-volume RSU using three tape drives, enter:

```
SERVICE ALL 181 182 183
```

- To install service to CMS only, from the third tape of a stacked RSU, enter:

```
SERVICE CMS 183
```

- To install service to TCPIP using the COR envelope file UI54321 SERVLINK, enter:

```
SERVICE TCPIP UI54321
```

- To install service to all of the products on a two-volume RSU envelope, enter:

```
SERVICE ALL RSUENV1 RSUENV2
```

- To restart installing service after reworking local modifications, enter:

```
SERVICE RESTART
```

- To build all components in the SYSSUF table, enter:

## SERVICE EXEC

```
SERVICE ALL BUILD
```

- To build all components in the list input file, enter:

```
SERVICE ALL BUILD LIST
```

If a component name is specified, the LIST operand is ignored:

```
SERVICE compname BUILD LIST
```

- To display the RSU level of CP, enter:

```
SERVICE CP STAT RSU
```

- To see if PTF UM12345 has been applied, built or put into production for CMS, enter:

```
SERVICE CMS STAT UM12345
```

- To see if APAR VM56789 has been applied, built or put into production for an installed product, enter:

```
SERVICE ALL STAT VM56789
```

### Input and Output Files

SERVICE calls the VMFSUFIN exec, which in turn calls the VMFSETUP, VMFMRDSK, VMFINS, VMFPSU, VMFAPPLY, VMFREC, and VMFBLD execs, therefore, SERVICE uses all of the input and output files for these EXECs. The input and output files unique to SERVICE are:

#### Input Files

##### ***ppfname* PPF**

The usable form product parameter file

##### ***appid* SRVAPPS**

The service-level apply status table

##### ***recid* SRVREQT**

The service-level requisite table

##### ***recid* SRVRECS**

The service-level receive status table

##### **fn SERVLINK**

RSU and COR envelopes.

##### **SERVICE \$PTFS**

Applied PTFs by component.

#### Input/Output Files

##### **SERVICE \$RESTART**

The service restart file.

##### **SERVICE \$CONS**

The console listing for the current invocation.

##### **SERVICE \$PRODS**

The put into production file that contains the list of products and objects that were serviced using the SERVICE EXEC. For more information, see [“The SERVICE \\$PRODS File ” on page 135.](#)

##### **VM SYSPINV**

The system-level Product Inventory table, which specifies which products are installed on which systems or members.

##### **VM SYSSUF**

The system-level service update facility table.

##### **VM SYSREST**

The system-level Restart table.



**VM SYSLMOD**

The system-level Local Modification table.

**VM SYSTEMEMO**

The system-level Memo table.

**Output Files****prodid SRVPROD**

The service-level production status table.

**systemid \$PRODS**

The put into production file, for a specific system, that is a list of products and objects that were serviced. For more information, see [“The systemid \\$PRODS file” on page 137.](#)

**\$VMFSRV \$MSGLOG A**

The SERVICE message log.

**INSTSERV \$RESTRT\$**

The service restart command file for automated installation use.

**Messages and Return Codes**

For a complete explanation of each message, use the z/VM help facility to view the message explanation online or see the appropriate message documentation. To display information about a specific message (VMF002E, for example), enter:

```
help msg vmf002e
```

To display the main z/VM help menu, enter:

```
help
```

For information about the HELP command, enter:

```
help cms help
```

Appendix D, [“Module Identifiers for VMSES/E Messages,”](#) on page 749 shows the format of VMSES/E messages and lists the identifiers for each VMSES/E EXEC.

The SERVICE EXEC issues the following return codes:

Return Code	Explanation
0	Command completed successfully.
2	Command completed successfully but extraneous data was encountered.
4	Command completed with one or more warning conditions.
5	Command not complete because service that affects core VMSES/E components has been identified.
6	Command not completed because local modifications were found.
12	Command failed because of an internal error.
24	Command failed because of a command line syntax error.
28	Command failed because a required file was not found.
36	Command failed because a target disk or directory was not available.
100	Command failed because of an external error.

### Recovery Information

If the SERVICE EXEC does not complete successfully, it can be restarted by reissuing the command after correcting the errors or reworking the local modifications. The format of the command is dependent on whether a SERVICE \$RESTART file was created. Message VMFSRV2310W will be issued if the restart file was created.

If the restart file was *not* created, reissue the original SERVICE command.

If the restart file was created, issue the SERVICE RESTART command displayed in message VMFSRV2310W.

## SERVMGR EXEC

---

Use the SERVMGR EXEC to handle all tasks associated with z/VM Centralized Service Management (z/VM CSM). Based on the keywords and operands supplied with it, SERVMGR calls applicable sub-function EXECs to perform specific types of processing.

For more information, see:

- [“SERVMGR INITIALIZE” on page 270](#)
- [“SERVMGR SYSTEM” on page 272](#)
- [“SERVMGR SRVLVL” on page 279](#)
- [“SERVMGR REMOVE” on page 285](#)
- [“SERVMGR MANAGED” on page 288](#)

**Note:** All SERVMGR operands and options *must* be specified in the order shown in the SERVMGR syntax diagrams.

## SERVMGR INITIALIZE

**Note:** SERVMGR INITIALIZE operands *must* be specified in the order shown in the SERVMGR INITIALIZE syntax diagram.

```
►► SERVMGR — INITialize — vrm ◄◄
```

### Purpose

Use the SERVMGR INITIALIZE command to create the initial structures for z/VM Centralized Service Management (z/VM CSM).

### Operands

#### INITialize

Creates the initial structures for z/VM CSM. The SERVMGR INITIALIZE command creates initial tables and creates and populates *vrm*-level Shared File System directories.

#### *vrm*

Represents the specific version/release/modification level of information for which z/VM CSM is to be initialized.

### Usage Notes

1. During SERVMGR command processing, the MAINTCSM 191 disk is accessed at file mode Z, with this access file mode maintained upon command completion. It is advised that files that are customized for using CSM (such as a MAINTCSM NAMES file) or for productivity (a PROFILE XEDIT file) be maintained on the MAINTCSM 191 disk, so these are available for reference, when needed.

### Example

1. To initialize z/VM CSM on your z/VM 730 system, enter:

```
SERVMGR INIT 730
```

### Input/Output Files

The input and output files unique to SERVMGR are:

#### Input/Output Files

##### **\$CSMAGT \$MSGLOG**

CSMAGENT message log.

##### **\$CSMCMG \$MSGLOG**

SERVMGR message log.

##### **CSM SVCSTAT**

z/VM CSM service status table.

##### **CSM SYSSTAT**

z/VM CSM system status table.

## Messages and Return Codes

For a complete explanation of each message, use the z/VM help facility to view the message explanation online or see the appropriate message documentation. To display information about a specific message (VMF002E, for example), enter:

```
help msg vmf002e
```

To display the main z/VM help menu, enter:

```
help
```

For information about the HELP command, enter:

```
help cms help
```

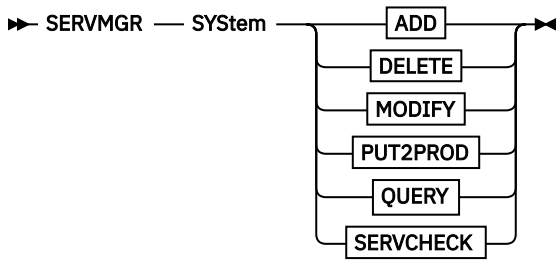
Appendix D, “Module Identifiers for VMSES/E Messages,” on page 749 shows the format of VMSES/E messages and lists the identifiers for each VMSES/E EXEC.

The SERVMGR EXEC issues the following return codes:

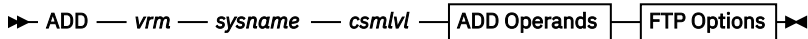
Return Code	Explanation
0	Command processing completed successfully.
4	Command processing completed with warnings.
5	Command processing stops.
6	Command processing stops.
8	Command processing stops.
9	Command processing stops.
12	Command processing stops.
13	Command processing stops.
21	Command processing stops.
24	Command processing stops.
28	Command processing continues or stops.
32	Command processing stops.
33	Command processing stops.
34	Command processing stops.
35	Command processing stops.
36	Command processing stops.
40	Command processing stops.
97	Command processing stops.
98	Command processing stops.
99	Command processing stops.
100	Command processing stops.
4113	Command processing stops.

## SERV MGR SYSTEM

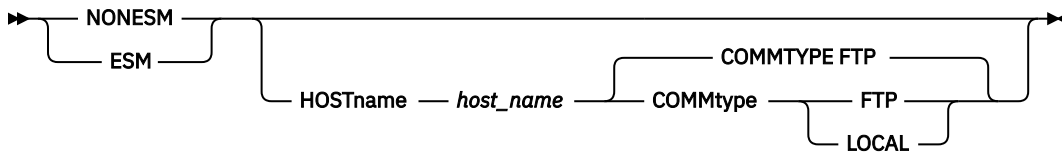
**Note:** SERVMGR SYSTEM operands *must* be specified in the order shown in the SERVMGR SYSTEM syntax diagram.



### ADD



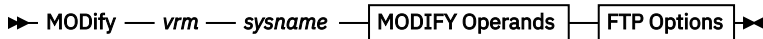
### ADD Operands



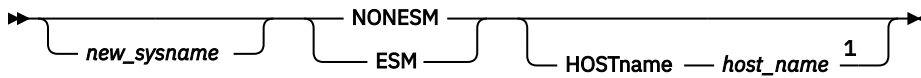
### DELETE



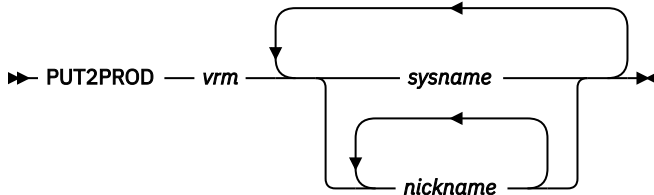
### MODIFY



### MODIFY Operands

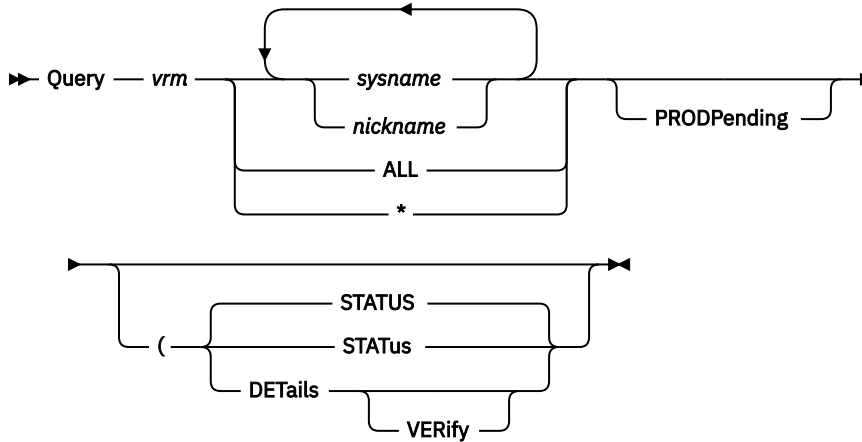


### PUT2PROD



Notes:

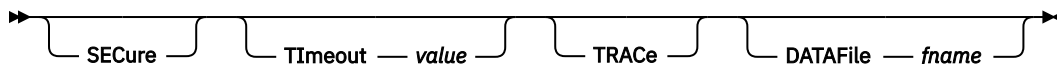
<sup>1</sup> The HOSTNAME operand is optional when supplied with a SYSTEM MODIFY command.

**QUERY****SERVCHECK**

➤ SERvcheck — *vrm* — *sysname* ➤

**FTP Options**

➤<sup>1</sup> ( Group 1 Group 2 ) ➤

**Group 1****Group 2**

➤➤

Notes:

<sup>1</sup> FTP options are valid with only the SYSTEM ADD and SYSTEM MODIFY commands.

**Purpose**

Use the SERVMGR SYSTEM command to handle all system tasks associated with z/VM Centralized Service Management (z/VM CSM).

**Operands****SYSTEM**

Performs functions related to maintaining a z/VM CSM system group. The SERVMGR SYSTEM command adds information about a selected *vrm*-level system to the z/VM CSM system status table of a pertinent principal system. The SERVMGR SYSTEM command is also used to maintain this information - to modify and query information, and if necessary, to delete a managed system.

***vrm***

Represents the specific version, release, and modification level of information that z/VM CSM is to reference, in relation to a given command. The specified *vrm* level determines which z/VM CSM system status table data and z/VM CSM service status table data is referenced as a z/VM CSM command is processed.

***sysname***

The 1- through 8-character system name (reported by the CP QUERY USERID command on the subject system) for the z/VM system that is to be serviced using z/VM CSM support.

***nickname***

A name assigned to a group of one or more z/VM systems that are to be serviced using z/VM CSM.

***csmlvl***

A 1- through 16-character identifier for a z/VM CSM service level. The *csmlvl* name must comply with naming conventions for SFS directories. For example, no dashes are allowed.

**ADD**

Indicates that a new *sysname* entry is to be created in the z/VM CSM group.

**NONESM**

Indicates that the system will not incorporate ESM controls and support.

**ESM**

Indicates that the system is to incorporate ESM controls and support.

**HOSTname *host\_name***

Identifies a TCP/IP fully-qualified internet host name or TCP/IP IP address (IPv4 dotted-decimal format or IPv6 colon-hexadecimal format).

**COMMtype *comm\_type***

Designates the type of communication protocol that is to be used for host-to-host data transfer and communication:

**FTP**

Indicates that the TCP/IP FTP protocol is to be used. This is the default.

**LOCAL**

Indicates that the system being added is the principal system (the communication protocol is not applicable).

**DELETE**

Indicates that the specified *sysname* is to be removed from the control of z/VM CSM. All information for the subject system is removed from z/VM CSM tables.

**MODify**

Indicates that changes are to be made to the FTP parameters for *sysname* or that *sysname* is to be renamed to *new\_sysname*.

**PUT2PROD**

Indicates that the VMSES/E PUT2PROD command is to be run on *sysname* or *nickname*.

**Query**

Indicates that a SYSTEM QUERY function is to be performed against the z/VM CSM system status table, to provide information about the systems defined in this table.

**ALL | \***

Indicates that information on every *sysname* in the z/VM CSM system status table is to be displayed.

**PRODPending**

Returns information about pending PUT2PROD operations on z/VM CSM managed systems.

**STATUS**

Indicates that the service level status information for *sysname* or *nickname* is to be displayed. This operand is the default operand for SYSTEM QUERY commands.

**DEtails**

Requests a list of more detailed information about system attributes and component-level service information maintained in the system status table on the principal system for the specified *sysname* or *nickname*.

**VERify**

Augments the detailed service level status information with information obtained directly from a managed system.



**SERVcheck**

Retrieves component bitmaps and local modification information from the designated managed system and compares this information with the z/VM CSM service level defined for this system.

**FTP Command Options**

For more information about the following command options, see *z/VM: TCP/IP User's Guide*.

**PORT *port\_number***

The TCP/IP port number at which a dedicated FTP server is configured to listen. The default listen port for this server is 4535.

**ADDRtype *address\_type***

The address type to use. Valid values are:

**ANY**

Indicates that any type (IPv4 or IPv6) of target host address is allowed.

**IPV4**

Indicates that the target host address must be in the AF\_INET (IPv4) address family.

**IPV6**

Indicates that the target host address must be in the AF\_INET6 (IPv6) address family.

**NOSECure**

Causes the FTP client not to attempt to secure data and control connections using TLS.

**SECure**

Causes the FTP client to attempt to secure data and control connection using TLS.

**TImeout *value***

Specifies the number of seconds to be used for various FTP timing parameters. *value* is a whole number ranging from 15 to 720, inclusive. If not specified, timing parameters are applied accordingly by the TCP/IP FTP command itself, inclusive of those defined in an FTP DATA file. Note that minimal TIMEOUT values are likely to cause command errors for those z/VM CSM commands that require communication with a managed system.

**TRACe**

Starts the generation of tracing output. TRACE is used to assist in debugging. Trace data is written to the console.

**DATAFile *sysname***

Specifies the file name of a host-specific FTP DATA file instance that is to provide system-unique FTP operational characteristics for a specific managed system. The file type of this file must be FTPDATA. Any *sysname* FTPDATA file that is created should be based on the FTP DATA file that is created and customized when z/VM CSM is initialized on a principal system. Like that file, the *sysname* FTPDATA should reside on the z/VM CSM root directory for a given release (VMPSFS:CSMvrm).

**Usage Notes**

1. During SERVMGR command processing, the MAINTCSM 191 disk is accessed at file mode Z, with this access file mode maintained upon command completion. It is advised that files that are customized for using CSM (such as a MAINTCSM NAMES file) or for productivity (a PROFILE XEDIT file) be maintained on the MAINTCSM 191 disk, so these are available for reference, when needed.

**Examples**

1. To add the principal system (SYSTEM1) as a locally-managed (local) system that incorporates external security manager (ESM) controls and support, enter:

```
SERVMGR SYSTEM ADD 730 SYSTEM1 RSU1_ESM ESM COMMTYPE LOCAL
```

2. To add a remote system named SYSTEM2 (which does not incorporate ESM controls and support), enter:

## SERVGR SYSTEM

```
SERVGR SYS ADD 730 SYSTEM2 RSU1 NONESM HOST 123.45.67.87 COMM FTP
```

3. To check system status information for SYSTEM2, enter:

```
SERVGR SYSTEM QUERY 730 SYSTEM2
```

4. To put service into production on managed system SYSTEM2, enter:

```
SERVGR SYSTEM PUT2PROD 730 SYSTEM2
```

5. To add SYSTEM3, which is running RACF, has IP address 123.45.67.88, and has RSU 7302 installed, enter:

```
SERVGR SYS ADD 730 SYSTEM3 RSU2_ESM ESM HOST 123.45.67.88 COMM FTP
```

6. To display summary information about all the systems that you are managing, enter:

```
SERVGR SYS Q 730 ALL
```

7. To display information from the SYSTEM2 \$PRODS file, enter:

```
SERVGR SYS Q 730 SYSTEM2 PRODPENDING
```

8. To display additional information about SYSTEM2, enter:

```
SERVGR SYS Q 730 SYSTEM2 (details
```

9. To change the name of SYSTEM3 and its associated IP information to SYSTEM4, enter:

```
SERVGR SYSTEM MOD 730 SYSTEM3 SYSTEM4 ESM HOSTNAME 123.45.67.89
```

10. To update the information for SYSTEM2 to enable the use of an ESM, enter:

```
SERVGR SYSTEM MOD 730 SYSTEM2 ESM
```

11. To update the information for SYSTEM4 to disable the use of an ESM, enter:

```
SERVGR SYSTEM MOD 730 SYSTEM4 NONESM
```

12. To delete a system that is no longer being managed by z/VM CSM from the z/VM CSM environment, enter:

```
SERVGR SYSTEM DEL 730 SYSTEM4
```

13. To compare service information on a managed system against the service information in the z/VM CSM service level for that system, enter:

```
SERVGR SYSTEM SERV 730 SYSTEM2
```

### Input/Output Files

The input and output files unique to SERVGR are:

#### Input/Output Files

##### **\$CSMAGT \$MSGLOG**

CSMAGENT message log.

##### **\$CSMCMG \$MSGLOG**

SERVGR message log.

##### **CSM SVCSTAT**

z/VM CSM service status table.

##### **CSM SYSSTAT**

z/VM CSM system status table.

## Messages and Return Codes

For a complete explanation of each message, use the z/VM help facility to view the message explanation online or see the appropriate message documentation. To display information about a specific message (VMF002E, for example), enter:

```
help msg vmf002e
```

To display the main z/VM help menu, enter:

```
help
```

For information about the HELP command, enter:

```
help cms help
```

Appendix D, “Module Identifiers for VMSES/E Messages,” on page 749 shows the format of VMSES/E messages and lists the identifiers for each VMSES/E EXEC.

The SERVMGR EXEC issues the following return codes:

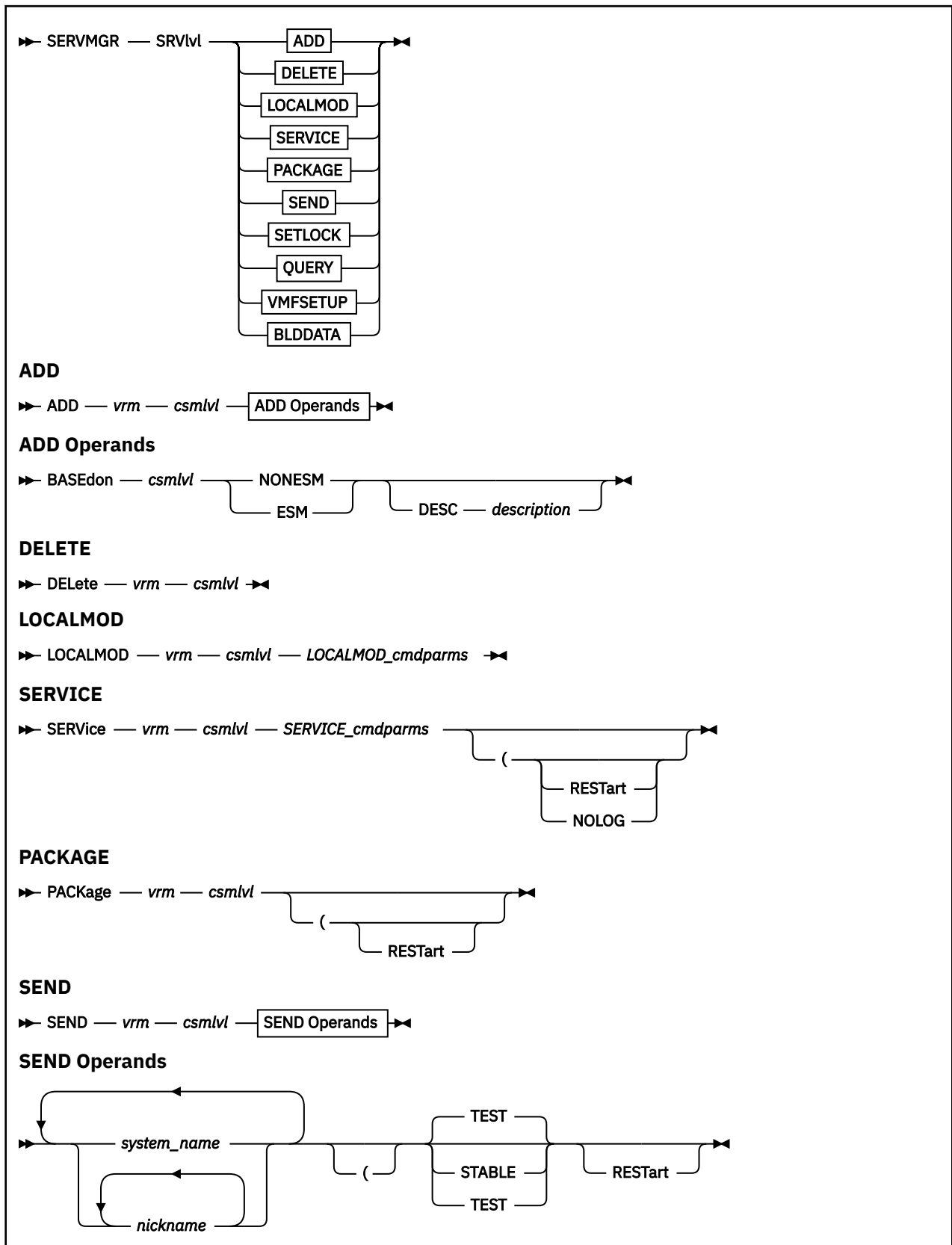
Return Code	Explanation
0	Command processing completed successfully.
2	Command processing completed successfully but additional action may be required.
4	Command processing completed with warnings.
5	Command processing stops.
6	Command processing stops.
8	Command processing stops.
9	Command processing stops.
12	Command processing stops.
13	Command processing stops.
21	Command processing stops.
24	Command processing stops.
28	Command processing continues or stops.
32	Command processing stops.
33	Command processing stops.
34	Command processing stops.
35	Command processing stops.
36	Command processing stops.
40	Command processing stops.
97	Command processing stops.
98	Command processing stops.
99	Command processing stops.
100	Command processing stops.

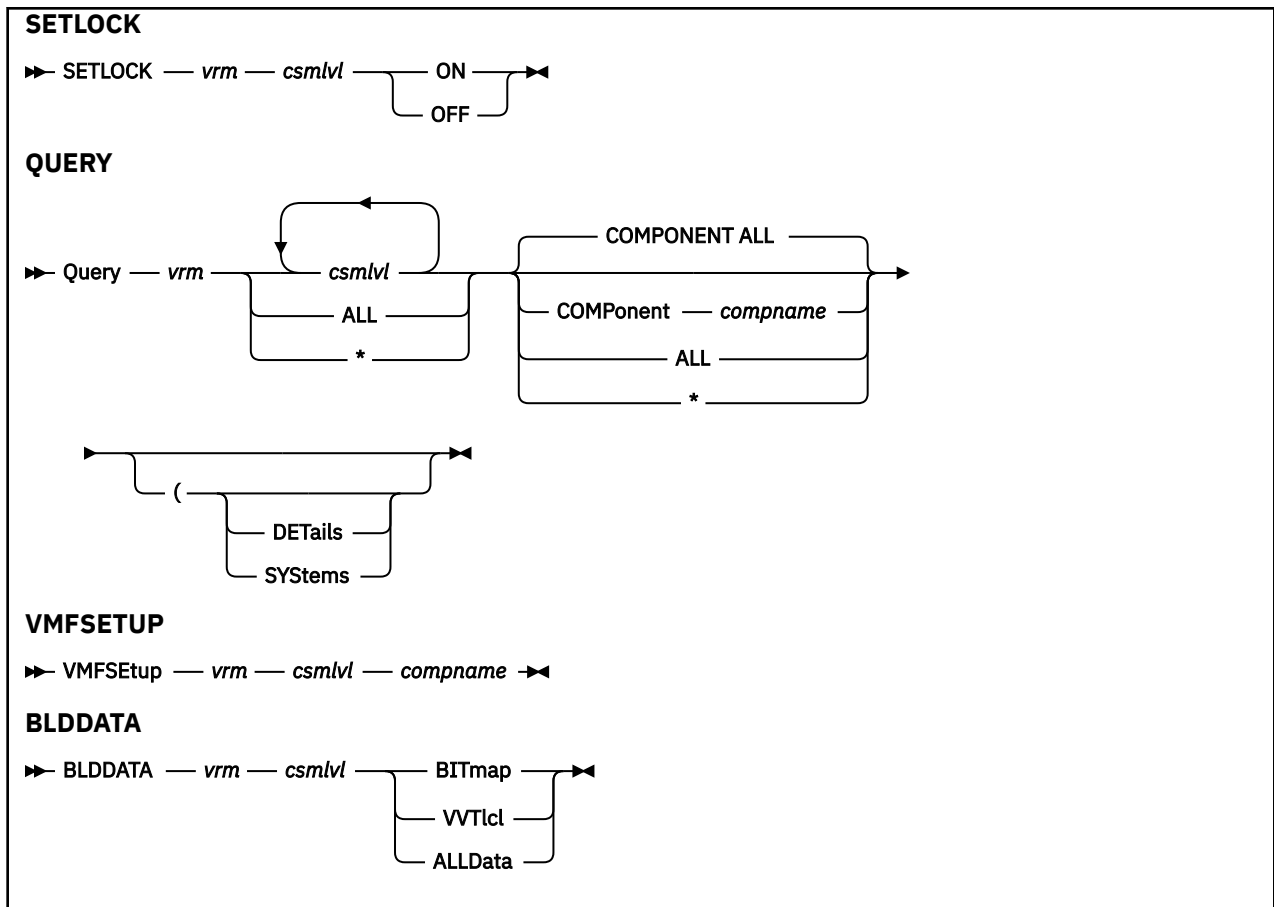
## SERVMGR SYSTEM

Return Code	Explanation
4113	Command processing stops.

## SERVMGR SRVLVL

**Note:** SERVMGR SRVLVL operands *must* be specified in the order shown in the SERVMGR SRVLVL syntax diagram.





## Purpose

Use the SERVMGR SRVLVL command to handle all service-level tasks associated with z/VM Centralized Service Management (z/VM CSM).

## Operands

### SRVLVL

Manages the unique service-level environments and gives the principal system control over all defined service levels that are available for the managed systems.

### *vrm*

Represents the specific version, release, and modification level of information that z/VM CSM is to reference, in relation to a given command. The specified *vrm* level determines which z/VM CSM system status table data and z/VM CSM service status table data is referenced as a z/VM CSM command is processed.

### *csmlvl*

A 1- through 16-character identifier for a z/VM CSM service level.

### ADD

Adds a service level to the z/VM CSM service status table and creates the SFS structure to contain the new *csmlvl*.

### BASEdon

Specifies the existing service level name from which the new *csmlvl* is initialized.

### NONESM

Indicates that the service level will not incorporate ESM controls and support.

### ESM

Indicates that the service level is to incorporate ESM controls and support.

**DESC description**

A free-form, text description for a service level that is being added. If the DESC operand is omitted or no descriptive text is supplied, the *csmlvl* identifier is used. Descriptive information is stored in the SERVLVL DESCRIPT file, which is maintained in the top-level SFS directory that pertains to *csmlvl*. For example:

```
VMPSFS:CSMVTM.ABC0118D_BLD01
```

**Note:** Although the description is free-form text, certain restrictions apply. Any characters used in the user's current CP TERMINAL command settings cannot be used. Additionally, parentheses are not allowed in the description text supplied on the command line.

**DELeTe**

Removes a service level from the z/VM CSM service status table and deletes the associated SFS directories.

See the **Usage Notes** for some service level deletion considerations.

**LOCALMOD**

Creates or reworks local modifications for a part maintained within the specified z/VM CSM service level.

**LOCALMOD\_cmdparms**

Any valid VMSES/E LOCALMOD command operands and options. See [“LOCALMOD EXEC” on page 245](#) for more information.

**SERVICE**

Receives and builds service for the specified *csmlvl* using the VMSES/E SERVICE command.

**SERVICE\_cmdparms**

Any valid VMSES/E SERVICE command operands and options. See [“SERVICE EXEC” on page 261](#) for more information.

**PACKage**

Indicates that serviced components in *csmlvl* are to be packaged for subsequent transport to one or more managed systems.

**SEND**

Indicates that the z/VM CSM service package for *csmlvl* is to be transported to the specified *system\_name* or *nickname*.

**TEST**

Indicates that the service level being transported has testing status. This status designation allows the content of the service level to be updated iteratively until it is designated as a stable service level.

**STABLE**

Indicates that the service level being transported has stabilized status. This status designation indicates that the content of the service level is to be maintained as it is now (with no further alteration possible) after it is transported to *any* managed system.

**SETLOCK**

Establishes an update activity lock for the specified *csmlvl*, to prevent or allow modification of its content using the SERVMGR SERVICE command or the SERVMGR LOCALMOD command.

**ON**

Prevents update activity for the specified *csmlvl*.

**OFF**

Allows update activity for the specified *csmlvl*.

**Query**

Processes the z/VM CSM service status table and returns information about a specific *csmlvl* or about all service levels defined in the z/VM CSM service status table.

**ALL | \***

Returns information about all service levels defined in the z/VM CSM service status table.

**COMPONENT**

Specifies when the list of levels is finished and demarks the *compname* variable's location.

***compname***

Denotes the name of the specific component being queried. ALL is the default value.

**DETAils**

Displays PTF and local modification information that pertains to the specified *csmlvl*. This information is listed on a component-by-component basis, with PTF data cited first, followed by any applicable local modification information.

**SYStems**

Reports systems where the specified *csmlvl* is in use.

**VMFSEtup**

Invokes the VMSES/E VMFSETUP command, to establish a CMS minidisk and SFS directory access order for the specified component, using resources that pertain to the specified *csmlvl*.

***compname***

The name of a component as it is specified on the :COMPNAME tag in a product parameter file (PPF). *compname* is a 1- through 16-character alphanumeric identifier.

**BLDDATA**

Generates new bitmaps (BITMAP), local version vector tables (VVTLCCL), or both (ALLDATA) for members of a service level.

**Options****REStart**

Indicates that SERVMGR processing is to be resumed, after a previous error condition was encountered (which since has been corrected). This option is valid for only the SERVMGR SRVLVL SERVICE, PACKAGE, and SEND functions.

**NOLOG**

Indicates that the messages produced by the SERVMGR SRVLVL command are not to be logged.

**Usage Notes**

1. When adding a new service level, the BASEDON service level will be made STABLE and will be locked from further updates.
2. In order to save space, it is recommended that service levels be deleted when they are no longer being used.
3. A service level cannot be deleted if it is the current or pending service level for a managed system or if it is the BASEDON service level for a level that is currently in use.
4. The BASE service level cannot be deleted or modified.
5. When a restart of the SERVICE command is needed in the z/VM CSM environment, RESTART is specified as a command *option* and not as a command *operand*.
6. Because the COMPONENT operand provides component-specific information about the PTFs applied for a given component, use of the DETAILS option with this operand is redundant (no additional information is available for the DETAILS option to operate upon).
7. During SERVMGR command processing, the MAINTCSM 191 disk is accessed at file mode Z, with this access file mode maintained upon command completion. It is advised that files that are customized for using CSM (such as a MAINTCSM NAMES file) or for productivity (a PROFILE XEDIT file) be maintained on the MAINTCSM 191 disk, so these are available for reference, when needed.

**Examples**

1. To create a new service level named RSU1 that is not ESM enabled, enter:



```
SERVGR SRV ADD 730 RSU1 BASEDON BASE NONESM DESC RSU1 shipped with z/VM
```

- To install a service file named RSU1FILE SERVLINK into service level RSU1, enter:

```
SERVGR SRVLVL SERVICE 730 RSU1 ALL RSU1FILE
```

- To restart SERVICE command processing for service level RSU2, after correcting reported errors, enter:

```
SERVGR SRVLVL SERVICE 730 RSU2 ( RESTART
```

- To create the service package for service level RSU1, enter:

```
SERVGR SRVLVL PACKAGE 730 RSU1
```

- To send service level RSU1 to remote managed system SYSTEM2, enter:

```
SERVGR SRVLVL SEND 730 RSU1 SYSTEM2
```

- To restart a SEND command to SYSTEM2, enter:

```
SERVGR SRVLVL SEND 730 RSU1 SYSTEM2 (RESTART
```

- To add a new service level named RSU2\_ESM that is based on service level RSU2, enter:

```
SERVGR SRVLVL ADD 730 RSU2_ESM BASEDON RSU2 ESM DESC RSU2 enabled for RACF
```

- To display summary information about all of the service levels that you have defined, enter:

```
SERVGR SRV Q 730 ALL
```

- To display PTF information for component CP in service level RSU1, enter:

```
SERVGR SRV Q 730 RSU1 COMPONENT CP
```

- To delete service level RSU1, enter:

```
SERVGR SRVLVL DEL 730 RSU1
```

## Input/Output Files

The input and output files unique to SERVMGR are:

### Input/Output Files

#### **\$CSMAGT \$MSGLOG**

CSMAGENT message log.

#### **\$CSMCMG \$MSGLOG**

SERVGR message log.

#### **CSM SVCSTAT**

z/VM CSM service status table.

#### **CSM SYSSTAT**

z/VM CSM system status table.

## Messages and Return Codes

For a complete explanation of each message, use the z/VM help facility to view the message explanation online or see the appropriate message documentation. To display information about a specific message (VMF002E, for example), enter:

```
help msg vmf002e
```

To display the main z/VM help menu, enter:

```
help
```

For information about the HELP command, enter:

```
help cms help
```

Appendix D, “Module Identifiers for VMSES/E Messages,” on page 749 shows the format of VMSES/E messages and lists the identifiers for each VMSES/E EXEC.

The SERVGR EXEC issues the following return codes:

<b>Return Code</b>	<b>Explanation</b>
0	Command processing completed successfully.
4	Command processing completed with warnings.
5	Command processing stops.
6	Command processing stops.
8	Command processing stops.
9	Command processing stops.
12	Command processing stops.
13	Command processing stops.
21	Command processing stops.
24	Command processing stops.
28	Command processing continues or stops.
32	Command processing stops.
33	Command processing stops.
34	Command processing stops.
35	Command processing stops.
36	Command processing stops.
40	Command processing stops.
97	Command processing stops.
98	Command processing stops.
99	Command processing stops.
100	Command processing stops.
4113	Command processing stops.

## SERVMGR REMOVE

**Note:** SERVMGR REMOVE operands *must* be specified in the order shown in the SERVMGR REMOVE syntax diagram.

```
►► SERVMGR — REMOVE — vrm ◄◄
```

### Purpose

Use the SERVMGR REMOVE command to delete a version/release/modification level from the z/VM Centralized Service Management (z/VM CSM) environment.

### Operands

#### REMOVE

Deletes a version/release/modification (*vrm*) level from the z/VM CSM environment. The SERVMGR REMOVE command deletes all tables and *vrm*-level Shared File System (SFS) directories that are associated with a subject *vrm* level. Before the *vrm* level is deleted, the SERVMGR message log file, \$CSMCMG \$MSGLOG, is copied from the z/VM CSM root directory for *vrm* (VMPSFS:CSM*vrm*.) to the MAINTCSM 191 disk or to SFS directory VMPSFS:MAINTCSM. and saved as \$CSMCMG \$MSG*vrm*.

#### *vrm*

Represents the specific version/release/modification level of information for which z/VM CSM is to be deleted.

### Usage Note

1. No managed systems in the z/VM CSM group can be using the specified version/release/modification (*vrm*) level when it is deleted. Before running the SERVMGR REMOVE command, use the SERVMGR SYSTEM DELETE command to delete all systems that are using the specified *vrm* level from the z/VM CSM group.
2. During SERVMGR command processing, the MAINTCSM 191 disk is accessed at file mode Z, with this access file mode maintained upon command completion. It is advised that files that are customized for using CSM (such as a MAINTCSM NAMES file) or for productivity (a PROFILE XEDIT file) be maintained on the MAINTCSM 191 disk, so these are available for reference, when needed.

### Example

1. To delete the z/VM 7.2.0 *vrm* level from z/VM CSM, enter:

```
SERVMGR REMOVE 720
```

### Input/Output Files

The input and output files unique to SERVMGR are:

#### Input/Output Files

##### \$CSMAGT \$MSGLOG

CSMAGENT message log.

##### \$CSMCMG \$MSGLOG

SERVMGR message log.

##### \$CSMCMG \$MSG*vrm*

SERVMGR message log for a deleted version/release/modification level.

#### CSM SVCSTAT

z/VM CSM service status table.

**CSM SYSSTAT**

z/VM CSM system status table.

**Messages and Return Codes**

For a complete explanation of each message, use the z/VM help facility to view the message explanation online or see the appropriate message documentation. To display information about a specific message (VMF002E, for example), enter:

```
help msg vmf002e
```

To display the main z/VM help menu, enter:

```
help
```

For information about the HELP command, enter:

```
help cms help
```

Appendix D, “[Module Identifiers for VMSES/E Messages](#),” on page 749 shows the format of VMSES/E messages and lists the identifiers for each VMSES/E EXEC.

The SERVMGR EXEC issues the following return codes:

<b>Return Code</b>	<b>Explanation</b>
0	Command processing completed successfully.
4	Command processing completed with warnings.
5	Command processing stops.
6	Command processing stops.
8	Command processing stops.
9	Command processing stops.
12	Command processing stops.
13	Command processing stops.
21	Command processing stops.
24	Command processing stops.
28	Command processing continues or stops.
32	Command processing stops.
33	Command processing stops.
34	Command processing stops.
35	Command processing stops.
36	Command processing stops.
40	Command processing stops.
97	Command processing stops.
98	Command processing stops.
99	Command processing stops.

Return Code	Explanation
100	Command processing stops.
4113	Command processing stops.

## SERVMGR MANAGED

**Note:** SERVMGR MANAGED operands *must* be specified in the order shown in the SERVMGR MANAGED syntax diagram.



### Purpose

Use the SERVMGR MANAGED command for emergency service situations, to remove a managed system from the z/VM Centralized Service Management (z/VM CSM) environment, or to enable a new z/VM-related product. SERVMGR MANAGED can be used for enablement of all of the z/VM-related products except RACF.

### Operands

#### MANAGED

Is used for emergency service situations, to enable a new z/VM-related product, or to remove a managed system from z/VM CSM.

#### OFF

Removes a managed system from the z/VM CSM environment.

#### ON

Puts a managed system in the z/VM CSM environment.

### Usage Notes

#### 1. Emergency service situations while running in a z/VM CSM environment:

If a situation arises for which you need to temporarily remove a managed system from a z/VM CSM environment and apply service manually, you can do that. This is *not* recommended by IBM because of the risk of getting your service levels out of sync with the managed system, and should be used with caution. Entered on the MAINT`vrm` user ID on the managed system, the SERVMGR MANAGED OFF command removes a managed system from the z/VM CSM environment. After the managed system has been removed, if the service is updated, the system will have to be managed manually until you re-synchronize it with a service level on the principal system. After you have re-synchronized the service, you can add the system back into your z/VM CSM group by entering the SERVMGR MANAGED ON command from the MAINT`vrm` user ID on the managed system.

### Example

- To enable Performance Toolkit on a managed system, first (on the managed system), enter:

```
SERVMGR MANAGED OFF
```

Next, issue the appropriate enablement command as provided in the licensing document (MEMO file).

Then, enter:

```
SERVMGR MANAGED ON
```

### Input/Output Files

The input and output files unique to SERVMGR are:

**Input/Output Files****\$CSMAGT \$MSGLOG**

CSMAGENT message log.

**\$CSMCMG \$MSGLOG**

SERVMGR message log.

**CSM SVCSTAT**

z/VM CSM service status table.

**CSM SYSSTAT**

z/VM CSM system status table.

**Messages and Return Codes**

For a complete explanation of each message, use the z/VM help facility to view the message explanation online or see the appropriate message documentation. To display information about a specific message (VMF002E, for example), enter:

```
help msg vmf002e
```

To display the main z/VM help menu, enter:

```
help
```

For information about the HELP command, enter:

```
help cms help
```

Appendix D, “Module Identifiers for VMSES/E Messages,” on [page 749](#) shows the format of VMSES/E messages and lists the identifiers for each VMSES/E EXEC.

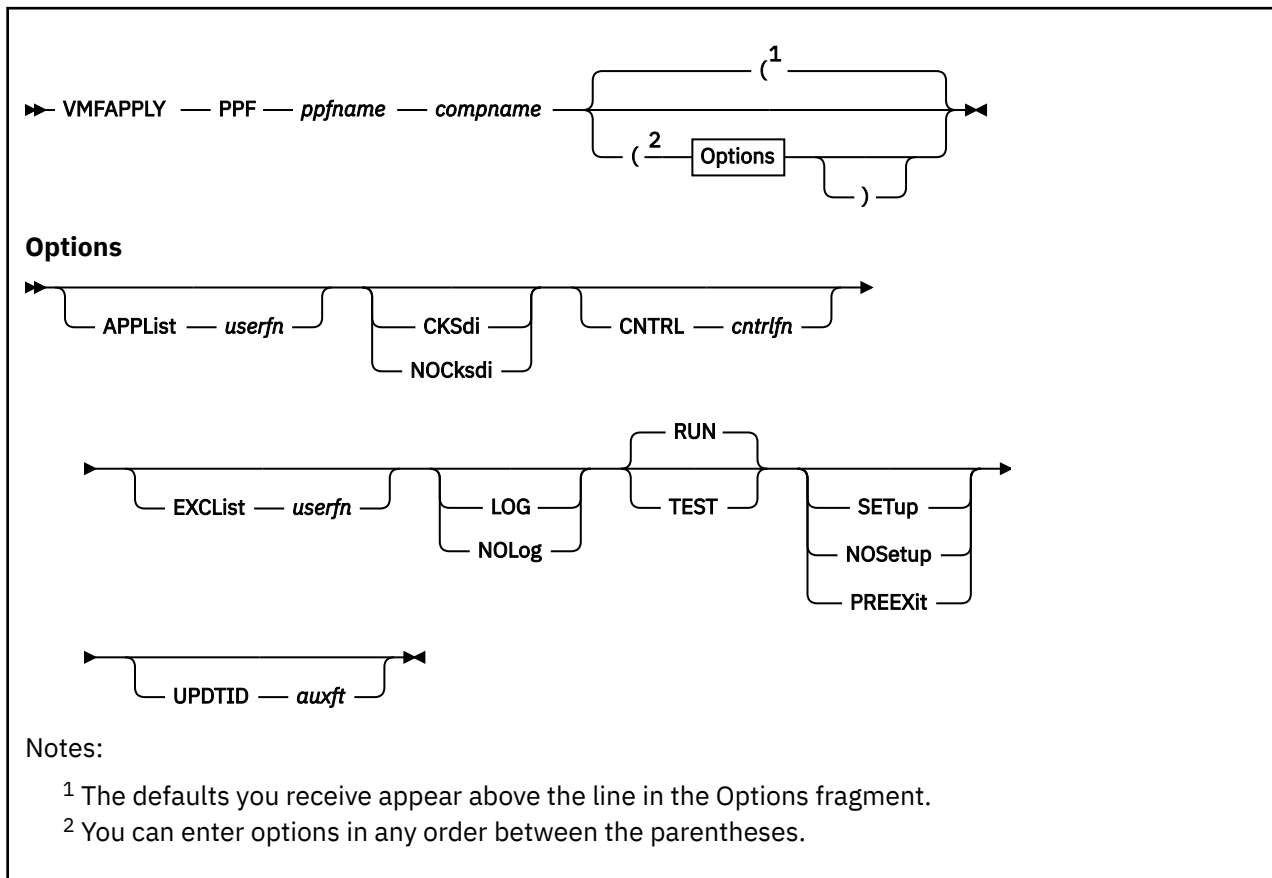
The SERVMGR EXEC issues the following return codes:

<b>Return Code</b>	<b>Explanation</b>
0	Command processing completed successfully.
4	Command processing completed with warnings.
5	Command processing stops.
6	Command processing stops.
8	Command processing stops.
9	Command processing stops.
12	Command processing stops.
13	Command processing stops.
21	Command processing stops.
24	Command processing stops.
28	Command processing continues or stops.
32	Command processing stops.
33	Command processing stops.
34	Command processing stops.
35	Command processing stops.

<b>Return Code</b>	<b>Explanation</b>
36	Command processing stops.
40	Command processing stops.
97	Command processing stops.
98	Command processing stops.
99	Command processing stops.
100	Command processing stops.
4113	Command processing stops.



## VMFAPPLY EXEC



### Purpose

VMFAPPLY updates the maintenance level of the specified component based on the set of PTFs defined in the specified apply and exclude lists. The maintenance level of the component is defined by the apply status table and the version vector tables. The apply status table defines the PTFs that have been successfully processed by VMFAPPLY. The version vector tables define the service history for all parts affected by PTFs in the apply status table.

### Operands

#### PPF

indicates that the specified product parameter file is to be used for apply processing.

#### ppfname

is the file name of the usable form product parameter file. The file type must be PPF.

#### compname

is the name of the component as it is specified on the :COMPNAME tag in the product parameter file. *compname* is a 1- to 16-character alphanumeric identifier.

### Options

#### APPList

defines the file name of a user-supplied apply list.

***userfn***

is the file name of the apply list. The file type of the apply list is \$APPLIST. This file is used instead of the IBM-supplied file. This value overrides the value on the :AXLIST tag in the PPF.

**CKSDI**

compares the self-documenting information contained in parts that have the SDI option specified in the \$PTFPART file to the version vector table when the apply process completes.

**NOCKSDI**

does not compare the self-documenting information to the version vector tables.

**Note:** If the CKSDI and NOCKSDI options are omitted, the VMFAPPLY EXEC uses the value of the :CKSDI tag in the product parameter file to determine whether to compare the self-documenting information to the version vector tables.

**CNTRL**

defines the file name of the control file used to identify the AUX file and version vector table structure.

***cntrlfn***

is the file name of the control file. The file type of the control file is CNTRL. This value overrides the value on the :CNTRL tag in the PPF.

**EXCLIST**

defines the file name of a user-supplied exclude list.

***userfn***

is the file name of the exclude list. The file type of the exclude list is \$EXCLIST. This file is used with the IBM-supplied apply list. This value overrides the value on the :EXCLIST tag in the PPF.

**LOG**

writes VMFAPPLY messages into the apply message log (\$VMFAPP \$MSGLOG).

No messages are logged until initial validation of the command is complete.

**NOLOG**

does not write VMFAPPLY messages into the apply message log (\$VMFAPP \$MSGLOG).

**Note:** If the LOG and NOLOG options are omitted, the VMFAPPLY EXEC uses the value of the :LOG tag in the product parameter file to determine whether to log messages into the apply message log.

**RUN**

requests a complete run of the apply process. The Software Inventory files are updated when the apply process is complete and no errors are encountered. RUN is the default.

**TEST**

requests a dry run of the apply process. All apply processing steps are completed, but the Software Inventory files are not updated.

**SETUP**

sets up a minidisk/directory access order for the apply function according to entries in the :MDA section of the product parameter file. If a user exit is specified in the product parameter file, setup will occur after the user exit is called.

**NOSETUP**

does not set up a new access order.

**PREEXIT**

sets up a minidisk or SFS directory access order for the apply function according to entries in the :MDA section of the product parameter file. If a user exit is specified in the product parameter file, setup will occur before the user exit is called.

**Note:** If the SETUP, PREEXIT, and NOSETUP options are omitted, the VMFAPPLY EXEC uses the value of the :SETUP tag in the product parameter file to determine whether to set up a new access order.

**UPDTID**

defines the file type of the version vector table that is updated by the VMFAPPLY EXEC. It also defines the file type of the AUX file that is generated for parts that have the AUX option specified in the \$PTFPART file. This value overrides the value on the :UPDTID tag in the PPF.

***auxft***

is the file type of the version vector table.

**Usage Notes**

1. VMFAPPLY requires the alternate APPLY disk to be accessed as read-write.
2. VMFAPPLY requires the entire APPLY, DELTA, and LOCAL strings to be accessed.

**Examples**

- To run VMFAPPLY using the IBM-supplied defaults, enter:

```
VMFAPPLY PPF ppfname compname
```

- To run VMFAPPLY using a user-specified apply list (named MYLIST \$APPLIST), enter:

```
VMFAPPLY PPF ppfname compname (APPLIST MYLIST
```

- To use VMFAPPLY to do a dry run of the apply process, enter:

```
VMFAPPLY PPF ppfname compname (TEST
```

**Input and Output Files****Input Files*****ppfname* PPF**

The usable form product parameter file.

***prodid* SRVREQT**

The service-level requisite table.

***fn* \$APPLIST**

The apply list, which identifies the PTFs to process.

***fn* \$EXCLIST**

The exclude list, which identifies PTFs that are not processed even if they are specified in the apply list.

***cntrlfn* CNTRL**

The control file identified by the :CNTRL tag in the PPF or the CNTRL option on the VMFAPPLY command.

***appid* SRVAPPS**

The service-level apply status table.

***appid* VVTlvlid**

The version vector tables specified by the control file specified on the :CNTRL tag in the product parameter file.

***appid* \$SELECT**

The select data file.

**Input/Output Files*****appid* \$APRCVRY**

The existence of this file on the APPLY disk string indicates VMFAPPLY was interrupted during critical processing on the last invocation of VMFAPPLY for the specified component (used for recovery).

***appid* \$SRVAPPS**

The apply status table with a temporary file type (used for recovery).

***appid* \$VVTlvlid**

The version vector tables with temporary file types (used for recovery).

***appid* \$STATS**

This file contains the values of key variables that are required for recovery (used for recovery).

**Output Files*****appid* \$MISSING**

If TEST option is selected, this file is created on the A-disk and contains a list of parts that were identified as missing. This file is in CMS LISTFILE format.

**RETRY \$APPLIST**

If VMFAPPLY fails because of errors with selected PTFs, this file is created on the A-disk. This file is an apply list for the PTFs that did not encounter errors.

**VMSES PARTCAT**

The parts catalog table.

**\$VMFAPP \$MSGLOG**

The apply message log.

***partid* AUXlvlid**

AUX files for parts that have the AUX option specified in the \$PTFPART file.

**PPF Tags Used****:APPID**

The identifier of the product used during apply processing.

**:AXLIST**

Defines the apply and exclude list names supplied by IBM on the service tape.

**:EXCLIST**

Defines the file name of a user-supplied exclude list. This file is used with the IBM-supplied exclude list that has the file name specified on the :AXLIST tag.

**:CNTRL**

Defines the name of the control file.

**:MDA**

Defines symbolic strings and the minidisk or SFS directories associated with them.

**:UPDTID**

Defines the file type of the version vector table that is updated by the VMFAPPLY EXEC. It also defines the file type of the AUX file that is generated for parts that have the AUX option specified in the \$PTFPART file.

**:CKSDI**

Controls whether the self-documenting information in parts with the SDI option specified in the \$PTFPART file is compared to the version vector upon completion of the apply process.

**:LOG**

Controls whether messages are logged in the apply message log (\$VMFAPP \$MSGLOG).

**:SETUP**

Controls whether the VMFSETUP EXEC is called to access minidisks/directories.

**:USEREXIT**

Defines the file name of the user exit. If no value is specified, then no exit is invoked.

**Messages and Return Codes**

Appendix D, “[Module Identifiers for VMSES/E Messages](#),” on page 749 shows the format of VMSES/E messages and lists the identifiers used by each VMSES/E exec.

For a complete explanation of any VMSES/E message, use the HELP Facility to view the message explanation online or see the appropriate messages documentation. To display information on a specific message, for example VMF002E, enter:

```
help msg vmf002e
```

If you are unfamiliar with the z/VM HELP Facility, enter `help` to display the main HELP Menu. For information on the HELP command, enter:

```
help cms help
```

### PI

Return codes issued by the VMFAPPLY EXEC may be returned to a user exit. For more information about user exits, see [:USEREXIT.](#)

The VMFAPPLY EXEC issues the following return codes:

Return Code	Explanation
0	Command completed successfully.
2	Command completed successfully but extraneous data was encountered.
4	Command completed with one or more warning conditions.
8	Command completed but at least one major process failed.
12	Command failed because of an internal error.
24	Command failed because of a command line syntax error.
28	Command failed because a required file was not found.
36	Command failed because a target disk or directory was not available.
100	Command failed because of an external error.
500	User terminated the command from a prompt.

**Note:** The Software Inventory is updated if the return code is less than or equal to 4.

### PI end

## Recovery Information

The VMFAPPLY command can be restarted by reissuing the command.

**Case 1:** After the successful completion of VMFAPPLY, the same command is reissued.

Rerunning VMFAPPLY after it has successfully completed has no effect on the state of the Software Inventory. PTFs that are already applied are not processed. Therefore, no PTFs would be processed in this case.

**Case 2:** VMFAPPLY is interrupted while processing PTFs in the Apply list.

While processing PTFs, all data is maintained in storage. No data is written to the APPLY disk. The only external changes are in the \$VMFAPP \$MSGLOG. In terms of the Apply Disk (the maintenance level of the system), this is equivalent to the command not being issued.

**Case 3:** VMFAPPLY is interrupted while updating the Software Inventory.

After the processing of PTFs in the Apply list is complete, the Software Inventory (the APPLY disk) is updated. Because this is an interruptible process, a situation could exist where half the information is updated. To avoid this inconsistency, VMFAPPLY:

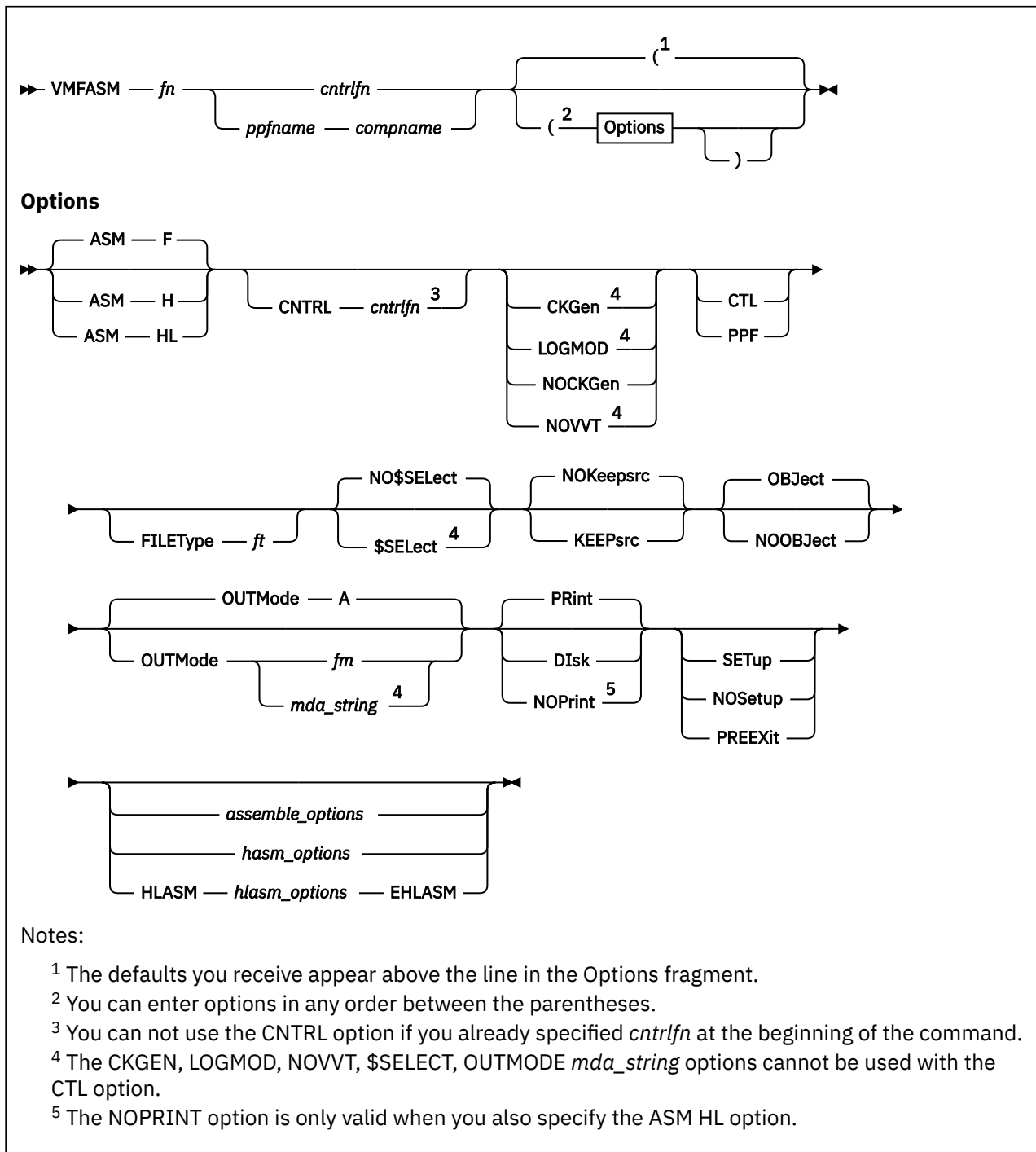
- Updates the Software Inventory (Apply Status table and Version Vectors) using temporary file types
- Saves key variables in a temporary file
- Sets the recovery flag (writes a file to the APPLY disk named *appid \$APRCVRY*)
- Copies the Software Inventory files to the real file types

## VMFAPPLY EXEC

- Creates AUX files
- Updates the \$SELECT file
- Resets the recovery flag (erases the file named *appid \$APRCVRY*)
- Erases all temporary files

If the process is interrupted while the recovery flag (*appid \$APRCVRY*) is set, VMFAPPLY detects this condition and completes the prior VMFAPPLY invocation based on the response from the user. The prior VMFAPPLY invocation is completed using the information saved in the temporary files.

## VMFASM EXEC



### Purpose

The VMFASM EXEC applies updates to a source file and assembles the file using either the ASSEMBLE command, the HASM command, or the HLASM command.

## Operands

### *fn*

is the file name of a source file to be updated and assembled. It must have a file type of ASSEMBLE.

### *cntrlfn*

is the file name of a control file. The file type must be CNTRL.

### *ppfname*

is the file name of a usable form product parameter file. It must have a file type of PPF. The name of the control file that is to be used to update the source file is obtained from this product parameter file.

### *compname*

is the name of the component (such as CP or CMS) as it is specified on the :COMPNAME tag in the product parameter file. *compname* is a 1- to 16-character alphanumeric identifier.

## Options

### ASM F

indicates that ASSEMBLE is used to assemble the part. ASM F is the default.

### ASM H

indicates that HASM is used to assemble the part.

### ASM HL

indicates that ASMAHL or HLASM is used to assemble the part.

### CNTRL

specifies that a control file is used to identify the AUX file structure.

### *cntrlfn*

is the file name of the control file that is used to identify the AUX file structure. The file type of the control file is CNTRL. This value overrides the value on the :CNTRL tag in the PPF. The CNTRL option can not be used if operands *fn* *cntrlfn* are specified.

### CKGen

requests validation of the AUX files against the version vector table and issues an error message if a mismatch is detected. The version vector tables are not updated.

### LOGMOD

requests validation of the AUX files against the version vector tables and automatically updates the local version vector tables when a mismatch is detected. When you specify the LOGMOD option, VMFASM modifies only the VVT files that are defined in the control file above the :UPDTID level defined in the product parameter file. All other VVT levels are only compared to the AUX files, and mismatches are displayed. You should only use the LOGMOD option when you are assembling files that have source updates. All LOCAL disks must be accessed as Read-Write.

When you use the LOGMOD option:

- If a version vector table does not exist on a LOCAL disk, it is created on the first disk in the LOCAL string.
- If the AUX file for a part is not found, the :PART entry (if found) is deleted from the version vector table.
- If the AUX file for a part is empty, the :MOD data is deleted from the version vector table for that part. The :PART entry is not deleted from the version vector table.

### NOCKGen

requests no validation of the AUX files against the version vector tables. The AUX file structure is used to update the source file, and the VVT structure is used to name the output file.

### NOVVT

requests no validation of the AUX files against the version vector tables (VVT). The AUX file structure is used to update the source file and name the output file.

**Note:** If you omit the CKGEN, LOGMOD, NOCKGEN, and NOVVT options, the VMFASM EXEC uses the value of the :CKGEN tag in the product parameter file to determine whether to validate the AUX files against



the version vector table. If the :CKGEN tag does not appear in the PPF, no validation is performed; and NOCKGEN is assumed.

**CTL**

indicates that the second operand in the command is the name of a control file. If CTL is specified, a product parameter file is not used.

**PPF**

indicates that the second parameter in the command is the name of a product parameter file that specifies the control file to be used to update the source file. The product parameter file also lists the minidisk/directory search order.

**Note:** If you do not enter a *compname* and you do not specify CTL or PPF, CTL is assumed.

**FILEType**

indicates the file type for the output file that is created. This option overrides any naming from the AUX or VVT structures.

***ft***

is the file type for the output file.

**NO\$SElect**

does not update the select data file (*appid* \$SELECT). NO\$SELECT is the default.

**\$SElect**

updates the *appid* \$SELECT file to indicate the text deck has been changed. The first APPLY disk specified in the :MDA section of the product parameter file must be accessed Read/Write. The other APPLY disks must be accessed.

**NOKeepsrsc**

erases the updated source file after it is assembled. NOKEEPSRC is the default.

**KEEPrsc**

indicates that the updated source file consisting of the source file and any updates will be saved on your A-disk. The file is named *\$fn* ASSEMBLE. *fn* is the source file name, truncated to seven characters when necessary.

**OBJect**

indicates that the output deck is to be created on your A-disk. OBJECT is the default.

**NOOBJect**

indicates that the output deck is not to be created on your A-disk.

**Note:** If you specify the OBJECT and NOOBJECT options between the HLASM and EHLASM keywords, they are ignored.

**OUTMode**

indicates the file mode for the output text and listing files created. This file mode must be accessed Read/Write.

**A**

creates the output files on file mode A. A is the default file mode.

***fm***

is the file mode for the output files.

***mda\_string***

is the name of the symbolic string of disks from the :MDA section of the product parameter file. The output is placed on the first disk specified in this string.

**PRint**

indicates the listing output is to be sent to the virtual printer. PRINT is the default.

**DIsk**

indicates the listing output is to be created on your A-disk.

**NOPrint**

suppresses the writing of the listing output.

**Note:** If you specify the PRINT, NOPRINT, and DISK options between the HLASM and EHLASM keywords, they are ignored.

### SETup

sets up a minidisk/directory access order for the assemble function according to entries in the :MDA section of the product parameter file. This option is valid only when using a product parameter file. If a user exit is specified in the product parameter file, setup will occur after the user exit is called.

### NOSetup

does not set up a new minidisk/directory access order.

### PREEXit

sets up a minidisk or SFS directory access order for the assemble function according to entries in the :MDA section of the product parameter file. If a user exit is specified in the product parameter file, setup will occur before the user exit is called.

**Note:** If the SETUP, PREEXIT, and NOSETUP options are omitted, the VMFASM EXEC uses the value of the :SETUP tag in the product parameter file to determine whether to set up a new access order.

## Usage Notes

1. VMFASM handles packed files.
2. When you assemble text decks for use in the VMSES/E environment, you must use a product parameter file.
3. If you receive warnings or errors from the UPDATE command, check the *fn* UPDLOG file for additional information.
4. The High Level Assembler is called with the TERM(NARROW) and NODECK options. If the High Level Assembler command name is ASMAHL it is also called with the following options, FLAG(NOCONT) and USING(NOWARN). To override these options, specify the desired options between the HLASM and EHLASM keywords.
5. VMFBLD uses the version vector tables to determine which level of a part to use during build processing. If you do not specify the LOGMOD option, you must either manually update the version vector tables before you run VMFBLD or you must rerun VMFASM and specify the LOGMOD option.
6. When you specify the \$SELECT option, the select data file (*appid* \$SELECT) is updated with a record consisting of either:
  - *fn* and the first 3 characters of the file type of the output file
  - *fn* and the full file type (when you also specify the FILETYPE option)

The select data file is used by VMFBLD to determine which objects need to be built using this text deck.
7. When you create local modifications, you can use the \$SELECT, LOGMOD, and OUTMODE options to eliminate some manual steps, such as updating the *appid* \$SELECT file, updating local version vector tables files, and saving the results on a LOCALMOD disk.

## Examples

- To run VMFASM, using the IBM-supplied defaults and a product parameter file, enter:

```
VMFASM DMSABC ppfname compname
```

- To run VMFASM, using the IBM-supplied defaults and a control file, enter:

```
VMFASM DMSABC cntrlfn (CTL
```

- To run VMFASM, using the H assembler and a product parameter file, enter:

```
VMFASM DMSABC ppfname compname (ASM H
```

- To run VMFASM, using the HL assembler and a product parameter file, enter:

VMFASM DMSABC *ppfname compname* (ASM HL)

## Input and Output Files

### Input Files

#### ***ppfname* PPF**

The usable form product parameter file.

#### ***cntrlfn* CNTRL**

The control file.

#### ***fn* ASSEMBLE**

The source file.

#### ***fn updtft***

Updates to the source file.

#### ***appid VVTlvlid***

The version vector table.

#### ***fn AUXlvlid***

The AUX file.

### Output Files

#### ***fn* TEXT**

#### ***fn* TXTnnnnn**

#### ***fn xxxnnnnn***

The assembled object deck (*xxx* is the file type abbreviation; *nnnnn* is a PTF number). You receive only one of these formats.

**Note:** The object deck is written to the A-disk only when the OBJECT option (the default) is specified.

#### ***appid VVTlvlid***

A version vector table.

#### ***appid \$SELECT***

The list of build requirements, when the \$SELECT option is specified.

#### ***\$fn* LISTING**

The assembler listing file.

#### ***fn* UPDLOG**

The update log file.

#### ***fn ctlfile***

The update information file.

**Note:** If listing output is generated during the assembly, the PRINT or DISK option determines where it will reside. The PRINT option (the default) causes all listing output to be sent to the virtual printer as *fn ctlfile*. The DISK option causes all listing output to be placed on the A-disk in two files, *\$fn* LISTING and *fn* UPDLOG.

### Temporary Files

#### ***\$fn* ASSEMBLE**

The updated source file.

#### ***\$fn* TEXT**

The temporary assembled object deck.

#### ***fn* UPDATES**

The update history file.

#### ***\$VMFSIM* CNTRL**

A control file used with the LOGMOD option.

#### ***fn* AUX\$\$\$\$**

An AUX file used with the LOGMOD option.

**PPF Tags Used**

**:APPID**

The identifier of the product, which is used to name the version vector table and the select data file.

**:CKGEN**

Controls the validation of AUX files against the version vector tables. Valid values are NO, YES, LOGMOD, and NOVVT.

**:COMPNAME**

Defines the component in the product parameter file to be used.

**:CNTRL**

Defines the name of the control file.

**:MDA**

Defines symbolic strings and the minidisks or SFS directories associated with them.

**:SETUP**

Controls whether the VMFSETUP EXEC is called to access the minidisks and directories.

**:USEREXIT**

Defines the file name of the user exit. If no value is specified, then no exit is invoked.

**Messages and Return Codes**

Appendix D, “Module Identifiers for VMSES/E Messages,” on page 749 shows the format of VMSES/E messages and lists the identifiers used by each VMSES/E exec.

For a complete explanation of any VMSES/E message, use the HELP Facility to view the message explanation online or see the appropriate messages documentation. To display information on a specific message, for example VMF002E, enter:

```
help msg vmf002e
```

If you are unfamiliar with the z/VM HELP Facility, enter help to display the main HELP menu. For information on the HELP command, enter:

```
help cms help
```

**PI**

Return codes issued by the VMFASM EXEC may be returned to a user exit. For more information about user exits, see [:USEREXIT.](#)

The VMFASM EXEC issues the following return codes:

Return Code	Explanation
0	Command completed successfully.
4	Command completed with one or more warning conditions.
8	Command completed but at least one major process failed.
24	Command failed because of a command line syntax error.
28	Command failed because a required file was not found.
36	Command failed because a target disk or directory was not available.
100	Command failed because of an external error.

**PI end**

**Recovery Information**

The VMFASM command can be restarted by reissuing the command.

## ASSEMBLE Options Supported by VMFASM

You can only enter *assemble\_options* when you use the ASM F option.

In the following table, the left column shows the options of the ASSEMBLE command. The right column shows how these options are supported by VMFASM when invoking the ASSEMBLE command. Also shown are the default values (underlined> of these options. The ASSEMBLE defaults are used wherever possible. Keyword-function options must be entered without the parentheses.

ASSEMBLE Option	VMFASM Option
ALIGN NOALIGN ALGN NOALGN	same
ALOGIC NOALOGIC	same
BUFSIZE (STD) BUFSIZE(MIN) BUFSIZE(MAX)	BUFSIZE STD BUFSIZE MIN BUFSIZE MAX
DECK NODECK	same
ESD NOESD	same
FLAG (0) FLAG(n)	FLAG 0 FLAG n
LIBMAC NOLIBMAC	same
LINECOUN (55) LINECOUN(nn)	LINECOUN 55 LINECOUN nn
LIST NOLIST	same
MCALL NOMCALL	same
MLOGIC NOMLOGIC	same
NUMbe <del>r</del>  NONUM	same
OB <del>J</del> ect NOOB <del>J</del> ect	same
PR <del>i</del> nt NOPR <del>i</del> nt DI <del>s</del> k	PR <del>i</del> nt DI <del>s</del> k
RENT NORENT	same
RLD NORLD	same
STMT NOSTMT	same
SYSPARM(string) SYSPARM(?) SYSPARM()	SYSPARM string SYSPARM ? SYSPARM SUP SUP SYSPARM EXP EXP
TERMi <del>n</del> al NOTERM	same
TEST NOTEST	same
WORKSIZE (2048K) WORKSIZE(nnnnK)	WORKSIZE 2048K WORKSIZE nnnnK
XREF (SHORT) XREF(FULL) NOXREF	XREF SHORT XREF FULL NOXREF
YFLAG NOYFLAG	same

**Note:** The defaults appear highlighted.

The SYSPARM SUP option suppresses the expansion of macros. The SYSPARM EXP option activates the expansion of macros. SYSPARM SUP is the default.

## HASM Options Supported by VMFASM

You can only enter *hasm\_options* when you use the ASM H option.

In the following table, the left column shows the options of the HASM command. The right column shows how these options are supported by VMFASM when invoking the HASM command. Also shown are the default values of these options. The HASM defaults are used wherever possible. Keyword-function options must be entered without the parentheses.

<b>HASM Option</b>	<b>VMFASM Option</b>
ALIGN NOALIGN	same
BATCH NOBATCH	same
DBCS NODBCS	same
DECK NODECK	same
ESD NOESD	same
FLAG (0) FLAG(n)	FLAG 0 FLAG n
LINECOUN (55) LINECOUN(nn)	LINECOUN 55 LINECOUN nn
LIST NOLIST	same
NUM NONUM	same
OBJECT NOOBJECT	OBJect NOOBJect
PRINT NOPRINT DISK	PRint Disk
RENT NORENT	same
RLD NORLD	same
STMT NOSTMT	same
SYSPARM(string) SYSPARM(?) <u>SYSPARM()</u>	SYSPARM string SYSPARM ? SYSPARM SUP SUP SYSPARM EXP EXP
TERM NOTERM	same
TEST NOTEST	same
XREF (FULL) XREF(SHORT) NOXREF	XREF FULL XREF SHORT NOXREF

**Note:** The defaults appear highlighted.

The SYSPARM SUP option suppresses the expansion of macros. The SYSPARM EXP option activates the expansion of macros. SYSPARM SUP is the default.

### **HLASM**

indicates the beginning of the HLASM options, which are passed directly to the HLASM command. VMFASM does not parse these options, the HLASM command performs the parsing.

#### ***hlasm\_options***

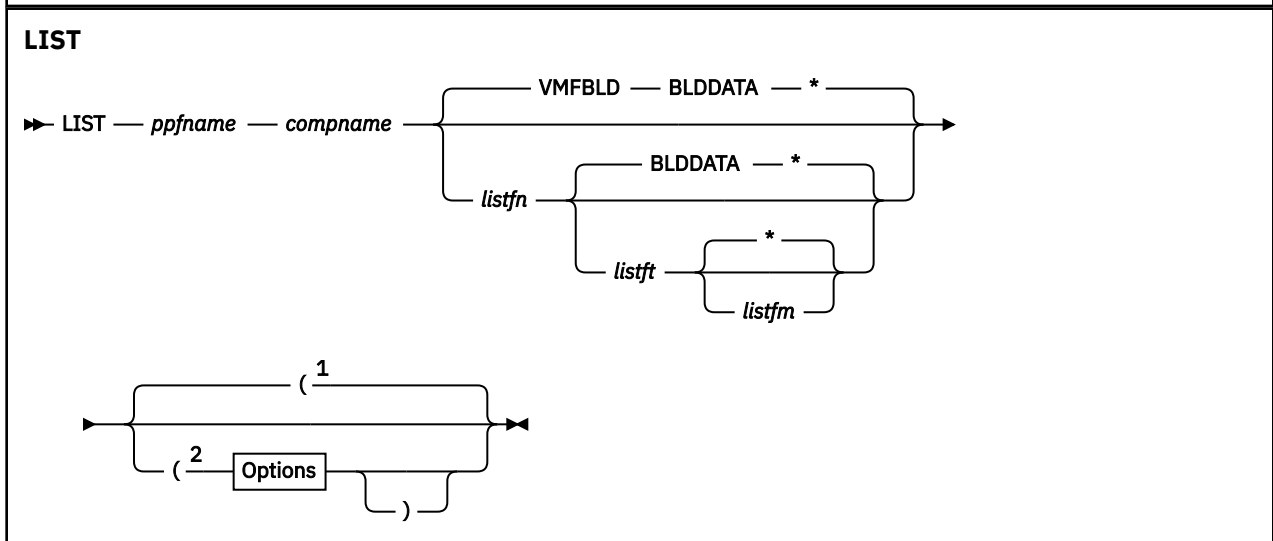
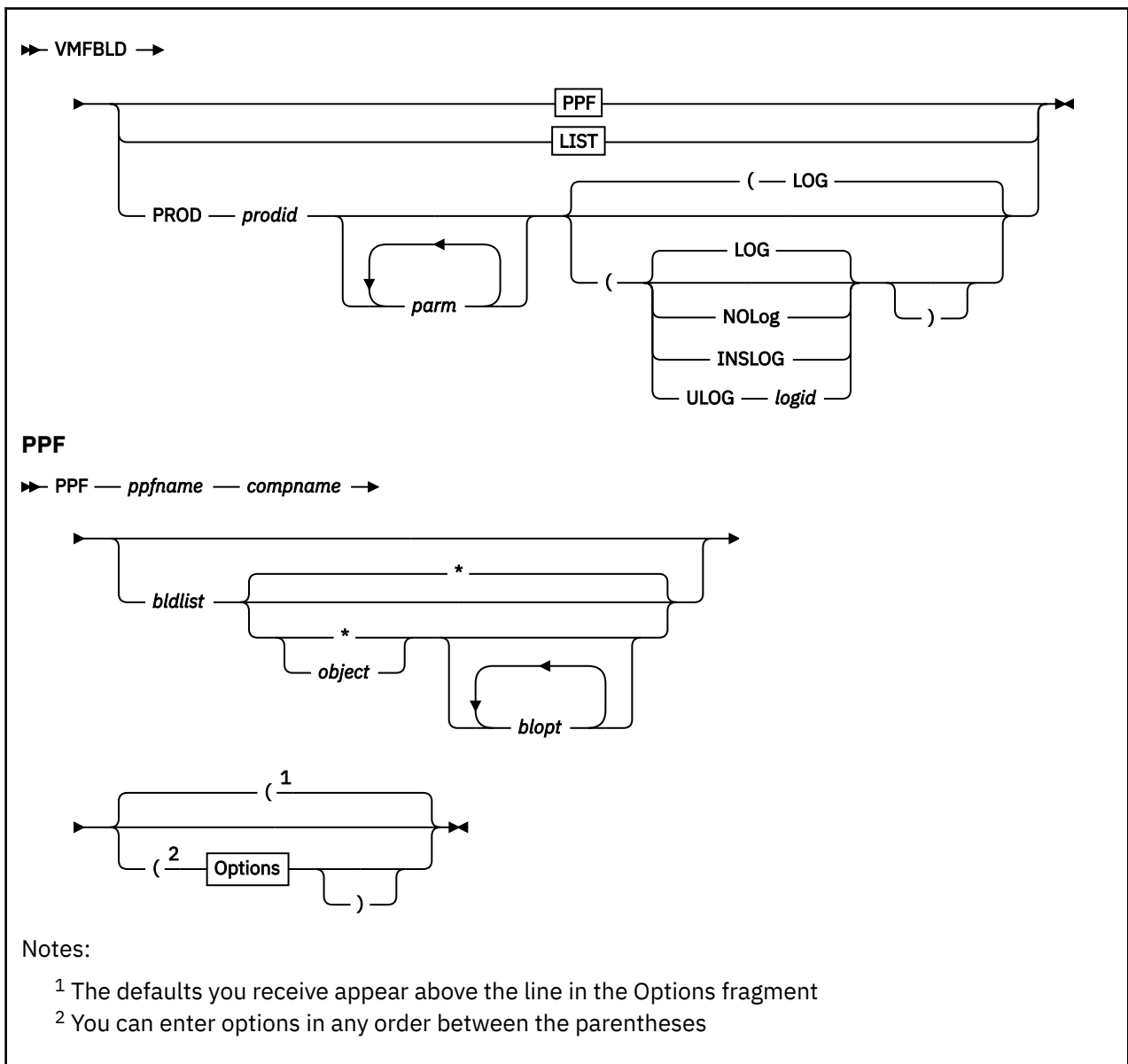
are the HLASM options. You can only enter *hlasm\_options* when you use the ASM HL option.

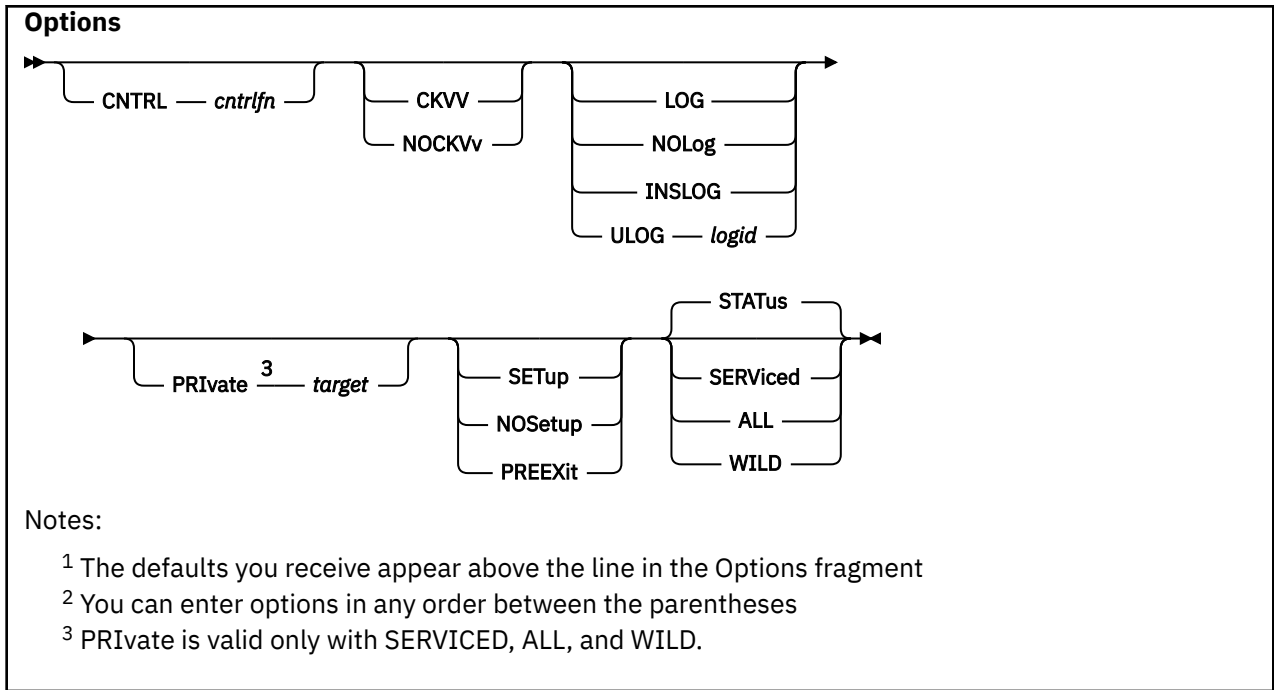
For a description of the HLASM options, see *IBM High Level Assembler/MVS & VM & VSE Programmer's Guide, SC26-4941*.

### **EHLASM**

indicates the end of the HLASM options.

# VMFBLD EXEC





## Purpose

Use the VMFBLD EXEC to build objects (such as MODULE files, nucleus load decks, saved segments, execs, XEDIT files, and so on). See the description of the STATUS option for information on how VMFBLD determines the build requirements.

## Operands

### PPF

indicates the product is fully supported by VMSES/E and the specified product parameter file is to be used for build processing.

This keyword is used when building any of the saved segments identified in a system saved segment build list, which may include entries for saved segments defined by several products, some of which may not be in VMSES/E format. Build parameters for saved segments used by non-VMSES/E products are defined in the saved segment data file associated with the system saved segment build list.

### ppfname

is the file name of the usable form product parameter file. The file type must be PPF.

When building saved segments, this is the saved segment product parameter file.

### compname

is the name of the component as it is specified on the :COMPNAME tag in the product parameter file. *compname* is a 1- to 16-character alphanumeric identifier.

When building saved segments, this is the name of the section in the saved segment product parameter file that contains control information for building all the saved segments on the system.

### bldlist

is the file name of a build list you want to process.

When building saved segments, this is the name of the system saved segment build list. It is also the name of the saved segment data file that contains build information for each saved segment.

### \*

indicates all objects in the build list should be built.

If an asterisk (\*) is used for *object*, the VMFBLD EXEC builds all objects in the specified build list that meet the build criteria established by any other options you include on the command.



**object**

is the name of an object in a build list that you want to build.

For format 1 build lists, the object name is BLDLIST.

For format 2 build lists, *object* can be either:

- the file name of the object, which builds all objects that have that file name.
- a fully-qualified object name, a file name and file type joined by a period (.), for example RECEIVE.EXEC. When you specify a fully-qualified object name, only that object is built by VMFBLD.

**blopt**

are build list options. Valid options depend on the part handler being used. See [“Build List Options”](#) on page 312 for valid build list options.

Build list options are generally operands or options of the command used to build objects. Build list options can be specified in the :BLD section of the product parameter file or when you enter a VMFBLD command. When you enter build list options on the VMFBLD command, they override the corresponding values in :BLD section of the product parameter file. They do not override corresponding object parameters that may appear in the build list. For more permanent overrides to the IBM supplied values, create product parameter file overrides.

**LIST**

indicates the product is fully supported by VMSES/E and the specified product parameter file is to be used to process all build lists and objects specified in the input file (VMFBLD BLDDATA A by default). See [“VMFBLD BLDDATA File”](#) on page 311 for the correct format for the input file.

**ppfname**

is the file name of the usable form product parameter file. The file type must be PPF.

**compname**

is the name of the component as it is specified on the :COMPNAME tag in the product parameter file. *compname* is a 1- to 16-character alphanumeric identifier.

**listfn**

is the file name of the CMS file containing the list of objects to build. The default is VMFBLD.

**listft**

is the file type of the CMS file containing the list of objects to build. When you specify *listft*, you must also specify *listfn*. The default is BLDDATA.

**listfm**

is the file mode of the disk containing the list of objects to build. When you specify *listfm*, you must also specify *listft*. The default is the \*.

**PROD**

indicates that the product to be built (*prodid*) is not fully supported by VMSES/E, and a product-specific exec is to be used to build it.

**prodid**

is the product identification number and the file name of the product-specific exec. The file type must be EXEC.

**parm**

are parameters you want to pass to the product-specific exec.

**Note:** VMFBLD enters the string 'BUILD' as the first parameter if it is not passed as a parameter by the product-specific exec.

**Options****CNTRL**

specifies that a control file is used to identify the AUX file structure.

***cntrlfn***

is the file name of the control file that is used to identify the AUX file structure. The file type of the control file is CNTRL. This value overrides the value on the :CNTRL tag in the PPF.

**CKVV**

requests validation of the AUX files against the version vector tables.

**NOCKVV**

requests no validation of the AUX files against the version vector tables.

**Note:** If the CKVV and NOCKVV options are omitted, the VMFBLD EXEC uses the value of the :CKVV tag in the product parameter file to determine whether to validate the AUX files against the version vector tables.

The CKVV and NOCKVV options are only used by part handlers that use the CMS UPDATE command to generate the serviceable parts that are included in the objects being built.

**LOG**

writes VMFBLD messages to the build message log (\$VMFBLD \$MSGLOG).

No messages are logged until initial validation of the command is complete.

**NOLOG**

does not write VMFBLD messages to the build message log (\$VMFBLD \$MSGLOG).

**Note:** If the LOG and NOLOG options are omitted, the VMFBLD EXEC uses the value of the :LOG tag in the product parameter file to determine whether to log VMFBLD messages in the build message log.

**INSLOG**

logs the messages in the \$VMFINS \$MSGLOG file.

**Note:** The INSLOG option is reserved for use by VMSES/E.

**ULOG**

writes VMFBLD messages to a user message log (\$VMFxxx \$MSGLOG).

***logid***

is a three-character message log identifier, for example:

<i>logid</i>	Type of LOG
XYZ	The user message log (\$VMFXYZ \$MSGLOG)

**PRIVate**

builds a private copy of each object on the target specified. The target specified on the PRIVATE option overrides the value specified in the :BLD section of the product parameter file for the build list being processed. The target must be accessed as Read/Write. When PRIVATE is specified, the service-level build status table is not updated. You must specify the SERVICED, ALL, or WILD option.

**Note:** When the PRIVATE option is specified, VMFBLD does not process any objects that have a status of DELETE.

***target***

is the virtual address of a minidisk or Shared File System (SFS) directory; file mode of a minidisk or SFS directory; or the name of a symbolic string of disks from the :MDA section of the PPF where the object is to be placed.

If a file mode is used with the SETUP operand, the SETUP is performed before a file mode is resolved. If you use the name of a symbolic string the output is placed on the first disk specified in this string.

When specifying the PRIVATE option there may be some ambiguity between the *fm* and the *disk* values. If for some reason you have a disk address that is A, B, C, D, E, or F you must specify the address with a leading **0**; otherwise the letter value is assumed to be a file mode (*fm*).

**SETup**

sets up a minidisk or directory access order for the build function according to entries in the :MDA section of the product parameter file (*ppfname*). If a user exit is specified in the product parameter file, setup will occur after the user exit is called.

**NOSetup**

does not set up a new access order.

**PREEXit**

sets up a minidisk or SFS directory access order for the build function according to entries in the :MDA section of the product parameter file. If a user exit is specified in the product parameter file, setup will occur before the user exit is called.

**Note:** If the SETUP, PREEXIT, and NOSETUP options are omitted, the VMFBLD EXEC uses the value of the :SETUP tag in the product parameter file to determine whether to set up a new access order.

**STATus**

identifies build requirements. A build requirement exists for an object when any of the following are true:

- Any serviceable parts included in the object have been serviced. All select data files (\$SELECT) identified on the :APPID tag in the product parameter file are processed to determine which parts have been serviced.
- Its object definition has been changed by service.
- It was requested from the VMFBLD command.
- It has a requisite for an object that meets any of the previous conditions.
- It has been deleted from the current level of the build list. These objects are given a status of DELETE.

New build requirements are added to the service-level build status table with a status of SERVICED or DELETE. You can use this as a planning step to see which objects have been serviced.

As part of STATUS option processing, newly-serviced source product parameter files are built to the A-disk. STATUS is the default.

**SERViced**

performs the STATUS function if any select data file (*appid \$SELECT*) specified on the :APPID tag in the product parameter file has been updated. Then builds the serviced objects as specified in [Table 18 on page 309](#). When you do not specify PRIVATE, SERVICED also builds objects flagged as DELETE in the service-level build status table.

**ALL**

performs the SERVICED function and in addition, builds all objects except wildcard objects specified on the command line. An initial status of BUILDALL is assigned to the objects in the service-level build status table.

**WILD**

performs the ALL function and in addition, builds all wildcard objects.

[Table 18 on page 309](#) shows how to specify what you want to build.

Bldlist	Object	Option	Objects Built
		STATUS	None
		SERVICED	All build requirements in the service-level build status table
		ALL	All non-wildcard objects plus all wildcard objects with build requirements in all build lists in the PPF

Table 18. VMFBLD EXEC Parameter Specifications and Objects Built (continued)

Bldlist	Object	Option	Objects Built
		WILD	All objects in all build lists in the PPF
X		STATUS	None
X		SERVICED	All build requirements for the specified build list in the service-level build status table
X		ALL	All non-wildcard objects plus all wildcard objects with build requirements in the specified build list
X		WILD	All objects in the specified build list
X	X	STATUS	None
X	X	SERVICED	The specified object, if there is a build requirement for it
X	X	ALL	The specified object if it is not a wildcard object or if it has a build requirement
X	X	WILD	The object specified
<p><b>Note:</b></p> <ul style="list-style-type: none"> <li>• The Objects Built column does not include any serviced source product parameter files that are automatically built as part of STATUS option processing.</li> <li>• When you use the LIST operand, each entry in the input file is processed as described in this table.</li> <li>• When you specify the SERVICED, ALL, and WILD options, VMFBLD also builds all build requisites for the object being built that have a status of SERVICED or BUILDALL. VMFBLD also processes all objects in the service-level build status table that have a status of DELETE, regardless of the build list or objects specified on the command line (except if the private option is specified).</li> </ul>			

## VMFBLD BLDDATA File

You can use the VMFBLD command with the LIST operand to process all build lists or objects specified in an input file. The default file name for this input file is VMFBLD BLDDATA. You can, however, give the input file another name. The contents of the file should conform to the following syntax.

### File Syntax

When you create the input file, *listfn listft* (or VMFBLD BLDDATA by default), make sure you follow this syntax.

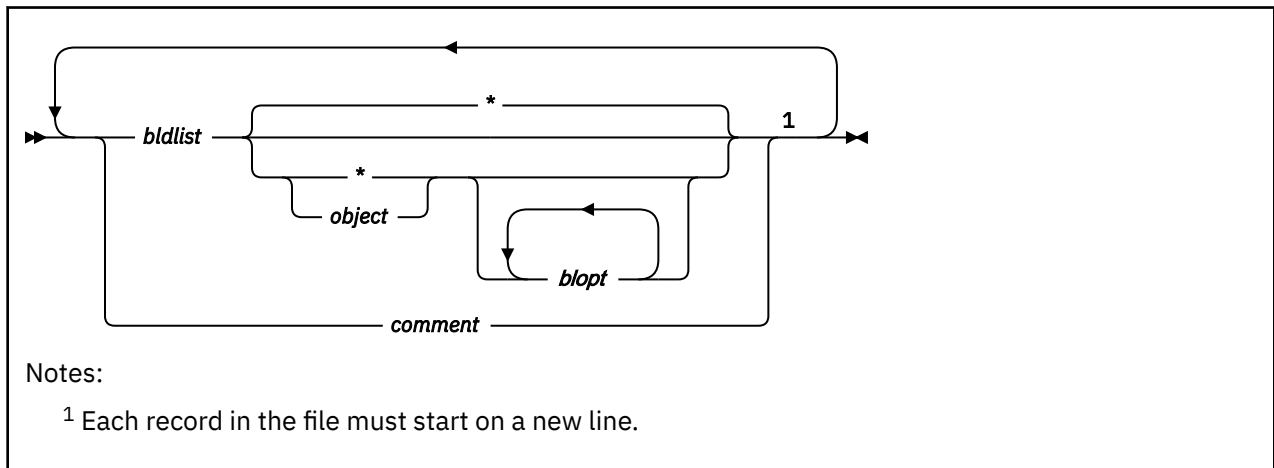


Figure 140. Syntax for VMFBLD LIST Input File (VMFBLD BLDDATA)

#### **bldlist**

is the file name of a build list you want to process.

**\***

indicates all objects in the build list should be built.

If you use an asterisk (\*) for *object*, the VMFBLD EXEC builds all objects in the specified build list that meet the build criteria established by any other options you include on the command.

#### **object**

is the name of an object in a build list that you want to build.

For format 1 build lists, the object name is BLDLIST.

For format 2 build lists, *object* can be either:

- The file name of the object, which builds all objects that have that file name.
- A fully-qualified object name, which is a file name and file type joined by a period (.), for example RECEIVE.EXEC. When you specify a fully-qualified object name, only that object and any required build requisites are built by VMFBLD.

For Format 3 build lists, the object name is the name of a member in a library.

#### **blopt**

are build list options. Valid options depend on the part handler being used. See [“Build List Options”](#) on page 312 for a complete description of build list options.

#### **comment**

represents comment records in the BLDDATA file. All records that start with an asterisk (\*) are treated as comment records and are ignored. Blank lines are also ignored.

## Build List Options

Table 19. Build List Options

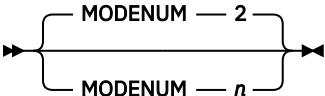
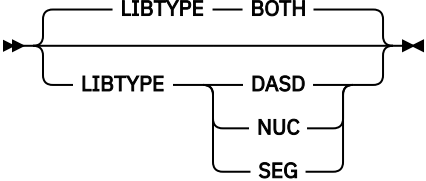
If You Are Building	Build List Options
Byte File System (VMFBDBFS)	<p>➤ loadbfs_options ➤</p> <p>specifies options that are passed to the LOADBFS command.</p>
Callable Services Libraries (CSL) (VMFBDCLB)	 <p>specifies the file mode number that is appended to the target mode of all objects in the build list. <i>n</i> must be a valid CMS file mode number. The default is 2.</p>  <p>specifies the format for the callable services library.</p> <p><b>BOTH</b> indicates two callable services libraries are to be built. One is formatted for DASD, and it has a file type of CSLLIB. One is formatted for use in a logical saved segment, and it has a file type of CSLSEG. BOTH is the default.</p> <p><b>DASD</b> indicates the callable services library is to be formatted for DASD, and the library will have a file type of CSLLIB.</p> <p><b>NUC</b> indicates the CSL is to be formatted for inclusion within the CMS nucleus, and the library will have a file type of TEXT.</p> <p><b>SEG</b> indicates the callable services library is to be formatted for use in a logical saved segment, and the library will have a file type of CSLSEG.</p>

Table 19. Build List Options (continued)

If You Are Building	Build List Options
Replacement Objects (VMFBDCOM)	<p data-bbox="479 252 625 283">▶▶ UPPER ◀◀</p> <p data-bbox="479 315 1485 388">indicates all files copied to the build target should use the uppercase option of the COPYFILE command.</p> <div data-bbox="479 409 803 514"> </div> <p data-bbox="479 546 1485 619">specifies the file mode number that is appended to the target mode of all objects in the build list. <i>n</i> must be a valid CMS file mode number. The default is 2.</p> <p data-bbox="479 640 641 672">▶▶ WRKST ◀◀</p> <p data-bbox="479 703 1485 777">indicates all files copied to the build target should also be downloaded to the workstation.</p> <p data-bbox="479 798 657 829">▶▶ HLVLCHK ◀◀</p> <p data-bbox="479 861 1485 997">specifies that VMFBDCOM will do highest release level checking when processing objects in the build list. Objects in the build list will always be built in a single-system environment, but will only be built when running the build on a member of an SSI cluster that is running the highest release level of the associated product.</p>
Replacement Text Only (VMFBDCPY)	<p data-bbox="479 1039 950 1176">▶▶ BLDREQ — <i>reqbldlist</i> — ◀◀</p> <div data-bbox="641 1050 917 1165"> </div> <p data-bbox="479 1207 1485 1333">specifies a build requisite. A build requisite must be built prior to the object that specifies it. <i>reqbldlist</i> is the build list that contains the requisite object, and <i>reqobj</i> is the requisite object. If <i>reqobj</i> is not specified, all objects in the specified build list are requisites.</p>
Full minidisks (VMFBDDDR)	<p data-bbox="479 1354 730 1396">No build list options.</p>
DOS Libraries (VMFBDDL B)	<div data-bbox="479 1480 803 1585"> </div> <p data-bbox="479 1617 1485 1690">specifies the file mode number that is appended to the target mode of all objects in the build list. <i>n</i> must be a valid CMS file mode number. The default is 2.</p> <p data-bbox="479 1711 779 1753">▶▶ DOSSEG — <i>segname</i> ◀◀</p> <p data-bbox="479 1785 1485 1900">specifies the name of the segment to use when VMFBDDL B issues SET DOS ON. If DOSSEG is not specified and DOS is not already set ON, the DOSINST segment is used.</p>

Table 19. Build List Options (continued)

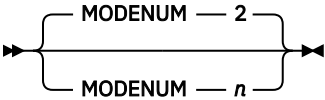
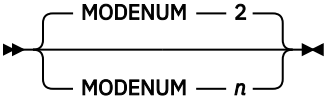
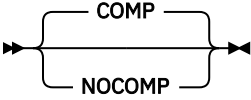
If You Are Building	Build List Options
Generated Objects (VMFBDGEN)	
	<p>specifies the file mode number that is appended to the target mode of all objects in the build list. <i>n</i> must be a valid CMS file mode number. The default is 2.</p>
	<p>➤ CNTRL — <i>cntrlfn</i> ➤</p>
	<p>specifies the file name that is used to identify the AUX file structure. The file type of the control file is CNTRL. This value overrides the value on the :CNTRL tag in the PPF.</p>
LOADLIBs (VMFBDLLB)	
	<p>specifies the file mode number that is appended to the target mode of all objects in the build list. <i>n</i> must be a valid CMS file mode number. The default is 2.</p>
MACLIBs (VMFBDMLB)	<p>➤ CNTRL — <i>cntrlfn</i> ➤</p>
	<p>specifies the file name of the control file to use when building the MACLIB. The control file specified on the build list option overrides the control file specified in the CNTRLOP section of the product parameter file.</p>
	<p>➤ ALTCNTRL ➤</p>
	<p>specifies the alternate control file, if it is specified in the CNTRLOP section of the product parameter file, is to be used instead of the control file when building the MACLIB.</p>
	
	<p>specifies whether the MACLIB should be compacted (unused space is removed). COMP is the default.</p>



Table 19. Build List Options (continued)

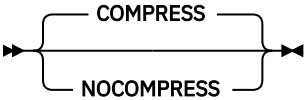
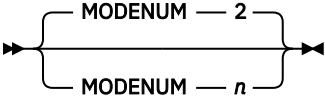
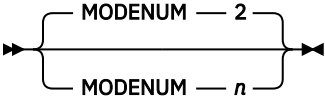
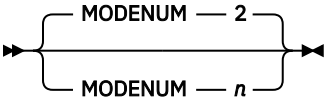
If You Are Building	Build List Options
MACLIBs (VMFBTMLB) <i>con't</i>	 <p data-bbox="483 401 1430 457">specifies whether the MACLIB should be compressed (comments are removed). COMPRESS is the default.</p>
Modules (VMFBDMOD)	 <p data-bbox="483 632 1471 688">specifies the file mode number that is appended to the target mode of all objects in the build list. <i>n</i> must be a valid CMS file mode number. The default is 2.</p>
Modules (VMFBDMOD)	 <p data-bbox="483 873 1471 930">specifies the file mode number that is appended to the target mode of all objects in the build list. <i>n</i> must be a valid CMS file mode number. The default is 2.</p> <p data-bbox="483 968 656 993">▶▶ HLVLCHK ▶▶</p> <p data-bbox="483 1031 1455 1157">specifies that VMFBDMOD will do highest release level checking when processing objects in the build list. Objects in the build list will always be built in a single-system environment, but will only be built when running the build on a member of an SSI cluster that is running the highest release level of the associated product.</p>
Modules with CPLINK (VMFBPMD)	 <p data-bbox="483 1346 1471 1402">specifies the file mode number that is appended to the target mode of all objects in the build list. <i>n</i> must be a valid CMS file mode number. The default is 2.</p> <p data-bbox="483 1440 656 1465">▶▶ HLVLCHK ▶▶</p> <p data-bbox="483 1503 1455 1629">specifies that VMFBPMD will do highest release level checking when processing objects in the build list. Objects in the build list will always be built in a single-system environment, but will only be built when running the build on a member of an SSI cluster that is running the highest release level of the associated product.</p>

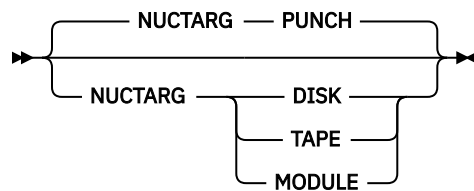
Table 19. Build List Options (continued)

If You Are Building	Build List Options
Nuclei (VMFBDNUC)	<p data-bbox="483 254 704 279">▶▶ CNTRL-cntrlfn ▶▶</p> <p data-bbox="529 312 1456 405">specifies the filename of the control file to use when building the nucleus. The control file specified on the build list option overrides the control file specified in the CNTRLOP section fo the product parameter file.</p> <p data-bbox="483 443 669 468">▶▶ ALTCNTRL ▶▶</p> <p data-bbox="483 501 1435 594">specifies the alternate control file, if it is specified in the CNTRLOP section of the product parameter file, is to be used instead of the control file when building the nucleus.</p> <p data-bbox="483 632 669 657">▶▶ FASTPATH ▶▶</p> <p data-bbox="483 690 1419 751">specifies the loader should use the text deck entries from the previous nucleus build. These are included in the <i>bldlist</i> \$NUCEXEC file.</p>

Table 19. Build List Options (continued)

If You Are Building	Build List Options
---------------------	--------------------

Nuclei  
(VMFBNUC)  
*con't*



specifies the target for the nucleus or nucleus module.

**PUNCH**

causes the nucleus to be created as an IPLable file in the virtual reader. NUCTARG PUNCH is the default.

**DISK**

causes the nucleus to be link-edited and loaded directly to the system residence device.

**TAPE**

causes the nucleus to be created as an IPLable file on the tape device at virtual address 182.

**MODULE**

causes the nucleus to be created as a link-edited CMS module on the A-disk.



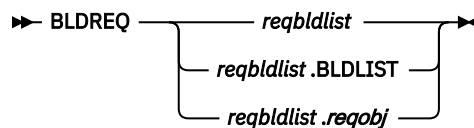
specifies which loader to call. If no loader is specified, the default is the loader identified in the nucleus build list.

➤ **RLDSAVE** ➤

indicates the relocation dictionary is to be saved. This option is only valid when NUCTARG MODULE is specified.

➤ **MODNAME** — *fn* ➤

specifies the name of the nucleus module. This option is only valid when NUCTARG MODULE is specified. If you do not specify MODNAME, the build list name is used as the default module name.



specifies a build requisite. A build requisite must be built prior to the object that specifies it. *reqbldlist* is the build list that contains the requisite object, and *reqobj* is the requisite object. If *reqobj* is not specified, all objects in the specified build list are requisites.

**Note:** The NUCTARG MODULE and RLDSAVE options are only valid when HCPLDR is used.

---

*Table 19. Build List Options (continued)*

---

<b>If You Are Building</b>	<b>Build List Options</b>
Segments (VMFBDSBR)	No build list options.

---

Table 19. Build List Options (continued)

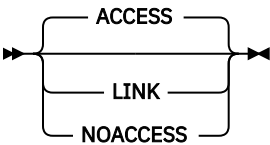
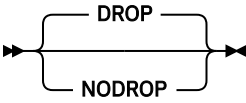
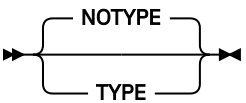
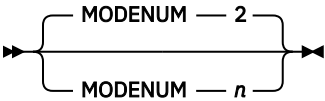
If You Are Building	Build List Options
Segments (VMFBDSEG)	
	<p>indicates whether VMFBDSEG should and how VMFBDSEG should call VMFSETUP when you build a segment that uses a product parameter file. The product parameter file is specified on the :BLDPARMS tag for the segment in the SEGDATA file.</p>
	<p><b>ACCESS</b> calls VMFSETUP to access the product disks (the disks must already be linked). The accessed disks are released after the segment is built. ACCESS is the default.</p>
	<p><b>LINK</b> calls VMFSETUP with the LINK option to link and access the product disks. The accessed disks are released, and the linked disks are detached after the segment is built.</p>
	<p><b>NOACCESS</b> does not call VMFSETUP. The disks must already be linked and accessed.</p>
	<p><b>Note:</b> These build list options do not affect the SETUP and NOSETUP options specified on the VMFBLD command.</p>
	
	<p>indicates whether VMFBDSEG should drop all or just new nucleus extensions after each segment is built.</p>
	<p><b>DROP</b> indicates VMFBDSEG should release all segments on entry. Before building the first segment and after each segment is built, all nucleus extensions are dropped. DROP is the default.</p>
	<p><b>NODROP</b> indicates VMFBDSEG should not release all existing segments. Before building the first segment and after each segment is built, only those nucleus extensions that were not already loaded on entry to VMFBDSEG are released.</p>
	
	<p>indicates whether VMFBDSEG should show the messages issued by SEGGEN.</p>
	<p><b>TYPE</b> indicates VMFBDSEG should show the messages issued by SEGGEN. This option is passed to SEGGEN.</p>
	<p><b>NOTYPE</b> indicates VMFBDSEG should not show the messages issued by SEGGEN. This option is passed to SEGGEN.</p>

Table 19. Build List Options (continued)

If You Are Building	Build List Options
TXTLIBs (VMFBDTLB)	
	<p>specifies the file mode number that is appended to the target mode of all objects in the build list. <i>n</i> must be a valid CMS file mode number. The default is 2.</p>

## Usage Notes

1. Most VMFBLD part handlers ignore the CKVV option, because they deal with parts supported by the version vector table only.
2. VMFBLD requires the alternate APPLY disk and any build disks that are the targets of specified objects to be accessed as Read-Write.
3. VMFBLD requires the entire APPLY, DELTA, LOCAL, and BASE strings to be accessed.
4. When you use the LIST operand:
  - VMFBLD does not necessarily build the objects in the list file in the order they are specified. The order in which they are built is determined by build requisite relationships and by grouping objects in the same build list to minimize the number of calls to the part handlers.
  - Options specified on the VMFBLD command apply to all of the objects being built. If you specify build list options in the list, VMFBLD concatenates all of the options specified for the objects that are being built as a group.
5. When you use the PRIVATE option, the results of the VMFBLD status function are used to determine what needs to be built. The service-level build status table, however, is not updated with the results from the status function.
 

**Note:** The target on the PRIVATE option is ignored when you are building some objects, such as saved segments and the GCS and CMS nucleus.
6. Build list syntax is defined in [“Build Lists” on page 141](#).
7. You should not remove build lists from product parameter files.
8. VMFBLD determines the objects to be deleted by reading the select data file to determine which build lists have been serviced. (When a build list has been serviced, a third token is added to the select data file entry.) VMFBLD then compares the previous level of the build list to the new level of the build list. VMFBLD gives the status of DELETE to any object found in the previous level of the build list that is not found in the new serviced level.
9. Unless the NODROP build list option is specified, all saved segments are released and nucleus extensions are dropped before saved segments are built. Specify NODROP when there are segments or nucleus extensions you do not want VMFBDSEG to release. Because VMFBDSEG will not release storage reserved by any segments or nucleus extensions in this case, it is more likely the storage needed by a segment to be built will not be available. To minimize this possibility, release all but the needed segments and nucleus extensions before you invoke VMFBLD.
10. When NUC is specified for VMFBDCLB:
  - All history is stored in the nucleus map.
  - No checking of the existence of the TEXT files associated to a ROUTINE will be done. It is assumed they already reside within the CMS nucleus. These text names are still needed to set the proper linkage within the library TEXT file.
  - A CSL that is defined to be built with the NUC option, can still be built using the options DASD, SEG or BOTH.

- Changes to text files associated with a ROUTINE result in rebuilding a CSL. If the NUC option is specified for the build, however, the generated TEXT output does not change because the updated text files are not used until the nucleus build.
11. When specifying the PRIVATE option there may be some ambiguity between the *fm* and the *disk* values. If for some reason you have a disk address that is A, B, C, D, E, or F you must specify the address with a leading **0**; otherwise the letter value is assumed to be a file mode (*fm*).
  12. If the WILD option is specified, VMFBLD builds wildcard objects by first searching for parts that have a local, serviced, or base file type that matches the object's :PARTID tag. VMFBLD also searches any BASE disks for parts with the real file type of the object. All parts found are then built using the highest level of each part as determined by the version vector tables.

## Examples

- To use VMFBLD to determine the build requirements for a fully-supported product using the IBM-supplied defaults, enter:

```
VMFBLD PPF ppfname compname
```

- To run VMFBLD to build all serviced objects for a fully-supported product using the IBM-supplied defaults, enter:

```
VMFBLD PPF ppfname compname (SERVICED
```

- To run VMFBLD to build all objects (except for wildcard objects that are not serviced) for a fully-supported product using the IBM-supplied defaults, enter:

```
VMFBLD PPF ppfname compname (ALL
```

- To run VMFBLD to build all objects (including wildcard objects that are not serviced) for a fully-supported product using the IBM-supplied defaults, enter:

```
VMFBLD PPF ppfname compname (WILD
```

- To run VMFBLD to build all serviced objects in a single build list for a fully-supported product using the IBM-supplied defaults, enter:

```
VMFBLD PPF ppfname compname bldlist (SERVICED
```

- To run VMFBLD for a product that is not fully supported, using the IBM-supplied defaults, enter:

```
VMFBLD PROD prodid parms
```

- To run VMFBLD to build all the saved segments defined in the system saved segment build list, enter:

```
VMFBLD PPF ppfname compname bldlist (ALL
```

- To run VMFBLD to build a list of segments that have been serviced, enter:

```
VMFBLD LIST ppfname compname listfn listft listfm (SERVICED
```

- To run VMFBLD to build a test version of an object and not update the service-level build status table, enter:

```
VMFBLD PPF ppfname compname bldlist object (ALL PRIVATE 191
```

- To run VMFBLD to build a specific saved segment whether or not it has been serviced, enter:

```
VMFBLD PPF ppfname compname bldlist segname (ALL
```

- To run VMFBLD to build a specific saved segment only if it has been serviced, enter:

```
VMFBLD PPF ppfname compname bldlist segname (SERVICED
```

- To build all serviced segments using the LINK build list option to link and detach product disks, enter:

```
VMFBLD PPF ppfname compname bldlist * LINK (SERVICED
```

- To build all serviced segments using the NODROP option to not drop the segments or nucleus extensions issue:

```
VMFBLD PPF SEGBLD ESASEGS * NODROP (SERVICED
```

## Input and Output Files

### Input Files

#### ***ppfname* PPF**

The usable form product parameter file.

#### ***appid* \$SELECT**

The select data file.

#### ***appid* SRVAPPS**

The service-level apply status table.

#### ***appid* VVTlvlid**

The version vector tables specified by the control file specified on the :CNTRL tag in the product parameter file.

#### ***listfn* listft**

The file containing the list of build lists and objects to process.

#### ***appid* \$APRCVRY**

Used for apply recovery. The existence of this file on the APPLY string indicates VMFAPPLY was interrupted during critical processing on the last invocation of VMFAPPLY for the specific component.

#### ***cntrlfn* CNTRL**

The control file identified by the :CNTRL tag in the product parameter file (PPF) or the CNTRL option of the VMFBLD command.

#### **VM SYSABRVT**

The file type abbreviation table.

#### **VMFNLS LANGLIST**

National language support table.

#### **VMSESE PROFILE**

The file that identifies the minidisk or directory location of the VMSBR \$SELECT file.

#### ***bldlist* EXCnnnnn**

The build list.

#### ***bldlist* SEGDATA**

The saved segment data file.

#### **SERVICE \$PRODS**

The place into production file that is a list of products and objects that were serviced. For more information, see [“Place Into Production Files” on page 135](#).

#### **VM SYSAPARS**

The system-level Base APAR table.

#### **VM SYSPINV**

The system-level product inventory table.

### Input/Output Files

#### ***bldid* SRVBLDS**

The service-level build status table.

#### **VMSBR \$SELECT**

The select data file for system objects, such as saved segments.



**appid \$HLVLSRV**

List of serviced parts and associated build requirements for cross-system highest release level objects.

**Output Files****\$VMFBLD \$MSGLOG**

The build message log.

**VMSES PARTCAT**

The parts catalog table.

**filename TEXT**

TEXT files created when you use the COPY option on the CSLGEN command.

**fname DLKEDERR**

The DOS library error file that is created or updated. *fname* is the name of the phase.

**libname CSLLIB**

The DASD version of the Callable Services Library that is created.

**libname CSLSEG**

The segment version of the Callable Services Library that is created.

**libname DLKEDIT**

The DOS library history file that is created or updated.

**libname DOSLIB**

The DOS library that is created or updated.

**libname LIBMAP**

The map for the DASD version of the Callable Services Library that is created.

**libname SEGMAP**

The map for the segment version of the Callable Services Library that is created.

**libname TEXT**

File containing the nucleus version of the Callable Services Library.

**libname TXTLIB**

The text library for the Callable Services Library that is created.

**objname MAP**

Load map created when a map is requested.

**objname \$MAP**

Load map created when a module build fails and a map is requested.

**objname LISTING**

Listing created when the DISK option is specified on the generation routine.

**objname cntrlfn**

Listing created when the DISK and NOLIST options are specified on the generation routine.

**bldlist \$NUCEXEC**

Load list created with all file specifications resolved.

**bldlist MAP**

Load map created when a map is requested.

**Temporary Files****\$LLB\$ LOADLIB**

Work copy of the LOADLIB being serviced.

**\$LLB\$ LKEDMAP**

Work copy of the link editor load map created when servicing a LOADLIB.

**\$LLB\$ TEXT**

File containing the link editor control statements to service a LOADLIB member.

**\$MLB\$ MACLIB**

Work copy of the MACLIB being serviced.

**\$MLB\$ COPY**

File created and included in the MACLIB being serviced to indicate the level of all members.

***objname* \$HISTORY**

File containing a header record and service level summary for TXTLIB and MACLIB members.

***objname* UPDATES**

File created to be used in generating the service level summary.

***\$objname* MODULE**

Work copy of the module being created.

***\$objname* MAP**

Work copy of the load map created when building a module.

**\$\$\$TLL\$ EXEC**

Temporary load list.

**\$TLB\$ TXTLIB**

Work copy of the TXTLIB being serviced.

**\$TLB\$ TEXT**

Text file with the service level summary of all TXTLIB members imbedded.

**\$TLB\$ \$HISTORY**

Collection of the \$HISTORY files created for each member in a TXTLIB.

***objname* TIMESTMP**

File created to record time stamp and user ID information that is used to create the service level summary.

**\$CLB\$ CSLCNTRL**

The CSLCNTRL file created from the build list.

**\$CLB\$ CSLLIB**

The DASD version of the Callable Services Library that is created.

**\$CLB\$ CSLSEG**

The segment version of the Callable Services Library that is created.

**\$CLB\$ TXTLIB**

The text library for the Callable Services Library that is created.

**\$CLB\$ LIBMAP**

The map for the DASD version of the Callable Services Library that is created.

**\$CLB\$ SEGMAP**

The map for the segment version of the Callable Services Library that is created.

***partname* ASSEMBLE**

A temporary copy of the source assemble file.

**\$DLB\$ DOSLIB**

A temporary copy of the DOS library.

**\$DLBB\$ DOSLIB**

A backup temporary copy of the DOS library.

**\$DLB\$ DLKEDIT**

A temporary copy of the DOS library history file.

***fname* DOSLNK**

A link edit control file that is used as input to the DOSLKED command. *fname* is the name of the phase.

***partname* objtype**

A temporary copy of the text file that is created.

**PPF Tags Used****:BLD**

Defines the build lists for the product and the part handlers and target strings associated with them.

**:CNTROL**

Defines the control options for the VMFBLD command and the VMFBLD part handlers.

**:DABBV**

Defines file type abbreviations specific to a product and the real and base file types associated with them.

**:MDA**

Defines symbolic strings and the minidisks or SFS directories associated with them.

## Messages and Return Codes

Appendix D, “Module Identifiers for VMSES/E Messages,” on page 749 shows the format of VMSES/E messages and lists the identifiers used by each VMSES/E exec.

For a complete explanation of any VMSES/E message, use the HELP Facility to view the message explanation online or see the appropriate messages documentation. To display information on a specific message, for example VMF002E, enter:

```
help msg vmf002e
```

If you are unfamiliar with the z/VM HELP Facility, enter help to display the main HELP Menu. For information on the HELP command, enter:

```
help cms help
```

**PI**

These return codes may be returned to a user exit by VMFBLD. For more information about user exits, see [:USEREXIT.](#)

The VMFBLD EXEC issues the following return codes:

Return Code	Explanation
0	Command completed successfully.
4	Command completed with one or more warning conditions.
8	Command completed but at least one major process failed.
12	Command failed because of an internal error.
24	Command failed because of a command line syntax error.
28	Command failed because a required file was not found.
36	Command failed because a target disk or directory was not available.
100	Command failed because of an external error.
500	User terminated the command from a prompt.

**PI end**

## Recovery Information

The VMFBLD command can be restarted by reissuing the command.

## Creating Objects with VMFBLD

VMFBLD uses a number of part handlers to process different types of parts and objects. The following sections briefly describe each part handler, the function it provides, and the available options and parameters. Usage notes and examples are also provided.

The following terms are used throughout this section of the book:

## Callable Services Libraries (CSL)

### **build target**

is a minidisk or Shared File System directory that is defined to be part of a BUILD string in the :MDA (minidisk/directory assignment) section of the product parameter file. The BUILD string contains the objects after they are built.

### **build list options**

are options that apply to an entire build list. They are used by the build part handlers or other commands that are used to generate the object.

### **object parameters**

are values that apply to only one object definition in a build list. They are used by the build part handlers or other commands that are used to generate the object. Object parameters take precedence over build list options.

### **part options**

are options that apply to one or more serviceable parts in an object definition. An :OPTIONS record remains in effect for each of the remaining parts of an object unless it is superseded by another :OPTIONS record. You can use a blank :OPTIONS record to turn off the previous :OPTIONS record. They are used by the build part handlers or other commands that are used to generate the object. Part options take precedence over object parameters.

## Callable Services Libraries (CSL)

VMFBLD uses the VMFBDCLB part handler to build callable services libraries using the information in the VMSES/E build list.

VMFBDCLB produces a CSL library, CSLLIB, CSLSEG or TEXT. In addition you can get TXTLIBs, maps, and text decks depending on the build list options specified. VMFBDCLB also provides information to the VMFBLD command so the service-level build status tables are updated.

VMFBDCLB supports the function of all CSL control statements, except the CSLCNTRL statement, as build list part options. For more information, see [“Part Options \(CSL\)”](#) on page 328. VMFBDCLB supports all valid CSLCNTRL statement options as part options on a :OPTIONS tag, except the FILETYPE option on the ROUTINE statement. All other options are passed to the CSLGEN command, and VMFBDCLB performs no error checking on them.

### **Build List Format (CSL)**

VMFBDCLB uses format 3 build lists. For more information on build lists, see [“Build Lists”](#) on page 141.

### **Example (CSL)**

[Figure 141](#) on page 327 is an example of a format 3 build list used by VMFBDCLB.

```

:FORMAT. 3
:LIBNAME. ACSLLIB
:GBLDREQ. DMSBLBAS.BLDLIST
:OBJNAME. ACSLLIB
:OPTIONS. TXTLIB DMSBASE DMSAENV
:OPTIONS. ROUTINE VMABND PATH 1024.080 MP CSECT VMABND
:PARTID. DMSABM TXT
:PARTID. DMSABM TMP
:OPTIONS. ROUTINE VMEVMCR PATH 1024.094 MP CSECT VMEVMCR
:PARTID. DMSEMC TXT
:PARTID. DMSEMC TMP
:OPTIONS. ALIAS VTEVMCR PATH 1024.094 MP COPY
:OPTIONS. ROUTINE DMSEXIST
:PARTID. DMSJEX TXT
:PARTID. DMSJEX TMP
:OPTIONS. INCLUDE
:PARTID. DMSCSL TXT
:OPTIONS. ROUTINE DMSQWUID
:PARTID. DMSJQW TXT
:PARTID. DMSJQWTP TMP
:EOBJNAME.

```

Figure 141. Example Build List Used by VMFBDCLB

Entering this command:

```
vmfbld ppf ppfname compname bldlist acsllib (all
```

causes the object, ACSLLIB, to be built and creates a callable services library named ACSLLIB.

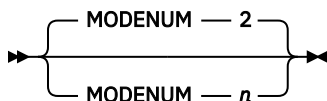
## Build List Restrictions (CSL)

VMFBDCLB:

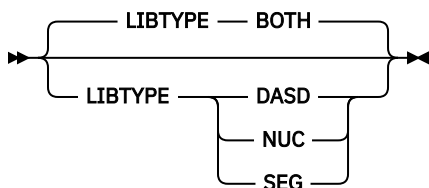
- Requires exactly one object block
- Requires at least one :PARTID tag in the object block
- Ignores all :GLOBAL and :GGLOBAL values.

## Build List Options (CSL)

VMFBDCLB uses the following build list options:



specifies the file mode number that is appended to the target mode of all CSL generated files except TEXT files. CSL generated TEXT files retain the file mode number set by CSLGEN. *n* must be a valid CMS file mode number. The default is 2.



specifies the format for the callable services library.

### DASD

indicates the callable services library is to be formatted for DASD, and the library will have a file type of CSLLIB. The associated map has a file type of LIBMAP.

## Callable Services Libraries (CSL)

### SEG

indicates the callable services library is to be formatted for use in a logical saved segment, and the library will have a file type of CSLSEG. The associated map has a file type of SEGMAP.

### BOTH

indicates two callable services libraries are to be built. One is formatted for DASD, and it has a file type of CSLLIB. One is formatted for use in a logical saved segment, and it has a file type of CSLSEG. The associated maps have a file type of LIBMAP and SEGMAP. BOTH is the default.

### NUC

indicates the callable services library is to be formatted for inclusion within the CMS nucleus, and the library will have a file type of TEXT.

## Library Parameters (CSL)

VMFBDCLB uses the following library parameters:

### ►► NOERASE ◄◄

is tolerated and has no effect on CSLLIB part handler since a CSLLIB can only be updated by being totally rebuilt.

## Object Parameters (CSL)

VMFBDCLB uses the following object parameters:

### ►► LDRTBLS — *nnn* ◄◄

defines the number of pages of storage to be used for loader tables by issuing the SET LDRTBLS command. *nnn* is a whole number ranging from 1 to 127 inclusive. The value is reset once the object build is complete.

## Part Options (CSL)

VMFBDCLB uses the following part options:

### ►► IGNORE ◄◄

indicates the part can be ignored if it does not exist. By ignoring a part, an object can still be considered built even if the part is not included in it. A part can only be ignored if it has no service history. If a service level is defined for the part and it is missing, it is an error condition. This part option is intended for parts for which no base level part is shipped (for example, help files and user exits).

### ►► LANGFUNC — *exec* ◄◄

specifies a language function to be used in the selection of the correct serviceable part.

### ►► NOGETLVL ◄◄

indicates the VMFSIM GETLVL function should not be used for this part. The values provided on the :PARTID tag are the specifications of the file used to build this part.

### ►► ROUTINE — *rtname* — *rtopts* ◄◄

defines a callable services library routine.

#### ***rtname***

is the CSL ROUTINE name.

***rtnopts***

are any CSL ROUTINE options. These are only valid on the :OPTIONS tags which precede the first :PARTID tag in a ROUTINE object block.

**➤ INCLUDE ➤**

indicates that a callable services library INCLUDE statement is being defined.

**➤ ALIAS — *rtnalias* — PATH — *path* — *alopts* ➤**

defines a callable services library routine alias.

***rtnalias***

is the CSL alias name.

**PATH**

is a CSL keyword.

***path***

is the path assigned for this alias name.

***alopts***

are any CSL ALIAS options. These are only valid on the :OPTIONS tags which precede the first :PARTID tag in an ALIAS object block.

**➤ TEXT ➤**

indicates a member of the callable services library TXTLIB is being defined.

**➤ TXTLIB ➤**

indicates a TXTLIB that must be made global when the callable services library is built.

**Usage Notes (CSL)**

1. The :LIBNAME tag is not required. If it is omitted, the library file name is the same as the build list file name.
2. The entire callable services library is built if any part has been serviced.
3. If no objects are listed in the build list and a build is requested, VMFBDCLB erases the CSLLIB, CSLSEG, TXTLIB, and maps (LIBMAP and SEGMAP). No text decks are erased if they were created.
4. If you specified the target disk as the A-disk and the disk fills up before the library is generated, the existing library remains unchanged. If the library is generated (that is, the CSLGEN command completes successfully) and the A-disk fills up before VMFBLD processing completes, the A-disk may contain both old and new levels of the library files.
5. For more information on callable services library routines and aliases, see [z/VM: CMS Application Development Guide for Assembler](#).
6. When NUC is specified for VMFBDCLB:
  - All history is stored in the nucleus map.
  - No checking of the existence of the TEXT files associated to a ROUTINE will be done. It is assumed that they already reside within the CMS nucleus. These text names are still needed to set the proper linkage within the library TEXT file.
  - A callable services library, that is defined to be built with the NUC option, can still be built using the options DASD, SEG, or BOTH.
  - Changes to text files associated with a ROUTINE result in rebuilding a callable services library. However, if the NUC option is specified for the build, the generated TEXT output does not change since the updated text files are not used until the nucleus build.

## Restore from DDR Image Files

VMFBLD uses the VMFBDDDR part handler to select the latest level of DDR image files and sets up the environment to allow PUT2PROD to restore the DDR image files to the appropriate production minidisk.

### Build List Format (DDR Image Files)

VMFBDDDR uses format 2 build lists. For more information on build lists, see [“Build Lists” on page 141](#).

#### Example (DDR Image Files)

Figure 142 on page 330 is an example of a format 2 build list used by VMFBDDDR.

```
:FORMAT. 2
:OBJNAME. LOHCOST.DDR
:PARTID. LOH1 CKD
:PARTID. LOH2 CKD
:PARTID. LOH1 FBA
:PARTID. LOH2 FBA
:EOBJNAME.
```

Figure 142. Example Build List Used by VMFBDDDR

Entering this command:

```
vmfbl d ppf ppfname compname bldlist lohcost.ddr ( all
```

causes the highest level of the DDR image file(s) to be copied to the test disk, and the environment is primed for PUT2PROD to be able to restore the DDR image file(s) to the production minidisk.

### Build List Restrictions (DDR Image Files)

VMFBDDDR requires at least one :PARTID tag per object block. Since each DDR image file is device dependent, there should be at least one image file for CKD devices and one image file for FBA devices. If the minidisk is large enough, there can be multiple image files for each minidisk. Each individual image file can be a portion of the minidisk, but there should be image files to cover all cylinders or blocks of the minidisk.

### Build List Options (DDR Image Files)

None.

### Object Parameters (DDR Image Files)

None.

### Part Options (DDR Image Files)

None.

### Usage Notes (DDR Image Files)

1. The :OBJNAME tags are not used in the build process except to identify the objects.
2. If an object is removed from a VMFBDDDR build list, the object is marked as DELETED, but no DDR image file processing occurs.



## CMS/DOS Phase Libraries (DOSLIB)

VMFBLD uses the VMFBDDL part handler to build CMS/DOS phase libraries (DOSLIBs) using the information in the VMSES/E build list. Each DOS library is represented by a build list that contains one or more object blocks. Each object represents a phase (member) in the DOS library.

VMFBDDL produces a DOSLNK file for each phase in the DOSLIB build list that is to be added or replaced, and calls the DOSLKED command to update the library. Phases are deleted from the library using the DOSLIB DEL command.

### Build List Format (DOSLIB)

VMFBDDL uses format 3 build lists. For more information on build lists, see [“Build Lists” on page 141](#).

#### Example (DOSLIB)

Figure 143 on page 331 is an example of a format 3 build list used by VMFBDDL. In this example, the second :OPTIONS record is necessary to turn off the parts in that object.

```
:FORMAT. 3
:LIBNAME. CMSVSAM
:OBJNAME. IKQVVS ACTION NOAUTO
:PARTID. IKQVVS TXT
:OPTIONS. ENTRY DMSVIP
:PARTID. DMSVIP TXT
:OPTIONS.
:PARTID. IKQSMDBM TXT
:EOBJNAME.
:OBJNAME. IKQVVN PHPARMS ,*,NOAUTO
:PARTID. IKQVVN TXT
:EOBJNAME.
:OBJNAME. IKQVRT PHPARMS ,+0
:PARTID. IKQVRT TXT
:EOBJNAME.
```

Figure 143. Example Build List Used by VMFBDDL

Entering this command:

```
vmfbld ppf ppfname compname bldlist ikqvvs (all
```

causes the phase IKQVVS to be built in the CMSVSAM DOSLIB.

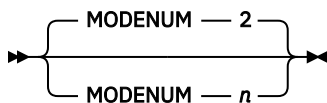
### Build List Restrictions (DOSLIB)

VMFBDDL:

- Requires at least one :PARTID tag per object block
- Ignores all :GGLOBAL and :GLOBAL values

### Build List Options (DOSLIB)

VMFBDDL uses the following build list options:



specifies the file mode number that is appended to the target mode of all objects in the build list. *n* must be a valid CMS file mode number. The default is 2.

➤ DOSSEG — *segname* ➤

specifies the name of the segment to use when VMFBDDL issues SET DOS ON. If DOSSEG is not specified and DOS is not already set ON, the DOSINST segment is used.

### Library Parameters (DOSLIB)

VMFBDDL uses the following library parameters:

#### ►► NOERASE ◄◄

indicates to the part handler the library is not to be erased during the build.

### Object Parameters (DOSLIB)

VMFBDDL uses the following object parameters:

#### ►► PHPARMS — *phaseparms* ◄◄

are a string of values that are valid following the phase name on a PHASE statement in a DOSLNK file. Each value must be preceded by a comma, and there must be no blanks in the string. The first value must be the origin value. If PHPARMS is not specified, the origin value defaults to '\*'.

#### ►► ACTION — *actionparms* ◄◄

specifies an ACTION record that will be the first input record to DOSLKED. *actionparms* is a string of values that are valid on an ACTION statement in a DOSLNK file. The parameters are separated by commas, and there must be no blanks in the string.

For more information on CMS/DOS libraries, see "Developing VSE Programs under CMS" in [z/VM: CMS Application Development Guide for Assembler](#). For more information on the statements in a DOSLNK file, see [VSE/ESA 1.3.0 System Control Statements](#), SC33-6513.

### Part Options (DOSLIB)

VMFBDDL uses the following part options:

#### ►► ENTRY — \* — ◄◄ *entrypoint*

specifies a valid DOS link edit ENTRY record, which will be added to the DOSLNK file after the INCLUDE record for the part (or as the next record if no part is specified). Use an \* to indicate the ENTRY record does not include an entry point. The ENTRY record is not added if a part is specified, and it is not found.

#### ►► REMPHASE ◄◄

indicates a PHASE record at the beginning of the control records section of the text deck should be removed. The record is changed to a comment.

#### ►► IGNORE ◄◄

indicates the part can be ignored if it does not exist. By ignoring a part, an object can still be considered built even if the part is not included in it. A part can only be ignored if it has no service history. If a service level is defined for the part and it is missing, it is an error condition. This part option is intended for parts for which no base level part is shipped (e.g. help files and user exits).

#### ►► LANGFUNC — *exec* ◄◄

specifies a language function to be used in the selection of the correct serviceable part.

➤ NOGETLVL ➤

indicates the VMFSIM GETLVL function should not be used for this part. The values provided on the :PARTID tag are the specifications of the file used to build this part.

## Usage Notes (DOSLIB)

1. The :LIBNAME tag is not required. If it is omitted, the library file name is the same as the build list file name.
2. If no objects are listed in the build list and a build is requested, VMFBDDL B erases the DOS library and the DOS library history file, unless the NOERASE library parameter is specified.
3. If the DOS library does not exist or the DOS library history file is not on the same disk as the DOS library, a message is issued and the entire DOS library and DOS library history file are built. If the NOERASE library parameter is specified this occurs only if the DOS library does not exist.
4. The A-disk should not be used as the target disk because VMFBDDL B uses the A-disk as a work disk to create and erase temporary files.
5. If the updated DOS library has been copied to the target disk, but the DOS library history file was not copied to the target disk; use the VMFCOPY command to copy the temporary history file (\$DLB\$ DLKEDIT A) to the target disk.
6. If none of the text decks that make up a phase are found and the IGNORE option was specified for each one, the phase is considered to have been built. Any version of the phase currently in the DOS library remains in the library.
7. If DOS is set to ON before VMFBLD is invoked, it is not set ON by the part handler.
8. The DOS Library history file (*libname* DLKEDIT) contains the following information for each phase:
  - The name of the phase
  - A list of the text files that are included in the phase by VMFBDDL B
  - The map output from the DOSLKED invocation to build the phase. The NOERASE parameter is useful if a library is to be updated from multiple products, each of which has its own build list for the library. Without NOERASE, if a product erases and rebuilds the library, any objects (members) that have been added by other products are lost.

The history information from each text deck is included within the map output portion of the DLKEDIT file. The map output from DOSLKED is erased after it is imbedded in the DLKEDIT file.

9. The NOERASE parameter is useful if a library is to be updated from multiple products, each of which has its own build list for the library. Without NOERASE, if a product erases and rebuilds the library, any objects (members) that have been added by other products are lost.

## Generated Objects

VMFBLD uses the VMFB DGEN part handler to build generated objects, such as text decks. VMFB DGEN:

- Selects the correct level of the source part, unpacks the source file, and puts it on the A-disk
- Calls the generation routine specified in the build list
- Places the file, which is a result of the generation routine, on the target disk using the file specifications that are specified on the :OBJNAME tag
- Erases the file staged to the A disk
- Deletes objects as requested
- Provides the status of the object to VMFBLD

## Build List Format (Generated Objects)

VMFB DGEN uses format 2 build lists. For more information on build lists, see [“Build Lists” on page 141](#).

### Example (Generated Objects)

Figure 144 on page 334 is an example of a format 2 build list used by VMFBDGEN.

```

:FORMAT. 2
:GOBJPARM. GENPARMS VMFASM NOLIST NOTERM EGENPARMS
:OBJNAME. FILE1.TEXT
:PARTID. FILE1 ASM
:EOBJNAME.
:OBJNAME. FILE2.TXT GENPARMS VMFASM NOLIST NOTERM EGENPARMS GETLVL
:PARTID. VMFZZZ ASM
:EOBJNAME.
:OBJNAME. FILE3.TXTGCS
:PARTID. FILE3 ASM
:EOBJNAME.
:OBJNAME. FILE4.TEXT
:OPTIONS. NOGETLVL
:PARTID. MYOWN ASSEMBLE
:EOBJNAME.

```

Figure 144. Example Build List Used by VMFBDGEN

Entering this command:

```
vmfbld ppf ppfname compname bldlist (serviced
```

causes all serviced objects in the build list to be built.

Entering this command:

```
vmfbld ppf ppfname compname bldlist (all
```

causes all objects in the build list to be built.

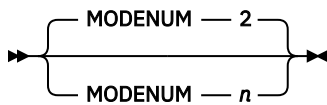
### Build List Restrictions (Generated Objects)

VMFBDGEN:

- Requires exactly one :PARTID tag per object block
- Ignores all :GGLOBAL and :GLOBAL values

### Build List Options (Generated Objects)

VMFBDGEN uses the following build list options:



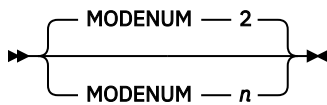
specifies the file mode number that is appended to the target mode of all objects in the build list. *n* must be a valid CMS file mode number. The default is 2.

➤ CNTRL — *cntrlfn* ➤

specifies the name of the control file that is used to identify the AUX file structure. The file type of the control file is CNTRL. This value overrides the value on the :CNTRL tag in the PPF.

### Object Parameters (Generated Objects)

VMFBDGEN uses the following object parameters:



specifies the file mode number that is appended to the target mode for this object. *n* must be a valid CMS file mode number. The default is 2.

►► GENPARMS — *command* — *options* — EGENPARMS ◄◄

specifies the generation routine to use to build this object.

***command***

is the generation routine to be used to build the object. The generation routine must be an EXEC.

***options***

are the options to be provided when the command is invoked.

►► GETLVL ◄◄

calls VMFSIM GETLVL to determine the file type of the output file. The values provided on the :OBJNAME tag are the file name and file type abbreviation of the output file. The PTF number is included in the file type.

The VMFBLD command uses the control file structure to determine the file type of the resulting output, unless you used the FILETYPE option. For more information, see [“How VMSES/E Uses Control Files” on page 117](#).

## Part Options (Generated Objects)

VMFBDGEN uses the following part options:

►► IGNORE ◄◄

indicates the part can be ignored if it does not exist. By ignoring a part, an object can still be considered built even if the part is not included in it. A part can only be ignored if it has no service history. If a service level is defined for the part and it is missing, it is an error condition. This part option is intended for parts for which no base level part is shipped (for example, help files and user exits).

►► LANGFUNC — *exec* ◄◄

specifies a language function to be used in the selection of the correct serviceable part.

►► NOGETLVL ◄◄

indicates the VMFSIM GETLVL function should not be used for this part. The values provided on the :PARTID tag are the specifications of the file used to build this part.

## Usage Notes (Generated Objects)

1. VMFBDGEN handles packed files.
2. The NOOBJECT option on the VMFASM, VMFHASM, VMFHLASM, and VMFNLS commands is ignored by VMFBDGEN.
3. When you apply local modifications to parts processed by VMFBDGEN, you should log them in the local version vector table using the source file type (for example, ASSEMBLE, REPOS, and DLCS).
4. The tokens, which are joined by a period (.), on the :OBJNAME tag are used to determine the file name and file type of the executable form created by VMFBLD part handlers that process format 2 build lists. The information on the :PARTID tag is used to determine which part(s) are used to create the output object. The following example demonstrates some of the capabilities provided by this convention:

## Load to the Byte File System

```
:FORMAT. 2
:GOBJPARM. GENPARMS VMFASM NOLIST NOTERM EGENPARMS
:OBJNAME. FILE1.TEXT
:PARTID. FILE1 ASM
:EOBJNAME.
:OBJNAME. FILE2.TXT GENPARMS VMFASM NOLIST NOTERM EGENPARMS GETLVL
:PARTID. VMFZZZ ASM
:EOBJNAME.
:OBJNAME. FILE3.TXTGCS
:PARTID. FILE3 ASM
:EOBJNAME.
:OBJNAME. FILE4.TEXT
:OPTIONS. NOGETLVL
:PARTID. MYOWN ASSEMBLE
:EOBJNAME.
```

## Load to the Byte File System

VMFBLD uses the VMFBDBFS part handler to select the latest level of a LOADBFS control file and uses the LOADBFS command to move files into the Byte File System.

**Note:** The VMFBDBFS part handler runs the CMS LOADBFS command. The LOADBFS command used to install certain utilities must be run on the z/VM system where the file pool resides. LOADBFS cannot be used to install data into a file pool on a remote z/VM system. The BFS limits the amount of data that can be sent to a remote file pool and LOADBFS sends more data than that limit.

### Build List Format (Byte File System)

VMFBDBFS uses format 2 build lists. For more information on build lists, see [“Build Lists” on page 141](#).

### Example (Byte File System)

Figure 145 on page 336 is an example of a format 2 build list used by VMFBDBFS.

```
:FORMAT. 2
*
:OBJNAME. SHELL.LOADBFS
:BLDREQ. GSUBLSYS GSUBLLIB GSUMLOAD
:PARTID. SHELL LBF
:EOBJNAME.
```

Figure 145. Example Build List Used by VMFBDBFS

Entering this command:

```
vmfbld ppf ppfname compname bldlist shell.loadbfs (all
```

causes the z/VM Shell and Utilities files to be loaded into Byte File System using the highest level of the control file SHELL LBF.

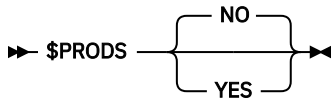
### Build List Restrictions (Byte File System)

VMFBDBFS requires at least one :PARTID tag per object block. The first :PARTID tag identifies the control file; any others identify the files to be loaded.

### Build List Options (Byte File System)

➤ *loadbfs\_options* ➤

*loadbfs\_options* specifies options that are passed to the LOADBFS command.



If \$PRODS YES is specified, a record is written to the SERVICE \$PRODS file and the byte file system is not updated. If \$PRODS NO is specified, the byte file system is updated. \$PRODS NO is the default.

## Object Parameters (Byte File System)

None.

## Part Options (Byte File System)

VMFBDBFS uses the following part options:

➤ LANGFUNC — *exec* ➤

specifies a language function to be used in the selection of the correct serviceable part.

➤ NOGETLVL ➤

indicates the VMFSIM GETLVL function should not be used for this part. The values provided on the :PARTID tag are the specifications of the file to use for this part.

## Usage Notes (Byte File System)

1. LOADLIBs specified on the :GLOBAL or :GGLOBAL tags in the build list will be made global when building an object. Any other types of libraries specified are ignored.
2. The first :PARTID tag in an object identifies the LOADBFS control file to be used. The latest service level is used. All other :PARTID tags in an object identify the files to be loaded into the Byte File System. The latest service levels of these files are not used. The parts that are actually loaded are defined in the LOADBFS control file.
3. The :OBJNAME tags are not used in the build process except to identify the objects.
4. The target disk is not used. The target is the Byte File System as defined in the LOADBFS control file.
5. If an object is removed from a VMFBDBFS build list, the object is marked as DELETED, but no Byte File System processing occurs.

## Objects Serviced by Complete Replacement

VMFBLD uses the VMFBDCOM part handler to build objects that are serviced by complete replacement of the serviceable part, for example execs, XEDIT macros, and help files. Many of these parts may also be supported by updates, in addition to the replacement parts.

The VMFBDCOM part handler selects the latest levels of serviceable parts, copies them to the appropriate build target, and renames them to the appropriate object names. For example, the serviceable part VMFREC EXC12345 would be copied and renamed to VMFREC EXEC.

## Build List Format (Replacement Objects)

VMFBDCOM uses format 2 build lists. For more information on build lists, see [“Build Lists” on page 141](#).

### **Example (Replacement Objects)**

[Figure 146 on page 338](#) is an example of a format 2 build list used by VMFBDCOM.

```

:FORMAT. 2
:OBJNAME. MYOWN.EXEC
:PARTID. MYOWN EXC
:EOBJNAME.
:OBJNAME. MYOWN.XEDIT
:PARTID. MYOWN XED
:EOBJNAME.
:OBJNAME. MYOWN2.EXEC
:PARTID. MYOWN2 EXC
:EOBJNAME.
:OBJNAME. =.HELPCP
:OPTIONS. IGNORE
:PARTID. * HCP
:EOBJNAME.
    
```

Figure 146. Example Build List Used by VMFBDCOM

Entering this command:

```
vmfbld ppf ppfname compname bldlist myown.exec (all
```

causes only the object MYOWN.EXEC to be built, which results in an output file on the target disk or directory MYOWN EXEC.

Entering this command:

```
vmfbld ppf ppfname compname bldlist myown (all
```

causes all objects whose file name portion of the :OBJNAME tag is MYOWN to be built. This causes MYOWN EXEC and MYOWN XEDIT to be built on the target disk or directory.

**Note:** The object =.HELPCP represents all files with a file type of HELPCP. If the correct level of serviceable part required to build any part of this object were the base level and that level could not be found, specification of the IGNORE option would cause an informational message to be issued and allow the object to be successfully built.

## Build List Restrictions (Replacement Objects)

VMFBDCOM:

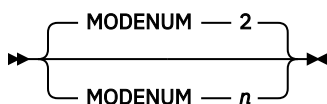
- Requires at least one :PARTID tag per object block
- Allows only one file type abbreviation in a :PARTID record
- Ignores :GLOBAL and :GGLOBAL tags

## Build List Options (Replacement Objects)

VMFBDCOM uses the following build list options:

➤ UPPER ➤

indicates all files copied to the build target should use the uppercase option of the COPYFILE command.



specifies the file mode number that is appended to the target mode of all objects in the build list. *n* must be a valid CMS file mode number. The default is 2.

➤ WRKST ➤



indicates all files copied to the build target should also be downloaded to the workstation.

►► HLVLCHK ◄◄

specifies that VMFBDCOM will do highest release level checking when processing objects in the build list. Objects in the build list will always be built in a single-system environment, but will only be built when running the build on a member of an SSI cluster that is running the highest release level of the associated product.

## Object Parameters (Replacement Objects)

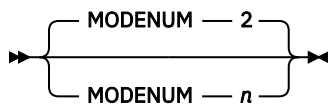
VMFBDCOM uses the following object parameters:

►► BYPASS ◄◄

indicates that the associated object is not to be built, regardless of its service history. This object parameter is intended for objects that need to be listed in a build list, but that should not be built by VMFBLD at any time.

►► UPPER ◄◄

indicates all files copied to the build target should use the uppercase option of the COPYFILE command.



specifies the file mode number that is appended to the target mode of this object in the build list. *n* must be a valid CMS file mode number. The default is 2.

## Part Options (Replacement Objects)

VMFBDCOM uses the following part options:

►► IGNORE ◄◄

indicates the part can be ignored if it does not exist. By ignoring a part, an object can still be considered built even if the part is not included in it. A part can only be ignored if it has no service history. If a service level is defined for the part and it is missing, it is an error condition. This part option is intended for parts for which no base level part is shipped, for example, for help files and user exits.

►► LANGFUNC — *exec* ◄◄

specifies a language function to be used in the selection of the correct serviceable part.

►► UPPER ◄◄

indicates all files copied to the build target should use the uppercase option of the COPYFILE command.

►► NOGETLVL ◄◄

indicates file types are not to be resolved. All associated :PARTIDs must contain two tokens. They are used as the file name and file type of the serviceable part that is required to build this object.

►► BYPASS ◄◄

indicates that the associated part is not to be built, regardless of its service history. This part option is intended for parts that need to be listed in a build list, but that should not be built by VMFBLD at any time.



specifies a build requisite. A build requisite must be built prior to the object that specifies it. *reqbldlist* is the build list that contains the requisite object, and *reqobj* is the requisite object. If *reqobj* is not specified, all objects in the specified build list are requisites.

## Object Parameters (Text Objects)

None.

## Part Options (Text Objects)

VMFBDCPY uses the following part options:

➤ LANG ➤

specifies the part is language sensitive and VMFLANG is to be used as the language function. If a part type is specified for the part in the build list, this option is ignored.

## Usage Notes (Text Objects)

VMFBDCPY is supported for compatibility with previous releases of VM/ESA. If you are creating build lists for text objects serviced by complete replacement, use a format 2 build list and the VMFBDCOM part handler.

## LOADLIBS

VMFBLD uses the VMFBDLLB part handler to select the latest levels of serviceable text decks, TXTLIB members, LOADLIB members, and create load modules as members of a LOADLIB. For example, the serviceable text decks DMSABC TXT12345 and DMSXYZ TXT23456 would be link edited into a load module that is placed into a LOADLIB. The LOADLIB is then copied to the appropriate build target.

## Build List Format (LOADLIBS)

VMFBDLLB uses format 3 build lists. For more information on build lists, see [“Build Lists” on page 141](#).

### Example (LOADLIBS)

Figure 148 on page 341 is an example of a format 3 build list used by VMFBDLLB.

```
:FORMAT. 3
:LIBNAME. ALOADLIB
:OBJNAME. TEXT1
:PARTID. MXRTXT1 TXT
:EOBJNAME.
:OBJNAME. TEXT2
:PARTID. MXRTXT2A TXT
:PARTID. MXRTXT2B TXT
:EOBJNAME.
:OBJNAME. TEXT3
:BLDREQ. TXTLIBA.TEXTA8
:OPTIONS. INCLUDE TXTLIBA(TEXTA8) INCLUDE DD3(LOAD3)
:OPTIONS. DDNAME DD3 LOADLIBA LOADLIB
:OPTIONS. ENTRY TEXTA8
:EOBJNAME.
:OBJNAME. TEXT4
:OPTIONS.INCLUDE TXTLIBX(TEXTX1)
      CONCAT SYSLIB TXTLIBP|TXTLIBA
      CONCAT SYSLIB TXTLIBQ
      CONCAT SYSLIB TXTLIBX
      CONCAT SYSLIB TXTLIBZ
:EOBJNAME.
```

Figure 148. Example Build List Used by VMFBDLLB

Entering this command:

## LOADLIBS

```
vmfbld ppf ppfname compname bldlist (all
```

causes ALOADLIB LOADLIB to be rebuilt using the latest level of serviceable part for MXRTEXT1, MXRTEXT2A, and MXRTEXT2B text files. It would also contain member TEXTA8 from TXTLIBA TXTLIB, member LOAD3 from LOADLIBA LOADLIB, member TEXTX1 from TXTLIBX TXTLIB, and any other text members that TEXTX1 brings with it from the libraries being concatenated.

Entering this command:

```
vmfbld ppf ppfname compname bldlist text1 (all
```

causes only the object TEXT1 to be built in ALOADLIB LOADLIB using the latest level serviceable part for MXRTEXT1 text file.

**Note:** If the object TEXTA8 in TXTLIBA TXTLIB were serviced, it would cause object TEXT3 to be rebuilt following the successful build of TEXTA8.

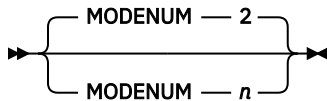
### Build List Restrictions (LOADLIBS)

VMFBDLLB:

- Does not allow wildcard objects.
- Ignores :GLOBAL and :GGLOBAL tags

### Build List Options (LOADLIBS)

VMFBDLLB uses the following build list options:



specifies the file mode number that is appended to the target mode of all objects in the build list. *n* must be a valid CMS file mode number. The default is 2.

### Library Parameters (LOADLIBS)

VMFBDLLB uses the following library parameters:

➤ BIND ➤

indicates that the LOADLIB is built with the Program Management Binder for CMS (Binder).

➤ NOERASE ➤

indicates to the part handler the library is not to be erased during the build.

### Object Parameters (LOADLIBS)

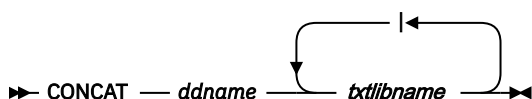
VMFBDLLB uses the following object parameter:

➤ LEPARMS — *opts* ➤

are the LKED options or BIND options for the member. The Binder SNAME option is not allowed as it is generated automatically from the :OBJNAME tag.

### Part Options (LOADLIBS)

VMFBDLLB uses the following part options:



assigns a single data definition name to two or more TXTLIBS. These TXTLIBS are then made global in the same order as the `CONCAT` statements are specified.

If the `CONCAT` option is specified, it must be followed by two tokens. The first token (*ddname*) is used as the data definition name. The second token (*txtlibname*) is used as the TXTLIB name. If you use the OR (|) operator, the first *txtlibname* found in the set of specified *txtlibnames* is used as the TXTLIB name. Also, if you use the OR (|) operator and one of the libraries is null and no libraries are found, no processing occurs for this `CONCAT` option. There are no spaces between the *txtlibnames* and the OR operator (for example, `TEXTLIB1|TEXTLIB2|TEXTLIB3`).

You can specify any number of `CONCAT` options in an object or a build list.

For more information on data definition names, see the description of the `FILEDEF` command in [z/VM: CMS Commands and Utilities Reference](#).

#### ►► IGNORE ◄◄

indicates the part can be ignored if it does not exist. Only parts defined on a `:PARTID` tag may be ignored. By ignoring a part, an object can still be considered built even if the part is not included in it. A part can only be ignored if it has no service history. If a service level is defined for the part and it is missing, it is an error condition. This part option is intended for parts for which no base level part is shipped, for example, for help files and user exits.

Parts that are specified with the `NOGETLVL` part option can also specify `IGNORE` to be ignored if they do not exist.

#### ►► LANGFUNC — exec ◄◄

specifies a language function to be used in the selection of the correct serviceable part.

#### ►► bind\_statements ◄◄

are any binder control statements, except for the `NAME` statement. The `NAME` statement is generated automatically from the `:OBJNAME` tag. Each binder control statement has to be on its own `:OPTIONS` tag.

#### ►► lked\_statements ◄◄

are any linkage editor control statements, except the `NAME` statement. The `NAME` statement is generated automatically from the `:OBJNAME` tag. The linkage editor control statements may all be listed on the same physical line with the `:OPTIONS` tag, or they may be split onto separate lines. `INCLUDE` statements are used to include `TXTLIB` or `LOADLIB` members. A `DDNAME` option is used to define the library that contains the member. If no `DDNAME` option matches the `INCLUDE`, the library defaults to a `TXTLIB` with the name specified on the `INCLUDE`.

#### ►► NOGETLVL ◄◄

indicates file types are not to be resolved. All associated `:PARTIDs` must contain two tokens. They are used as the file name and file type of the serviceable part that is required to build this object.

#### ►► DDNAME — ddname — libname — libtype ◄◄

defines the library (`TXTLIB` or `LOADLIB`) that is searched for the member specified on an `INCLUDE` statement. The format of the `INCLUDE` is `INCLUDE ddname(member)`.

## Usage Notes (LOADLIBS)

1. The :LIBNAME tag is not required. If it is omitted, the library file name is the same as the build list file name.
2. Format 3 part handlers can selectively service members based on the build requirements established by VMFBLD.
3. Format 3 part handlers attempt to service an existing library, but will rebuild it if the library does not exist or if all members must be rebuilt. If the NOERASE library parameter is specified the library will be rebuilt only if it does not exist.
4. The NOERASE parameter is useful if a library is to be updated from multiple products, each of which has its own build list for the library. Without NOERASE, if a product erases and rebuilds the library, any objects (members) that have been added by other products are lost.
5. VMSES/E uses the part options IGNORE, LANGFUNC, and NOGETLVL to determine the latest level of parts defined by :PARTID tags. When these options are found on an :OPTIONS tag, they apply to all :PARTID tags that follow in the object block in which they are found until the next :OPTIONS tag is encountered. The next :OPTIONS tag resets these options. All other part options are used to build the input to the LKED command for the object block in which they are found. These options are used as they are encountered on :OPTIONS tags and are not reset by subsequent :OPTIONS tags.

## MACLIBS

VMFBLD uses the VMFBTMLB part handler to select the latest level serviceable MACRO and COPY files and create members of a MACLIB. For example, the serviceable parts MACRO MAC12345 and COPY4 CPY54321 are placed into a MACLIB that is then copied to the appropriate build target. If files other than MACRO or COPY are selected they are first renamed to COPY before being placed into the MACLIB.

### Build List Format (MACLIBS)

VMFBTMLB uses format 3 build lists. For more information on build lists, see [“Build Lists” on page 141](#).

#### Example (MACLIBS)

Figure 149 on page 344 is an example of a format 3 build list used by VMFBTMLB.

```
:FORMAT. 3
:LIBNAME. AMACLIB
:OBJNAME. MEMBER1
:PARTID. MACR01 MAC
:EOBJNAME.
:OBJNAME. MEMBER2
:PARTID. COPY1 COPY
:EOBJNAME.
:OBJNAME. MEMBER3
:PARTID. COPY2 CPY
:EOBJNAME.
:OBJNAME. MEMBER4
:PARTID. MACR02 MACRO
:EOBJNAME.
```

Figure 149. Example Build List Used by VMFBTMLB

Entering this command:

```
vmfbld ppf ppfname compname bldlist (all
```

causes AMACLIB MACLIB to be rebuilt using the highest level of serviceable parts for MACRO1 MAC and COPY2 CPY and using the highest levels of MACRO2 MACRO and COPY1 COPY, which are created using their update files.

Entering this command:

```
vmfbld ppf ppfname compname bldlist member3 (all
```

causes only the object MEMBER3 to be built in AMACLIB MACLIB using the latest level serviceable part for COPY2 CPY.

**Note:**

1. If any member of a MACLIB is to be deleted, VMFBTMLB will rebuild the MACLIB omitting the deleted member unless NOERASE library parameter is specified. In this case, the deleted member will be removed from the MACLIB without a complete rebuild.
2. Specifying a real file type, rather than an abbreviation, on the :PARTID tag signals the VMFBTMLB part handler that the UPDATE command must be called to create the highest level of this part. If the real file type is 3 characters in length, the UPDATE part option must be used to signal the use of the UPDATE command.

**Build List Restrictions (MACLIBs)**

VMFBTMLB:

- Requires exactly one :PARTID tag per object block
- Does not allow wildcard objects
- Ignores :GLOBAL and :GGLOBAL tags

**Build List Options (MACLIBs)**

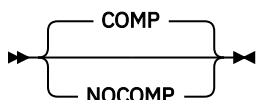
VMFBTMLB uses the following build list options:

►► CNTRL — *cntrlfn* ◄◄

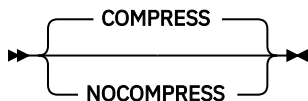
specifies the name of the control file. The control file must have a file type of CNTRL. The control file specified on the build list option overrides the control file specified in the CNTRLOP section of the product parameter file.

►► ALTCNTRL ◄◄

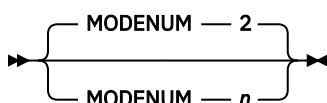
specifies the alternate control file, if it is specified in the CNTRLOP section of the product parameter file, is to be used instead of the control file.



specifies whether the MACLIB should be compacted (unused space is removed). COMP is the default.



specifies whether the MACLIB should be compressed (comments are removed). COMPRESS is the default.



specifies the file mode number that is appended to the target mode of all objects in the build list. *n* must be a valid CMS file mode number. The default is 2.

## Library Parameters (MACLIBs)

VMFBDMLB uses the following library parameters:

### ►► NOERASE ◄◄

indicates to the part handler the library is not to be erased during the build.

## Object Parameters (MACLIBs)

VMFBDMLB uses no object parameters.

## Part Options (MACLIBs)

VMFBDMLB uses the following part options:

### ►► IGNORE ◄◄

indicates the part can be ignored if it does not exist. By ignoring a part, an object can still be considered built even if the part is not included in it. A part can only be ignored if it has no service history. If a service level is defined for the part and it is missing, it is an error condition. This part option is intended for parts for which no base level part is shipped, for example, for help files and user exits.

Parts that are specified with the NOGETLVL part option can also specify IGNORE to be ignored if they do not exist.

### ►► LANGFUNC — *exec* ◄◄

specifies a language function to be used in the selection of the correct serviceable part.

### ►► NOGETLVL ◄◄

indicates file types are not to be resolved. All associated :PARTIDs must contain two tokens. They are used as the file name and file type of the serviceable part that is required to build this object.

### ►► UPDATE ◄◄

indicates the UPDATE command must be called to create the highest level of this part.

## Usage Notes (MACLIBs)

1. The :LIBNAME tag is not required. If it is omitted, the library file name is the same as the build list file name.
2. Format 3 part handlers can selectively service members based on the build requirements established by VMFBLD.
3. Format 3 part handlers attempt to service an existing library, but will rebuild it if the library does not exist or if all members must be rebuilt. If the NOERASE library parameter is specified this occurs only if the library does not exist.
4. When servicing a MACLIB using VMSES/E, there should be twice the number of free blocks of DASD on your A-disk than the MACLIB being serviced requires. This provides more space for such functions as staging the MACLIB, updating serviced files, and compressing the MACLIB.
5. The NOERASE parameter is useful if a library is to be updated from multiple products, each of which has its own build list for the library. Without NOERASE, if a product erases and rebuilds the library, any objects (members) that have been added by other products are lost.



## Executable Modules

The VMFBDMOD part handler selects the latest level of serviceable text decks and uses the LOAD, INCLUDE, and GENMOD commands to create executable MODULEs. For example, DMSABC TXT12345, DMSXYZ TXT23456, and DMSA2Z are loaded into storage and used to create a module with the specified name on the appropriate build target.

### Build List Format (Modules)

VMFBDMOD uses format 2 build lists. For more information on build lists, see [“Build Lists” on page 141](#).

#### Example (Modules)

Figure 150 on page 347 is an example of a format 2 build list used by VMFBDMOD.

```
:FORMAT. 2
:OBJNAME. FIRSTMOD.MODULE NOMAP
:OPTIONS. NOMAP
:PARTID. FIRSTTEXT TXT
:EOBJNAME.
:OBJNAME. SECNDMOD.MODULE NOMAP
:OPTIONS. NOMAP CLEAR RLDSAVE NCHIST NOUNDEF
:PARTID. SECTEXT1 TXT
:PARTID. SECTEXT2 TXT
:PARTID. SECTEXT3 TXT
:OPTIONS. NOGETLVL SAME
:PARTID. DMSCSL TEXT
:OPTIONS. NOMAP CLEAR RLDSAVE NCHIST NOUNDEF
:PARTID. SECTEXT4 TXT
:OPTIONS. UNDEF SAME
:PARTID. SECLAST TXT
:EOBJNAME.
```

Figure 150. Example Build List Used by VMFBDMOD

Entering this command:

```
vmfbld ppf ppfname compname bldlist firstmod.module (all
```

causes the object FIRSTMOD MODULE to be built on the target disk or directory using the highest level of the specified text deck.

Entering this command:

```
vmfbld ppf ppfname compname bldlist secndmod (all
```

causes the object SECNDMOD MODULE to be built on the target disk or directory using the highest level of the specified text decks for all parts except DMSCSL TEXT. The NOGETLVL option causes normal VMFSIM GETLVL processing to be bypassed for this part, and the first accessed version of the part name provided on the :PARTID tag is used.

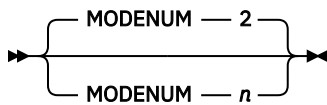
### Build List Restrictions (Modules)

VMFBDMOD:

- Requires at least one :PARTID tag per object block
- Does not allow wildcard objects
- Does not allow consecutive :OPTIONS tags

### Build List Options (Modules)

VMFBDMOD uses the following build list options:



specifies the file mode number that is appended to the target mode of all objects in the build list. *n* must be a valid CMS file mode number. The default is 2.

▶▶ **HLVLCHK** ▶◀

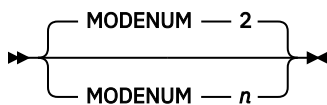
specifies that VMFBDMOD will do highest release level checking when processing objects in the build list. Objects in the build list will always be built in a single-system environment, but will only be built when running the build on a member of an SSI cluster that is running the highest release level of the associated product.

### Object Parameters (Modules)

VMFBDMOD uses the following object parameters:

▶▶ **genmodopts** ▶◀

are any GENMOD command options.



specifies the file mode number that is appended to the target mode of this object in the build list. *n* must be a valid CMS file mode number. The default is 2.

▶▶ **LDRTBLS** — *nnn* ▶◀

defines the number of pages of storage to be used for loader tables by issuing the SET LDRTBLS command. *nnn* is a whole number ranging from 1 to 127 inclusive. The value is reset once the object build is complete.

### Part Options (Modules)

VMFBDMOD uses the following part options:

▶▶ **LANGFUNC** — *exec* ▶◀

specifies a language function to be used in the selection of the correct serviceable part.

▶▶ **loadopts** ▶◀

are any LOAD command options. These are only valid on the :OPTIONS tags which precede the first :PARTID tag in an object block.

▶▶ **inclopts** ▶◀

are any INCLUDE command options. These are only valid on the :OPTIONS tags which follow the first :PARTID tag in an object block.

▶▶ **NOGETLVL** ▶◀

indicates file types are not to be resolved. All associated :PARTIDs must contain two tokens. They are used as the file name and file type of the serviceable part that is required to build this object.

## Usage Notes (Modules)

1. If you are building modules and you require TXTLIBs to be made global, specify them on the :GLOBAL or :GGLOBAL tags in the build list.
2. The :OBJNAME tags specified in the build list have a clear and concise purpose. They provide the file name and file type of the resulting usable form.
3. The tokens, which are joined by a period (.), on the :OBJNAME tag are used to determine the file name and file type of the executable form created by VMFBLD part handler handlers that process format 2 build lists. The information on the :PARTID tag is used to determine which part(s) are used to create the output object.

## Executable Modules When Using CPLINK or BIND

The VMFBPMD part handler selects the latest level of serviceable text decks and uses the CPLINK, LOAD, and GENMOD commands to create executable modules. The specified object parameter determines which command(s) will be used to build the modules:

- If the C89 object parameter is specified, the C89 command is used to build the modules.
- If the BIND object parameter is specified, the C89 and BIND commands are used to build the modules.
- If the XPLINK object parameter is specified, the C89, XPLINK, and BIND commands are used to build the modules.

### **Build List Format (Modules using CPLINK or BIND)**

VMFBPMD uses format 2 build lists. For more information on build lists, see [“Build Lists” on page 141](#).

*Example (Modules using CPLINK or BIND)*

Figure 151 on page 349 is an example of a format 2 build list used by VMFBPMD.

```
:FORMAT. 2
:GLOBAL. LOADLIB SCEERUN
:OBJNAME. FIRSTMOD.MODULE NOMAP
:OPTIONS. NOMAP
:PARTID. FIRSTTEXT TXT
:EOBJNAME.
:OBJNAME. SECNDMOD.MODULE NOMAP
:OPTIONS. NOMAP CLEAR RLDSAVE NCHIST
:PARTID. SECTEXT1 TXT
:PARTID. SECTEXT2 TXT
:PARTID. SECTEXT3 TXT
:OPTIONS. NOGETLVL
:PARTID. DMSCSL TEXT
:OPTIONS.
:PARTID. SECTEXT4 TXT
:PARTID. SECLAST TXT
:OPTIONS. UPCASE
:EOBJNAME.
```

*Figure 151. Example Build List Used by VMFBPMD*

Entering this command:

```
vmfblld ppf ppfname compname bldlist firstmod.module (all
```

causes the object FIRSTMOD MODULE to be built on the target disk or directory using the highest level of the specified text deck. The CPLINK, LOAD, and GENMOD commands are used to build the MODULE.

Entering this command:

```
vmfblld ppf ppfname compname bldlist secndmod (all
```

causes the object SECNDMOD MODULE to be built on the target disk or directory using the highest level of the specified text decks for all parts except DMSCSL TEXT. The NOGETLVL option causes normal VMFSIM

GETLVL processing to be bypassed for this part, and the first accessed version of the part name provided on the :PARTID tag is used. UPCASE is a CPLINK option.

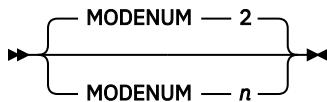
### **Build List Restrictions (Modules using CPLINK or BIND)**

VMFBDPMD:

- Requires at least one :PARTID tag per object block
- Does not allow wildcard objects

### **Build List Options (Modules using CPLINK or BIND)**

VMFBDPMD uses the following build list options:



specifies the file mode number that is appended to the target mode of all objects in the build list. *n* must be a valid CMS file mode number. The default is 2.

▶▶ HLVLCHK ▶◀

specifies that VMFBDPMD will do highest release level checking when processing objects in the build list. Objects in the build list will always be built in a single-system environment, but will only be built when running the build on a member of an SSI cluster that is running the highest release level of the associated product.

### **Object Parameters (Modules using CPLINK or BIND)**

VMFBDPMD uses the following object parameters:

▶▶ C89 ▶◀

specifies the C89 command will be used to build the module.

▶▶ BIND ▶◀

specifies the C89 and the BIND commands will be used to build the module.

▶▶ XPLINK ▶◀

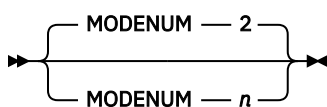
specifies the C89, XPLINK, and BIND commands will be used to build the module.

▶▶ *genmodopts* ▶◀

are any GENMOD command options.

▶▶ *bindopts* ▶◀

are any BIND command options.



specifies the file mode number that is appended to the target mode of this object in the build list. *n* must be a valid CMS file mode number. The default is 2.

►► LDRTBLS — *nnn* ◄◄

defines the number of pages of storage to be used for loader tables by issuing the SET LDRTBLS command. *nnn* is a whole number ranging from 1 to 127 inclusive. The value is reset once the object build is complete.

►► LOADEXIT — *fn* ◄◄

is an EXEC called between CPLINK and LOAD processing. The output from CPLINK is passed to the LOADEXIT with *fn* being the file name of the EXEC. Output from the LOADEXIT is passed to LOAD. This exit must accept 2 arguments: the file name of the input text file followed by the file name of the output file.

### **Part Options (Modules using CPLINK or BIND)**

VMFBDPMD uses the following part options:

►► LANGFUNC — *exec* ◄◄

specifies a language function to be used in the selection of the correct serviceable part.

►► *loadopts* ◄◄

are any LOAD command options. These are only valid on the :OPTIONS tag which precede the first :PARTID tag in an object block. LOAD options are ignored if the BIND object parameter is specified.

►► NOGETLVL ◄◄

indicates file types are not to be resolved. All associated :PARTIDs must contain two tokens. They are used as the file name and file type of the serviceable part that is required to build this object.

►► *cplinkopts* ◄◄

are any CPLINK command options. These are only valid on the :OPTIONS tag which follows the last :PARTID tag in an object block. CPLINK options are ignored if the BIND object parameter is specified.

### **Usage Notes (Modules using CPLINK or BIND)**

1. If you specify the C89 object parameter, VMFBDPMD requires that any part specified on a :PARTID tag must be available on an accessed disk or directory. VMFBDPMD does not search global TXTLIBs for specified parts when using the C89 command.
2. If you specify the C89 object parameter with the DLL CPLINK command option, an exported variables and functions list file (file type of EXP) is created. In this situation, the LOADEXIT object parameter is not supported and is ignored if it is specified.
3. If you specify the BIND object parameter, the C89 object parameter is specified by default.
4. If you specify the XPLINK object parameter, the C89 and BIND object parameters are specified by default.
5. If you are building modules and you require TXTLIBs or LOADLIBs to be made global, specify them on the :GLOBAL or :GGLOBAL tags in the build list. If you do not specify the C89 object parameter, you should include the SCEERUN LOADLIB on a global tag.
6. The :OBJNAME tags specified in the build list have a clear and concise purpose. They provide the file name and file type of the resulting usable form.
7. The tokens, which are joined by a period (.), on the :OBJNAME tag are used to determine the file name and file type of the executable form created by VMFBLD part handler handlers that process format 2 build lists. The information on the :PARTID tag is used to determine which part(s) are used to create the output object.

## Nuclei

When building a nucleus, VMFBLD uses the VMFBDNUC part handler to select the latest level of serviceable text decks and build a temporary load list to be used by the system loader to build a nucleus. For example, all file types for all entries in CPLOAD EXEC would be resolved and HCPLDR would be called with this information, and any other information, based on options or parameters that have been supplied.

### Build List Format (Nuclei)

VMFBDNUC uses format 1 build lists. For more information on build lists, see [“Build Lists” on page 141](#).

#### Example (Nuclei)

Figure 152 on page 352 is an example of a build list used by VMFBDNUC.

```
&TRACE OFF
&1 &2 &3 HCPLDR LOADER
&1 &2 &3 HCPSYS
&1 &2 &3 HCPRIO
*LDT
&1 &2 &3 HCPBOX
*LDT HCPGENUC
```

Figure 152. Example Build List Used by VMFBDNUC

Entering this command:

```
vmfblld ppf ppfname compname bldlist * nuctarg module modname nuc (all
```

causes a nucleus to be generated as a loadable module on the target disk with the file identifier of NUC MODULE *fm*. If the MODNAME option was not specified, the resulting file name would be the same as *bldlist*.

Entering this command:

```
vmfblld ppf ppfname compname bldlist * fastpath (all
```

causes a nucleus to be generated to the virtual reader using the existing *bldlist* \$NUCEXEC as a load list. If this file does not exist, an error message is issued and processing terminates.

**Note:** Since VMFBDNUC generates the correct control card for HCPGENUC or HCPLODNC based on the NUCTARG value, you could encounter problems using the FASTPATH option. If the value in the \$NUCEXEC file does not match the value specified (or the default) on the command, an error message is issued and processing terminates.

### Build List Restrictions (Nuclei)

VMFBDNUC requires at least one part in the build list.

### Build List Options (Nuclei)

VMFBDNUC uses the following build list options:

➤ CNTRL-cntrlfn ➤

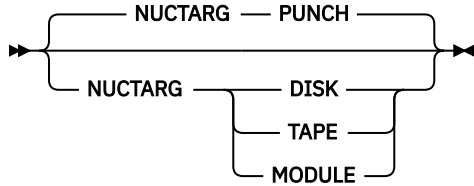
specifies the filename of the control file. The control file must have a filetype of CNTRL. The control file specified on the build list option overrides the control file specified in the CNTRLOP section for the product parameter file.

➤ ALTCNTRL ➤

specifies the alternate control file, if it is specified in the CNTRLOP section of the product parameter file, is to be used instead of the control file.

►► FASTPATH ◄◄

specifies the loader should use the text deck entries from the previous nucleus build. These are included in the *bdlist* \$NUCEXEC file.



specifies the target for the nucleus or nucleus module.

**PUNCH**

causes the nucleus to be created as an IPLable file in the virtual reader. NUCTARG PUNCH is the default.

**DISK**

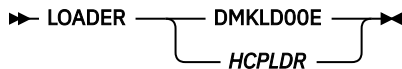
causes the nucleus to be link-edited and loaded directly to the system residence device.

**TAPE**

causes the nucleus to be created as an IPLable file on the tape device at virtual address 182.

**MODULE**

causes the nucleus to be created as a link-edited CMS module on the target disk.



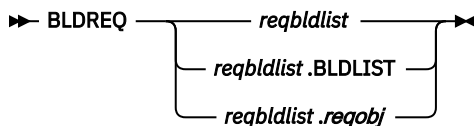
specifies which loader to call. If no loader is specified, the default is the loader identified in the nucleus build list.

►► RLDSAVE ◄◄

indicates the relocation dictionary is to be saved. This option is only valid when NUCTARG MODULE is specified.

►► MODNAME — *fn* ◄◄

specifies the name of the nucleus module. This option is only valid when NUCTARG MODULE is specified. If you do not specify MODNAME, the build list name is used as the default module name.



specifies a build requisite. A build requisite must be built prior to the object that specifies it. *reqbldlist* is the build list that contains the requisite object, and *reqobj* is the requisite object. If *reqobj* is not specified, all objects in the specified build list are requisites.

**Note:** The NUCTARG MODULE and RLDSAVE options are only valid when HCPLDR is used.

## Object Parameters (Nuclei)

None.

### Part Options (Nuclei)

VMFBGNUC uses the following part options:

► LANG ◄

specifies the part is language sensitive and VMFLANG is to be used as the language function. If a part type is specified for the part in the build list, this option is ignored.

### Usage Notes (Nuclei)

1. VMFBGNUC must insert the correct control card, either HCPGENUC or HCPLODNC, for the NUCTARG value specified. If the control card value in the *bldlist* \$NUCEXEC conflicts with the NUCTARG value that is specified when FASTPATH is requested, you receive an error message and processing terminates.
2. The CP nucleus can be built only with NUCTARG MODULE.

## Identifying System Objects to be Built

VMFBLD uses the VMFBDSBR part handler to identify system objects (such as saved segments) that need to be built because they contain parts that have been serviced. VMFBDSBR updates the VMSBR \$SELECT file.

### Build List Format (System Objects)

VMFBDSBR uses format 2 product saved segment build lists. For more information on build lists, see [“Build Lists” on page 141](#).

#### Example (System Object)

Figure 153 on page 354 is an example of a product saved segment build list used by VMFBDSBR.

```
:FORMAT. 2
:OBJNAME. CMSFILES.SEGMENT
:BLDREQ.  SERVLOAD.DMSDAC.MODULE
          SERVLOAD.DMSSAC.MODULE
          DMSBL493.DMSDAC.LSEG
          DMSBL493.DMSSAC.LSEG
:OPTIONS. LOADFUNC(LSEG DMSDAC)
          LOADFUNC(LSEG DMSSAC)
:EOBJNAME.
```

Figure 153. Example Product Saved Segment Build List Used by VMFBDSBR and VMFBDSEG

### Build List Restrictions (System Objects)

None.

### Build List Options (System Objects)

None.

### Object Parameters (System Objects)

VMFBDSBR uses the following object parameters:

► SBR — *sbrname* — *sbrtype* ◄



specifies the file name and file type of a part or a system object to be identified in the select data file as a build requirement instead of the file name and file type of this build list. If no object parameters are specified, this build list is identified as a build requirement.

## Part Options (System Objects)

Part options are used by VMFBDSEG, but not by VMFBDSBR.

## Usage Notes (System Objects)

1. The :VERSION tag in the product parameter file that identifies the product saved segment build list must be at least VM/ESA 1.2.0.
2. The first :SHRDISK tag in the first VMSESE PROFILE file found is used to locate the minidisk or SFS directory on which to update or create the VMSBR \$SELECT file.
3. Minidisks might be released during processing and reaccessed on termination.
4. The VMSBR \$SELECT file might be updated even if the VMFBDSBR return code is not zero. Check the time stamp in the file. If it is current, you do not need to run VMFBLD again for this build list.

## Saved Segments

VMFBLD uses the VMFBDSEG part handler to build saved segments.

### Build List Format (Saved Segments)

VMFBDSEG uses two kinds of format 2 build lists:

- A system saved segment build list that identifies all of the saved segments on the system
- Product saved segment build lists that contain build information for saved segments defined by VMSES/E format products

For more information on build lists, see [“Build Lists” on page 141](#).

### Example (Saved Segments)

Figure 154 on page 355 is an example of a system saved segment build list used by VMFBDSEG. Figure 153 on page 354 is an example of a product saved segment build list used by VMFBDSEG.

```

:FORMAT . 2
:OBJNAME . CMSAMS.SEGMENT
:PARTID . CMSAMS DMY
:EOBJNAME .
:OBJNAME . CMSBAM.SEGMENT
:PARTID . DMSBBAM EXC
:PARTID . CMSBAM DMY
:EOBJNAME .
:OBJNAME . CMSDOS.SEGMENT
:BLDREQ . SEGBLIST.DOSINST.SEGMENT
:PARTID . DMSBDOS EXC
:PARTID . CMSDOS DMY
:EOBJNAME .
:
:OBJNAME . CMSFILES.SEGMENT
:PARTID . DMSBSFS EXC
:PARTID . CMSFILES DMY
:EOBJNAME .
:
:OBJNAME . DOSINST.SEGMENT
:PARTID . DMSBDOS EXC
:PARTID . DOSINST DMY
:EOBJNAME .

```

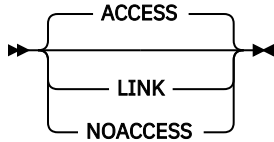
Figure 154. Example System Saved Segment Build List Used by VMFBDSEG

## Build List Restrictions (Saved Segments)

VMFBDSEG does not allow wildcard objects.

## Build List Options (Saved Segments)

VMFBDSEG uses the following build list options to link and detach the product disks when the segments for the product are built.



indicates whether VMFBDSEG should and how VMFBDSEG should call VMFSETUP when you build a segment that uses a product parameter file. The product parameter file is specified on the :BLDPARMS tag for the segment in the SEGDATA file.

### ACCESS

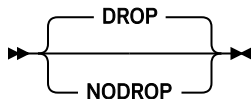
calls VMFSETUP to access the product disks (the disks must already be linked). The accessed disks are released after the segment is built. ACCESS is the default.

### LINK

calls VMFSETUP with the LINK option to link and access the product disks. The accessed disks are released, and the linked disks are detached after the segment is built.

### NOACCESS

does not call VMFSETUP. The disks must be already linked and accessed.



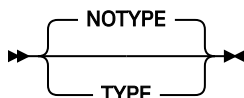
specifies whether VMFBDSEG should drop all or just new nucleus extensions after each segment is built.

### DROP

indicates VMFBDSEG should release all segments on entry. Before building the first segment and after each segment is built, all nucleus extensions are dropped.

### NODROP

indicates VMFBDSEG should not release any existing segments. Before building the first segment and after each segment is built, only those nucleus extensions that were not already loaded on entry to VMFBDSEG are released.



specifies whether VMFBDSEG should show the messages issued by SEGGEN.

### TYPE

indicates VMFBDSEG should show the messages issued by SEGGEN. This option is passed to SEGGEN.

### NOTYPE

indicates VMFBDSEG should not show the messages issued by SEGGEN. This option is passed to SEGGEN.

## Object Parameters (Saved Segments)

None.

## Part Options (Saved Segments)

When processing a system saved segment build list, VMFBDSEG uses no part options.

When processing a product saved segment build list, VMFBDSEG uses the following part options:

►► **LOADFUNC** — (*parmstring*) ◄◄

specifies the function that VMFBDSEG calls to load and save the saved segment (after VMFBDSEG has issued the DEFSEG command to define the saved segment to CP), where *parmstring* is one of the following:

►► *loadfunc* ————— ◄◄  
                   └── *loadparms* ─┘

specifies the name of the routine used to load and save the saved segment, plus any parameters to be passed to the routine. The following built-in variables are also available to indicate data to be passed to the *loadfunc* routine:

**&ORIGIN**

Starting load address

**&RANGE**

Page range, page descriptor codes, and optional DEFSEG operands

**&SEGNAME**

Saved segment name

**&SPACE**

Primary segment space name

When the saved segment is built, VMFBDSEG resolves the variables by obtaining the values from the saved segment definition record in the saved segment data (SEGDATA) file that has the same file name as the system saved segment build list specified on the VMFBLD command.

Return codes from product or user build functions are processed as follows:

**0**

Saved segment built without any errors

**1-4**

Saved segment built, one or more warnings issued

**all others**

Saved segment build failed

►► **LOADSAVE** — *modfn* ————— ◄◄  
                                   └── **ORIGIN** — *hexloc* ─┘

specifies the built-in LOADSAVE function, which issues the LOADMOD command to load the specified relocatable module file (*modfn* MODULE). LOADSAVE then issues the SAVESEG command to save the module as a saved segment.

**Note:** It is recommended that the ORIGIN keyword be specified and the built-in variable &ORIGIN be used to indicate the storage location. When the saved segment is built, VMFBDSEG resolves the &ORIGIN variable to get the load address from the saved segment definition in the SEGDATA file.

If the ORIGIN keyword is not specified, CMS selects any available storage location. Coding ORIGIN with a specific value for *hexloc* means the user cannot customize the load address. If a fixed address is necessary, it must be within the range specified in the DEFPARMS field of the definition in the SEGDATA file.

## Saved Segments



is a logical saved segment definition, which identifies a CMS logical saved segment to be included in a physical saved segment.

**Note:** The LOADFUNC part option can be repeated only when *parmstring* is a logical saved segment definition.

### ►► UNKNOWN ◄◄

indicates a build function for this object cannot be issued by VMFBDSEG. This forces VMFBDSEG to issue only the DEFSEG command to define the saved segment to CP. After VMFBLD has completed, the user must issue the function that actually loads and saves the saved segment.

## Usage Notes (Saved Segments)

1. VMFBDSEG does not support segments defined above 2 GB.
2. The :VERSION tag in the saved segment product parameter file (the product parameter file that identifies the system saved segment build list) must be at least VM/ESA 1.2.0.
3. The user ID issuing the VMFBLD command must have:
  - the authority to issue the CP DEFSEG, SAVESEG, QUERY NSS, and PURGE NSS commands.
  - enough virtual storage to contain the range of each saved segment to be built (except those whose load function is UNKNOWN) in addition to the storage used by CMS.
4. All saved segments are released and nucleus extensions dropped before saved segments are built.
5. To avoid problems when building the individual members of a segment space, build all the members at the same time by creating a VMFBLD input list (VMFBLD BLDDATA file) containing the members. Then issue the VMFBLD command with the LIST option.
6. If you are building segments from different products at the same time, use the LINK build list option to call VMFSETUP and link and access the product disks. The ACCESS build list option also calls VMFSETUP, but it does not link the product disks. For more information, see [“Build List Options \(Saved Segments\)”](#) on page 356.
7. A saved segment is deleted if any of the following are true:
  - The saved segment has been deleted through VMFSGMAP. In this case, the definition for the segment has been marked DELETED in the SEGDATA file; and the segment has been removed from the system saved segment build list.
  - The saved segment is built entirely from build lists having a status of DELETED.

In the first case, the build status of the saved segment is set to DELETED. In the second case, the build status of the saved segment is set to BUILT, because the saved segment is still in the system saved segment build list.

8. Before building a saved segment, VMFBDSEG purges any existing Class S (skeleton) system data file for that saved segment. In addition, when building a member saved segment in a Class S segment space, if a Class A (active) system data file for that member already exists, VMFBDSEG purges the Class A system data file. Note that this means the Class A member is removed from all the other segment spaces to which it belongs, even if those spaces are active.
9. When building a member of an existing Class A (active) or Class R (restricted) segment space, the unchanged members of the space are copied (using the SAME operand on the DEFSEG command) to the new Class S (skeleton) segment space. This does not copy the attributes (such as RSTD) with which these saved segments might have been created. To give the Class S segment space these attributes, they must be specified in the DEFPARMS field of the definition for the member being built.

10. When the PPF keyword is specified in the BLDPARMS field of the saved segment definition, VMFBDSEG activates any global libraries specified in the product saved segment build list as it uses each load function in the build list to build the saved segment.
11. When building a physical saved segment containing CMS logical saved segments:
  - a. VMFBDSEG copies the SYSTEM SEGID file from the S disk to the build target disk, if it is not already there. VMFBDSEG then invokes the SEGEN command to build the physical and logical saved segments and update the SYSTEM SEGID file on the target disk.
 

After all the saved segments are built, the updated SYSTEM SEGID file may need to be copied to the S disk. Copying the file is not necessary if data has been added, deleted, or changed in an existing saved segment. Copying the file is *required* if a new logical or physical saved segment has been created, an existing logical or physical saved segment has been deleted, or the relationship between the logical and physical saved segments has been changed (for example, a logical saved segment has been copied or moved from one physical saved segment to another). A message is issued if copying the file is necessary. If the message is issued after building a critical segment, such as CMSINST, the file must *immediately* be copied to the S disk. Note that updating a file on the system disk invalidates the current shared S-STAT (the S disk file directory). Therefore, after the file is copied to the system disk, the CMS named saved system must be resaved to update the S-STAT.
  - b. VMFBDSEG creates the physical segment definition file (default file type PSEG) from the logical saved segment definitions:
    - That follow the PROD keyword in the BLDPARMS field of the saved segment definition
    - That are specified in the build lists that follow the PPF keyword in the BLDPARMS field of the saved segment definition

Then VMFBDSEG issues the SEGEN command to build the physical and logical saved segments. VMFBDSEG creates logical saved segment profiles and epifiles (named \$VMFPn EXEC and \$VMFEn EXEC) as needed. These files are erased if SEGEN is successful.
  - c. PROFILE and EPIFILE cannot be used as the file type of a logical segment definition file.
  - d. The PSEGMAP and LSEGMAP load map files created by SEGEN are stored on the target disk.
12. When building a saved segment that does not contain CMS logical saved segments:
  - a. Only one set of build parameters can be specified in the BLDPARMS field of the saved segment definition. If the BLDPARMS field contains the PPF keyword only one load function can be specified in the product saved segment build list.
  - b. If a routine is called to load and save the saved segment, it must release all the storage that it obtains. A return code of 0, 1, 2, 3, or 4 from the routine is considered a successful build (resulting in a VMFBDSEG return code of 0). Any other return code from the routine is considered a severe error (VMFBDSEG return code 100). The routine should issue its own messages to indicate error conditions.

## TXTLIBs

VMFBLD uses the VMFBDTLB part handler to select the latest level of serviceable text decks and create members of a TXTLIB. For example, the serviceable parts DMSABC TXT12345 and DMSXYZ TXT54321 are placed into a TXTLIB that is then copied to the appropriate build target.

### Build List Format (TXTLIBs)

VMFBDTLB uses format 3 build lists. For more information on build lists, see [“Build Lists” on page 141](#).

### Examples (TXTLIBs)

[Figure 155 on page 360](#) is an example of a format 3 build list used by VMFBDTLB.

```

:FORMAT. 3
:LIBNAME. MYTXTLIB
:OBJNAME. MEMBER1
:PARTID. PART1 TXT
:EOBJNAME.
:OBJNAME. MEM2 NOFILENAME
:OPTIONS. NAME MEM2 ALIAS MEM3
:PARTID. PART2 TXT
:OPTIONS. ENTRY CSECT3
:PARTID. PART3 TXT
:OPTIONS.
:PARTID. PART4 TXT
:EOBJNAME.
    
```

*Figure 155. Example Build List Used by VMFBDTLB*

Entering this command:

```
vmfbld ppf ppfname compname bldlist (all
```

creates MYTXTLIB TXTLIB with two members. MEMBER1 is created from PART1 with an entry point of CSECT1. The second member, MEM2, is created from PART2, PART3, and PART4. It has an alias name of MEM3 and an entry point of CSECT3 in PART3.

Figure 156 on page 360 is another example of a format 3 build list used by VMFBDTLB.

```

:FORMAT. 3
:LIBNAME. MYTXTLIB C370LIB
:OBJNAME. MEMBER1
:OPTIONS. ENTRY CSECT1
:PARTID. PART1 TXT
:EOBJNAME.
    
```

*Figure 156. Example of Format 3 Build List for TXTLIB*

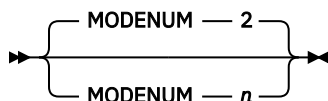
The C370LIB parameter will be used by Licensed Products who wish to have their TXTLIBs built using the C370LIB family of commands. By not specifying C370LIB as a library parameter on the :LIBNAME tag, VMFBDTLB will default to the use of TXTLIB commands.

### **Build List Restrictions (TXTLIBs)**

VMFBDTLB does not allow wildcard objects.

### **Build List Options (TXTLIBs)**

VMFBDTLB uses the following build list options:



specifies the file mode number that is appended to the target mode of all objects in the build list. *n* must be a valid CMS file mode number. The default is 2.

### **Library Parameters (TXTLIBs)**

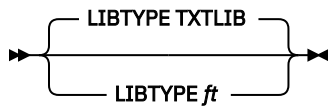
VMFBDTLB uses the following library parameters:

► C370LIB ◄

indicates to VMFBDTLB the C370LIB family of commands are to be used for TXTLIB manipulation instead of the TXTLIB family of commands.

➤ NOERASE ➤

indicates to the part handler the library is not to be erased during the build.



specifies the filetype of the TXTLIB that is to be created or updated.

### Object Parameters (TXTLIBs)

VMFBDTLB uses the following object parameters:



specifies whether VMFBDTLB should add the object to the TXTLIB using the FILENAME option on the TXTLIB command.

#### FILENAME

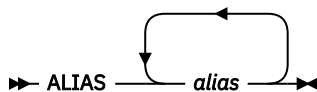
creates a TXTLIB member with the same name as the object name in the build list. FILENAME is the default.

#### NOFILENAME

creates a TXTLIB member named according to the rules for the TXTLIB command.

### Part Options (TXTLIBs)

VMFBDTLB uses the following part options:



is a TXTLIB link edit ALIAS record. This record is added to the text deck after the END statement.

➤ ENTRY — *entryname* ➤

is a TXTLIB link edit ENTRY record. This record is added to the text deck after the END statement.

➤ IGNORE ➤

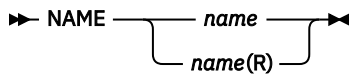
indicates the part can be ignored if it does not exist. By ignoring a part, an object can still be considered built even if the part is not included in it. A part can only be ignored if it has no service history. If a service level is defined for the part and it is missing, it is an error condition. This part option is intended for parts for which no base level part is shipped, for example, for help files and user exits.

Parts that are specified with the NOGETLVL part option can also specify IGNORE to be ignored if they do not exist.

➤ LANGFUNC — *exec* ➤

specifies a language function to be used in the selection of the correct serviceable part.

## SMAPI Appliance Servers and Stand-Alone Dump Utility



is a TXTLIB link edit NAME record. This record is added to the text deck after the END statement. The NAME record is added after all the other types of link edit records. The *name* on the NAME option must match the *objectname* specified on the :OBJNAME tag in the build list.

▶ NOGETLVL ▶

indicates file types are not to be resolved. All associated :PARTIDs must contain two tokens. They are used as the file name and file type of the serviceable part that is required to build this object.

▶ SETSSI — xxxxxxxx ▶

is a TXTLIB link edit SETSSI record. This record is added to the text deck after the END statement.

**Note:** You can specify any number of NAME, ENTRY, ALIAS, and SETSSI statements on an :OPTIONS record. VMFBDTLB adds the statements to the text deck, identified on the :PARTID record, directly following the :OPTIONS statement. The statements are added to the text deck after the END statement. The NAME statement appears last.

### Usage Notes (TXTLIBs)

1. The :LIBNAME tag is not required. If it is omitted, the library file name is the same as the build list file name.
2. Format 3 part handlers can selectively service members based on the build requirements established by VMFBLD.
3. Format 3 part handlers attempt to service an existing library, but will rebuild it if the library does not exist or if all members must be rebuilt. If the NOERASE library parameter is specified the library will be rebuilt only if it does not exist.
4. For successful delete processing, the object name in the TXTLIB build list must be the same as the name of the TXTLIB member that is built from that object.
5. Multiple :PARTID records can be specified for an object in a TXTLIB build list. Each :PARTID record can be preceded by an :OPTIONS record that specifies the link edit statements for that part. As each :PARTID record is processed, VMFBDTLB adds the link edit statements after the END statement for that part. It then appends the part to the first part for that object. The combined parts are staged on the A-disk and used to create the member.
6. The NOERASE parameter is useful if a library is to be updated from multiple products, each of which has its own build list for the library. Without NOERASE, if a product erases and rebuilds the library, any objects (members) that have been added by other products are lost.

## SMAPI Appliance Servers and Stand-Alone Dump Utility

VMFBLD uses the VMFBDSSP part handler to notify the service installer that certain Systems Management Application Programming Interface (SMAPI) appliance servers need to be restarted, or that the Stand-Alone Dump utility needs to be rebuilt using the information in the VMSES/E build list.

### Build List Format (SMAPI/Stand-Alone Dump Utility)

VMFBDSSP uses format 2 build lists. For more information on build lists, see [“Build Lists” on page 141](#).

#### *Example (SMAPI/Stand-Alone Dump Utility)*

[Figure 157 on page 363](#) is an example of a format 2 build list used by VMFBDSSP.



```

:FORMAT. 2
:OBJNAME. LOHCOST.SERVER
:BLDREQ. DMSBL400.SSPK71.IMAGE
:BLDREQ. DMSBL400.SSPI71.IMAGE
:BLDREQ. DMSBL400.SSP71.IMAGE
:BLDREQ. DMSBL400.LOHCOST.IMAGE
:EOBJNAME.

```

Figure 157. Example Build List Used by VMFBDSSP

When SERVICE issues the command:

```
vmfbld ppf ppfname compname bldlist lohcost.server (all
```

it causes the message VMFBLD2899W to be issued, telling the service installer that the LOHCOST appliance server needs to be restarted.

### **Build List Restrictions (SMAPI/Stand-Alone Dump Utility)**

VMFBDSSP requires at least one :BLDREQ tag per object block.

### **Build List Options (SMAPI/Stand-Alone Dump Utility)**

None.

### **Object Parameters (SMAPI/Stand-Alone Dump Utility)**

None.

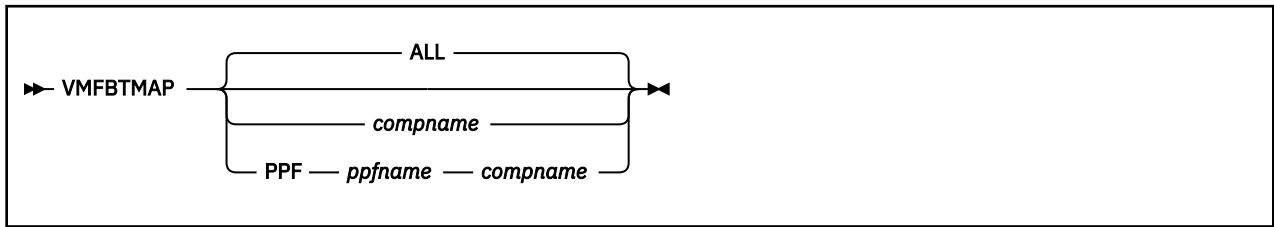
### **Part Options (SMAPI/Stand-Alone Dump Utility)**

None.

### **Usage Notes (SMAPI/Stand-Alone Dump Utility)**

1. The :OBJNAME tags are not used in the build process except to identify the objects (i.e. the SMAPI appliance server or the Stand-Alone Dump utility).

## VMFBTMAP EXEC



### Purpose

The VMFBTMAP EXEC creates or updates a file, (one for each PTF prefix associated with a component), that contains a bitmap representation of all PTFs received, applied, or superseded on the z/VM system for the PTF prefix. It also creates a file that lists all of the products that are installed on the system.

### Operands

#### ALL

indicates PTF bitmap files are created for all products in the VM SYSSUF table. This is the default.

#### *compname*

is the name of the component as it is specified on the :PRODID tag in the system-level service update facility table (VM SYSSUF). It indicates that bitmap files will be created or updated for the specific component. *compname* is a 1-16 character alphanumeric identifier.

#### PPF

identifies the product parameter file:

- *ppfname* is the file name of the PPF file.
- *compname* is the component name specified in the PPF file. *compname* is a 1-16 character alphanumeric identifier.

### Usage Notes

1. If VMFREM with the UNRECEIVE option has been executed, you need to run VMFBTMAP with the ALL operand to remove PTFs from the BITMAP files.
2. The BITMAP files are written to the VMSES/E Software Inventory Disk.
3. The VMPFXALL BITMAP and VMPRODS BITMAP files always get rebuilt.
4. VMFSUFTB needs to be run prior to running VMFBTMAP to place all installed products into the VM SYSSUF table.

### Examples

- To create a bitmap of PTFs for CP, enter:

```
VMFBTMAP CP
```

- To create a bitmap of PTFs for all products defined in the VM SYSSUF table, enter:

```
VMFBTMAP
```

### Input and Output Files

#### Input Files

**VM SYSSUF**

The system-level service update facility table

**ppfname PPF**

The usable form product parameter file

**recid SRVRECS**

The service-level receive status table

**appid SRVAPPS**

The service-level apply status table

**Input/Output Files****VMPFX-aa BITMAP**

Bitmap representation file for PTF PREFIX *aa*

**Output Files****VMPFXALL BITMAP**

File containing a list of products installed, by PRODID, and bitmap representation of all PTFs installed

**VMPRODS BITMAP**

List of products installed on the system

**Messages and Return Codes**

Appendix D, “Module Identifiers for VMSES/E Messages,” on page 749 shows the format of VMSES/E messages and lists the identifiers used by each VMSES/E exec.

For a complete explanation of any VMSES/E message, use the HELP Facility to view the message explanation online or see the appropriate messages documentation. To display information on a specific message, for example VMF002E, enter:

```
help msg vmf002e
```

If you are unfamiliar with the z/VM HELP Facility, enter `help` to display the main HELP Menu. For information on the HELP command, enter:

```
help cms help
```

The VMFBTMAP EXEC issues the following return codes:

Return Code	Explanation
0	Command completed successfully.
4	Command completed with one or more warning conditions.
8	Command completed but at least one major process failed.
12	Command failed because of an internal error.
24	Command failed because of a command line syntax error.
28	Command failed because a required file was not found.
36	Command failed because a target disk or directory was not available.
100	Command failed because of an external error.

**The BITMAP File Structure**

The structure of the VMPRODS and VMPFX-*aa* BITMAP files follows. The VMPFXALL BITMAP file contains the VMPRODS BITMAP file and all of the VMPFX-*aa* BITMAP files appended together. All of the files reside on the VMSES/E Software Inventory Disk; the default is the 51D (D) disk.

The VMPRODS BITMAP file consists of 157 80-byte records with the following format:

```

Bytes 01-08   PRODLIST
Bytes 09-14   processor number (from Q CPUID command)
Bytes 15-18   model number      (from Q CPUID command)
Bytes 19-26   VM v.r.m e.g. VM 7.3.0 (from Q CPLEVEL command)
Bytes 27-32   date in format of yymmdd
Bytes 33-43   not used
Bytes 44-47   SESE
Bytes 48-60   not used
Bytes 61-12560 Blank-delimited PRODLIDs

```

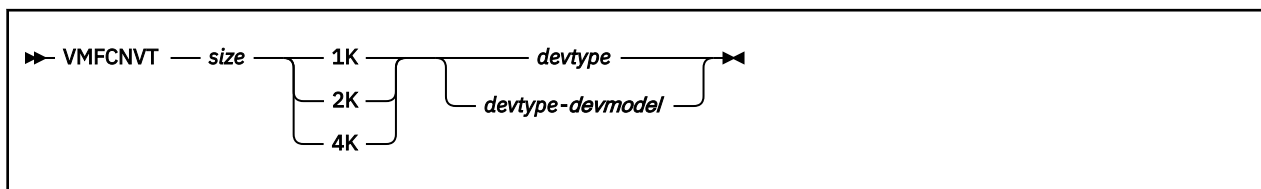
Each VMPFX-*aa* BITMAP file (one for each PTF prefix) consists of 157 80-byte records with the following format:

```

Bytes 01-06   BITPTF
Bytes 07-08   ptf prefix e.g. UA
Bytes 09-14   processor number (from Q CPUID command)
Bytes 15-18   model number      (from Q CPUID command)
Bytes 19-26   VM v.r.m e.g. VM 7.3.0 (from Q CPLEVEL command)
Bytes 27-32   date in format of yymmdd
Bytes 33-43   not used
Bytes 44-47   SESE
Bytes 48-60   not used
Bytes 61-12560 each bit represents PTF within this prefix
                1st bit in byte 61 indicates PTF aa00000
                Last bit in byte 12560 indicates PTF aa99999
                bit on, '1', represents PTF is received

```

## VMFCNVT EXEC



### Purpose

The VMFCNVT command converts a given number of blocks of DASD to an equivalent number of cylinders and displays the result. Output depends on the type of DASD being used.

### Operands

#### *size*

is the number of blocks to be converted.

#### **1K**

indicates the blocks to be converted are 1K in size.

#### **2K**

indicates the blocks to be converted are 2K in size.

#### **4K**

indicates the blocks to be converted are 4K in size.

#### *devtype*

is the device type of the direct access storage device (DASD) for which you want information. The device type (for example, 3380 or 3390) must be listed in the \$DASD\$ CONSTS file.

#### *devmodel*

is the model of the DASD. For example, 3390-1 indicates the 3390 model one. (-1 is the model indicator.) The device model must be listed in the \$DASD\$ CONSTS file. If you do not specify *devmodel*, conversions are calculated for all models of the specified DASD.

### Usage Notes

1. The \$DASD\$ CONSTS file, which is shipped with the VM/ESA product, must be available.

### Examples

- To determine the number of 3380 cylinders required for 50 4K blocks, enter:

```
vmfcnvt 50 4k 3380
```

You receive this response:

```
50 4K BLOCKS = 1 3380 CYLINDERS
50 4K BLOCKS = 1 3380-E4 CYLINDERS
50 4K BLOCKS = 1 3380-K4 CYLINDERS
READY;
```

### Input and Output Files

#### Input Files

#### **\$DASD\$ CONSTS**

Contains DASD-dependent conversion factors

## Messages and Return Codes

For a complete explanation of each message, use the HELP Facility to view the message explanation online or see the appropriate messages documentation. To display information on a specific message, for example VMF002E, enter:

```
help msg vmf002e
```

If you are unfamiliar with the z/VM HELP Facility, enter help to display the main HELP Menu. For information on the HELP command, enter:

```
help cms help
```

Appendix D, “[Module Identifiers for VMSES/E Messages](#),” on page 749 shows the format of VMSES/E messages and lists the identifiers for each VMSES/E exec.

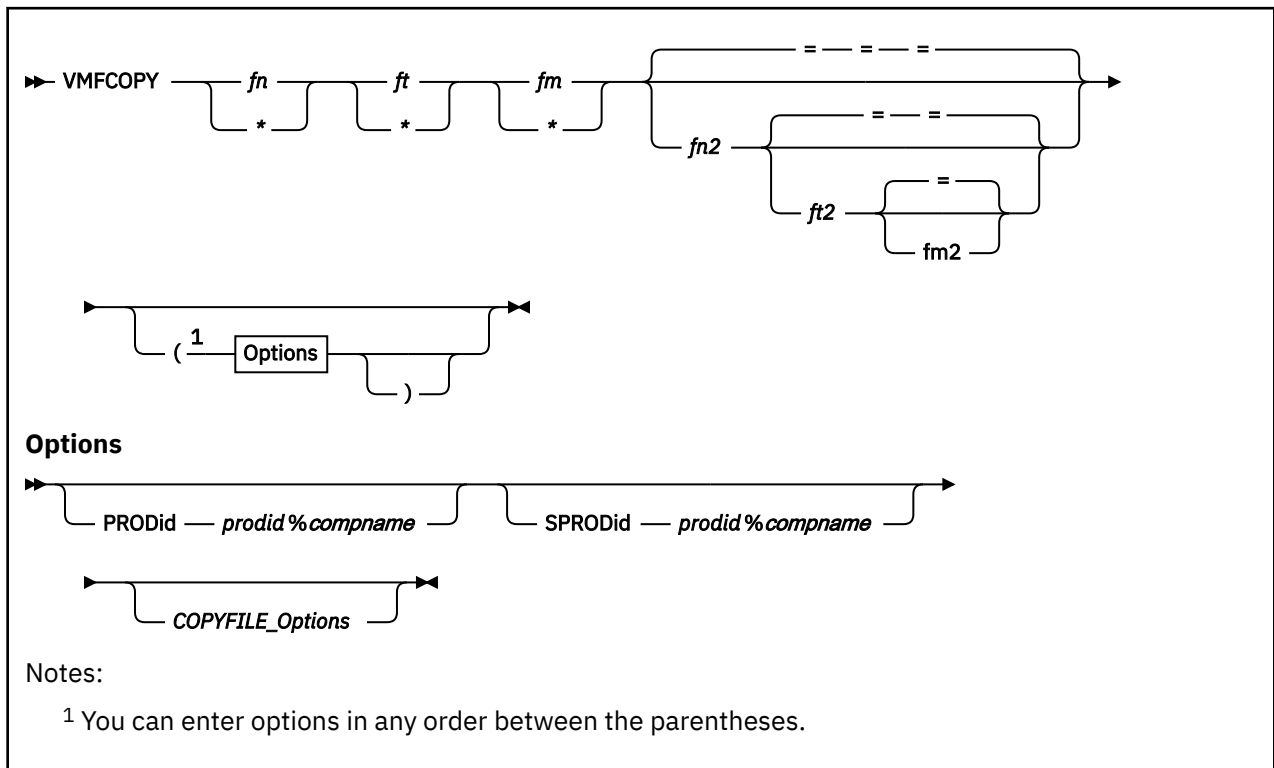
VMFCNVT issues the following return codes:

Return Code	Explanation
0	Command completed successfully.
24	Command failed because of a command line syntax error.
28	Command failed because a required file was not found.
100	Command failed because of an external error.

## Recovery Information

The VMFCNVT command can be restarted by reissuing the command.

## VMFCOPY EXEC



### Purpose

The VMFCOPY EXEC copies files to a VMSES/E target minidisk or SFS directory and, optionally, updates the VMSES PARTCAT file on that target.

### Operands

#### *fn*

is the file name of the input file. You must indicate a specific file name or use special characters (\* and %) as part of the file name to request a specific subset of files to copy.

#### *ft*

is the file type of the input file. You must indicate a specific file type or use special characters (\* and %) as part of the file name to request a specific subset of files to copy.

#### *fm*

is the file mode of the input file. You must indicate a specific file mode or use an asterisk (\*) as a wildcard indicator.

#### *fn2*

is the file name of the output file. You must indicate a specific file name or use an equal sign (=) to indicate the same file name as the input file. The default is =.

#### *ft2*

is the file type of the output file. You must indicate a specific file type or use an equal sign (=) to indicate the same file type as the input file. The default is =.

#### *fm2*

is the file mode of the output file. You must indicate a specific file mode or use an equal sign (=) to indicate the same file mode as the input file. The default is =.

## Options

### PRODID

indicates the identifier for the product.

#### **prodid**

is the 7- to 8-character alphanumeric identifier assigned to the product by IBM.

#### **%**

is a delimiter.

#### **compname**

is the 1- to 16-character alphanumeric identifier assigned to the component by IBM (for example, CMS). The component name can be found on the :PRODID tag in the product management files, such as the PRODPART file, the product parameter file, and the system-level Software Inventory tables.

### SPRODID

is the product identifier for the files to be copied. This is only applicable if the VMSES PARTCAT has a catalog entry for the file.

#### **prodid**

is the 7- to 8-character alphanumeric identifier assigned to the product by IBM.

#### **%**

is a delimiter.

#### **compname**

is the 1- to 16-character alphanumeric identifier assigned to the component by IBM (for example, CMS). The component name can be found on the :PRODID tag in the product management files, such as the PRODPART file, the product parameter file, and the system-level Software Inventory tables.

### **COPYFILE\_Options**

are the options supported by the CMS COPYFILE command. The CMS COPYFILE command is described in [z/VM: CMS Commands and Utilities Reference](#).

## Usage Notes

1. When you enter the VMFCOPY command with the PRODID option, the VMFCOPY EXEC calls the COPYFILE command to copy the source file(s) to the target, then updates the VMSES PARTCAT file on the target disk or SFS directory.
2. When you enter the VMFCOPY command without the PRODID option, VMFCOPY simply calls the COPYFILE command to copy the source file(s) to the target.
3. Any option supported by the CMS COPYFILE command is supported by VMFCOPY.
4. Although the CMS COPYFILE command allows you to specify multiple file identifiers, VMFCOPY only allows you to specify two — one for the source and one for the target.
5. You cannot use VMFCOPY to copy VMSES PARTCAT files from one disk to another, because the VMSES PARTCAT file lists all files that are related to a specific product on a specific disk or directory.  
VMFCOPY does not copy the VMSES PARTCAT file on the source disk to the target disk, but it does update the VMSES PARTCAT file on the target disk.
6. If the SPRODID option is specified, only those files that are cataloged in the VMSES PARTCAT file with the *prodid%compname* and meet the file selection criteria are copied to the target location.
7. If \* is specified as *fn* and the same *fn* and *ft* are on different disks, VMFCOPY will not append the files to the target (*fn2 ft2 fm2*) as does the CMS COPYFILE command. If the REPLACE option is not specified, only the first file in the search order is copied to the target location. The remaining files will result in error message DMSCPY024E, stating the file (*fn ft fm*) already exists. If the REPLACE option is specified, each file will be copied to the target (*fn2 ft2 fm2*), replacing the previous file. The last *fn ft* in the CMS search order will be the resulting file on the target location.



8. Special pattern matching characters \* and % are supported for copying a subset of files to the target location.
9. Alias file support has been added to VMFCOPY. The VMFCOPY command will copy files to an alias target file provided that:
  - The user ID has WRITE and NEWWRITE authority to the target file pool directory.
  - The user ID has only READ authority to the files currently in the target directory.
  - The user ID has only READ authority to the base files and base file pool directory.

### Examples

- To use VMFCOPY to copy the CMS file MYFILE DATA A to MYFILE DATA C and update the VMSES PARTCAT file on the target, enter:

```
VMFCOPY MYFILE DATA A = = C (PRODID 1VMVMC23%MYCOMP
```

- To use VMFCOPY to copy the file TEST DATA A to TEST DATA C, preserving the original file creation date on the target file, without updating the VMSES PARTCAT file on the target, enter:

```
VMFCOPY TEST DATA A = = C (OLDDATE
```

- To use VMFCOPY to copy all files from the A disk to the C disk that are cataloged in VMSES PARTCAT A for product 1VMVMC23%MYCOMP and to update VMSES PARTCAT C, enter:

```
VMFCOPY * * A = = C (SPRODID 1VMVMC23%MYCOMP PRODID 1VMVMC23%MYCOMP
```

## Input and Output Files

### Input Files

#### *fn ft fm*

The source file(s) to be copied.

### Output Files

#### *fn2 ft2 fm2*

The target file of the copy.

### VMSES PARTCAT

The parts catalog table.

## Messages and Return Codes

Appendix D, “Module Identifiers for VMSES/E Messages,” on page 749 shows the format of VMSES/E messages and lists the identifiers used by each VMSES/E exec.

For a complete explanation of any VMSES/E message, use the HELP Facility to view the message explanation online or see the appropriate messages documentation.

To display information on a specific message, for example VMF002E, enter:

```
help msg vmf002e
```

If you are unfamiliar with the z/VM HELP Facility, enter help to display the main HELP Menu. For information on the HELP command, enter:

```
help cms help
```

The VMFCOPY EXEC issues the following return codes:

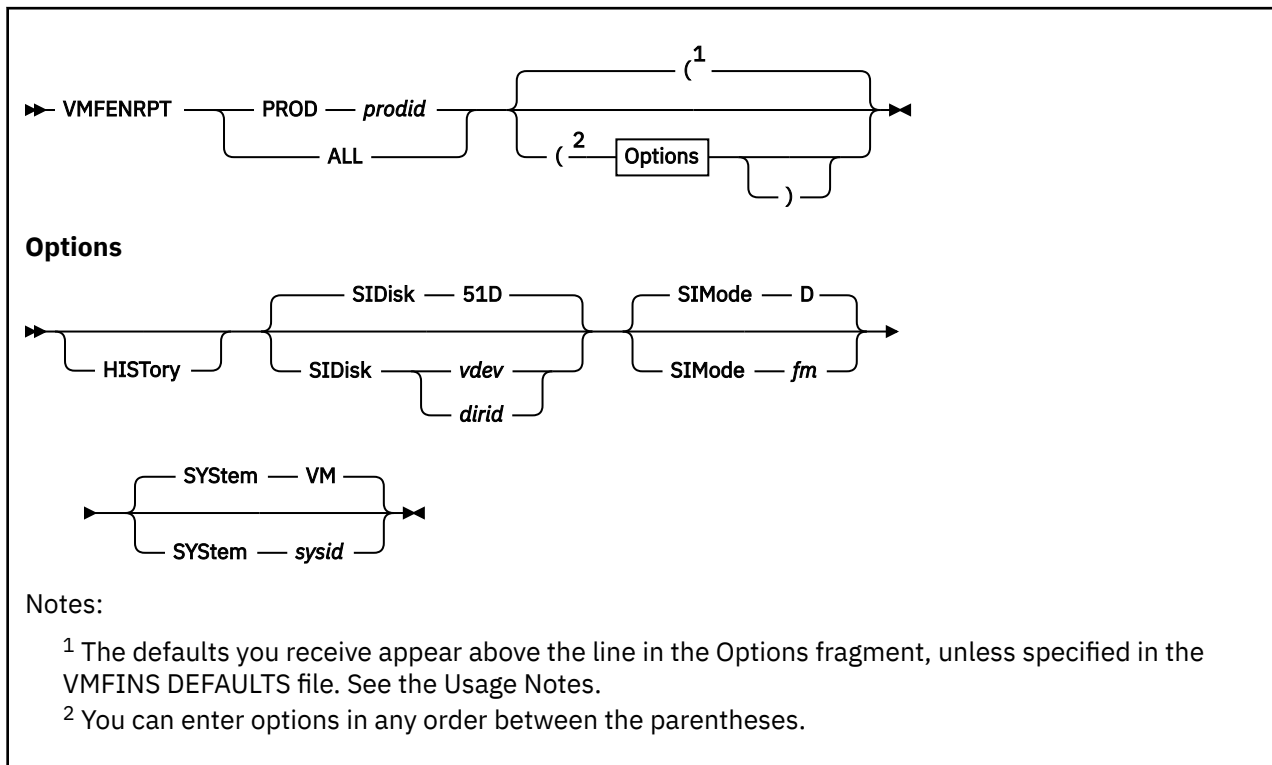
Return Code	Explanation
0	Command completed successfully.

<b>Return Code</b>	<b>Explanation</b>
2	Command completed successfully but extraneous data was encountered.
4	Command completed with one or more warning conditions.
8	Command completed but at least one major process failed.
24	Command failed because of a command line syntax error.
28	Command failed because a required file was not found.
36	Command failed because a target disk or directory was not available.
100	Command failed because of an external error.

**Recovery Information**

The VMFCOPY command can be restarted by reissuing the command.

## VMFENRPT EXEC



### Authorization

Privilege Class: E

User IDs that issue this EXEC must have a CP privilege class of E. The CP QUERY PRODUCT command requires this authority.

### Purpose

The VMFENRPT EXEC creates a report of the products that are enabled and disabled on your system. The file name of the report is VMFENRPT REPORT, and it is placed on the A-disk. This report may contain the following information, depending on the options you specify:

- The system and VMSES/E current enablement state of the products. This is the default.
- The system and VMSES/E enablement state history, which includes the current state of the products.

### Operands

#### PROD

identifies the product for which you want a report.

#### *prodid*

is the 7- or 8-character alphanumeric identifier for the product.

#### ALL

indicates you want a report on all products.

### Options

#### HISTory

indicates that you want the enablement history.

**SIDisk**

identifies where the system-level Software Inventory resides. This can be the virtual address of the minidisk or the name of the Shared File System directory.

**51D**

is the default minidisk address.

**vdev**

is the virtual address of the Software Inventory minidisk.

**dirid**

is the name of the SFS directory.

**SIMode**

identifies the file mode for the Software Inventory disk.

**D**

is the default.

**fm**

identifies the file mode for the Software Inventory disk. If you are going to continue to use this file mode, create a product parameter file override to list it on the :RETAIN tag in the product parameter file so it is not changed by future invocations of the VMFSETUP command.

**Note:** The Software Inventory disk must be accessed as read-write during all VMFINS processing. If it is not accessed as read-write, VMFINS tries to access it.

**SYStem**

identifies the name of the system-level Software Inventory.

**VM**

is the default.

**sysid**

is the name of the system-level Software Inventory.

**Usage Notes**

1. Each time you run VMFENRPT, a new report is appended to the top of the VMFENRPT REPORT file.
2. Your A-disk must be accessed as read-write.
3. This command uses the VMFINS DEFAULTS file to determine the default value for options. If you do not specify an option and there is no value assigned for that option in the VMFINS DEFAULT file, the command uses the default value that is shown in the syntax diagram. VMSES/E uses the first VMFINS DEFAULTS file found in the CMS search order. For more information about the VMFINS DEFAULTS file, see [“Changing the VMFINS Command Defaults” on page 48.](#)

**Examples**

- To run VMFENRPT, using the IBM-supplied defaults, to obtain the current state of a product identified by the *prodid*, enter:

```
VMFENRPT PROD 5735FALQ
```

- To run VMFENRPT, using the IBM-supplied defaults, to obtain the current state and the enablement history on all products, enter:

```
VMFENRPT ALL (HISTORY
```

```

*****
****  ENABLEMENT REPORT: ALL  HISTORY                      USERID: MAINT          1
****
*****
****  Date: 30 DEC 2021          Time: 08:00:18
****
*****
Product          Enablement State          Description          2
-----
5735FALQ        TCPIP          System:  DISABLED  00/00/00.00:00:00.$BASEDDR  TCP/IP LEVEL 730 - TCP/IP FEATURE (BASE)
5735FALQ        TCPIP          VMSES/E:  DISABLED  10/11/21.09:12:30.P735FALQ  TCP/IP LEVEL 730 - TCP/IP FEATURE (BASE)
                    INSTALLED  10/11/21.09:12:30.P735FALQ
-----
5735NFSQ        NOCOMP        System:  DISABLED  00/00/00.00:00:00.$BASEDDR  TCP/IP NFS FEATURE LEVEL 730 - Part of TCP/IP base
5735NFSQ        NOCOMP        VMSES/E:  DISABLED  10/11/21.09:12:30.P735FALQ  TCP/IP NFS FEATURE LEVEL 730 - Part of TCP/IP base
-----
5684096K        RSCS          System:  ENABLED    10/20/21.13:13:13.P684096K  RSCS Networking Version 7 Release 3 Modification 0
5684096K        RSCS          VMSES/E:  ENABLED    10/20/21.13:13:13.P684096K  RSCS Networking Version 7 Release 3 Modification 0
                    DISABLED  10/11/21.09:12:30.P684096K
                    INSTALLED  10/11/21.09:12:30.P684096K
-----
9999TST        TTESTSFS     System:  ENABLED    12/05/21.13:13:13.MAINT      Test Product
9999OVER       TTESTSFS     VMSES/E:  DISABLED  12/01/21.08:10:45.CTTEST     Test Product Override to change SFS filepool  3
9999TST        TTESTSFS     VMSES/E:  ENABLED    11/01/21.15:20:20.CTTEST     Test Product
                    INSTALLED  11/01/21.15:20:20.CTTEST
-----
5799ABC        System:  ENABLED
VMSES/E:  NONE          4
-----
5654A22C        CCXX         System:  NONE          5
5654A22C        CCXX         VMSES/E:  DISABLED  12/01/21.08:10:45.5654A22C  IBM XL C/C++ for z/VM Compiler
                    DELETED   12/01/21.08:10:45.5654A22C  IBM XL C/C++ for z/VM Compiler
                    ENABLED   10/31/21.17:50:00.5654A22C
                    INSTALLED 10/31/21.17:50:00.5654A22C

```

Figure 158. Product Enablement Report

Figure 158 on page 375 is an example of the VMFENRPT REPORT file. The data in this report was generated using the ALL operand with the HISTORY option. A report without the HISTORY option is similar. The difference is that for VMSES/E only the most recent enablement status is displayed (a single line of VMSES/E status is displayed or a double line of VMSES/E status is displayed if the timestamps match between two adjacent entries).

Each time you run VMFENRPT, a new report is appended to the top of the VMFENRPT REPORT file. If a VMFENRPT REPORT file does not exist, VMFENRPT creates one.

Section 1 is the report header. The information in the report header shows the VMFENRPT function used, the user ID that issued the command, and the date and time the command was issued.

Section 2 is the column header.

Under the Product column, the information shown is: 1) The *prodids* listed in the output of the QUERY PRODUCT command and 2) For VMSES/E, any product parameter file (PPF) found in the system-level apply status table (VM SYSAPPS), for the specified *prodid*, with a :ESTAT value.

Under the Enablement State column, the system data is part of the output from the QUERY PRODUCT command. The VMSES/E data is the information on the :ESTAT tag in the system-level apply status table (VM SYSAPPS). A system enablement state that contains 00/00/00.00:00:00.\$BASEDDR means this product's enablement was set on the base DDR. If no timestamp appears for the system enablement state no timestamp was present in the product description data.

Under the Description column, the system description data is part of the output from the QUERY PRODUCT command. The VMSES/E description data is taken from the system-level description table (VM SYSDESC). The description is only shown once for each PPF shown. The description could be blank, for system or VMSES/E, if none was found.

Section 3 shows there were two PPFs, 9999OVER and 9999TST, that have been used for *prodid* 9999TST.

Section 4 shows a state of NONE for VMSES/E. This means the product did not have any ESTAT value in the VM SYSAPPS table.

Section 5 shows a state of NONE for System. This means the product did not appear in the QUERY PRODUCT output.

## Input and Output Files

### Input Files

**sysid SYSAPPS**

The system-level apply status table.

**sysid SYSDSCT**

The system-level description table.

**VMFINS DEFAULTS**

The file containing the VMFINS option defaults (the established command option defaults, the overrides created by you on your A-disk, or both).

**Output Files****VMFENRPT REPORT**

The file containing the report on the enablement current state or history of the product, which is stored on the A-disk.

**Messages and Return Codes**

Appendix D, “Module Identifiers for VMSES/E Messages,” on page 749 shows the format of VMSES/E messages and lists the identifiers used by each VMSES/E EXEC.

For a complete explanation of any VMSES/E message, use the HELP Facility to view the message explanation online or see the appropriate messages documentation. To display information about a specific message - VMF002E, for example - enter:

```
help msg vmf002e
```

If you are not familiar with the z/VM HELP Facility, enter help to display the main HELP Menu. For information about the HELP command, enter:

```
help cms help
```

Appendix D, “Module Identifiers for VMSES/E Messages,” on page 749 shows the format of VMSES/E messages and lists the identifier for each VMSES/E EXEC. The VMFENRPT EXEC issues the following return codes:

Return Code	Explanation
0	Command completed successfully.
4	Command completed with one or more warning conditions.
24	Command failed because of a command line syntax error.
36	Command failed because a target disk or directory was not available.
100	Command failed because of an external error.



**LIST**

specifies the file that identifies the files to be erased.

***fn***

is the file name of the input file that identifies the files to be erased.

***ft***

is the file type of the input file that identifies the files to be erased.

**\***

is the default file mode.

***fm1***

is the file mode of the input file that identifies the files to be erased.

The input file can contain one of the following:

1. A collection of the file identifiers (file names and file types) for the parts to be erased from the parts catalog table. The file can be in the format of a CMS exec generated by the LISTFILE command with the EXEC option.

**Note:** If you specify a file mode as part of the file identifier, it is ignored.

2. A collection of TDATA statements, in the following form, that identify the files to be erased:

```
TDATA
:PARTID TEST1 FILE
:PRODID 1234567%ABCD
```

There can be multiple TDATA statements, one for each file to be erased. You can use the VMFSIM QUERY function to create the file that contains these TDATA statements. You can enter the VMFERASE command using the same file.

**FROM**

specifies the target minidisk or SFS directory where the files reside. The specified files are erased from this target, and the VMSES PARTCAT on this target is updated.

***fm2***

is the file mode of the target minidisk or SFS directory where the files reside.

**FILE**

identifies the file or files to be erased. When you enter an asterisk (\*) for the *fn* or *ft*, all file names or file types are used.

***fn***

is the file name of the files to be erased. An asterisk (\*) indicates that all files with file types that match the specified file type are to be erased from all specified file modes.

***ft***

is the file type of the files to be erased. An asterisk (\*) indicates that all files with file names that match the specified file name are to be erased from all specified file modes.

**A**

is the default file mode for the minidisk or directory from which the files are erased.

***fm***

is the file mode for the minidisk or directory from which the files are erased. If you use an asterisk (\*) as the file mode, you must specify either the file name or the file type (or both) by name. If this field is omitted, VMFERASE searches only the minidisk or directory accessed as A.

**Note:** If you specify an asterisk (\*) for both *fn* and *ft*, you must specify a valid 2-character file mode.

**LOG**

identifies the type of message logging to be done. Messages are logged in the specified message log, as well as written to the terminal.

No messages are logged until initial validation of the command is complete.

**Note:** The LOG option is reserved for use by VMSES/E.



**NONE**

is the default. If NONE is specified, messages are written only to the terminal.

**logid**

is a three-character message log identifier, for example:

**logid****Type of Log****APP**

the apply message log (\$VMFAPP \$MSGLOG A)

**BLD**

the build message log (\$VMFBLD \$MSGLOG A)

**XYZ**

the user message log (\$VMFZYX \$MSGLOG A)

**LOGLVL**

identifies the level of message logging to be done.

**Note:** The LOGLVL option is ignored if the LOG option is NONE.

**I**

is the default. 'I' logs all informational messages, warning messages, error messages, severe error messages, and terminating errors.

**mlvl**

is the message severity level. Messages are logged in the specified message log if they have a severity level equal or above the *mlvl* specified. The message levels are shown below, in order of severity. Messages of level 'R' have the lowest severity, and messages of level 'T' have the highest severity.

**mlvl****Level of logging****R**

Response required

**I**

Informational Message

**W**

Warning Message

**E**

Error Message

**S**

Severe Error Message

**T**

Terminating Error

**Usage Notes**

1. If a specified file is not found, no message is issued.
2. When you specify:

```
vmferase file fn ft *
```

VMFERASE erases files matching *fn ft* from any mode accessed Read-Write. If the file is found on a Read-Only disk, no message is issued.

3. If you specify a VMSES PARTCAT file as one of the files to be erased by a VMFERASE command with the LIST or FILE option, it is ignored. You cannot use VMFERASE to directly erase a VMSES PARTCAT file. The VMSES PARTCAT file is automatically erased, however, when VMFERASE erases the last file listed in the VMSES PARTCAT file.

**Examples**

- To use VMFERASE to erase the file, MYFILE DATA C, and update the VMSES PARTCAT file on C-disk, enter:

```
VMFERASE FILE MYFILE DATA C
```

- To use VMFERASE to erase all files from the MYCOMP component on the A-disk, enter:

```
VMFERASE PROD 1VMVMC23%MYCOMP FROM A
```

**Input and Output Files**

**Input Files**

*fn ft fm*

The file to be erased or the file containing the file identifiers of files to be erased.

**Output Files**

**VMSES PARTCAT**

The parts catalog table.

**Messages and Return Codes**

Appendix D, “Module Identifiers for VMSES/E Messages,” on page 749 shows the format of VMSES/E messages and lists the identifiers used by each VMSES/E exec.

For a complete explanation of any VMSES/E message, use the HELP Facility to view the message explanation online or see the appropriate messages documentation.

To display information on a specific message, for example VMF002E, enter:

```
help msg vmf002e
```

If you are unfamiliar with the VM HELP Facility, enter help to display the main HELP Menu. For information on the HELP command, enter:

```
help cms help
```

The VMFERASE EXEC issues the following return codes:

Return Code	Explanation
0	Command completed successfully.
8	Command completed but at least one major process failed.
24	Command failed because of a command line syntax error.
28	Command failed because a required file was not found.
36	Command failed because a target disk or directory was not available.
100	Command failed because of an external error.

**Recovery Information**

The VMFERASE command can be restarted by reissuing the command.



***ppfname***

is the file name of a usable form Product Parameter File (PPF). It must have a file type of PPF. The name of the control file that is to be used to update the source file is obtained from this PPF.

***compname***

is the name of the component (such as CP or CMS) as it is specified on the :COMPNAME tag in the PPF. *compname* is a 1-16 character alphanumeric identifier.

**Options****CKGen**

requests validation of the AUX files against the version vector tables (VVT) and issues an error message if a mismatch is detected. The version vector tables are not updated.

**LOGMOD**

requests validation of the AUX files against the version vector table (VVT) and automatically updates the local VVT when a mismatch is detected. When you specify the LOGMOD option, VMFEXUPD modifies only the VVTs that are defined in the control file above the :UPDTID level defined in the PPF. All other VVT levels are only compared to the AUX files, and mismatches are displayed. You should only use the LOGMOD option when you are processing files that have source updates. All LOCAL disks must be accessed as Read-Write.

When you use the LOGMOD option:

- If a version vector table does not exist on a LOCAL disk, it is created on the first disk in the LOCAL string.
- If the AUX file for a part is not found, the :PART entry (if found) is deleted from the version vector table.
- If the AUX file for a part is empty, the :MOD data is deleted from the version vector table for that part. The :PART entry is not deleted from the version vector table.

**NOCKGen**

requests no validation of the AUX files against the version vector tables. The AUX file structure is used to update the source file, and the VVT structure is used to name the output file.

**NOVVT**

requests no validation of the AUX files against the version vector tables. The AUX file structure is used to update the source file and name the output file.

**Note:** If you omit the CKGEN, LOGMOD, NOCKGEN, and NOVVT options, the VMFEXUPD EXEC uses the value of the :CKGEN tag in the PPF to determine whether to validate the AUX files against the VVT. If the :CKGEN tag does not appear in the PPF, no validation is performed; and NOCKGEN is assumed.

**CNTRL**

specifies that a control file is used to identify the AUX file structure.

***cntrlfn***

is the file name of the control file that is used to identify the AUX file structure. The file type of the control file is CNTRL. This value overrides the value on the :CNTRL tag in the PPF.

**FILEType**

indicates the file type for the output file that is created. This option overrides any naming from the AUX or VVT structures.

***out\_ft***

is the file type for the output file.

**FTAbbr**

is used to name the file type for the output file that is created. This option overrides the three character abbreviation that is obtained from the VM SYSABRVT table.

***ftabbr***

is the three character abbreviation used with the file type. For example, a file type of SXE12345 has SXE as the *ftabbr*.

**OUTMode**

indicates the file mode for the executable version of the source program that is created. This file mode must be accessed Read/Write.

**A**

indicates file mode A is the file mode for the executable version of the source program. A is the default.

***fm***

is the file mode for the executable version of the source program.

***mda\_string***

is the name of a symbolic string of disks from the :MDA section of the product parameter file. The executable version of the source program is placed on the first disk specified in this string.

**NO\$SElect**

does not update the *appid* \$SELECT file. NO\$SELECT is the default.

**\$SElect**

updates the *appid* \$SELECT file to indicate the executable version has been changed. The first APPLY disk specified in the :MDA section of the product parameter file must be accessed Read/Write. The other APPLY disks must be accessed.

**NOKeepsrc**

erases the updated source file after it is converted to an executable version. NOKEEPSRC is the default.

**KEEpsrc**

saves the updated source file, which consists of the source file and any updates, on the *outmode* disk. If OUTMODE is not specified, the default is the A-disk. The file is named *\$fn \$ft*. *fn* is the name of the source file, which is truncated to seven characters when necessary. *\$ft* is the file type of the source file.

**SETup**

sets up a minidisk or SFS directory access order for the VMFEXUPD function according to entries in the :MDA section of the PPF. If a user exit is specified in the product parameter file, setup will occur after the user exit is called.

**NOSetup**

does not set up a new access order.

**PREEXit**

sets up a minidisk or SFS directory access order for the VMFEXUPD function according to entries in the :MDA section of the product parameter file. If a user exit is specified in the product parameter file, setup will occur before the user exit is called.

**Note:** If the SETUP, PREEXIT, and NOSETUP options are omitted, the VMFEXUPD EXEC uses the value of the :SETUP tag in the product parameter file to determine whether to set up a new access order.

**Usage Notes**

1. VMFEXUPD handles packed files.
2. If you receive warnings or errors from the UPDATE command, check the *fn* UPDLOG file for additional information.
3. When you specify the \$SELECT option, the select data file (*appid* \$SELECT) is updated with a record consisting of either:
  - *fn* and the first 3 characters of the file type of the output file
  - *fn* and the full file type (when you also specify the FILETYPE option).

The select data file is used by VMFBLD to determine which objects need to be built using the updated executable file.

4. When you create local modifications, you can use the \$SELECT, LOGMOD, and OUTMODE options to eliminate some manual steps, such as updating the *appid* \$SELECT file, updating local version vector tables files, and saving the results on a LOCALMOD disk.
5. VMFBLD uses the version vector tables to determine the correct level of the part to use during build processing. If you do not specify the LOGMOD option, you must either manually update the version vector tables before you run VMFBLD or you must rerun VMFEXUPD and specify the LOGMOD option.
6. You must use the FTABBR *ftabbr* option when using VMFEXUPD with compiled REXX/VM parts.

### Examples

- To create an executable version of the source file, SENDFILE \$EXEC, and remove comments from the file, enter:

```
vmfexupd sendfile exec ppfname compname (comp noup
```

The COMPRESS option is a default; you do not have to specify it. The NOUPDATE option ignores update file processing and produces a file named SENDFILE EXEC from the SENDFILE \$EXEC. SENDFILE EXEC does not contain the comments.

## Input and Output Files

### Input Files

#### ***ppfname* PPF**

The usable form product parameter file.

#### ***cntrlfn* CNTRL**

The control file.

#### ***fn \$ft***

The source file.

#### ***fn updtft***

Updates to the source file.

#### ***fn AUXlvlid***

The auxiliary control file.

#### ***appid* VVTlvlid**

The version vector table.

### Output Files

#### ***fn out\_ft***

The updated source file, when the FILETYPE option is specified.

#### ***fn xxxnnnnn***

The updated source file (*xxx* is the file type abbreviation; *nnnnn* is a PTF number or LOCALMODID).

**Note:** You receive only one of the above formats.

#### ***fn* UPDLOG**

The UPDATE log file.

#### ***appid* VVTlvlid**

A version vector table when the LOGMOD option is specified.

#### ***\$fn \$ft***

The updated source file when the KEEPSRC option is specified.

#### ***appid* \$SELECT**

The list of build requirements when the \$SELECT option is specified.

### Temporary Files

#### ***fn* UPDATES**

The update history file.

**\$fn \$ft**

The updated source file unless the KEEPSRC option is specified.

**PPF Tags Used****:APPID**

The identifier of the product, which is used to name the VVT and the \$SELECT file.

**:CKGEN**

Controls the validation of AUX files against the VVT. Valid values are NO, YES, LOGMOD, and NOVVT.

**:CNTRL**

Defines the name of the control file.

**:MDA**

Defines symbolic strings and the minidisks or SFS directories associated with them.

**:SETUP**

Controls whether the VMFSETUP EXEC is called to access minidisks/directories.

**:USEREXIT**

Defines the file name of the user exit. If no value is specified, then no exit is invoked.

**Messages and Return Codes**

Appendix D, “Module Identifiers for VMSES/E Messages,” on page 749 shows the format of VMSES/E messages and lists the identifiers used by each VMSES/E exec.

For a complete explanation of any VMSES/E message, use the HELP Facility to view the message explanation online or see the appropriate messages documentation. To display information on a specific message, for example VMF002E, enter:

```
help msg vmf002e
```

If you are unfamiliar with the z/VM HELP Facility, enter help to display the main HELP Menu. For information on the HELP command, enter:

```
help cms help
```

**PI**

Return codes issued by the VMFEXUPD EXEC may be returned to a user exit. For more information about user exits, see [:USEREXIT..](#)

The VMFEXUPD EXEC issues the following return codes:

Return Code	Explanation
0	Command completed successfully.
4	Command completed with one or more warning conditions.
8	Command completed but at least one major process failed.
24	Command failed because of a command line syntax error.
28	Command failed because a required file was not found.
36	Command failed because a target disk or directory was not available.
100	Command failed because of an external error.

**PI end****Recovery Information**

The VMFEXUPD command can be restarted by reissuing the command.

## **EXECUPDT Options Supported by VMFEXUPD**

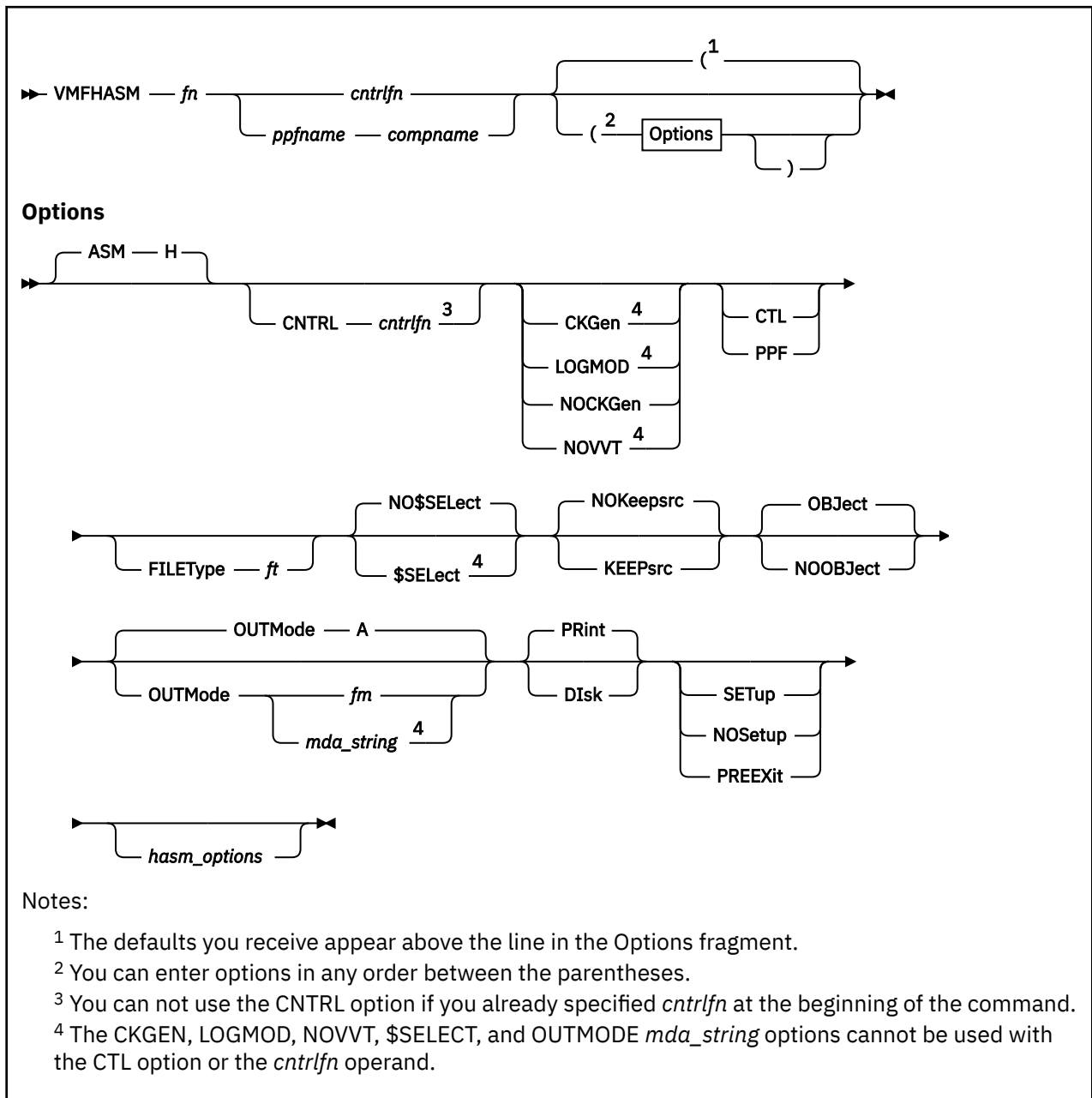
VMFEXUPD accepts all EXECUPDT command options. The EXECUPDT command is called with the HISTORY and SID options, which are not the EXECUPDT defaults. To override these options, specify NOHISTORY or NOSID on the VMFEXUPD command line. For a complete list of EXECUPDT options, see [\*z/VM: CMS Commands and Utilities Reference\*](#).

## **UPDATE Options Supported by VMFEXUPD**

VMFEXUPD accepts all UPDATE command options. For a complete list of UPDATE options, see [\*z/VM: CMS Commands and Utilities Reference\*](#).



## VMFHASM EXEC



### Purpose

The VMFHASM EXEC calls the VMFASM EXEC with the ASM H option to apply updates to a source file and then assembles the file using the H assembler.

### Operands

#### *fn*

is the file name of a source file to be updated and assembled. The source file must have a file type of ASSEMBLE.

***cntrlfn***

is the file name of a control file. The control file must have a file type of CNTRL. If there is a product parameter file with the same file name, you must specify the CTL option. If you do not, VMFHASM uses the control file that is specified in the product parameter file to update the source file and not the control file specified on the command line.

***ppfname***

is the file name of a usable form product parameter file. It must have a file type of PPF. The control file used to update the source file is obtained from this PPF.

***compname***

is the name of the component (such as CP or CMS) as it is specified on the :COMPNAME tag in the product parameter file. *compname* is a 1-16 character alphanumeric identifier.

**Options****ASM H**

tells VMFHASM to use the H assembler. ASM H is always used by the VMFHASM command. The version vector tables are not updated.

**CNTRL**

specifies a control file is used to identify the AUX file structure.

***cntrlfn***

is the file name of the control file that is used to identify the AUX file structure. The file type of the control file is CNTRL. This value overrides the value on the :CNTRL tag in the PPF. The CNTRL option can not be used if operands *fn cntrlfn* are specified.

**CKGen**

requests validation of the AUX files against the version vector tables (VVT) and issues an error message if a mismatch is detected.

**LOGMOD**

requests validation of the AUX files against the version vector tables and automatically updates the local version vector tables when a mismatch is detected. When you specify the LOGMOD option, VMFHASM modifies only the VVT files that are defined in the control file above the :UPDTID level defined in the product parameter file. All other VVT levels are only compared to the AUX files, and mismatches are displayed. You should only use the LOGMOD option when you are assembling files that have source updates. All LOCAL disks must be accessed as Read-Write.

When you use the LOGMOD option:

- If a version vector table does not exist on a LOCAL disk, it is created on the first disk in the LOCAL string.
- If the AUX file for a part is not found, the :PART entry (if found) is deleted from the version vector table.
- If the AUX file for a part is empty, the :MOD data is deleted from the version vector table for that part. The :PART entry is not deleted from the version vector table.

**NOCKGen**

requests no validation of the AUX files against the version vector tables. The AUX file structure is used to update the source file, and the VVT structure is used to name the output file.

**NOVVT**

requests no validation of the AUX files against the version vector tables (VVT). The AUX file structure is used to update the source file and name the output file.

**Note:** If you omit the CKGEN, LOGMOD, NOCKGEN, and NOVVT options, the VMFHASM EXEC uses the value of the :CKGEN tag in the product parameter file to determine whether to validate the AUX files against the version vector tables. If the :CKGEN tag does not appear in the product parameter file, no validation is performed; and NOCKGEN is assumed.

**CTL**

indicates the second operand in the command is the name of a control file. If CTL is specified, a product parameter file is not used.

**PPF**

indicates the product parameter file specifies which control file to use to update the source file. The product parameter file also lists the minidisk and directory search order.

**Note:** If you do not enter a *compname* and you do not specify CTL or PPF, CTL is assumed.

**FILEType**

indicates the file type for the output file that is created. This option overrides any naming from the AUX or VVT structures.

*ft*

is the file type for the output file.

**NO\$SElect**

does not update the *appid* \$SELECT file. NO\$SELECT is the default.

**\$SElect**

updates the *appid* \$SELECT file to indicate the text deck has been changed. The first APPLY disk specified in the :MDA section of the product parameter file must be accessed Read/Write. The other APPLY disks must be accessed.

**NOKeepsrsc**

erases the updated source file after it is assembled. NOKEEPSRC is the default.

**KEEPrsc**

indicates the updated source file, which consists of the source file and any updates, will be saved on the user's A-disk. The file is named \$*fn* ASSEMBLE. *fn* is the source file name, which is truncated to seven characters when necessary.

**OBJect**

creates the output deck on the user's A-disk. OBJECT is the default.

**NOOBJect**

does not create the output deck on the user's A-disk.

**OUTMode**

indicates the file mode for the output text and listing files created. This file mode must be accessed Read/Write.

**A**

creates the output files on file mode A. A is the default file mode.

*fm*

is the file mode for the output files.

*mda\_string*

is the name of the symbolic string of disks from the :MDA section of the product parameter file. The output is placed on the first disk specified in this string.

**PRint**

sends the listing output to the virtual printer. PRINT is the default.

**Disk**

creates the listing output on the user's A-disk.

**SETup**

sets up a minidisk or SFS directory access order for the assemble function according to the entries in the :MDA section of the product parameter file. This option is valid only when using a product parameter file. If a user exit is specified in the product parameter file, setup will occur after the user exit is called.

**NOSetup**

does not set up a new access order.

**PREEXit**

sets up a minidisk or SFS directory access order for the assemble function according to entries in the :MDA section of the product parameter file. If a user exit is specified in the product parameter file, setup will occur before the user exit is called.

**Note:** If the SETUP, PREEXIT, and NOSETUP options are omitted, the VMFHASM EXEC uses the value of the :SETUP tag in the product parameter file to determine whether to set up a new access order.

**HASM OPTIONS SUPPORTED BY VMFHASM AND VMFHASM**

In the following table, the left column shows the options of the HASM command. The right column shows how these options are supported by VMFHASM when invoking the HASM command. Also shown are the default values (underlined>) of these options. The HASM defaults are used wherever possible. Keyword-function options must be entered without the parentheses.

<b>HASM Option</b>	<b>VMFHASM Option</b>
ALIGN NOALIGN	same
BATCH NOBATCH	same
DBCS NODBCS	same
DECK NODECK	same
ESD NOESD	same
FLAG (0) FLAG(n)	FLAG 0 FLAG n
LINECOUN (55) LINECOUN(nn)	LINECOUN 55 LINECOUN nn
LIST NOLIST	same
NUM NONUM	same
OBJECT NOOBJECT	OBJect NOOBJect
PRINT NOPRINT DISK	PRint DIsk
RENT NORENT	same
RLD NORLD	same
STMT NOSTMT	same
SYSPARM(string) SYSPARM(?) SYSPARM()	SYSPARM string SYSPARM ? SYSPARM SUP SUP SYSPARM EXP EXP
TERM NOTERM	same
TEST NOTEST	same
XREF (FULL) XREF(SHORT) NOXREF	XREF FULL XREF SHORT NOXREF

**Note:** The defaults appear highlighted.

The SYSPARM SUP option suppresses the expansion of macros. The SYSPARM EXP option activates the expansion of macros. SYSPARM SUP is the default.

**Usage Notes**

1. VMFHASM handles packed files.
2. When assembling text decks for use in the VMSES/E environment, you must use a product parameter file.
3. If you receive warnings or errors from the UPDATE command, check the *fn* UPDLOG file for additional information.

4. VMFBLD uses the version vector tables to determine which level of a part to use during build processing. If you do not specify the LOGMOD option, you must either manually update the version vector tables before you run VMFBLD or you must rerun VMFHASM and specify the LOGMOD option.
5. When you specify the \$SELECT option, the select data file (*appid \$SELECT*) is updated with a record consisting of either:
  - *fn* and the first 3 characters of the file type of the output file
  - *fn* and the full file type (when you also specify the FILETYPE option)

The select data file is used by VMFBLD to determine which objects need to be built using this text deck.
6. When you create local modifications, you can use the \$SELECT, LOGMOD, and OUTMODE options to eliminate some manual steps, such as updating the *appid \$SELECT* file, updating local version vector tables files, and saving the results on a LOCALMOD disk.

## Examples

- To run VMFHASM using the IBM-supplied defaults and a product parameter file, enter:

```
VMFHASM DMSABC ppfname compname
```

- To run VMFHASM using the IBM-supplied defaults and a control file, enter:

```
VMFHASM DMSABC cntrlfn
```

## Input and Output Files

### Input Files

#### ***ppfname* PPF**

The usable form product parameter file.

#### ***cntrlfn* CNTRL**

The control file.

#### ***fn* ASSEMBLE**

The source file.

#### ***fn updtft***

Updates to the source file.

#### ***fn AUXlvlid***

The auxiliary control file.

#### ***appid VVTlvlid***

The version vector table.

### Output Files

#### ***fn* TEXT**

#### ***fn* TXTnnnnn**

#### ***fn* xxxnnnnn**

The assembled object deck (*xxx* is the file type abbreviation; *nnnnn* is a PTF number). You receive only one of these formats.

**Note:** The object deck is written to the A-disk only when the OBJECT option (the default) is specified.

#### ***\$fn* ASSEMBLE**

The updated source file, when you use the KEEPSRC option.

#### ***appid VVTlvlid***

A version vector table.

#### ***appid \$SELECT***

The list of build requirements, when you specify the \$SELECT option.

***\$fn* LISTING**

The assembler listing file.

***fn* UPDLOG**

The update log file.

***fn ctlfile***

The update information file.

**Note:** If listing output is generated during the assembly, the PRINT or DISK option determines where it will reside. The PRINT option (the default) causes all listing output to be sent to the virtual printer as *fn ctlfile*. The DISK option causes all listing output to be placed on the A-disk in two files (*\$fn* LISTING and *fn* UPDLOG).

**Temporary Files*****\$fn* ASSEMBLE**

The updated source file.

***\$fn* TEXT**

The assembled object deck.

***fn* UPDATES**

The update history file.

***\$VMFSIM* CNTRL**

A control file used with the LOGMOD option.

***fn* AUX\$\$\$\$**

An AUX file used with the LOGMOD option.

**PPF Tags Used****:APPID**

The identifier of the product, which is used to name the version vector tables and the \$SELECT file.

**:CKGEN**

Controls the validation of AUX files against the version vector tables. Valid values are NO, YES, LOGMOD, and NOVVT.

**:CNTRL**

Defines the name of the control file.

**:MDA**

Defines symbolic strings and the minidisks or SFS directories associated with them.

**:SETUP**

Controls whether the VMFSETUP EXEC is called to access minidisks/directories.

**:USEREXIT**

Defines the file name of the user exit. If no value is specified no exit is invoked.

**Messages and Return Codes**

Appendix D, “Module Identifiers for VMSES/E Messages,” on page 749 shows the format of VMSES/E messages and lists the identifiers used by each VMSES/E exec.

For a complete explanation of any VMSES/E message, use the HELP Facility to view the message explanation online or see the appropriate messages documentation. To display information on a specific message, for example VMF002E, enter:

```
help msg vmf002e
```

If you are unfamiliar with the z/VM HELP Facility, enter help to display the main HELP Menu. For information on the HELP command, enter:

```
help cms help
```

**PI**

Return codes issued by the VMFHASM EXEC may be returned to a user exit. For more information about user exits, see :USEREXIT..

The VMFHASM EXEC issues the following return codes:

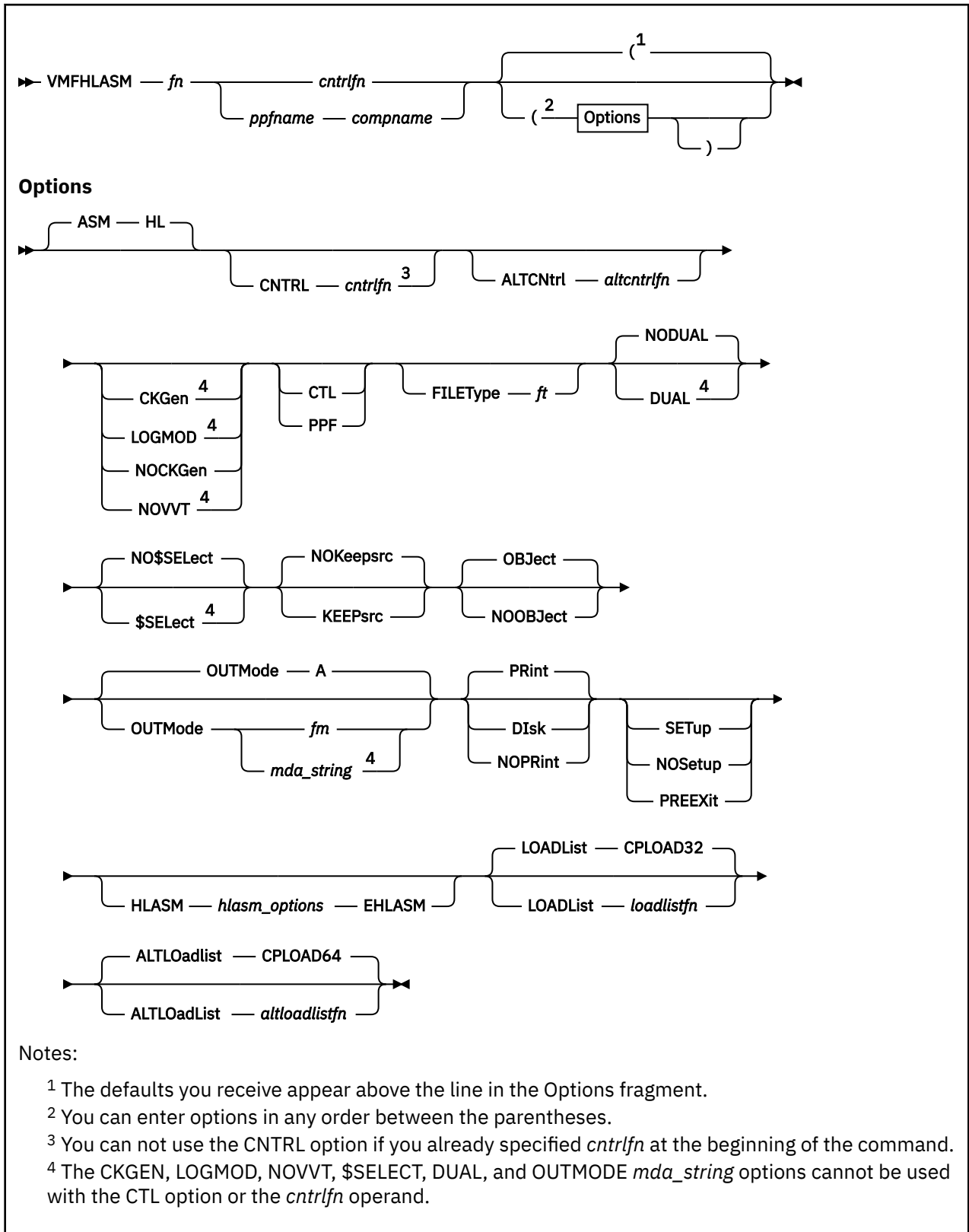
<b>Return Code</b>	<b>Explanation</b>
0	Command completed successfully.
4	Command completed with one or more warning conditions.
8	Command completed but at least one major process failed.
24	Command failed because of a command line syntax error.
28	Command failed because a required file was not found.
36	Command failed because a target disk or directory was not available.
100	Command failed because of an external error.

**PI end**

## **Recovery Information**

The VMFHASM command can be restarted by reissuing the command.

# VMFHLASM EXEC





## Purpose

The VMFHLASM EXEC calls the VMFASM EXEC with the ASM HL option to apply updates to a source file and then assembles the file using the IBM High Level Assembler (HLASM).

## Operands

### *fn*

is the file name of a source file to be updated and assembled. The source file must have a file type of ASSEMBLE.

### *cntrlfn*

is the file name of a control file. The control file must have a file type of CNTRL.

### *ppfname*

is the file name of a usable form product parameter file. It must have a file type of PPF. The control file used to update the source file is obtained from this product parameter file.

### *compname*

is the name of the component (such as CP or CMS) as it is specified on the :COMPNAME tag in the product parameter file. *compname* is a 1-16 character alphanumeric identifier.

## Options

### ASM HL

tells VMFASM to use the IBM High Level Assembler. ASM HL is always used by the VMFHLASM command. The version vector tables are not updated.

### CNTRL

specifies a control file is used to identify the AUX file structure.

#### *cntrlfn*

is the file name of the control file that is used to identify the AUX file structure. The file type of the control file is CNTRL. This value overrides the value on the :CNTRL tag in the PPF. The CNTRL option can not be used if operands *fn cntrlfn* are specified.

### ALTCNtrl

specifies a control file is used to identify the alternate AUX file structure.

#### *altcntrlfn*

is the file name of the alternate control file that is used to identify the alternate AUX file structure. The file type of the alternate control file is CNTRL. This value overrides the value on the :ALTCNTRL tag in the PPF. The ALTCNTRL option cannot be used if operands *fn cntrlfn* are specified.

### CKGen

requests validation of the AUX files against the version vector tables and issues an error message if a mismatch is detected. The version vector tables are not updated.

### LOGMOD

requests validation of the AUX files against the version vector tables and automatically updates the local version vector tables when a mismatch is detected. When you specify the LOGMOD option, VMFHLASM modifies only the VVT files that are defined in the control file above the :UPDTID level defined in the product parameter file. All other VVT levels are only compared to the AUX files, and mismatches are displayed. You should only use the LOGMOD option when you are assembling files that have source updates. All LOCAL disks must be accessed as Read-Write.

When you use the LOGMOD option:

- If a version vector table does not exist on a LOCAL disk, it is created on the first disk in the LOCAL string.
- If the AUX file for a part is not found, the :PART entry (if found) is deleted from the version vector table.

- If the AUX file for a part is empty, the :MOD data is deleted from the version vector table for that part. The :PART entry is not deleted from the version vector table.

**NOCKGen**

requests no validation of the AUX files against the version vector tables. The AUX file structure is used to update the source file, and the VVT structure is used to name the output file.

**NOVVT**

requests no validation of the AUX files against the version vector tables (VVT). The AUX file structure is used to update the source file and name the output file.

**Note:** If you omit the CKGEN, NOCKGEN, LOGMOD, and NOVVT options, the VMFHLASM EXEC uses the value of the :CKGEN tag in the product parameter file to determine whether to validate the AUX files against the version vector tables. If the :CKGEN tag does not appear in the product parameter file, no validation is performed; and NOCKGEN is assumed.

**CTL**

indicates the second operand in the command is the name of a control file. If CTL is specified, a product parameter file is not used.

**PPF**

indicates the product parameter file specifies which control file to use to update the source file. The product parameter file also lists the minidisk and directory search order.

**Note:** If you do not enter a *compname* and you do not specify CTL or PPF, CTL is assumed.

**FILEType**

indicates the file type for the output file that is created. This option overrides any naming from the AUX or VVT structures.

*ft*

is the file type for the output file.

**DUAL**

indicates the file will be assembled once or twice depending on the contents of the specified load lists. If the file, *fn*, is found in the *loadlistfn*, the file is assembled using the *cntrlfn* control file. If the file, *fn*, is found in the *altloadlistfn*, the file is assembled using the *altcntrlfn* control file. If the file is not found in either load list, the file is assembled using the *cntrlfn* control file.

**NODUAL**

indicates the file will be assembled once using the *cntrlfn* control file.

**NO\$SElect**

does not update the *appid* \$SELECT file. NO\$SELECT is the default.

**\$SElect**

updates the *appid* \$SELECT file to indicate the text deck has been changed. The first APPLY disk specified in the :MDA section of the product parameter file must be accessed Read/Write. The other APPLY disks must be accessed.

**NOKeepsrsc**

erases the updated source file after it is assembled. NOKEEPSRC is the default.

**KEEPrsc**

indicates the updated source file, which consists of the source file and any updates, will be saved on your A-disk. The file is named *\$fn* ASSEMBLE. *fn* is the source file name, which is truncated to seven characters when necessary.

**OBJect**

creates the output deck on your A-disk. OBJECT is the default.

**NOOBJect**

does not create the output deck on your A-disk.

**Note:** If the OBJECT and NOOBJECT options are specified between the HLASM and EHLASM keywords, they are ignored.

**OUTMode**

indicates the file mode for the output text and listing files created. This file mode must be accessed Read/Write.

**A**

creates the output files on file mode A. A is the default file mode.

***fm***

is the file mode for the output files.

***mda\_string***

is the name of the symbolic string of disks from the :MDA section of the product parameter file. The output is placed on the first disk specified in this string.

**PRint**

sends the output to the virtual printer in a file, *fn ctlfile*. PRINT is the default.

**DIsk**

creates the output on your A-disk in files, *\$fn LISTING* (or *\$fn LIST32* and *\$fn LIST64*) and *fn UPDLOG*.

**NOPRint**

suppresses the writing of the listing output.

**Note:** If the PRINT, NOPRINT, and DISK options are specified between the HLASM and EHLASM keywords, they are ignored.

**SETup**

sets up a minidisk or SFS directory access order for the assemble function according to the entries in the :MDA section of the product parameter file. This option is valid only when you use a product parameter file. If a user exit is specified in the product parameter file, setup will occur after the user exit is called.

**NOSetup**

does not set up a new access order.

**PREExit**

sets up a minidisk or SFS directory access order for the assemble function according to entries in the :MDA section of the product parameter file. If a user exit is specified in the product parameter file, setup will occur before the user exit is called.

**Note:** If the SETUP, PREEXIT, and NOSETUP options are omitted, the VMFHLASM EXEC uses the value of the :SETUP tag in the product parameter file to determine whether to set up a new access order.

**HLASM**

indicates the beginning of the HLASM options, which are passed directly to the HLASM command. VMFHLASM does not parse these options.

***hlasm\_options***

are the HLASM options. You can only enter *hlasm\_options* when you use the ASM HL option.

For a description of the HLASM options, see *IBM High Level Assembler/MVS & VM & VSE Programmer's Guide, SC26-4941*.

**EHLASM**

indicates the end of the HLASM options.

**LOADlist**

specifies the filename of the primary loadlist

**CPLOAD32**

is the default primary load list file name.

***loadlistfn***

is the primary load list file name.

**ALTLoadlist**

specifies the filename of the alternate load list.

**CPLOAD64**

is the default alternate load list filename.

**altloadlistfn**

is the alternate load list filename.

**Usage Notes**

1. VMFHLASM handles packed files.
2. When assembling text decks for use in the VMSES/E environment, you must use a product parameter file.
3. If you receive warnings or errors from the UPDATE command, check the *fn* UPDLOG file for additional information.
4. The High Level Assembler is called with the TERM(NARROW) and NODECK options. If the High Level Assembler command name is ASMAHL, it is also called with the FLAG(NOCONT) and USING(NOWARN) options. To override these options, specify the desired options between the HLASM and EHLASM keywords.
5. To suppress the printing of macro expansions, specify SYSPARM(SUP) between the HLASM and EHLASM keywords.
6. VMFBLD uses the version vector tables to determine which level of a part to use during build processing. If you do not specify the LOGMOD option, you must manually update the version vector tables before you run VMFBLD or you must rerun VMFHLASM and specify the LOGMOD option.
7. When you specify the \$SELECT option, the select data file (*appid* \$SELECT) is updated with a record consisting of one of the following:
  - a. *fn* and the first three characters of the file type of the output file
  - b. *fn* and the full file type (when you also specify the FILETYPE option)

The select data file is used by VMFBLD to determine which objects need to be built using this text deck.
8. When you create local modifications, you can use the \$SELECT, LOGMOD, and OUTMODE options to eliminate some manual steps, such as updating the *appid* \$SELECT file, updating local version vector tables files, and saving the results on a LOCALMOD disk.
9. HLASM 1.1 uses the command name HLASM. HLASM 1.2 uses the command name ASMAHL.

**Examples**

- To run VMFHLASM using the IBM-supplied defaults, a product parameter file, and the HLASM LINECOUNT option, enter:

```
VMFHLASM DMSABC ppfname compname (HLASM LINECOUNT(65) EHLASM
```

- To run VMFHLASM using the IBM-supplied defaults and a control file, enter:

```
VMFHLASM DMSABC cntrlfn (ctl
```

- To assemble DMSXMPL ASSEMBLE, issue the following command:

```
VMFHLASM DMSXMPL ppfname compname (CNTRL cntrlfn
```

**Input and Output Files****Input Files*****ppfname* PPF**

The usable form product parameter file.

***cntrlfn* CNTRL**

The control file.

***altcntrlfn* CNTRL**

The alternate control file.

***loadlistfn* EXEC*loadlistfn* EXC*nnnnn***

The primary load list.

***altloadlistfn* EXEC*altloadlistfn* EXC*nnnnn***

The alternate load list.

***fn* ASSEMBLE**

The source file.

***fn updtft***

Updates to the source file.

***fn* AUX*lvlid***

The auxiliary control file.

***appid* VVT*lvlid***

The version vector table.

**Output Files*****fn* TEXT*****fn* TXT*nnnnn******fn* xxx*nnnnn***

The assembled object deck (*xxx* is the file type abbreviation; *nnnnn* is a PTF number). You receive only one of these formats.

**Note:** The object deck is written to the A-disk only when the OBJECT option (the default) is specified.

***appid* VVT*lvlid***

A version vector table.

***appid* \$SELECT**

The list of build requirements, when you specify the \$SELECT option.

***\$fn* LISTING**

The assembler listing file.

***\$fn* LIST32 | *\$fn* LIST64**

The assembler listing files when two files are produced.

***fn* UPDLOG**

The update log file.

***fn* ctlfile**

The update information file.

**Note:** If listing output is generated during the assembly, the PRINT or DISK option determines where it will reside. The PRINT option (the default) causes all listing output to be sent to the virtual printer as *fn* *ctlfile*. The DISK option causes all listing output to be placed on the A-disk in files, *\$fn* LISTING (or *\$fn* LIST32 and *\$fn* LIST64) and *fn* UPDLOG.

**Temporary Files*****\$fn* ASSEMBLE**

The updated source file.

***\$fn* TEXT**

The assembled object deck.

***fn* UPDATES**

The update history file.

***\$VMFSIM* CNTRL**

A control file used with the LOGMOD option.

***fn* AUX\$\$\$\$**

An AUX file used with the LOGMOD option.

**PPF Tags Used**

**:APPID**

The identifier of the product, which is used to name the version vector tables and the \$SELECT file.

**:CKGEN**

Controls the validation of AUX files against the version vector tables. Valid values are NO, YES, LOGMOD, and NOVVT.

**:COMPNAME**

Defines the component in the product parameter file to be used.

**:CNTRL**

Defines the name of the control file.

**:ALTCNTRL**

Defines the name of the alternate control file.

**:MDA**

Defines symbolic strings and the minidisks or SFS directories associated with them.

**:SETUP**

Controls whether the VMFSETUP EXEC is called to access minidisks/directories.

**:USEREXIT**

Defines the file name of the user exit. If no value is specified, then no exit is invoked.

**Messages and Return Codes**

Appendix D, “Module Identifiers for VMSES/E Messages,” on page 749 shows the format of VMSES/E messages and lists the identifiers used by each VMSES/E exec.

For a complete explanation of any VMSES/E message, use the HELP Facility to view the message explanation online or see the appropriate messages documentation. To display information on a specific message, for example VMF002E, enter:

```
help msg vmf002e
```

If you are unfamiliar with the z/VM HELP Facility, enter help to display the main HELP menu. For information on the HELP command, enter:

```
help cms help
```

**PI**

Return codes issued by the VMFHLASM EXEC may be returned to a user exit. For more information about user exits, see :USEREXIT..

The VMFHLASM EXEC issues the following return codes:

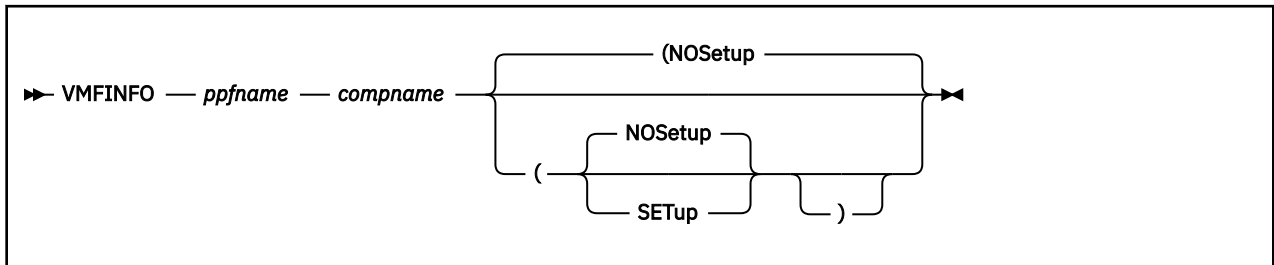
Return Code	Explanation
0	Command completed successfully.
4	Command completed with one or more warning conditions.
8	Command completed but at least one major process failed.
24	Command failed because of a command line syntax error.
28	Command failed because a required file was not found.
36	Command failed because a target disk or directory was not available.
100	Command failed because of an external error.

**PI end**

## Recovery Information

The VMFHLASM command can be restarted by reissuing the command.

## VMFINFO EXEC



### Purpose

Use the VMFINFO command to query the Software Inventory files. With VMFINFO, you can select specific query topics from the VMFINFO panels, and VMFSIM performs the queries.

### Operands

#### *ppfname*

is the file name of the usable form product parameter file (PPF) to be used for the VMFSIM queries.

#### *compname*

is the name of the component (for example, CP or CMS) to use for the queries. *compname* is a 1-16 character alphanumeric identifier.

### Options

#### **NOSetup**

does not automatically set up a new minidisk/directory access order. If the environment is not correct, incorrect results may be returned. To force a set up, change the value for "Setup" on the VMFINFO panels to YES. NOSETUP is the default.

#### **SETup**

sets up a minidisk/directory access order according to the entries in the :MDA section of the product parameter file. When SETUP is specified, a setup is performed when the command is issued or any other time there is a change to the PPF name or component name.

### Usage Notes

1. If you do not specify the PPF file name (*ppfname*) and component name (*compname*) when you issue the VMFINFO command, the PPF Fileid - Help panel is displayed. You must select a PPF from this panel. After you select the PPF, the Component Name - Help panel is displayed. You must select a component name.

If there is only one *ppfname*, it is automatically selected for you; and you do not receive the PPF Fileid - Help panel.

2. When you specify both the PPF name (*ppfname*) and the component name (*compname*), the VMFINFO Main Panel is displayed.
3. You can use the PF5 key to save the output in a file called VMFINFO *mmddhhtt*. *mmddhhtt* is the month (*mm*), day (*dd*), hour (*hh*), and minute (*tt*).

When you use the PF5 key, consecutive queries within a single invocation of VMFINFO are appended to the bottom of the file. A new file is created for each consecutive invocation.

### Examples

- To select a product parameter file to use for the queries, enter:



```
vmfinfo
```

- To access the VMFINFO Main Panel, enter:

```
vmfinfo ppfname compname
```

For more information on using the VMFINFO command, see [Chapter 17, “Using the VMFINFO Panels,”](#) on page 199.

## Input and Output Files

### Input Files

VMFINFO uses all of the Software Inventory tables, product parameter files, build lists, and control files.

### Output Files

#### VMFINFO *mmddhhtt*

The output file for the queries, if you use PF5 to save the output.

## Messages and Return Codes

For a complete explanation of any VMSES/E message, use the HELP Facility to view the message explanation online or see the appropriate messages documentation. To display information on a specific message, for example VMF002E, enter:

```
help msg vmf002e
```

If you are unfamiliar with the z/VM HELP Facility, enter `help` to display the main HELP Menu. For information on the HELP command, enter:

```
help cms help
```

Appendix D, “Module Identifiers for VMSES/E Messages,” on page 749 shows the format of VMSES/E messages and lists the identifiers for each VMSES/E exec.

VMFINFO issues the following return codes:

Return Code	Explanation
0	Command completed successfully.
12	Command failed because of an internal error.
24	Command failed because of a command line syntax error.
28	Command failed because a required file was not found.
100	Command failed because of an external error.

## Recovery Information

The VMFINFO command can be restarted by reissuing the command.

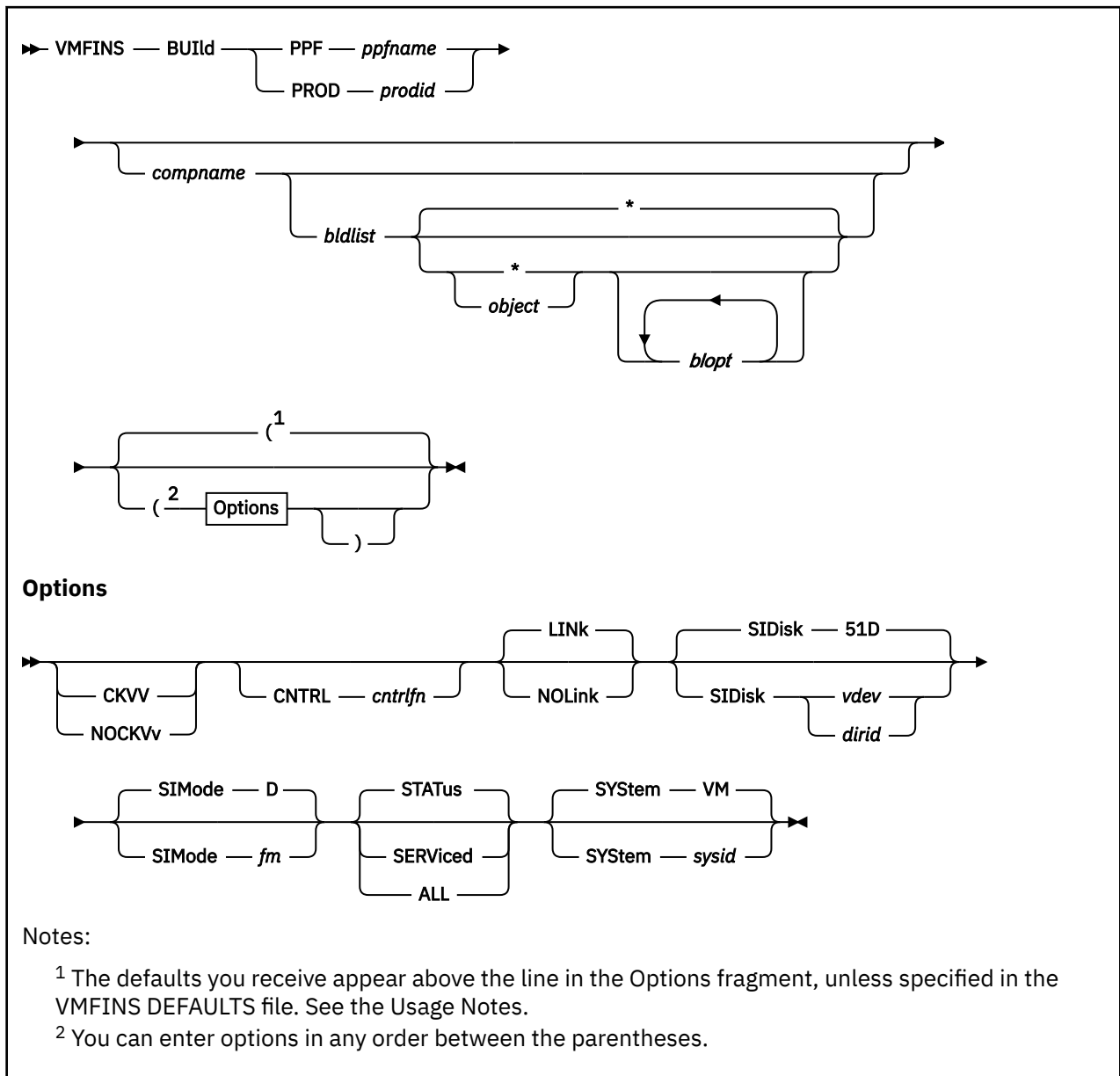
## VMFINS EXEC

---

The VMFINS EXEC provides a single, consistent, and flexible process for installing products on your z/VM system. With the VMFINS EXEC, you can:

- Print the *Memo-to-Users* for products on the installation media
- Plan to install, migrate, and delete products
- Install products
- Specify installation-related parameters
- Maintain multiple levels of products on your system
- Migrate products from one version or release to another
- Build products
- Delete products

## VMFINS BUILD Command



### Purpose

Use the VMFINS BUILD command to build a product after it has been installed or migrated on your system. VMFINS BUILD updates the system-level Software Inventory and calls the VMFBLD command.

Before you enter a VMFINS BUILD command, see [“Who Can Use VMFINS?”](#) on page 10 to make sure you have everything you need.

### Operands

#### BUILD

builds the product, updates the Software Inventory status tables, saves any segments, and verifies the product was installed or migrated correctly.

### PPF

identifies the product you want to build using a product parameter file name (*ppfname*).

#### *ppfname*

is the file name of the usable form product parameter file after all overrides have been applied. The product parameter file must have a file type of PPF.

### PROD

identifies the product you want to build using the product identifier (*prodid*).

#### *prodid*

is a 7-8 alphanumeric identifier assigned to the product by IBM.

### *compname*

is the name of the component associated with product parameter file (*ppfname*) or the product identifier (*prodid*), such as CP or CMS, or the name of a component parameter override area containing alternate parameters for the component. *compname* is a 1-16 character alphanumeric identifier.

If you do not provide a valid component name or product parameter file override name, or there is more than one to choose from, VMFINS shows you a list of names and asks you to choose one. If there is only one component, it is automatically selected for processing.

### *bldlist*

is the file name of the build list that you want to process. The build list must have a file type of EXEC. When you specify this option, you must also specify the component name (*compname*).

\*

builds all objects in the specified build list that meet the build criteria established by any other options you include on the command line.

### *object*

is the name of an object in the build list that you want to build.

**Note:** For format 1 build lists, the object name is BLDLIST.

### *blopt*

are build list options. Valid options depend on the part handler being used.

Build list options are generally operands or options of the command used to build objects. Build list options can be specified in the :BLD section of the product parameter file or when you enter a VMFINS BUILD command. When you enter build list options on the VMFINS BUILD command, they override the corresponding values in :BLD section of the product parameter file. They do not override corresponding object parameters that may appear in the build list. For more permanent overrides to the IBM supplied values, create product parameter file overrides. See [“Build List Options” on page 312](#) for valid build list options.

## Options

### CKVV

requests validation of the AUX files against the version vector tables.

### NOCKVV

requests no validation of the AUX files against the version vector tables.

**Note:** If the CKVV and NOCKVV options are omitted, the VMFINS BUILD command uses the value of the :CKVV tag in the product parameter file to determine whether to validate the AUX files against the version vector tables.

The CKVV and NOCKVV options are only used by part handlers that use the CMS UPDATE command to generate the serviceable parts for the objects being built. VMFBDMLB and VMFBDGEN are the only VMSES/E part handlers that use these options.

### CNTRL

specifies a control file is used to identify the AUX/VVT file structure.

***cntrlfn***

is the file name of the control file that is used to identify the AUX file structure. The control file must have a file type of CNTRL. This value overrides the value on the :CNTRL tag in the product parameter file.

**LINK**

links the minidisks and accesses the minidisks and SFS directories that are common to both the :MDA and :DCL sections in the product parameter file. LINK is the default.

**NOLink**

assumes you have already linked the proper minidisks and accesses the minidisks and SFS directories that are defined in the :MDA section of the product parameter file.

**SIDisk**

specifies where the Software Inventory resides. This can be either the virtual address of the minidisk or the name of the Shared File System directory.

**51D**

is the default minidisk address.

***dirid***

is the name of the Shared File System directory.

***vdev***

is the virtual address of the Software Inventory minidisk.

**SIMode**

specifies the file mode for the Software Inventory disk.

**D**

is the default.

***fm***

specifies the file mode for the Software Inventory disk. If you are going to continue to use this file mode, create a product parameter file override to list it on the :RETAIN tag in the product parameter file so it is not changed by future invocations of the VMFSETUP command.

**Note:** The Software Inventory disk must be accessed as read-write during all VMFINS processing. If it is not already accessed as read-write, VMFINS tries to access it.

**STATUS**

identifies build requirements. A build requirement exists for an object when any of the following are true:

- Any serviceable parts included in the object have been serviced. All \$SELECT files identified on the :APPID tag in the PPF file are processed to determine which parts have been serviced.
- Its object definition has been changed by service.
- It was requested from the VMFBLD command.
- It has a requisite for an object that meets any of the previous conditions.
- It has been deleted from the current level of the build list. These objects are given a status of DELETE.

New build requirements are added to the service-level build status table with a status of SERVICED or DELETE. You can use this as a planning step to see which objects have been serviced.

As part of STATUS option processing, newly-serviced source product parameter files are built to the A-disk. STATUS is the default.

**SERViced**

Performs the status function if any select data file (*appid* \$SELECT) specified on the :APPID tag in the product parameter file has been updated. Then builds the serviced objects as specified in [Table 20 on page 408](#). In addition, SERVICED also builds all objects flagged as SERVICED, DELETE, or BUILDALL in the service-level build status table.

**ALL**

Performs the status function if any select data file (*appid \$SELECT*) identified on the :APPID tag in the product parameter file has been updated. In addition, ALL builds all objects specified on the command line and assigns an initial status of BUILDALL to the objects in the service-level build status table.

The following table shows how to specify what you want to build.

*Table 20. VMFBLD EXEC Parameter Specifications and Objects Built*

<b>Bldlist</b>	<b>Object</b>	<b>Option</b>	<b>Objects Built</b>
		STATUS	None
		SERVICED	All build requirements in the service-level build status table
		ALL	All objects in all build lists in the PPF
X		STATUS	None
X		SERVICED	All build requirements for the specified build list in the service-level build status table
X		ALL	All objects in the specified build list
X	X	STATUS	None
X	X	SERVICED	The specified object, if there is a build requirement for it
X	X	ALL	The object specified

**Note:**

- The “Objects Built” column does not include any serviced build lists or product parameter source files that are automatically built as part of STATUS option processing.
- When you specify the LIST operand, VMFBLD processes each entry in the input file as described in this table.
- When you specify the SERVICED and ALL options, VMFBLD also builds all build requisites of the object being built that have a status of SERVICED or BUILDALL. VMFBLD also processes all objects in the service-level build status table that have a status of DELETE, regardless of the build list or objects specified on the command line (except if the private option is specified).

**SYStem**

specifies which system-level Software Inventory files to use.

**VM**

is the default.

**sysid**

is the file name of the system-level Software Inventory files you want to use.

**Build List Options**

See “Build List Options” on page 312 for valid build list options.

**Usage Notes**

1. If you have not spooled your console, VMFINS spools it to a reader file named VMFINS CONSOLE.
2. If a product is not in VMSES/E format, the only options valid are LINK, NOLINK, SIDISK, and SYSTEM.
3. This command uses the VMFINS DEFAULTS file to determine the default values for the LINK, NOLINK, SIDISK, SIMODE, and SYSTEM options. If you do not specify an option and there is no value assigned

for that option in the VMFINS DEFAULT file, the command uses the default value that is shown in the syntax diagram. VMSES/E uses the first VMFINS DEFAULTS file found in the CMS search order. For more information on the VMFINS DEFAULTS file, see [“Changing the VMFINS Command Defaults”](#) on page 48.

4. Build list syntax is defined in [“Build Lists”](#) on page 141 .

## Examples

For a detailed example of how to use the VMFINS BUILD command, see [“Scenario 1: Building a Product with the PPF Operand”](#) on page 77.

## Input and Output Files

### Input Files

#### ***prodid* \$PPF**

The source product parameter file.

#### ***ppfname* \$PPF**

The source or override product parameter file.

#### ***ppfname* PPF**

The usable form product parameter file.

### **VMFINS DEFAULTS**

The file containing the VMFINS option defaults (either the established command option defaults, the overrides created by you on your A-disk, or both).

### **SETUP \$LINKS**

The file containing the original minidisk access order and the modified access order as VMFINS processes.

#### ***sysid* SYSREQT**

The system-level requisite table.

#### ***sysid* SYSDESCT**

The system-level description table.

#### ***sysid* SYSRECS**

The system-level receive status table.

#### ***Bponum***

Product-specific exec sent with the product.

#### ***Vponum***

Product-specific exec sent with the product.

#### ***appid* \$SELECT**

The select data file.

#### ***appid* SRVAPPS**

The service-level apply status table.

#### ***appid* VVTlvlid**

The version vector tables specified by the control file specified on the :CNTRL tag in the product parameter file.

#### ***appid* \$APRCVRY**

(Used for apply recovery) the existence of this file on the APPLY disk string indicates VMFAPPLY was interrupted during critical processing on the last invocation of VMFAPPLY for the specific component.

### **VMFNLS LANGLIST**

The country code values for the language source files.

### **VM SYSABRVT**

The file type abbreviation table.

### Input/Output Files

### ***sysid* SYSAPPS**

The system-level apply status table.

### ***bldid* SRVBLDS**

The service-level build status table for the product.

### **Output Files**

#### **\$VMFINS \$MSGLOG**

The VMFINS message log, which is stored on your A-disk when VMFINS processing is complete.

#### **VMFINS CONSOLE**

The VMFINS console listing.

#### **VMSES PARTCAT**

The parts catalog.

### ***sysid* SYSBLDS**

The system-level build status table.

### **PPF Tags Used**

#### **:DCL**

Defines the minidisk links needed to build the product.

#### **:APPID**

The identifier of the product used during apply processing.

#### **:BLD**

Defines the buildlists for the product and the part handlers and target strings associated with them.

#### **:BLDID**

The identifier of the product used during build processing.

#### **:CKVV**

Controls the validation of AUX files against the version vector tables.

#### **:CNTRL**

Defines the name of the control file, which is used to identify the AUX file structure.

#### **:DABBV**

Defines file type abbreviations specific to a product and the real and base file types associated with them.

#### **:LOG**

Controls whether messages are logged.

#### **:MDA**

Defines symbolic strings and the minidisks and SFS directories associated with them.

#### **:NLS**

Defines the national language being used.

#### **:PRODID**

The identifier of the product, component, release, version, and modification level.

#### **:PTFPFX**

Defines the 2 letter PTF prefix for the product.

#### **:RECID**

The identifier of the product being received as it appears on the tape.

#### **:SETUP**

Controls whether the VMFSETUP EXEC is called to access minidisks or SFS directories.

#### **:USEREXIT**

Defines the file name of the user exit. If no value is specified, then no exit is invoked.



## Messages and Return Codes

For a complete explanation of any VMSES/E message, use the HELP Facility to view the message explanation online or see the appropriate messages documentation. To display information on a specific message, for example VMF002E, enter:

```
help msg vmf002e
```

If you are unfamiliar with the z/VM HELP Facility, enter help to display the main HELP Menu. For information on the HELP command, enter:

```
help cms help
```

Appendix D, “Module Identifiers for VMSES/E Messages,” on page 749 shows the format of VMSES/E messages and lists the identifier for each VMSES/E exec.

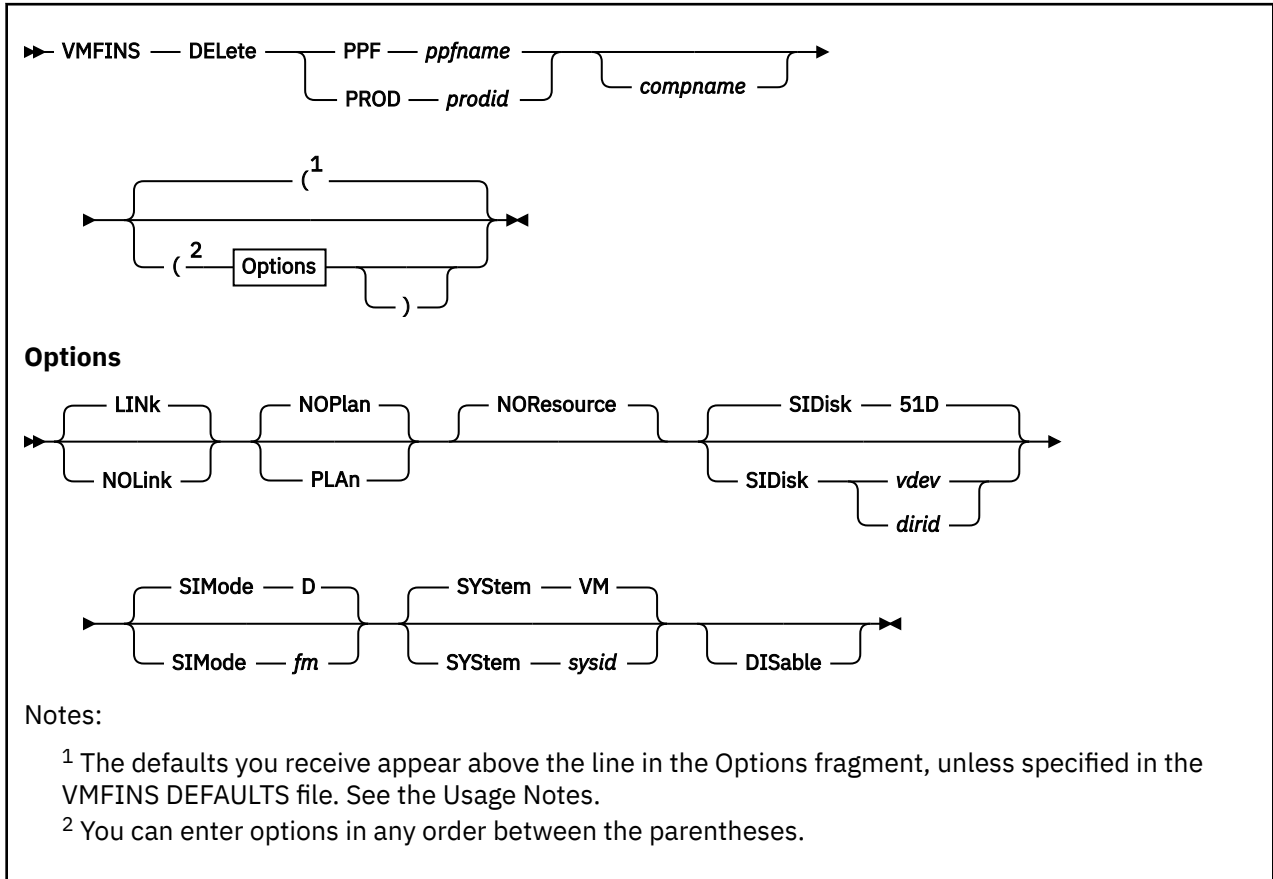
VMFINS BUILD issues the following return codes:

Return Code	Explanation
0	Command completed successfully.
2	Command completed successfully but extraneous data was encountered.
4	Command completed with one or more warning conditions.
8	Command completed but at least one major process failed.
12	Command failed because of an internal error.
24	Command failed because of a command line syntax error.
28	Command failed because a required file was not found.
36	Command failed because a target disk or directory was not available.
100	Command failed because of an external error.
500	User terminated the command from a prompt.

## Recovery Information

The VMFINS BUILD command can be restarted by reissuing the command.

## VMFINS DELETE Command



### Purpose

Use the VMFINS DELETE command to remove product code from the system, if the product was installed with VMSES/E.

Make sure you have everything you need before entering a VMFINS DELETE command. For more information, see [“Who Can Use VMFINS?”](#) on page 10.

### Operands

#### DELeTe

removes a product from the system.

#### PPF

identifies the product you want to delete using the product parameter file name (*ppfname*).

#### *ppfname*

is the file name of the usable form product parameter file after all overrides have been applied. The product parameter file must have a file type of PPF.

For more information about PPF files, see [Chapter 21, “Product Parameter File Syntax,”](#) on page 621.

#### PROD

identifies the product you want to delete using the product identifier (*prodid*).

#### *prodid*

is the 7-8 alphanumeric identifier for the product. You can use the VMFINS DELETE INFO command to get a list of *prodids*.

**compname**

is the name of the component associated with the product parameter file (*ppfname*) or the product identifier (*prodid*), such as CP or CMS, or the name of a component parameter override area containing alternate parameters for the component. *compname* is a 1-16 character alphanumeric identifier.

If you do not provide a valid component, or there is more than one to choose from, VMFINS shows you a list of names and asks you to choose one. If there is only one component, it is automatically selected for processing.

**Options****LINK**

links the minidisks and accesses the minidisks and SFS directories that are common to both the :MDA and :DCL sections of the product parameter file. LINK is the default.

**NOLink**

assumes you have already linked the proper minidisks and accesses the minidisks and SFS directories as they are defined in the :MDA section of the product parameter file.

**NOPlan**

deletes the product. NOPLAN is the default.

**PLAn**

creates a *prodid* PLANINFO file. This file contains information on product dependencies, as well as resources that will be deleted (user IDs, minidisks, and SFS directories). PLAN also creates the *ppfname* ERASE file that lists the files that will be erased during delete processing.

**Note:** When you use the PLAN option, NORESOURCE, and DISABLE are ignored.

**Delete processing does not occur.**

**NOResource**

prevents the VMFINS resource manager from automatically deleting system resources (which include user IDs, minidisks, and SFS directories), or from altering the CP directory or the CMS Shared File System in any manner. This operand is an unchangeable default and is in effect for all VMFINS DELETE commands, except those for which the PLAN operand also is specified (in which case, NORESOURCE has no meaning so is ignored). You must manually delete any system resources that are no longer required for the product code that has been removed upon successful completion of a VMFINS DELETE command that is issued for other than planning purposes.

**SIDisk**

specifies where the system-level Software Inventory resides. This can be either the virtual address of the minidisk or the name of the Shared File System directory.

**51D**

is the default minidisk address.

**vdev**

is the virtual address of the Software Inventory minidisk.

**dirid**

is the name of the SFS directory.

**SIMode**

specifies the file mode for the Software Inventory disk.

**D**

is the default.

**fm**

specifies the file mode for the Software Inventory disk. If you are going to continue to use this file mode, create a product parameter file override to list it on the :RETAIN tag in the product parameter file so it is not changed by future invocations of the VMFSETUP command.

**Note:** The Software Inventory disk must be accessed as read-write during all VMFINS processing. If it is not already accessed as read-write, VMFINS tries to access it.

## VMFINS DELETE

### **SYStem**

identifies the name of the system-level Software Inventory.

### **VM**

is the default.

### **sysid**

is the name of the system-level Software Inventory.

### **DISable**

sets up a product as disabled and deleted in VMSES/E. Deletes the product from the system enablement support.

## Usage Notes

1. Your A-disk must be accessed as read-write.
2. If you have not spooled your console, VMFINS spools it to a reader file named VMFINS CONSOLE.
3. This command uses the VMFINS DEFAULTS file to determine the default value for options. If you do not specify an option and there is no value assigned for that option in the VMFINS DEFAULT file, the command uses the default value that is shown in the syntax diagram. VMSES/E uses the first VMFINS DEFAULTS file found in the CMS search order. For more information on the VMFINS DEFAULTS file, see [“Changing the VMFINS Command Defaults” on page 48](#).

## Examples

For a detailed example of how to use the VMFINS DELETE command, see [“Scenario 1: Deleting a Product with the PPF Operand” on page 81](#).

## Input and Output Files

### Input Files

#### **prodid PRODPART**

The PRODPART file that is shipped with the product.

#### **ppfname \$PPF**

The source or override product parameter file.

#### **ppfname PPF**

The usable form product parameter file.

### **VMFINS DEFAULTS**

The file containing the VMFINS option defaults (either the established command option defaults, the overrides created by you on your A-disk, or both).

### **VMFRMT EXTENTS**

File telling the resource manager the available extents on the system on which to allocate space.

### **SETUP \$LINKS**

The file containing the original minidisk access order and the modified access order as VMFINS processes.

#### **sysid SYSREQT**

The system-level requisite table.

#### **sysid SYSDESCT**

The system-level description table.

### Input/Output Files

#### **VMSES PARTCAT**

The parts catalog.

#### **sysid SYSRECS**

The system-level receive status table.

### Output Files

**prodid PLANINFO**

Lists planning information for the product being deleted. Created when you enter a VMFINS DELETE command with the PLAN option.

**prodid \$\$EXEC\$\$**

The executable exec file containing the CP SET PRODUCT command, which is stored on the A-disk if an existing *prodid* EXEC that was not created by VMFINS was found.

**prodid EXEC**

The executable exec file containing the CP SET PRODUCT command, which is stored on the A-disk if the CP SET PRODUCT command did not complete successfully.

**prodid PRODSYS**

The file containing the CP PRODUCT system configuration statement, which is stored on the A-disk.

**ppfname ERASE**

Lists the files to be deleted for the product Created when you enter a VMFINS DELETE command with the PLAN option.

**\$VMFINS \$MSGLOG**

The VMFINS message log, which is stored on your A-disk when VMFINS processing is complete.

**VMFINS CONSOLE**

The VMFINS console log.

**sysid SYSAPPS**

The system-level apply status table.

**sysid SYSBLDS**

The system-level build status table.

**Temporary Files**

**These files are created and left on your system only if delete processing stops before it is complete.**

**ppfname \$PPFTEMP**

File created by VMFOVER; used and deleted by VMFPPF.

**PPF Tags Used****:DCL**

Defines user IDs, Shared File System directories, and links for minidisks required by the product.

**:MDA**

Identifies the minidisks or Shared File System directories that need to be accessed.

**Messages and Return Codes**

For a complete explanation of any VMSES/E message, use the HELP Facility to view the message explanation online or see the appropriate messages documentation. To display information on a specific message, for example VMF002E, enter:

```
help msg vmf002e
```

If you are unfamiliar with the z/VM HELP Facility, enter help to display the main HELP Menu. For information on the HELP command, enter:

```
help cms help
```

Appendix D, "Module Identifiers for VMSES/E Messages," on page 749 shows the format of VMSES/E messages and lists the identifier for each VMSES/E exec.

VMFINS DELETE issues the following return codes:

Return Code	Explanation
0	Command completed successfully.

## VMFINS DELETE

Return Code	Explanation
2	Command completed successfully but extraneous data was encountered.
4	Command completed with one or more warning conditions.
8	Command completed but at least one major process failed.
12	Command failed because of an internal error.
24	Command failed because of a command line syntax error.
28	Command failed because a required file was not found.
36	Command failed because a target disk or directory was not available.
100	Command failed because of an external error.
500	User terminated the command from a prompt.

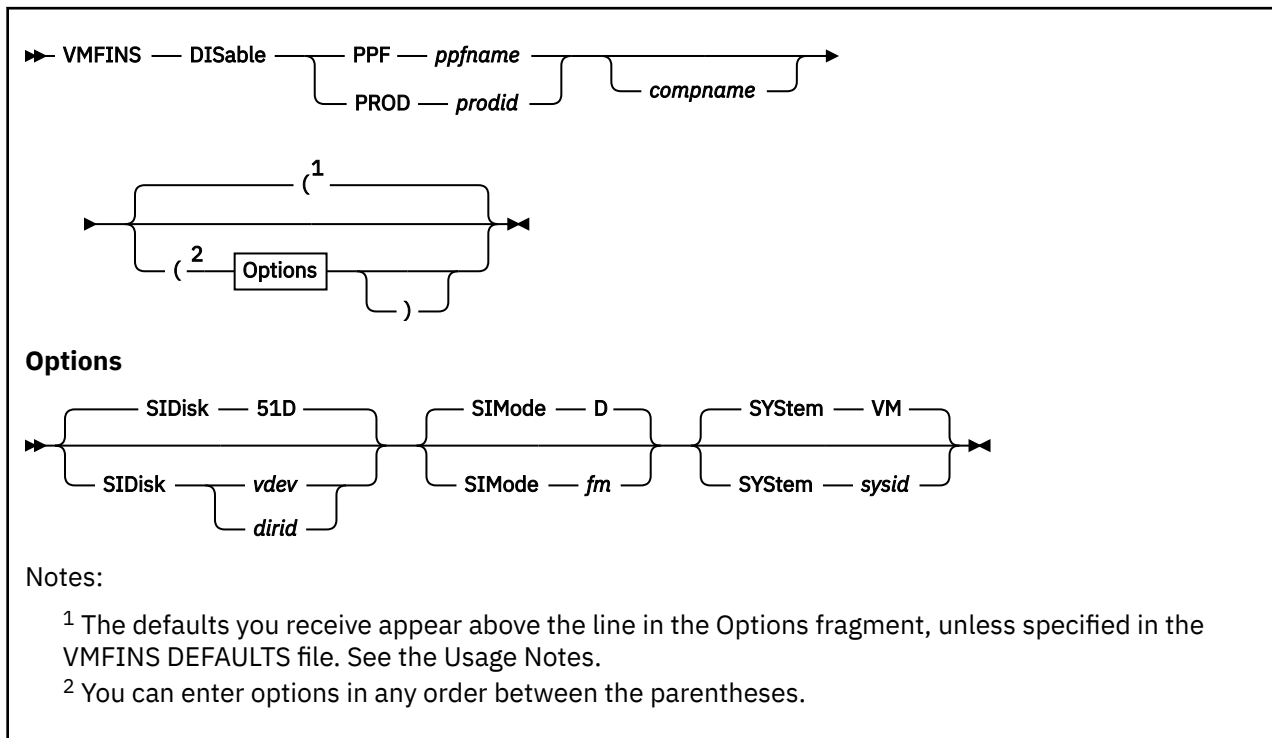
### Recovery Information

The VMFINS DELETE command can be restarted by reissuing the command.

If for some reason, a product is not successfully deleted, check the \$VMFINS \$MSGLOG file to see where an error occurred. See online help or the appropriate messages book for additional information on error messages and recommendations for fixing the errors.

Correct the problem. Then, reissue the original VMFINS DELETE command.

## VMFINS DISABLE Command



### Purpose

Use the VMFINS DISABLE command to change a product to a disabled state.

### Operands

#### DISable

changes a product to a disable state.

#### PPF

identifies the product you want to disable using the product parameter file name (*ppfname*).

#### *ppfname*

is the file name of the usable form product parameter file after all overrides have been applied. The product parameter file must have a file type of PPF.

For more information about PPF files, see [Chapter 21, “Product Parameter File Syntax,”](#) on page 621.

#### PROD

identifies the product you want to disable using the product identifier (*prodid*).

#### *prodid*

is the 7-8 alphanumeric identifier for the product.

You specify a product by entering the *prodid*. You specify a component by entering the *prodid* and the *compname*.

#### *compname*

is the name of the component associated with the product parameter override file (*ppfname*) or the product identifier (*prodid*), such as CP or CMS. *compname* is a 1-16 character alphanumeric identifier.

If you do not enter a component name, or there is more than one from which to choose, VMFINS shows you a list of names and asks you to choose one. If there is only one component, it is automatically selected for processing.

## Options

### SIDisk

identifies where the system-level Software Inventory resides. This can be either the virtual address of the minidisk or the name of the Shared File System directory.

#### 51D

is the default minidisk address.

#### vdev

is the virtual address of the Software Inventory minidisk.

#### dirid

is the name of the SFS directory.

### SIMode

identifies the file mode for the Software Inventory disk.

#### D

is the default.

#### fm

identifies the file mode for the Software Inventory disk. If you are going to continue to use this file mode, create a product parameter file override to list it on the :RETAIN tag in the product parameter file so it is not changed by future invocations of the VMFSETUP command.

**Note:** The Software Inventory disk must be accessed as read-write during all VMFINS processing. If it is not accessed as read-write, VMFINS tries to access it.

### SYStem

identifies the name of the system-level Software Inventory.

#### VM

is the default.

#### sysid

is the name of the system-level Software Inventory.

## Usage Notes

1. The product you want to disable must be in VMSES/E format and must have already been installed using VMSES/E.
2. Your A-disk must be accessed as read-write.
3. This command uses the VMFINS DEFAULTS file to determine the default value for options. If you do not specify an option and there is no value assigned for that option in the VMFINS DEFAULT file, the command uses the default value that is shown in the syntax diagram. VMSES/E uses the first VMFINS DEFAULTS file found in the CMS search order. For more information on the VMFINS DEFAULTS file, see [“Changing the VMFINS Command Defaults” on page 48](#).

## Examples

- To run VMFINS DISABLE using the IBM-supplied defaults and a product parameter file to disable a product, enter:

```
VMFINS DISABLE PPF ppfname compname
```

- To run VMFINS DISABLE using the IBM-supplied defaults and a product identifier, enter:

```
VMFINS DISABLE PROD prodid
```

## Input and Output Files

### Input Files



**ppfname \$PPF**

The source or override product parameter file.

**ppfname PPF**

The usable form product parameter file after all overrides have been applied.

**sysid SYSDESCT**

The system-level description table.

**VMFINS DEFAULTS**

The file containing the VMFINS option defaults (either the established command option defaults, the overrides created by you on your A-disk, or both).

**Output Files****prodid \$\$EXEC\$\$**

The executable exec file containing the CP SET PRODUCT command, which is stored on the A-disk if an existing *prodid* EXEC that was not created by VMFINS was found.

**prodid EXEC**

The executable exec file containing the CP SET PRODUCT command, which is stored on the A-disk if the CP SET PRODUCT command did not complete successfully.

**prodid PRODSYS**

The file containing the CP PRODUCT system configuration statement, which is stored on the A-disk.

**sysid SYSAPPS**

The system-level apply status table.

**\$VMFINS \$MSGLOG**

The VMFINS message log, which is stored on your A-disk when VMFINS processing is complete.

**Messages and Return Codes**

For a complete explanation of any VMSES/E message, use the HELP Facility to view the message explanation online or see the appropriate messages documentation. To display information on a specific message, for example VMF002E, enter:

```
help msg vmf002e
```

If you are unfamiliar with the z/VM HELP Facility, enter `help` to display the main HELP Menu. For information on the HELP command, enter:

```
help cms help
```

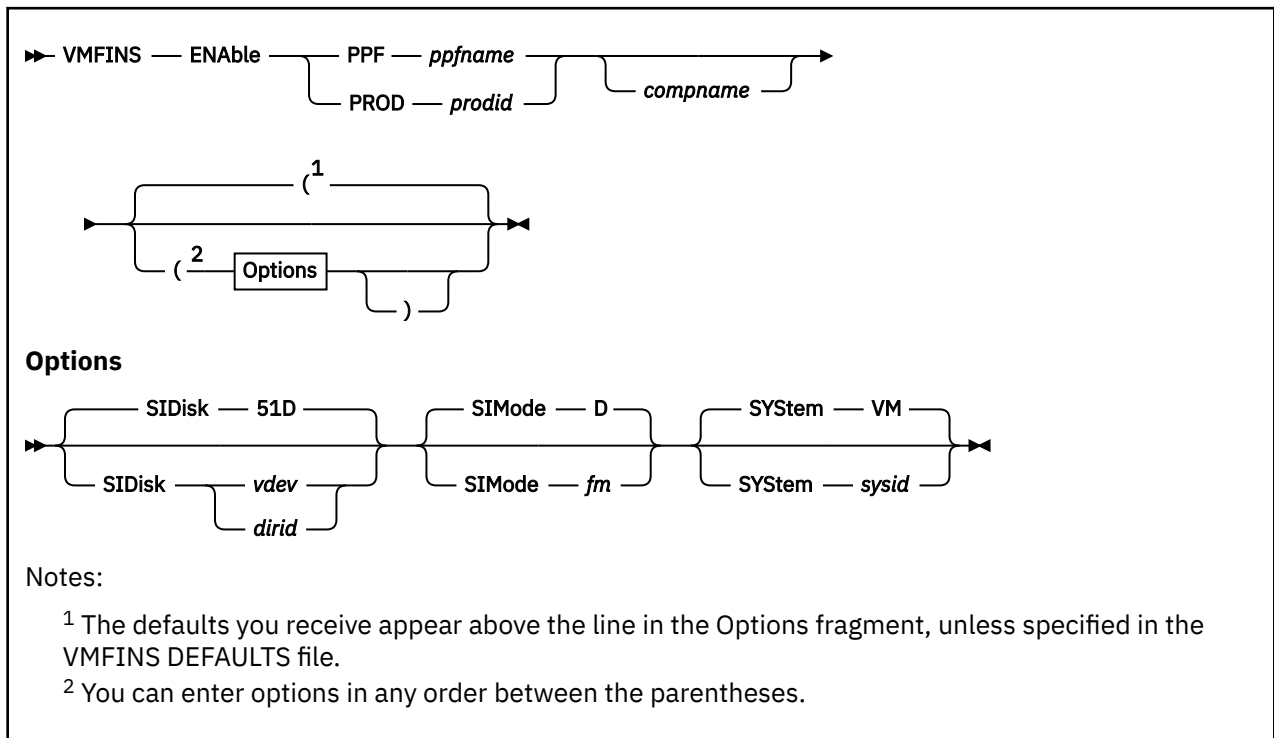
Appendix D, “Module Identifiers for VMSES/E Messages,” on page 749 shows the format of VMSES/E messages and lists the identifier for each VMSES/E exec. VMFINS DISABLE issues the following return codes:

Return Code	Explanation
0	Command completed successfully.
2	Command completed successfully but extraneous data was encountered.
4	Command completed with one or more warning conditions.
8	Command completed but at least one major process failed.
12	Command failed because of an internal error.
24	Command failed because of a command line syntax error.
28	Command failed because a required file was not found.
36	Command failed because a target disk or directory was not available.
100	Command failed because of an external error.

## VMFINS DISABLE

Return Code	Explanation
500	User terminated the command from a prompt.

## VMFINS ENABLE Command



### Purpose

Use the VMFINS ENABLE command to change a product to an enabled state.

### Operands

#### ENable

changes a product to an enabled state.

#### PPF

identifies the product you want to enable using the product parameter file name (*ppfname*).

#### *ppfname*

is the file name of the usable form product parameter file after all overrides have been applied. The product parameter file must have a file type of PPF.

For more information about PPF files, see [Chapter 21, “Product Parameter File Syntax,”](#) on page 621.

#### PROD

identifies the product you want to enable using the product identifier (*prodid*).

#### *prodid*

is the 7-8 alphanumeric identifier for the product.

You specify a product by entering the *prodid*. You specify a component by entering the *prodid* and the *compname*.

#### *compname*

is the name of the component associated with the product parameter override file (*ppfname*) or the product identifier (*prodid*), such as CP or CMS. *compname* is a 1-16 character alphanumeric identifier.

If you do not enter a component name, or there is more than one from which to choose, VMFINS shows you a list of names and asks you to choose one. If there is only one component, it is automatically selected for processing.

## Options

### SIDisk

identifies where the system-level Software Inventory resides. This can be either the virtual address of the minidisk or the name of the Shared File System directory.

#### 51D

is the default minidisk address.

#### vdev

is the virtual address of the Software Inventory minidisk.

#### dirid

is the name of the SFS directory.

### SIMode

identifies the file mode for the Software Inventory disk.

#### D

is the default.

#### fm

identifies the file mode for the Software Inventory disk. If you are going to continue to use this file mode, create a product parameter file override to list it on the :RETAIN tag in the product parameter file so it is not changed by future invocations of the VMFSETUP command.

**Note:** The Software Inventory disk must be accessed as read-write during all VMFINS processing. If it is not accessed as read-write, VMFINS tries to access it.

### SYSTEM

identifies the name of the system-level Software Inventory.

#### VM

is the default.

#### sysid

is the name of the system-level Software Inventory.

## Usage Notes

1. The product you want to enable must be in VMSES/E format and must have already been installed using VMSES/E.
2. Your A-disk must be accessed as read-write.
3. This command uses the VMFINS DEFAULTS file to determine the default value for options. If you do not specify an option and there is no value assigned for that option in the VMFINS DEFAULT file, the command uses the default value that is shown in the syntax diagram. VMSES/E uses the first VMFINS DEFAULTS file found in the CMS search order. For more information on the VMFINS DEFAULTS file, see [“Changing the VMFINS Command Defaults” on page 48](#).

## Examples

- To run VMFINS ENABLE using the IBM-supplied defaults and a product parameter file to enable a product, enter:

```
VMFINS ENABLE PPF ppfname compname
```

- To run VMFINS ENABLE using the IBM-supplied defaults and a product identifier, enter:

```
VMFINS ENABLE PROD prodid
```

## Input and Output Files

### Input Files

**ppfname \$PPF**

The source or override product parameter file.

**ppfname PPF**

The usable form product parameter file after all overrides have been applied.

**sysid SYDESCT**

The system-level description table.

**VMFINS DEFAULTS**

The file containing the VMFINS option defaults (either the established command option defaults, the overrides created by you on your A-disk, or both).

**Output Files****prodid \$\$EXEC\$\$**

The executable exec file containing the CP SET PRODUCT command, which is stored on the A-disk if an existing *prodid* EXEC that was not created by VMFINS was found.

**prodid EXEC**

The executable exec file containing the CP SET PRODUCT command, which is stored on the A-disk if the CP SET PRODUCT command did not complete successfully.

**prodid PRODSYS**

The file containing the CP PRODUCT system configuration statement, which is stored on the A-disk.

**sysid SYSAPPS**

The system-level apply status table.

**\$VMFINS \$MSGLOG**

The VMFINS message log, which is stored on your A-disk when VMFINS processing is complete.

**Messages and Return Codes**

For a complete explanation of any VMSES/E message, use the HELP Facility to view the message explanation online or see the appropriate messages documentation. To display information on a specific message, for example VMF002E, enter:

```
help msg vmf002e
```

If you are unfamiliar with the z/VM HELP Facility, enter `help` to display the main HELP Menu. For information on the HELP command, enter:

```
help cms help
```

Appendix D, “Module Identifiers for VMSES/E Messages,” on page 749 shows the format of VMSES/E messages and lists the identifier for each VMSES/E exec.

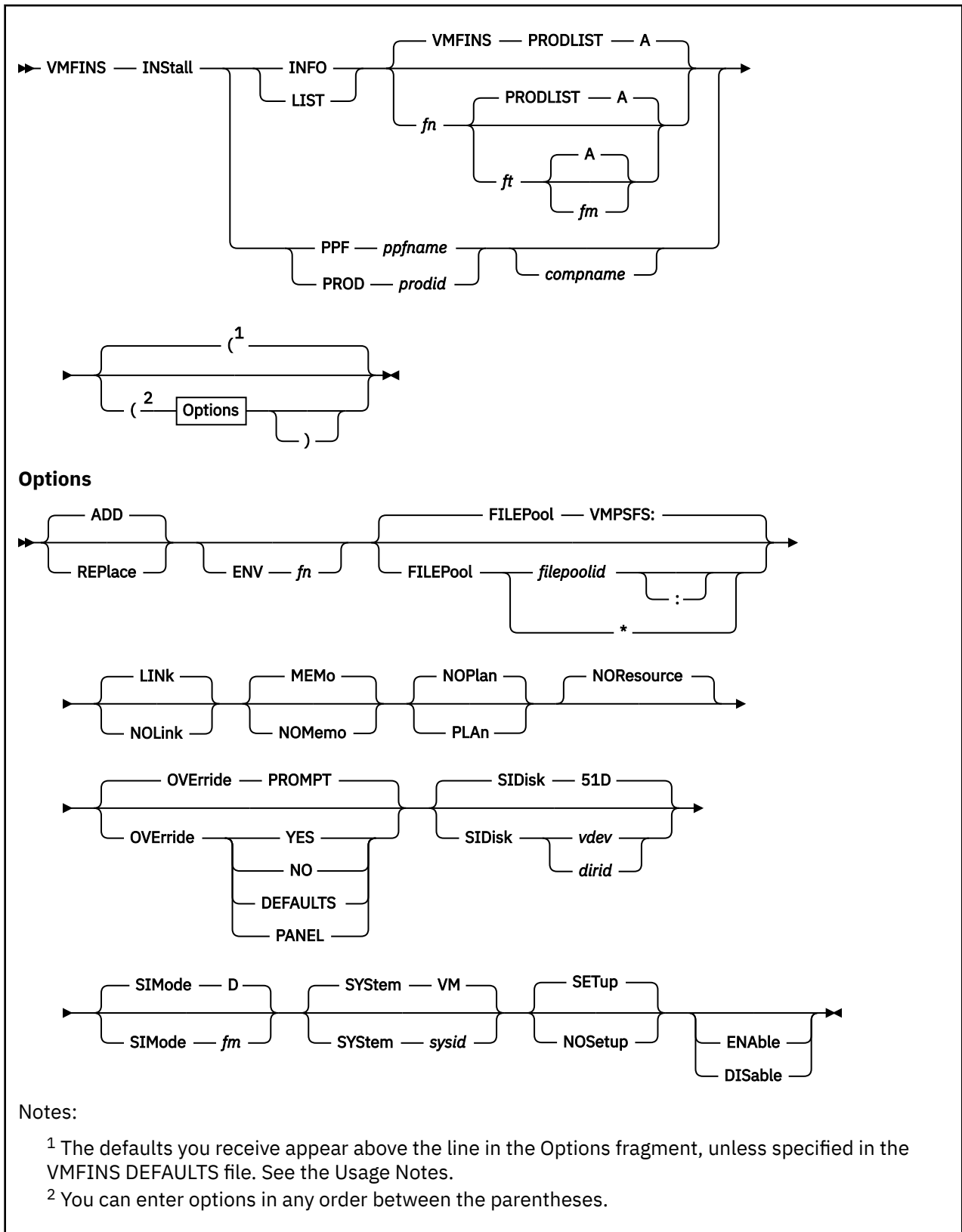
VMFINS ENABLE issues the following return codes:

Return Code	Explanation
0	Command completed successfully.
2	Command completed successfully but extraneous data was encountered.
4	Command completed with one or more warning conditions.
8	Command completed but at least one major process failed.
12	Command failed because of an internal error.
24	Command failed because of a command line syntax error.
28	Command failed because a required file was not found.
36	Command failed because a target disk or directory was not available.

**VMFINS ENABLE**

<b>Return Code</b>	<b>Explanation</b>
100	Command failed because of an external error.
500	User terminated the command from a prompt.

## VMFINS INSTALL Command



## Purpose

Use the VMFINS INSTALL command to add a copy of a new product to your system or to replace an existing copy with a new copy. Tailorings for the existing copy (not on the LOCAL disks) are not saved.

Make sure you have everything you need before entering the VMFINS INSTALL command. For more information, see [“Who Can Use VMFINS?” on page 10](#) .

## Operands

### INStall

puts a new copy of a product on the system.

### INFO

creates a list of products that can be installed and stores the list in a file on your A-disk. You can use the default file identifier for the output file, which is VMFINS PRODLIST A, or you can name the output file anything you want. When you use the INFO operand with the REPLACE option, the output file (VMFINS PRODLIST by default) lists products that are on the installation media and are currently installed on your system.

**Note:** When you use the INFO operand, the PLAN, NOPLAN, LINK, NOLINK, NORESOURCE, FILEPOOL, OVERRIDE, ENABLE, and DISABLE options are ignored.

#### *fn*

is the file name of the output file you want to create. The default is VMFINS.

#### *ft*

is the file type of the output file you want to create. The default is PRODLIST.

#### *fm*

is the file mode of the output file you want to create. The default is A.

Entries in the output file are in one of these two formats:

```
PROD prodid compname description
PPF ppfname compname description
```

The component name and description are optional. Make sure you use the correct format if you add information to this file. For more information, see [“The VMFINS PRODLIST File” on page 18](#) .

### LIST

indicates you want to install a list of products. The list of products is in an input file. If you used the defaults when you entered the VMFINS INSTALL INFO command, the file identifier is VMFINS PRODLIST A.

#### *fn*

is the file name of the input file you want to use. The default is VMFINS.

#### *ft*

is the file type of the input file you want to use. The default is PRODLIST.

#### *fm*

is the file mode of the input file you want to use. The default is A.

Entries in the input file can be in one of these two formats:

```
PROD prodid compname description
PPF ppfname compname description
```

The component name and description are optional. Make sure you use the correct format if you manually add information to this file. For more information, see [“The VMFINS PRODLIST File” on page 18](#).

### PPF

identifies the product you want to install using the product parameter file name (*ppfname*).



***ppfname***

is the file name of the usable form product parameter file after all overrides have been applied. The product parameter file must have a file type of PPF.

using the LIST operand and the PLAN option, you can find the *ppfname* in the VMFINS PRODLIST file. For more information on product parameter files, see [“The Source Product Parameter File”](#) on page 13 and Chapter 21, [“Product Parameter File Syntax,”](#) on page 621.

**PROD**

identifies the product you want to install using the product identifier (*prodid*). You specify a product by entering the *prodid*. You can specify a component by entering the *prodid* and the *compname*.

***prodid***

is the 7-8 alphanumeric identifier for the product. You can obtain a list of *prodids* by using the VMFINS INSTALL INFO command.

***compname***

is the name of the component you want to install, for example, CP or CMS. *compname* is a 1-16 character alphanumeric identifier.

If you do not enter a component name or you enter a component name that is not recognized by VMFINS, VMFINS shows you a list of valid names; and you are asked to select one. If there is only one component, it is automatically selected for processing.

**Options****ADD**

adds a new copy or an additional copy of a product or component to the system. ADD is the default.

**REPLace**

overlays an existing copy of a product or component on the system.

**ENV**

indicates the input media is an electronic envelope. If an electronically-packaged product requires more than one envelope file, you are prompted to enter the file name of the next envelope. This option is only valid for products in VMSES/E format.

***fn***

is the file name of the envelope to use. The file type must be SERVLINK. VMFINS INSTALL uses the first file found in the CMS search order with the file identifier *fn* SERVLINK.

**FILEPool**

identifies the product service Shared File System file pool. Product system data is maintained and serviced using this file pool.

**VMPSFS:**

is the default file pool ID for VMFINS processing if nothing is specified in the VMFINS DEFAULTS file.

***filepoolid***

the name (file pool ID) of the file pool. An asterisk (\*) indicates that the CMS file pool default (set by the CMS SET FILEPOOL command) is to be used. If no CMS file pool default is in effect, the VMFINS command default is used. That is, the default of VMPSFS: is used if the FILEPOOL option is omitted, or if no such option has been defined in the VMFINS DEFAULTS file.

**LINK**

issues the CP LINK commands defined in the :DCL section of the product parameter file. LINK is the default.

**NOLink**

does not issue the CP LINK commands defined in the :DCL section of the product parameter file.

**MEMo**

asks you to select the *Memo-to-Users* to print on your system printer. MEMO is the default.

**NOMemo**

does not ask you if you want to print the *Memo-to-Users*.

**Note:** Each time you enter a VMFINS INSTALL command, all *Memo-to-Users* are loaded to the Software Inventory disk from the installation media.

### **NOPlan**

installs the products. NOPLAN is the default.

### **PLAn**

creates a *prodid* PLANINFO file. This file contains product requisites and resources required for the product (user IDs, minidisks, and SFS directories). **PLAN does not generate, allocate, or commit any system resources.**

This option is not valid for the INFO operand.

If you specify the REPLACE option, PLAN also creates a *ppfname* ERASE file that lists:

- the files associated with the copy of the product you are replacing
- the resources associated with the copy of the product you are replacing

When you specify the LIST operand with the PLAN option, the VMFINS PRODLIST file is updated to show the product parameter file overrides created during PLAN processing. See [“Using the LIST Operand with the PLAN Option”](#) on page 21 for an example.

**Note:** When you use the PLAN option, the NORESOURCE, ENABLE, and DISABLE options are ignored.

### **NOResource**

prevents the VMFINS resource manager from automatically updating system resources (which includes user IDs, minidisks, and SFS directories), or from altering the CP directory or the CMS Shared File System in any manner. This operand is an unchangeable default and is in effect for all VMFINS INSTALL commands, except those for which the PLAN operand also is specified (in which case, NORESOURCE has no meaning so is ignored). You must ensure that any system resources required for the product code that is to be installed are available prior to the use of the VMFINS INSTALL command that is issued for other than planning purposes.

### **OVERRIDE**

controls the prompts for creating product parameter file overrides that are issued by VMFINS processing.

#### **PROMPT**

asks you if you want to create an override for the product you are installing. If you answer yes to this prompt, PROMPT also asks if you want to use the defaults for the product.

#### **YES**

does not ask you if you want to create an override. You are, however, asked if you want to use the defaults for the product.

#### **NO**

suppresses the prompts and does not display the Make Override Panel. NO uses the existing parameters in the product parameter file and does not give you an opportunity to change them.

#### **DEFAULTS**

suppresses both prompts and creates an override using the defaults for the product.

#### **PANEL**

suppresses both prompts and displays the Make Override Panel so you can enter new values for the installation parameters.

### **SIDisk**

specifies where the system inventory resides. This can be either the virtual address of the minidisk or the name of the Shared File System directory.

#### **51D**

is the default minidisk address.

#### **vdev**

is the virtual address of the system-level Software Inventory minidisk.

#### **dirid**

is the name of the Shared File System directory.

**SIMode**

specifies the file mode for the system-level Software Inventory disk.

**D**

is the default.

***fm***

specifies the file mode for the system-level Software Inventory disk. If you are going to continue to use this file mode, create a product parameter file override to list it on the :RETAIN tag in the product parameter file so it is not changed by future invocations of the VMFSETUP command.

**Note:** The Software Inventory disk must be accessed as read-write during all VMFINS processing. If it is not accessed as read-write, VMFINS tries to access it.

**SYSTEM**

specifies the name of the system-level Software Inventory.

**VM**

is the default.

***sysid***

is the name of the system-level Software Inventory.

**SETup**

sets up a minidisk or directory access order according to entries in the :MDA section of the product parameter file.

**NOSetup**

does not set up a new minidisk or directory access order.

**ENAb1e**

sets up the product as enabled.

**DISAb1e**

sets up the product as disabled.

**Usage Notes**

1. Your A-disk must be accessed as read-write.
2. If you have not spooled your console, VMFINS spools it to a reader file named VMFINS CONSOLE.
3. When you use the FILEPOOL option:
  - The file pool specified by *filepoolid* must be available in interactive mode.
  - You must have administrative authority for the file pool to enroll yourself in the file pool or increase the amount of space allocated.
  - You create the *filepoolid:userid.VMFINS* directory or enroll a user in the *filepoolid* file pool.
4. You should run the VMFINS BUILD command after each product installation. VMFINS BUILD builds the product on the system and updates the Software Inventory tables.
5. This command uses the VMFINS DEFAULTS file to determine the default value for options. If you do not specify an option and there is no value assigned for that option in the VMFINS DEFAULT file, the command uses the default value shown in the syntax diagram. VMSES/E uses the first VMFINS DEFAULTS file found in the CMS search order. For more information on the VMFINS DEFAULTS file, see [“Changing the VMFINS Command Defaults” on page 48](#).
6. There is a difference between requisite and prerequisite products if they are missing when you issue a VMFINS INSTALL command. If a requisite is missing, the install will complete and you will receive a message indicating there is a run time requisite but the install can complete and is successful. If a prerequisite is missing, the install is unsuccessful. This is considered an installation requisite so that is why the install does not complete.
7. If you choose to have VMFINS override the default installation parameters provided for the product, a product parameter file override is created for every component in the specified PPF. Any changes that

you provide using the Make Override Panel are made to the specified component and to each of the other components if applicable.

### Examples

For detailed examples of how to use the VMFINS INSTALL command, see [“Scenario 1: Installing a Product with the PPF Operand”](#) on page 54.

## Input and Output Files

### Input Files

#### ***prodid* PRODPART**

The PRODPART file that is shipped with the product.

#### ***ppfname1* \$PPF**

The source and override product parameter file.

#### ***ppfname1* PPF**

The usable form product parameter file.

#### **VMFINS PRODLIST**

The file created when you enter a VMFINS INSTALL command with the INFO operand.

#### **VMFNLS LANGLIST**

The country code values for the language source files.

#### **VMFINS DEFAULTS**

The file containing the VMFINS option defaults (either the established command option defaults, the overrides created by you on your A-disk, or both).

### Output Files

#### **VMFINS PRODLIST**

Created when you enter a VMFINS INSTALL command with the INFO operand.

#### ***prodid* PLANINFO**

Lists planning information for the product.

#### ***prodid* \$\$EXEC\$\$**

The executable exec file containing the CP SET PRODUCT command, which is stored on the A-disk if an existing *prodid* EXEC that was not created by VMFINS was found.

#### ***prodid* EXEC**

The executable exec file containing the CP SET PRODUCT command, which is stored on the A-disk if the CP SET PRODUCT command did not complete successfully.

#### ***prodid* PRODSYS**

The file containing the CP PRODUCT system configuration statement, which is stored on the A-disk.

#### ***ppfname* ERASE**

Lists files that will be deleted for the product.

#### **\$VMFINS \$MSGLOG**

The VMFINS message log, which is stored on your A-disk when VMFINS processing is complete.

#### **VMFINS CONSOLE**

The VMFINS console log.

#### ***ppfname2* \$PPF**

The override product parameter file.

#### ***ppfname2* PPF**

The usable form product parameter file.

#### ***sysid* SYSREQT**

The system-level requisite table.

#### ***sysid* SYSDESCT**

The system-level description table.

**sysid SYSRECS**

The system-level receive status table.

**sysid SYSAPPS**

The system-level apply status table.

**Temporary Files****VMFRMT EXTENTS**

The file listing the available extents on the system.

**VMFRMT \$NEWCP\$**

The temporary CP user directory used to build the new one.

**VMFRMT \$TMPCP\$**

The temporary CP user directory to backup original.

**PPF Tags Used****:DCL**

Defines user IDs, Shared File System directories, and links for minidisks required by the product.

**:MDA**

Identifies the section that lists the minidisks or Shared File System directories that need to be accessed.

**:RECINS**

Defines the tape files included on the installation media and the part handlers and target strings associated with them.

**Messages and Return Codes**

For a complete explanation of any VMSES/E message, use the HELP Facility to view the message explanation online or see the appropriate messages documentation. To display information on a specific message, for example VMF002E, enter:

```
help msg vmf002e
```

If you are unfamiliar with the z/VM HELP Facility, enter help to display the main HELP Menu. For information on the HELP command, enter:

```
help cms help
```

Appendix D, “Module Identifiers for VMSES/E Messages,” on page 749 shows the format of VMSES/E messages and lists the identifiers for each VMSES/E exec.

VMFINS issues the following return codes:

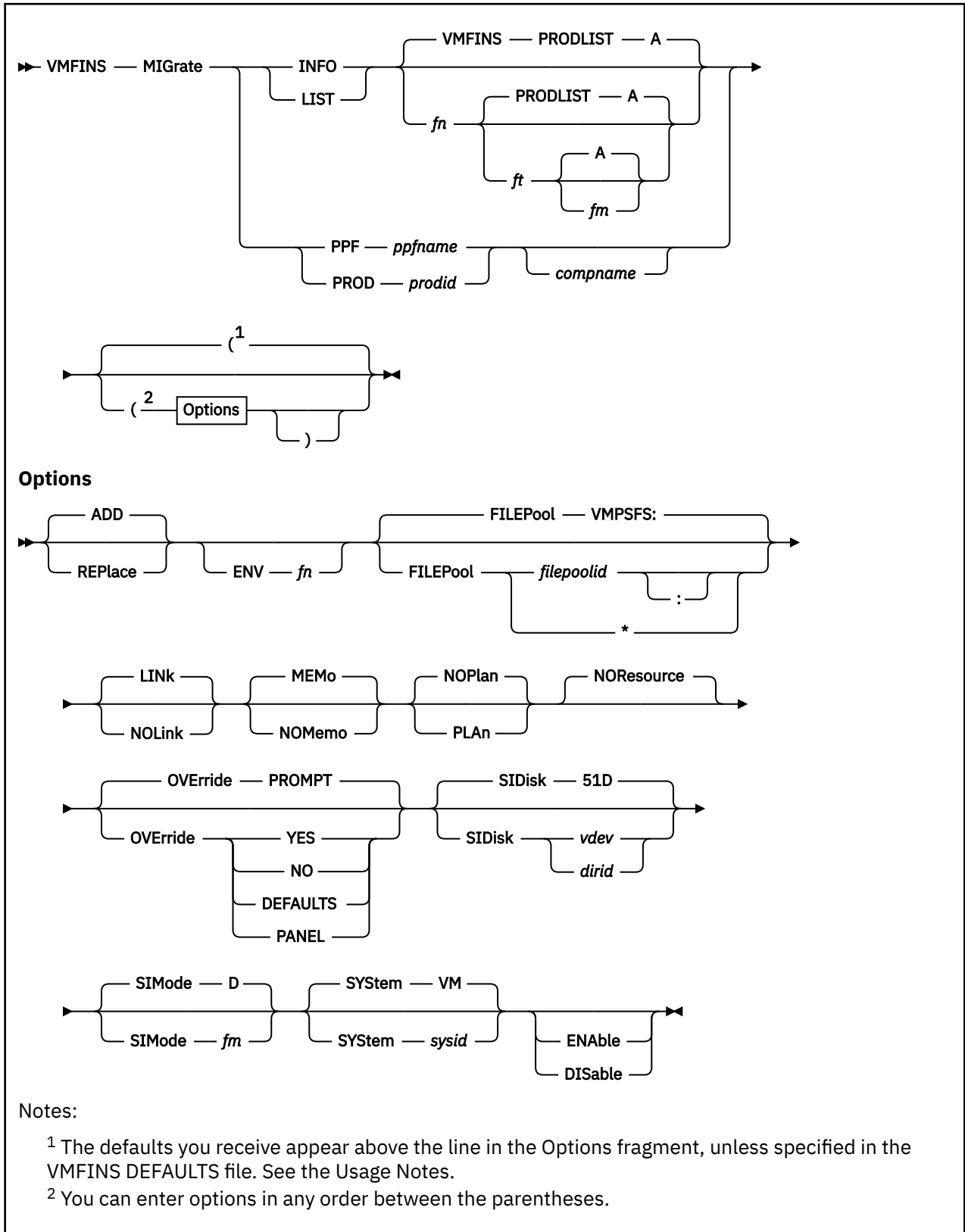
Return Code	Explanation
0	Command completed successfully.
2	Command completed successfully but extraneous data was encountered.
4	Command completed with one or more warning conditions.
8	Command completed but at least one major process failed.
12	Command failed because of an internal error.
24	Command failed because of a command line syntax error.
28	Command failed because a required file was not found.
36	Command failed because a target disk or directory was not available.
100	Command failed because of an external error.
500	User terminated the command from a prompt.

### Recovery Information

A VMFINS INSTALL command can be restarted by reissuing the command.

If an installation ends unsuccessfully, check the \$VMFINS \$MSGLOG and VMFINS CONSOLE files to see where the error occurred. For example, there may not be enough contiguous space available or there was an error in the product parameter file. See *z/VM: Other Components Messages and Codes* for additional information on error messages and recommendations for fixing the errors. Fix the problem and rerun the installation using the same command. If resources have already been generated, VMFINS will not create duplicate resources when you rerun install.

# VMFINS MIGRATE Command



## Purpose

Use the VMFINS MIGRATE command to put a new copy of a product or a list of products on your system while keeping any user tailorings, such as tailored files and SFS file authorizations and aliases. VMFINS MIGRATE also determines reach-ahead service, if there is any. You can migrate to a new copy of a product, if the product was installed with VMSES/E.

Make sure you have everything you need before entering the VMFINS MIGRATE command. For more information, see [“Who Can Use VMFINS?” on page 10](#).

## Operands

### MIGrate

puts a new copy of the product on the system and preserves the tailorings from a previously installed or migrated copy of the product.

### INFO

creates a list of products that can be migrated and stores the list in an output file. You can use the default file identifier for the output file (VMFINS PRODLIST A), or you can create a new file identifier.

#### *fn*

is the file name of the output file you are creating. The default is VMFINS.

#### *ft*

is the file type of the output file you are creating. The default is PRODLIST.

#### *fm*

is the file mode of the output file you are creating. The default is A.

Entries in the output list are in one of these two formats:

```
PPF ppfname compname description  
PROD prodid compname description
```

The component name and description are optional. Make sure the information is in the correct format if you manually add information to this file. For more information, see [“The VMFINS PRODLIST File” on page 18](#).

**Note:** When you use the INFO operand, the PLAN, NOPLAN, LINK, NOLINK, NORESOURCE, FILEPOOL, OVERRIDE, ENABLE, and DISABLE options are ignored.

### LIST

indicates you want to migrate a list of products. The list of products is in an input file. If you used the defaults when you entered the VMFINS MIGRATE INFO command, the file identifier is VMFINS PRODLIST A.

#### *fn*

is the file name of the input file you want to use. The default is VMFINS.

#### *ft*

is the file type of the input file you want to use. The default is PRODLIST.

#### *fm*

is the file mode of the input file you want to use. The default is A.

Entries in the input file are in one of these two formats:

```
PPF ppfname compname description  
PROD prodid compname description
```

The component name and description are optional. If you manually add information to this file, make sure the information is in the correct format. For more information, see [“The VMFINS PRODLIST File” on page 18](#).

### PPF

identifies the product you want to migrate using the product parameter file name (*ppfname*).



***ppfname***

is the file name of the usable form product parameter file after all overrides have been applied. The product parameter file must have a file type of PPF. For more information about PPF files, see Chapter 21, “Product Parameter File Syntax,” on page 621 the information about product parameter file syntax in *z/VM: VMSES/E Introduction and Reference*.

**PROD**

identifies the product you want to migrate using the product identifier (*prodid*).

***prodid***

is the 7-8 alphanumeric identifier for the product. You can use the VMFINS MIGRATE INFO command to get a list of *prodids*.

You specify a product by entering the *prodid*; you specify a component by entering the *prodid* and the *compname*.

***compname***

is the name of the component associated with the product parameter file override (*ppfname*) or the product identifier (*prodid*), such as CP or CMS. *compname* is a 1-16 character alphanumeric identifier.

If you do not enter a component name, or there is more than one to choose from, VMFINS shows you a list of valid names; and you are asked to select one. If there is only one component, it is automatically selected for processing.

**Options****ADD**

adds a new copy of the product to your system and preserves the tailorings from the current copy of the product. ADD is the default.

**REPlace**

lays the new version of the product over the existing copy on your system and preserves the tailorings from that existing copy of the product.

**ENV**

indicates the input media is an electronic envelope. If an electronically-packaged product requires more than one envelope file, you are prompted to enter the file name of the next envelope. This option is only valid for products in VMSES/E format.

***fn***

is the file name of the envelope to use. The file type must be SERVLINK. VMFINS MIGRATE uses the first file found in the CMS search order with the file identifier *fn* SERVLINK.

**FILEPool**

identifies the product service Shared File System file pool. Product system data is maintained and serviced using this file pool.

**VMPSFS:**

is the default file pool ID for VMFINS processing if nothing is specified in the VMFINS DEFAULTS file.

***filepoolid***

the name (file pool ID) of the file pool. An asterisk (\*) indicates that the CMS file pool default (set by the CMS SET FILEPOOL command) is to be used. If no CMS file pool default is in effect, the VMFINS command default is used. That is, the default of VMPSFS: is used if the FILEPOOL option is omitted, or if no such option has been defined in the VMFINS DEFAULTS file.

**LINK**

links the minidisks and accesses the minidisks and SFS directories that are common to both the :MDA and :DCL sections of the product parameter file. LINK is the default.

**NOLink**

accesses the minidisks and SFS directories that are common to both the :MDA and :DCL sections of the product parameter file. This option assumes you have already linked the proper minidisks.

## VMFINS MIGRATE

### MEMo

asks you to select the *Memo-to-Users* to print on your system printer. MEMO is the default.

### NOMemo

does not ask you if you want to print the *Memo-to-Users*.

**Note:** Each time you enter a VMFINS MIGRATE command, all *Memo-to-Users* are loaded to the Software Inventory disk from the installation media.

### NOPlAn

migrates the product. NOPLAN is the default.

### PLAn

creates a *prodid* PLANINFO file. This file contains information on product requisites and resources required for the product (user IDs, minidisks, and SFS directories).

If you specify the REPLACE option, PLAN also creates a *ppfname* ERASE file that lists:

- the files associated with the copy of the product you are replacing
- the resources associated with the copy of the product you are replacing

**PLAN does not generate, allocate, or commit any system resources.** This option is not valid with the INFO operand.

**Note:** When you use the PLAN option, the NORESOURCE, ENABLE, and DISABLE options are ignored.

### NOResource

prevents the VMFINS resource manager from automatically updating system resources (which include user IDs, minidisks, and SFS directories), or from altering the CP directory or the CMS Shared File System in any manner. This operand is an unchangeable default and is in effect for all VMFINS MIGRATE commands, except those for which the PLAN operand also is specified (in which case, NORESOURCE has no meaning so is ignored). You must ensure that any system resources required for the product code that is to be installed are available prior to the use of the VMFINS MIGRATE command that is issued for other than planning purposes.

### OVeRride

controls the prompts issued by VMFINS processing.

### PROMPT

asks you if you want to create an override for the product you are installing. If you answer yes to this prompt, PROMPT also asks if you want to use the defaults for the product.

### YES

does not ask you if you want to create an override. You are, however, asked if you want to use the defaults for the product.

### NO

suppresses the prompts and does not display the Make Override Panel. NO uses the existing parameters in the product parameter file and does not give you an opportunity to change them.

### DEFAULTS

suppresses both prompts and creates an override using the defaults for the product.

### PANEL

suppresses both prompts and displays the Make Override Panel so you can enter new values for the installation parameters.

### SIDisk

specifies where the system-level Software Inventory resides. This can be either the virtual address of the minidisk or the name of the Shared File System directory.

### 51D

is the default minidisk address.

### vdev

is the virtual address of the Software Inventory minidisk.

***dirid***

is the name of the Shared File System directory.

**SIMode**

specifies the file mode for the Software Inventory disk.

**D**

is the default.

***fm***

specifies the file mode for the Software Inventory disk. If you are going to continue to use this file mode, create a product parameter file override to list it on the :RETAIN tag in the product parameter file so it is not changed by future invocations of the VMFSETUP command.

**Note:** The Software Inventory disk must be accessed as read-write during all VMFINS processing. If it is not accessed as read-write, VMFINS tries to access it.

**SYSTEM**

specifies the name of the system-level Software Inventory.

**VM**

is the default.

***sysid***

is the name of the system-level Software Inventory.

**ENABLe**

sets up the product as enabled.

**DISAbLe**

sets up the product as disabled.

**Usage Notes**

1. The product you want to migrate from must be in VMSES/E format and must have already been installed using VMSES/E.
2. If you have not spooled your console, VMFINS spools it to a reader file named VMFINS CONSOLE.
3. Your A-disk must be accessed as read-write.
4. If you have specified a file pool in the product parameter file, you must have administrative authority for the file pool to:
  - Enroll users
  - Create directories for other users
  - Increase the amount of space allocated
5. VMFINS MIGRATE uses a temporary directory (*filepoolid:userid.VMFINS*) to migrate products. *filepoolid* is the value specified on the FILEPOOL option.
6. You must be enrolled in the Shared File System file pool specified in the VMFINS DEFAULTS file or specified on the VMFINS MIGRATE command (VMPSFS:*userid.VMFINS* by default), because VMFINS MIGRATE uses this Shared File System directory for its migration save area.
7. When you use the NORESOURCE option, you should be a SFS administrator or own the files in the SFS directories associated with the product you want to migrate. Otherwise, file level SFS authorizations and aliases might not be restored.
8. When you use the FILEPOOL option:
  - The file pool specified by *filepoolid* must be available in interactive mode.
  - You must have administrative authority for the file pool to enroll yourself in the file pool or increase the amount of space allocated.
  - You must create the *filepoolid:userid.VMFINS* directory or enroll a user in the *filepoolid* file pool.
9. This command uses the VMFINS DEFAULTS file to determine the default value for options. If you do not specify an option and there is no value assigned for that option in the VMFINS DEFAULT file, the

command uses the default value that is shown in the syntax diagram. VMSES/E uses the first VMFINS DEFAULTS file found in the CMS search order. For more information on the VMFINS DEFAULTS file, see [“Changing the VMFINS Command Defaults” on page 48](#).

10. If reach-ahead service for the product you are migrating is found on your system, a *prodid* \$APPLIST file is created. You will see several messages in the \$VMFINS \$MSGLOG file saying there are program temporary fixes (PTFs) that need to be re-applied. For more information on re-applying those PTFs, see the service documentation for the product you are migrating.
11. Run the VMFINS BUILD command after each product migration. The VMFINS BUILD command builds the product on the system and updates the Software Inventory tables to reflect the build.
12. If you choose to have VMFINS override the default installation parameters provided for the product, a product parameter file override is created for every component in the specified PPF. Any changes you provide using the Make Override Panel are made to the specified component and to each of the other components if applicable.

### Examples

For more information on how to use the VMFINS MIGRATE command, see [“The Source Product Parameter File” on page 13](#).

## Input and Output Files

### Input Files

#### ***prodid* PRODPART**

The source product parameter file.

#### ***ppfname* \$PPF**

The source or override product parameter file after all overrides have been applied.

#### ***ppfname* PPF**

The usable form product parameter file after all overrides have been applied.

### **VMFINS DEFAULTS**

The file containing the VMFINS option defaults (either the established command option defaults, the overrides created by you on your A-disk, or both).

### **VMFRMT EXTENTS**

File telling the resource manager the available extents on the system on which to allocate space.

### **SETUP \$LINKS**

The file containing the original minidisk access order and the modified access order as VMFINS processes.

### **Input/Output Files**

#### **VMFINS PRODLIST**

Created when you enter a VMFINS MIGRATE command with the INFO operand or used when you enter a VMFINS MIGRATE command with the LIST operand.

#### **VMSES PARTCAT**

Parts catalog table which identifies the product that owns all parts residing on a disk and the VMSES/E command that last modified or created the part.

#### ***sysid* SYSAPPS**

Software Inventory table containing information on which products have been applied on the system.

#### ***sysid* SYSREQT**

The system-level requisite table.

#### ***sysid* SYSDESCT**

The system-level description table.

#### ***sysid* SYSRECS**

The system-level receive status table.

### **Output Files**

***prodid* \$\$EXEC\$\$**

The executable exec file containing the CP SET PRODUCT command, which is stored on the A-disk if an existing *prodid* EXEC that was not created by VMFINS was found.

***prodid* EXEC**

The executable exec file containing the CP SET PRODUCT command, which is stored on the A-disk if the CP SET PRODUCT command did not complete successfully.

***prodid* PRODSYS**

The file containing the CP PRODUCT system configuration statement, which is stored on the A-disk.

***prodid* PLANINFO**

Lists planning information for the product being migrated. Created when you enter a VMFINS MIGRATE command with the PLAN option.

***ppfname* ERASE**

Lists the files to be deleted for the product. Created when you enter a VMFINS MIGRATE command with the REPLACE and PLAN options.

**\$VMFINS \$MSGLOG**

The VMFINS message log, which is stored on your A-disk when VMFINS processing is complete.

**VMFINS CONSOLE**

The VMFINS console log.

**Temporary Files****IBMSRC \$VMFREST**

File containing the IBM tailorable file information needed to migrate the product.

**CUSSRC \$VMFREST**

File containing your tailorable file information needed to migrate the product.

**VMFRMT EXTENTS**

The file listing the available extents on the system.

**VMFRMT \$NEWCP\$**

The temporary CP user directory used to build the new one.

**VMFRMT \$TMPCP\$**

The temporary CP user directory to backup the original.

**Note:** These files are created and left on your system only if the migration ends before it is complete. They are the work files that are found in the migration save area, which is created during the migration.

**PPF Tags Used****:DCL**

Defines user IDs, Shared File System directories, and links for minidisks required by the product.

**:MDA**

Identifies the minidisks or Shared File System directories that need to be accessed.

**:RECINS**

Defines the tape files included on the installation media and the part handlers and target strings associated with them.

**Messages and Return Codes**

For a complete explanation of any VMSES/E message, use the HELP Facility to view the message explanation online or see the appropriate messages documentation. To display information on a specific message, for example VMF002E, enter:

```
help msg vmf002e
```

If you are unfamiliar with the z/VM HELP Facility, enter help to display the main HELP Menu. For information on the HELP command, enter:

```
help cms help
```

Appendix D, “Module Identifiers for VMSES/E Messages,” on page 749 shows the format of VMSES/E messages and lists the identifier for each VMSES/E exec.

VMFINS MIGRATE issues the following return codes:

Return Code	Explanation
0	Command completed successfully.
2	Command completed successfully but extraneous data was encountered.
4	Command completed with one or more warning conditions.
8	Command completed but at least one major process failed.
12	Command failed because of an internal error.
24	Command failed because of a command line syntax error.
28	Command failed because a required file was not found.
36	Command failed because a target disk or directory was not available.
100	Command failed because of an external error.
500	User terminated the command from a prompt.

## Recovery Information

There are several reasons a migration process could complete unsuccessfully. Depending on the point in the migration where the error occurred, you could remigrate the product or do a few extra steps to complete your migration.

VMFINS creates a log of the messages (\$VMFINS \$MSGLOG) issued during your migration and stores it on your A-disk. You should look in this file for the commands entered and any error messages that were issued during your migration. The VMFINS CONSOLE file can also be helpful with the responses you entered during your migration. It is spooled to your reader during your migration. *You may want to print these files for reference during the following steps.* They will guide you through the recovery process.

You should complete these steps before using another VMFINS command.

### Step 1. Re-IPL CMS

Enter:

```
ipl cms
```

to return CMS to a running state.

### Step 2. Try to Determine What Stopped Your Migration

#### Comments

While your migration was processing, messages were displayed on your screen and logged in the \$VMFINS \$MSGLOG file and the VMFINS CONSOLE file. These files can give you other hints to what went wrong. See online help or the appropriate messages book for additional information on error messages and recommendations for fixing the errors.

If you are unable to determine what went wrong during your migration after looking at the \$VMFINS \$MSGLOG file, you must re-install your product, retailer the product files you had on your system for that product, and delete all files from the migration save area, instead of completing the migration. For information on re-installing the product, see [Chapter 4, “Installing Products with VMFINS,”](#) on page 49

## Step 3. Correct the Problem

### Comments

If you determine the error was system related, see online help or the appropriate messages book and contact your system administrator or your IBM representative.

After you have determined what went wrong during your migration, correct the problem and go to the next step.

## Step 4. Finish Your Migration

### Comments

When the problem has been fixed, there are two ways to continue – one for the ADD option; one for the REPLACE option: **If you used the ADD option,**

1. Access the migration save area as file mode C using:

```
set filepool vmpsfs:
```

```
access .vmfins c
```

**Note:** The VMFINS subdirectory is the default migration save area.

2. Erase all the files in the migration save area on your C disk.
3. Reissue the command you used to migrate your product.

**Note:** If you cannot remember the exact options you used, refer to the VMFINS CONSOLE file in your reader for the command line entry that started your migration.

### If you used the REPLACE option,

1. Determine where the migration stopped. Check the \$VMFINS \$MSGLOG file on your A-disk for the following message:

```
VMFDEF2739I Now deleting files for ppfname compname (prodid)
```

2. **If you do not find the message VMFDEF2739:**

- a. Access the migration save area as file mode C using these commands:

```
set filepool vmpsfs:
```

```
access .vmfins c
```

**Note:** The VMFINS subdirectory is the default migration save area.

- b. Erase all the files in the migration save area on your C disk.
- c. Reissue the command you used to migrate your product.

**Note:** If you cannot remember the exact options you used, refer to the VMFINS CONSOLE file in your reader for the command line entry that started your migration.

3. **If you found the message VMFDEF2739:**

- a. Look again in the \$VMFINS \$MSGLOG file for the following message:

```
VMFRES1960I VMFREST processing completed successfully
```

- b. If you find this message, the migration is complete.
- c. If you do not find this message, go on to the next step.

## Step 5. Manually Complete Your Migration (Optional)

### Comments

#### Note:

The following instructions are used only if you are recovering from a migration that was invoked with the REPLACE option and you cannot restart it.

By following the next set of steps, you can recover the product files that were set up for the old copy of the product.

*To ensure that a valid copy of the product is on the system, install the product using the REPLACE option.*

After you have successfully installed the product using the VMFINS INSTALL command with the REPLACE option, the following manual steps are required to retailor the product's system files and SFS attributes.

**Note:** The tailored files stored in the migration save area have modified file types. This is done to avoid a name conflict when a tailored file with an identical file name and file type is stored in the save area. The original file type can be determined by examining the \$VMFINS \$MSGLOG file as described in step [“1”](#) on [page 442](#) below.

1. To restore the tailored files for your product, look in the \$VMFINS \$MSGLOG file for the following message:

```
VMFSAV2715I Tailored part fn1 ft1 fm1 (vaddr1|dir1)
             saved as fn2 ft2 fm2 (vaddr2|dir2)
```

This message may appear more than once. It shows what file identifiers VMFINS MIGRATE gave to your tailored files when they were copied to the migration save area.

Use the following steps to manually tailor these files:

- a. Access the new product version of the file by entering:

```
access vaddr1|dir1 fm1
```

**Note:** Use the *vaddr1* or *dir1* and the first file mode specified in message VMFSAV2715I.

- b. Access your old version of the file by entering:

```
access vaddr2|dir2 fm2
```

**Note:** Use the *vaddr2* or *dir2* and the second file mode specified in message VMFSAV2715I.

- c. Compare these two files to see what changes from your file need to be incorporated into the product file. You can compare these by printing copies of the files or by using XEDIT and the SET SCREEN 2 XEDIT command to view both files in a split screen session.
  - d. Combine the tailorings from your file into the product file and save the product file on the disk or directory accessed as *fm1*.
  - e. Repeat steps [“1.a”](#) on [page 442](#) and [“1.b”](#) on [page 442](#) for **each** file that was identified with message VMFSAV2715I, then go on to the next step.
2. To restore your SFS file authorizations for your product, look in the \$VMFINS \$MSGLOG file for the following messages:

```
VMFSAV2715I Authorizations saved for fn ft dir
VMFSAV2121I grantee read write
```

Message VMFSAV2121I may appear more than once. These messages show the file in message VMFSAV2715I had all the file authorizations listed in messages VMFSAV2121I.

Use the following steps to manually grant these authorizations:



- a. Grant these file authorizations using:

```
grant authority fn ft dir to grantee (authority
```

*fn ft dir* are the same as in message VMFSAV2715I, and *grantee* is the same as in message VMFSAV2121I. *authority* is WRITE if message VMFSAV2121I has a “X” in the write column. Otherwise, *authority* is READ, which is the command default.

- b. Repeat this command for each *grantee* listed in the VMFSAV2121I messages.
- c. Repeat steps “2.a” on page 443 and “2.b” on page 443 for **each** file and directory pair identified with message VMFSAV2715I.
3. To restore your SFS file aliases for your product, look in the \$VMFINS \$MSGLOG file for the following messages:

```
VMFSAV2715I Aliases saved for fn1 ft1 dir1
VMFSAV2121I fn2 ft2 dir2
```

Message VMFSAV2121I may appear more than once. These messages show the file in message VMFSAV2715I had all the aliases listed in messages VMFSAV2121I.

Use the following steps to manually create your file aliases:

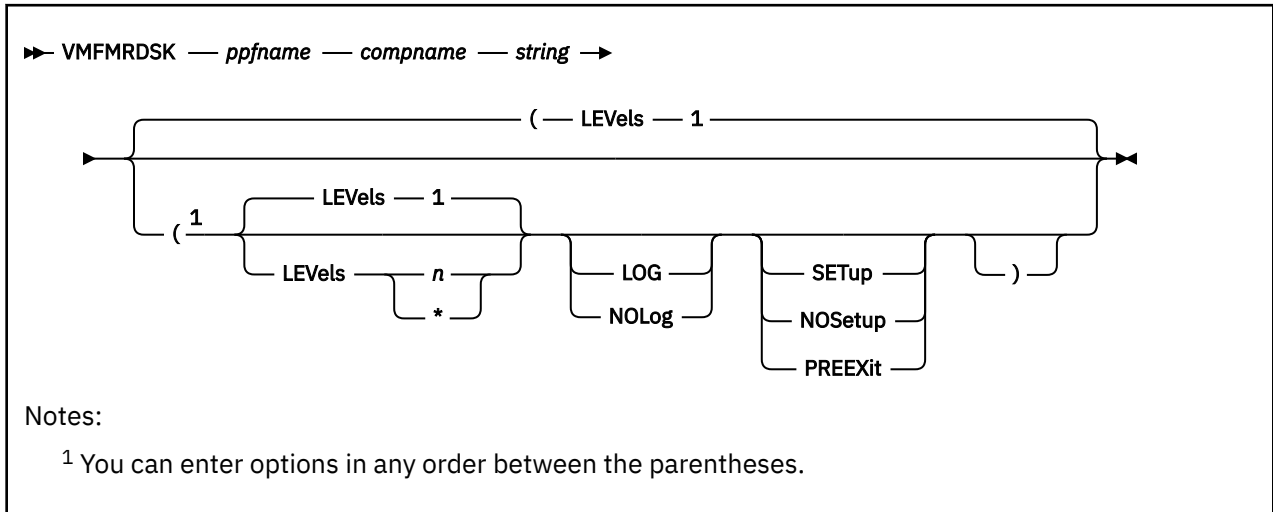
- a. Recreate these file aliases using:

```
create alias fn1 ft1 dir1 fn2 ft2 dir2
```

*fn1 ft1 dir1* are the same as in message VMFSAV2715I, and *fn2 ft2 dir2* is the same as in message VMFSAV2121I.

- b. Repeat this command for each file alias found in the VMFSAV2121I message.
- c. Repeat steps “3.a” on page 443 and “3.b” on page 443 for **each** file and directory pair identified with message VMFSAV2715I.

## VMFMRDSK EXEC



### Purpose

VMFMRDSK consolidates the contents of the specified minidisks or Shared File System directories within a string. Merging is done for a specific string in pairs of minidisks or directories, which are called levels. In a string, you can specify any number of levels to be merged.

### Operands

#### *ppfname*

is the file name of the usable form product parameter file. The product parameter file must have a file type of PPF.

#### *compname*

is the name of the component as it is specified on the :COMPNAME tag in the product parameter file. *compname* is a 1-16 character alphanumeric identifier.

#### *string*

is a symbolic name for a disk string (defined in the :MDA section of the product parameter file). Valid values are DELTAxxx, APPLYxxx, or LOCALxxx. The variable xxx is a qualifier for different disk strings within the same category of disks, such as LOCAL1 and LOCAL2.

### Options

#### LEVels

Indicates how many levels of disks to merge. Each level is made up of one disk pair, so the number of levels of disks that can be merged is one less than the total number of disks in the string. For example, with three disks in a string, there are two levels that can be merged in the string. If you specify more levels than the number of levels defined in the product parameter file, a message is issued; and no merge is performed.

#### 1

is the default. One disk level is to be merged.

#### n

is the specific number of levels to be merged.

#### \*

indicates that all disk levels are to be merged.

**LOG**

writes VMFMRDSK messages into the merge message log (\$VMFMRD \$MSGLOG).

No messages are logged until initial validation of the command is complete.

**NOLog**

does not write VMFMRDSK messages into the merge message log (\$VMFMRD \$MSGLOG).

**Note:** If the LOG and NOLOG options are omitted, the VMFMRDSK EXEC uses the value of the :LOG tag in the product parameter file to determine whether to log VMFMRDSK messages into the merge message log.

**SETup**

sets up a minidisk or SFS directory access order for the merge function according to entries in the :MDA section of the product parameter file. If a user exit is specified in the product parameter file, setup will occur after the user exit is called.

**NOSetup**

does not set up a new access order.

**PREEXit**

sets up a minidisk or SFS directory access order for the merge function according to entries in the :MDA section of the product parameter file. If a user exit is specified in the product parameter file, setup will occur before the user exit is called.

**Note:** If the SETUP, PREEXIT, and NOSETUP options are omitted, the VMFMRDSK EXEC uses the value of the :SETUP tag in the product parameter file to determine whether to set up a new access order.

**Usage Notes**

1. VMFMRDSK requires all source and target disks in the string being merged to be accessed as read-write.

**Examples**

The following examples assume the APPLY string defined in the :MDA section of the product parameter file is:

```
APPLY 004 003 002 001
```

- To merge one level of the APPLY string, enter:

```
VMFMRDSK ppfname compname APPLY (LEVELS 1
```

In this example,

1. The contents of 004 are copied to 003, and 004 is erased.

- To merge two levels of the APPLY string, enter:

```
VMFMRDSK ppfname compname APPLY (LEVELS 2
```

In this example,

1. The contents of 003 are copied to 002, and 003 is erased.
2. The contents of 004 are copied to 003, and 004 is erased.

- To merge all levels of the APPLY string, enter either of these two commands:

```
VMFMRDSK ppfname compname APPLY (LEVELS *
```

or

```
VMFMRDSK ppfname compnameAPPLY (LEVELS 3
```

In this example,

1. The contents of 002 are copied to 001, and 002 is erased.
2. The contents of 003 are copied to 002, and 003 is erased.
3. The contents of 004 are copied to 003, and 004 is erased.

## **Input and Output Files**

### **Input Files**

#### ***ppfname* PPF**

The usable form product parameter file.

#### **\$DASD\$ CONSTS**

The DASD constants file (used for space calculations).

#### ***appid* SRVAPPS**

The service-level apply status table.

#### ***appid* \$APRCVRY**

(used for recovery) The existence of this file on the APPLY disk string indicates VMFAPPLY was interrupted during critical processing on the last invocation of VMFAPPLY for the specified component.

#### **VMSES \$PARTCAT**

A temporary copy of the parts catalog table.

### **Output Files**

#### **\$CRDSK\$ \$FILES\$**

A file that allows you to read-only access source disks in the future by acting as a place holder.

#### **VMSES PARTCAT**

The parts catalog.

### **Temporary Files**

#### **\$TRGLST\$ EXEC**

List of files on the target disk.

#### **\$SRCLST\$ EXEC**

List of files on the source disk.

### **PPF Tags Used**

#### **:APPID**

The identifier of the product used during apply processing.

#### **:LOG**

Controls whether messages are logged.

#### **:MDA**

Defines symbolic strings and the minidisks or SFS directories associated with them.

#### **:PRODID**

The identifier of the product, component, release, version, and modification level.

#### **:RECID**

The identifier of the product used when it was received.

#### **:SETUP**

Controls whether the VMFSETUP EXEC is called to access minidisks/directories.

#### **:USEREXIT**

Defines the file name of the user exit. If no value is specified no exit is invoked.

## **Messages and Return Codes**

Appendix D, "Module Identifiers for VMSES/E Messages," on page 749 shows the format of VMSES/E messages and lists the identifiers used by each VMSES/E exec.

For a complete explanation of any VMSES/E message, use the HELP Facility to view the message explanation online or see the appropriate messages documentation. To display information on a specific message, for example VMF002E, enter:

```
help msg vmf002e
```

If you are unfamiliar with the z/VM HELP Facility, enter help to display the main HELP Menu. For information on the HELP command, enter:

```
help cms help
```

## PI

Return codes issued by the VMFMRDSK EXEC may be returned to a user exit. For more information about user exits, see [:USEREXIT.](#)

The VMFMRDSK EXEC issues the following return codes:

Return Code	Explanation
0	Command completed successfully.
4	Command completed with one or more warning conditions.
12	Command failed because of an internal error.
24	Command failed because of a command line syntax error.
28	Command failed because a required file was not found.
36	Command failed because a target disk or directory was not available.
100	Command failed because of an external error.

## PI end

## Recovery Information

The VMFMRDSK command can be restarted by reissuing the command.

If the target disk is not big enough, VMFMRDSK does not attempt the merge and issues an error message.



## Purpose

VMFNLS applies updates to a source file specific to a national language and then generates one or more object decks for that file.

## Operands

### *fn*

is the file name of a source file that is to be converted to text.

### **ASSEMBLE**

is the file type of an assembler source part that is to be converted to text. The text decks generated from assembler source parts are created using the ASSEMBLE command.

### **DLCS**

is the file type of a definition language for command syntax (DLCS) source parts that are to be converted to text. The text decks generated from DLCS parts are created using the GENCMD command.

### **REPOS**

is the file type of a message repository source part that is to be converted to text. The text decks generated from message repository source parts are created using the GENMSG command.

### *cntrlfn*

is the file name of a control file. The control file must have a file type of CNTRL. If there is a product parameter file with the same file name, you must specify the CTL option. If you do not, VMFNLS uses the control file that is specified in the product parameter file to update the source file and not the control file specified on the command line.

### *ppfname*

is the file name of a usable form product parameter file. The product parameter file must have a file type of PPF. This PPF contains the name of the control file to use to update the source file.

### *compname*

is the name of the component (such as CP or CMS) as it is specified on the :COMPNAME tag in the product parameter file. *compname* is a 1- to 16-character alphanumeric identifier.

## Options

### **CNTRL**

specifies a control file is used to identify the AUX file structure.

### *cntrlfn*

is the file name of the control file that is used to identify the AUX file structure. The file type of the control file is CNTRL. This value overrides the value on the :CNTRL tag in the PPF.

### **CKGen**

requests validation of the AUX files against the version vector tables and issues an error message if a mismatch is detected. The version vector tables are not updated.

### **LOGMOD**

requests validation of the AUX files against the version vector tables and automatically updates the local version vector tables when a mismatch is detected. When you specify the LOGMOD option, VMFNLS modifies only the VVT files that are defined in the control file above the :UPDTID level defined in the product parameter file. All other VVT levels are only compared to the AUX files, and mismatches are displayed. You should only use the LOGMOD option when you are assembling files that have source updates. All LOCAL disks must be accessed as Read/Write.

When you use the LOGMOD option:

- If a version vector table does not exist on a LOCAL disk, it is created on the first disk in the LOCAL string.
- If the AUX file for a part is not found, the :PART entry (if found) is deleted from the version vector table.

- If the AUX file for a part is empty, the :MOD data is deleted from the version vector table for that part. The :PART entry is not deleted from the version vector table.

**NOCKGen**

requests no validation of the AUX files against the version vector tables. The AUX file structure is used to update the source file, and the VVT structure is used to name the output file.

**NOVVT**

requests no validation of the AUX files against the version vector tables (VVT). The AUX file structure is used to update the source file and name the output file.

**Note:** If you omit the CKGEN, LOGMOD, NOCKGEN, and NOVVT options, the VMFNLS EXEC uses the value of the :CKGEN tag in the product parameter file to determine whether to validate the AUX files against the VVT. If the :CKGEN tag does not appear in the PPF, no validation is performed; and NOCKGEN is assumed.

**CTL**

indicates the third operand in the command is the name of a control file. If CTL is specified, a product parameter file is not used.

**PPF**

indicates the second parameter in the command is the name of a product parameter file that specifies the control file to be used to update the source file. The product parameter file also lists the minidisk and directory search order.

**Note:** If you do not enter a *compname* and you do not specify CTL or PPF, CTL is assumed.

**FILEType**

indicates the file type for the output file that is created. This option overrides any naming from the AUX or VVT structures.

*ft*

is the file type for the output file.

**NO\$SElect**

does not update the *appid* \$SELECT file. NO\$SELECT is the default.

**\$SElect**

updates the *appid* \$SELECT file to indicate the text deck has been changed. The first APPLY disk specified in the :MDA section of the product parameter file must be accessed Read/Write. The other APPLY disks must be accessed.

**NOKeepsrc**

erases the updated source file after it is converted to text. NOKEEPSRC is the default.

**KEEPsrc**

indicates the updated source file, which consists of the source file and any updates, will be saved on your A-disk. The file is named *\$fn ft*. *fn* is the source file name, which is truncated to 7 characters when necessary; and *ft* is the source file type.

**OBJect**

creates the output deck(s) on your A-disk. OBJECT is the default.

**NOOBJect**

does not create the output deck(s) on your A-disk.

**OUTMode**

indicates the file mode for the output text and listing files created. This file mode must be accessed Read/Write.

**A**

creates the output files on file mode A. A is the default file mode.

*fm*

is the file mode for the output files.



***mda\_string***

is the name of the symbolic string of disks from the :MDA section of the product parameter file. The output is placed on the first disk specified in this string.

**PRint**

sends the listing output to the virtual printer. PRINT is the default.

**Disk**

creates the listing output on your A-disk.

**SETup**

sets up a minidisk or SFS directory access order for the assemble function according to entries in the :MDA section of the product parameter file. This option is valid only when using a product parameter file. If a user exit is specified in the product parameter file, setup will occur after the user exit is called.

**NOSetup**

does not set up a new access order.

**PREExit**

sets up a minidisk or SFS directory access order for the assemble function according to entries in the :MDA section of the product parameter file. If a user exit is specified in the product parameter file, setup will occur before the user exit is called.

**Note:** If the SETUP, PREEXIT, and NOSETUP options are omitted, the VMFNLS EXEC uses the value of the :SETUP tag in the product parameter file to determine whether to set up a new access order.

**ASSEMBLE Options Supported by VMFNLS**

In the following table, the left column shows the options of the ASSEMBLE command. The right column shows how these options are supported by VMFNLS when invoking the ASSEMBLE command. Also shown are the default values (underlined> of these options. The ASSEMBLE defaults are used wherever possible. Keyword-function options must be entered without the parentheses.

<b>ASSEMBLE Option</b>	<b>VMFNLS Option</b>
ALIGN NOALIGN ALGN NOALGN	same
ALOGIC NOALOGIC	same
BUFSIZE (STD) BUFSIZE(MIN) BUFSIZE(MAX)	BUFSIZE <u>STD</u> BUFSIZE <u>MIN</u> BUFSIZE <u>MAX</u>
DECK NODECK	same
ESD NOESD	same
FLAG (0) FLAG(n)	FLAG <u>0</u> FLAG <u>n</u>
LIBMAC NOLIBMAC	same
LINECOUN (55) LINECOUN(nn)	LINECOUN <u>55</u> LINECOUN <u>nn</u>
LIST NOLIST	same
MCALL NOMCALL	same
MLOGIC NOMLOGIC	same
NUMber NONUM	same
OBJect NOOBJect	same
PRint NOPRint DIsk	<u>PRint</u>   <u>DIsk</u>
RENT NORENT	same
RLD NORLD	same
STMT NOSTMT	same

ASSEMBLE Option	VMFNLS Option
SYSPARM(string) SYSPARM(?) SYSPARM()	SYSPARM string SYSPARM ? SYSPARM SUP SUP SYSPARM EXP EXP
TERMinal NOTERM	same
TEST NOTEST	same
WORKSIZE (2048K) WORKSIZE(nnnnK)	WORKSIZE 2048K WORKSIZE nnnnK
XREF (SHORT) XREF(FULL) NOXREF	XREF SHORT XREF FULL NOXREF
YFLAG NOYFLAG	same
<p><b>Note:</b> The defaults appear highlighted.</p> <p>The SYSPARM SUP option suppresses the expansion of macros. The SYSPARM EXP option activates the expansion of macros. SYSPARM SUP is the default.</p>	

**GENCMD Options Supported by VMFNLS**

In the following table, the left column shows the options of the GENCMD command. The right column shows how these options are supported by VMFNLS when invoking the GENCMD command. The default values of these options are also shown. The GENCMD defaults are used wherever possible.

GENCMD Option	VMFNLS Option
OUTmode * OUTmode fm CCheck	OBJect NOOBJect
STACK FIFO LIFO	not supported
SYSTEM USER ALL	same
<p><b>Note:</b> The defaults appear highlighted.</p>	

**GENMSG Options Supported by VMFNLS**

In the following table, the left column shows the options of the GENMSG command. The right column shows how these options are supported by VMFNLS when invoking the GENMSG command. The default values of these options are also shown. The GENMSG defaults are used wherever possible.

GENMSG Option	VMFNLS Option
If the application ID is DMK or HCP, CP is used.	
Dbscs NODbscs	If the language is KANJI, DBCS is used.
List NOList	same
Margin 72 Margin nn	Margin 63 Margin nn
Object NOObject	OBJect NOOBJect
Xref NOXref	same
<p><b>Note:</b> The defaults appear highlighted.</p>	

**Usage Notes**

1. VMFNLS handles packed files.
2. When generating text decks for use in the VMSES/E environment, you must use a product parameter file.

3. If you receive warnings or errors from the UPDATE command, check the *fn* UPDLOG file for additional information.
4. VMFBLD uses the version vector tables to determine which level of a part to use during build processing. If you do not specify the LOGMOD option, you must update the version vector tables manually before you run VMFBLD or you must rerun VMFNLS and specify the LOGMOD option.
5. When you specify the \$SELECT option, the select data file (*appid* \$SELECT) is updated with a record consisting of one of the following:
  - *fn* and the first 3 characters of the file type of the output file
  - *fn* and the full file type (when you also specify the FILETYPE option)
 The select data file is used by VMFBLD to determine which objects need to be built using this text deck.
6. When you create local modifications, you can use the \$SELECT, LOGMOD, and OUTMODE options to eliminate some manual steps, such as updating the *appid* \$SELECT file, updating local version vector tables files, and saving the results on a LOCALMOD disk.
7. VMFNLS creates three output files from a CMS Pipelines system message repository file (FPLMESx REPOS). The system message file (FPLMESx TXTnnnnn) is created by the GENMSG command. The user message file (FPLUME TXTnnnnn) is a copy of the system message file. The merged message file (FPLNLS xxxnnnnn) is created by the LANGMERC command.

## Examples

- To run VMFNLS using the IBM-supplied defaults and a product parameter file to assemble a part, enter:

```
VMFNLS DMSABC ASSEMBLE ppfname compname
```

- To run VMFNLS using the IBM-supplied defaults and a control file, enter:

```
VMFNLS DMSABC ASSEMBLE cntrlfn
```

- To run VMFNLS using the IBM-supplied defaults and a product parameter file to generate commands, enter:

```
VMFNLS DMSSPA DLCS ppfname compname
```

- To run VMFNLS using the IBM-supplied defaults and a product parameter file to generate messages, enter:

```
VMFNLS DMSMES REPOS ppfname compname
```

## Input and Output Files

### Input Files

#### ***ppfname* PPF**

The usable form product parameter file.

#### ***cntrlfn* CNTRL**

The control file.

#### ***fn* ASSEMBLE*fn* DLCS*fn* REPOS**

The source file.

#### ***fn updtft***

Updates to the source file.

#### **VMFNLS LANGLIST**

National language support country code table.

#### ***fn AUXI*lid**

The auxiliary control file.

**appid VVTlvlid**

The version vector table.

**Output Files****fn TEXTfn TXTnnnnfn xxxnnnn**

The assembled object deck (*xxx* is the file type abbreviation; *nnnn* is a PTF number). You receive only one of these formats.

**Note:** The object deck is written to the A-disk only when the OBJECT option (the default) is specified.

**appid VVTlvlid**

A version vector table.

**appid \$SELECT**

The list of build requirements, when you specify the \$SELECT option.

**\$fn LISTING**

The assembler listing file.

**fn UPDLOG**

The update log file.

**fn ctlfile**

The update information file.

**Note:** If listing output is generated during the assembly, the PRINT or DISK option determines where it will reside. The PRINT option (the default) causes all listing output to be sent to the virtual printer as *fn ctlfile*. The DISK option causes all listing output to be placed on the A-disk in two files (*\$fn LISTING* and *fn UPDLOG*).

**Temporary Files****\$fn ASSEMBLE**

The updated source assemble file.

**\$fn TEXT**

The temporary generated object deck or decks.

**fn \$langid**

The updated source repos file.

**fn \$DLCS**

The updates source DLCS file.

**fn LISTING**

The temporary listing file.

**fn UPDATES**

The update history file.

**\$VMFSIM CNTRL**

A control file used with the LOGMOD option.

**fn AUX\$\$\$\$**

An AUX file used with the LOGMOD option.

**PPF Tags Used****:APPID**

The identifier of the product, which is used to name the version vector tables and the \$SELECT file.

**:CKGEN**

Controls the validation of AUX files against the version vector tables. Valid values are NO, YES, LOGMOD, and NOVVT.

**:CNTRL**

Defines the name of the control file.

**:MDA**

Defines symbolic strings and the minidisks or SFS directories associated with them.

**:SETUP**

Controls whether the VMFSETUP EXEC is called to access minidisks and SFS directories.

**:USEREXIT**

Defines the file name of the user exit. If no value is specified, no exit is called.

**Messages and Return Codes**

Appendix D, “Module Identifiers for VMSES/E Messages,” on page 749 shows the format of VMSES/E messages and lists the identifiers used by each VMSES/E EXEC.

For a complete explanation of any VMSES/E message, use the HELP Facility to view the message explanation online or see the appropriate messages documentation. To display information about a specific message - VMF002E, for example - enter:

```
help msg vmf002e
```

If you are not familiar with the z/VM HELP Facility, enter help to display the main HELP Menu. For information about the HELP command, enter:

```
help cms help
```

Appendix D, “Module Identifiers for VMSES/E Messages,” on page 749 shows the format of VMSES/E messages and lists the identifiers for each VMSES/E EXEC.

**PI**

Return codes issued by the VMFNLS EXEC might be returned to a user exit. For more information about user exits, see [:USEREXIT](#).

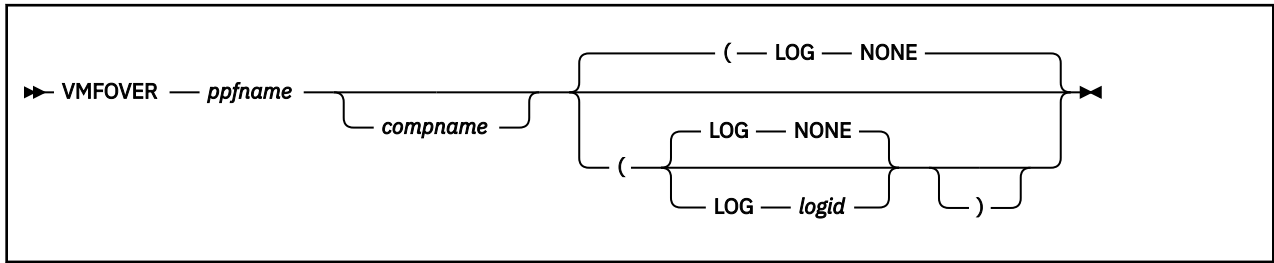
The VMFNLS EXEC issues the following return codes:

Return Code	Explanation
0	Command completed successfully.
4	Command completed with one or more warning conditions.
8	Command completed but at least one major process failed.
24	Command failed because of a command line syntax error.
28	Command failed because a required file was not found.
36	Command failed because a target disk or directory was not available.
100	Command failed because of an external error.

**PI end****Recovery Information**

The VMFNLS command can be restarted by reissuing the command.

## VMFOVER EXEC



### Purpose

The VMFOVER EXEC creates a temporary product parameter file by applying overrides to a source product parameter file. The file type of the temporary file is \$PPFTEMP. The file is resolved to one component (the component specified on invocation of VMFOVER) with all overrides applied.

Component parameter overrides (alternative or additional component parameters) can be defined in a component parameter override area either in the source product parameter file or in a separate product parameter override file.

Overrides can be chained so that an override can point to either a component area or another override area.

### Operands

#### *ppfname*

is the name of a source or override product parameter file. The file type must be \$PPF.

#### *compname*

is the name of the component (such as CP or CMS) or the name of a component area or an override area. *compname* is a 1-16 character alphanumeric identifier. If you do not specify a valid component or override name, VMFOVER shows you a list of names and asks you to choose one. You can specify only one name.

### Options

#### LOG

identifies the type of message logging to be done. Messages are logged in the specified message log as well as written to the terminal.

No messages are logged until initial validation of the command is complete.

**Note:** The LOG option is reserved for use by VMSES/E.

#### NONE

is the default. If NONE is specified, messages are written only to the terminal.

#### *logid*

can be one of the following:

#### *logid*

##### Type of Log

#### APP

The apply message log (\$VMFAPP \$MSGLOG A)

#### BLD

The build message log (\$VMFBLD \$MSGLOG A)

**INS**

The install message log (\$VMFINS \$MSGLOG A)

**MRD**

The merge message log (\$VMFMRD \$MSGLOG A)

**REC**

The receive message log (\$VMFREC \$MSGLOG A)

**Examples**

To run VMFOVER with a component name, enter:

```
VMFOVER ppfname compname
```

If the product parameter file you wish to process has only one component or you wish to be prompted for a component name, enter:

```
VMFOVER ppfname
```

For additional examples, see [“Examples of Overrides”](#) on page 655.

**Input and Output Files****Input Files*****ppfname* \$PPF**

The input file is either a source product parameter file or an override file.

**Temporary Files****\$\$\$PPFT\$\$ \$PPF**

\$\$\$PPFT\$\$ \$PPF is a working file used to process override files. It is deleted if VMFOVER completes successfully.

**Output Files*****ppfname* \$PPFTEMP**

*ppfname* \$PPFTEMP is the final outcome of running VMFOVER. All overrides have been processed for the given component name. This file is a source of information for VMFPPF EXEC.

**\$VMFlogid \$MSGLOG**

The message log.

**PPF Tags Used****:COMPLST**

The :COMPLST tag is used to identify the valid components for a product parameter file.

**:OVERLST**

The :OVERLST tag is used to identify the valid overrides. The overrides may be a separate override file, or part of a source product parameter file.

**All** of the PPF tags are relevant to VMFOVER. The tags listed are of particular importance when trying to accomplish specific tasks.

**Override Control Records****./DELETE**

The ./DELETE tag is used to identify a record to be deleted.

**./INSERT**

The ./INSERT tag is used to identify the start of a block to be inserted.

**Note:** For more information on the ./DELETE and ./INSERT tags, see [“Override Control Records”](#) on page 645.

## Messages and Return Codes

Appendix D, “Module Identifiers for VMSES/E Messages,” on page 749 shows the format of VMSES/E messages and lists the identifiers used by each VMSES/E exec.

For a complete explanation of any VMSES/E message, use the HELP Facility to view the message explanation online or see the appropriate messages documentation. To display information on a specific message, for example VMF002E, enter:

```
help msg vmf002e
```

If you are unfamiliar with the z/VM HELP Facility, enter help to display the main HELP Menu. For information on the HELP command, enter:

```
help cms help
```

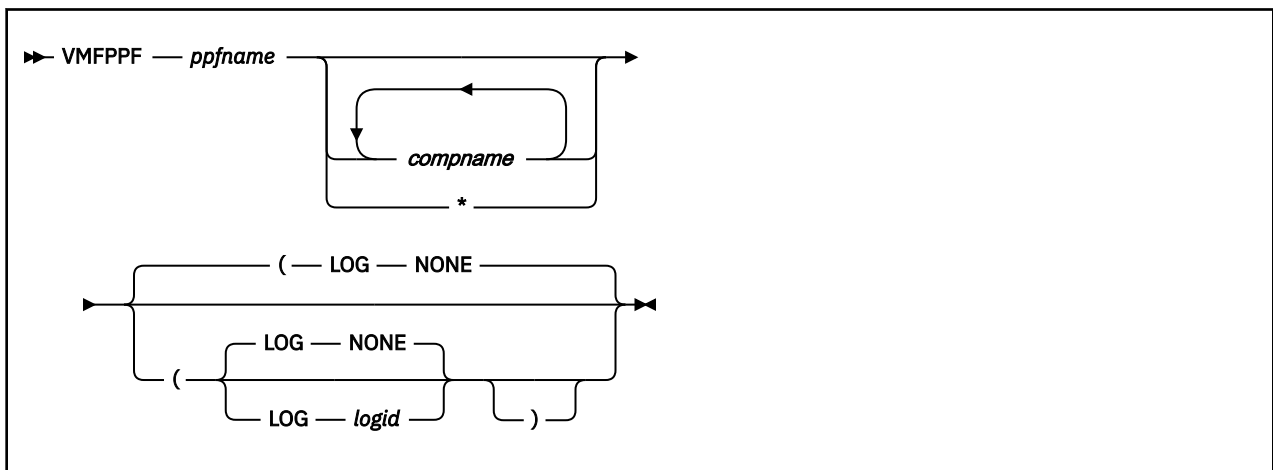
The VMFOVER EXEC issues the following return codes:

Return Code	Explanation
0	VMFOVER completed successfully.
4	A warning message has been issued.
12	A syntax error was found in the product parameter file.
24	An error was found when parsing the input to VMFOVER.
28	A product parameter file was not found.
100	An error occurred as a result of issuing an external command.

## Recovery Information

The VMFOVER command can be restarted by reissuing the command.

## VMFPPF EXEC



## Purpose

The VMFPPF EXEC updates and compiles a source product parameter file (with file type \$PPF) into its usable form (with file type PPF). The VMFPPF EXEC calls the VMFOVER EXEC to generate the temporary product parameter file which is used as input. The VMFPPF EXEC also resolves variables declared



in the :DCL section of the product parameter file, which may be referenced in the :MDA, :RECINS, and :RECSER sections. The VMFPPF EXEC also performs syntax checking as part of the compile process.

## Operands

### *ppfname*

is the name of a source or override product parameter file. The file type must be \$PPF.

### *compname*

is the name of the component (such as CP or CMS) to be added to the compiled product parameter file. *compname* is a 1-16 character alphanumeric identifier. The component must be specified on the :COMPLST or :OVERLST tags in the source product parameter file. You can specify any number of components.

\*

compiles all of the components listed on the :COMPLST and :OVERLST tags in the source product parameter file (\$PPF).

**Note:** If you do not specify an asterisk (\*) or a *compname*, a list of all of the components on the :COMPLST and :OVERLST tags in the source product parameter file (\$PPF) is displayed. You are prompted to select the ones to compile.

## Options

### LOG

identifies the type of message logging to be done. Messages are logged in the specified message log as well as written to the terminal.

No messages are logged until initial validation of the command is complete.

**Note:** The LOG option is reserved for use by VMSES/E.

### NONE

is the default. If NONE is specified, messages are written only to the terminal.

### *logid*

can be one of the following:

#### *logid*

##### Type of Log

#### APP

The apply message log (\$VMFAPP \$MSGLOG A)

#### BLD

The build message log (\$VMFBLD \$MSGLOG A)

#### INS

The install message log (\$VMFINS \$MSGLOG A)

#### MRD

The merge message log (\$VMFMRD \$MSGLOG A)

#### REC

The receive message log (\$VMFREC \$MSGLOG A)

## Examples

- To run VMFPPF to compile the usable form of the product parameter file for the CP component, enter:

```
VMFPPF ppfname CP
```

- To run VMFPPF to compile the usable form of the product parameter file for all components, enter:

```
VMFPPF ppfname *
```

- To run VMFPPF and be prompted for the components to compile, enter:

```
VMFPPF ppfname
```

## Input and Output Files

### Input Files

#### *ppfname* \$PPF

The source and override product parameter files used by the VMFOVER EXEC.

### Temporary Files

#### *ppfname* \$PPFTEMP

The temporary product parameter file produced by the VMFOVER EXEC. This file is used by VMFPPF and is temporarily stored on your A-disk.

### Output Files

#### *ppfname* PPF

The usable form of the product parameter file. This file is stored on your A-disk if the PPF does not already exist. If there is a previous copy of *ppfname* PPF, VMFPPF replaces the existing copy if it is on a read-write disk.

### PPF Tags Used

All PPF tags are used by the VMFPPF EXEC.

## Messages and Return Codes

Appendix D, “Module Identifiers for VMSES/E Messages,” on page 749 shows the format of VMSES/E messages and lists the identifiers used by each VMSES/E exec.

For a complete explanation of any VMSES/E message, use the HELP Facility to view the message explanation online or see the appropriate messages documentation. To display information on a specific message, for example VMF002E, enter:

```
help msg vmf002e
```

If you are unfamiliar with the z/VM HELP Facility, enter help to display the main HELP Menu. For information on the HELP command, enter:

```
help cms help
```

The VMFPPF EXEC issues the following return codes:

Return Code	Explanation
0	Command completed successfully.
12	Command failed because of an internal error.
24	Command failed because of a command line syntax error.
28	Command failed because a required file was not found.
36	Command failed because a target disk or directory was not available.
100	Command failed because of an external error.

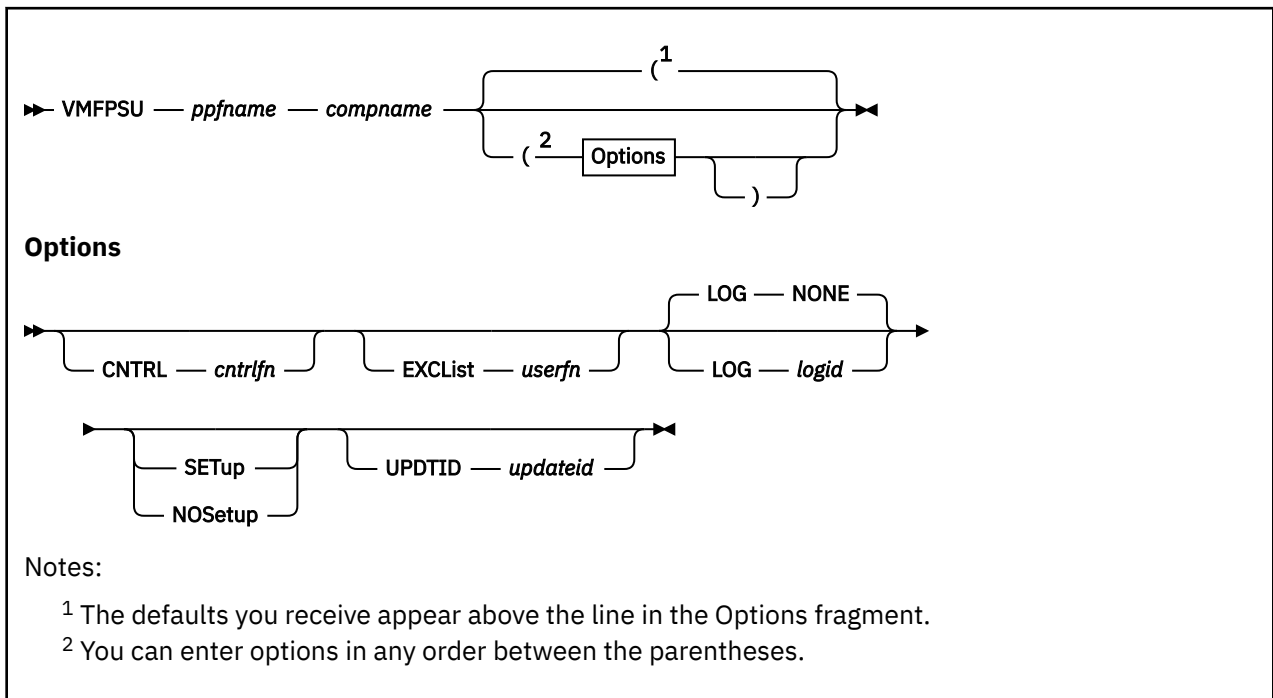
## Recovery Information

The compiled version of the product parameter file is not updated unless the return code from the VMFPPF EXEC is zero, with one exception. The following message indicates the VMFPPF EXEC failed while attempting to update the compiled version of the product parameter file.

```
VMFPPF1965E The command, COPYFILE VMFUT1 $TEMP A ppfname PPF  
fm failed with return code rc
```

In this case, the compiled version of the product parameter file may be in an inconsistent state. Correct the problem as indicated by the return code *rc* in the message and reissue the VMFPPF command.

## VMFPSU EXEC



### Purpose

VMFPSU produces a file (*appid* PSUPLAN) that contains:

- A list of all the PTFs that are contained on the Product Service Upgrade but not applied to the product.
- A list of all PTFs that are applied to the product and not on the PSU.
- A list of excluded PTFs.
- A list of the parts with local modifications that need to be reprocessed after you receive the PSU. Their local modification IDs (*modids*) are also provided.

VMFPSU also creates a select data file (*xxx*\$PSU\$ \$SELECT) with parts from local modifications that do not need to be reprocessed but do need to be rebuilt. The *xxx* in the *xxx*\$PSU\$ \$SELECT file will be the assigned component *partname* prefix.

### Operands

#### *ppfname*

is the file name of a usable form product parameter file. The product parameter file must have a file type of PPF.

#### *compname*

is the name of the component (such as CP or CMS) as it is specified on the :COMPNAME tag in the product parameter file. *compname* is a 1- to 16-character alphanumeric identifier.

### Options

#### CNTRL

indicates a specific control file is to be used to identify the version vector table structure.

#### *cntrlfn*

is the file name of the control file. This value overrides the value on the :CNTRL tag in the PPF. The file type of the control file is CNTRL.

**EXCList**

indicates a specific user-supplied exclude list is to be used.

***userfn***

is the file name of the exclude list. This value overrides the value on the :EXCLIST tag in the PPF. The file type of the exclude list is \$EXCLIST.

**LOG**

identifies the type of message logging to be done. Messages are written to the terminal or logged in the specified message log as well as written to the terminal, depending on the LOG option used.

No messages are logged until initial validation of the command is complete.

**Note:** The LOG option is reserved for use by VMSES/E.

**NONE**

is the default. If NONE is specified, messages are written only to the terminal.

***logid***

can be one of the following:

***logid*****Type of Log****APP**

The apply message log (\$VMFAPP \$MSGLOG A)

**BLD**

The build message log (\$VMFBLD \$MSGLOG A)

**INS**

The install message log (\$VMFINS \$MSGLOG A)

**MRD**

The merge message log (\$VMFMRD \$MSGLOG A)

**REC**

The receive message log (\$VMFREC \$MSGLOG A)

**SETup**

sets up a minidisk or SFS directory access order according to the entries in the :MDA section of the product parameter file.

**NOSetup**

does not set up a new access order.

**Note:** If the SETUP and NOSETUP options are omitted, the VMFPSU EXEC uses the value of the :SETUP tag in the product parameter file to determine whether to set up a new access order.

**UPDTID**

indicates a specific update level identifier is to be used to determine the file type of the maintenance level version vector table. The resulting file is compared with the VVT\$PSU\$ file shipped on the PSU media.

***updateid***

is the update level identifier used to determine the file type of the maintenance level version vector table. The identifier must begin with the string AUX (AUXVM, for example). This value overrides the value on the :UPDTID tag in the PPF.

**Usage Notes**

1. You must issue the VMFINS INSTALL INFO command before you use VMFPSU to ensure the appropriate version vector tables from the Product Service Upgrade media are available.
2. The APPLY and LOCAL strings must be accessed.
3. VMFPSU uses only the primary :APPID to identify the version vector tables.
4. The *appid* PSUPLAN output file is replaced when you run VMFPSU for the same *appid*.

5. All version vector tables (VVTs) above the value specified on the :UPDTID tag in the product parameter file, or on the command line, are checked for parts containing local modifications. Be careful when you change the UPDTID, because all local modifications may not be picked up.
6. If you see an entry containing question marks (???) under a heading in the *appid* PSUPLAN file, the information for that section is not available due to an error. You can check the console log to find out where the error occurred. You can also check the message log if you used the LOG *logid* option.

### Examples

To use the VMFPSU command for the MYCOMP component, enter these commands:

```
VMFINS INSTALL INFO
VMFPSU ESA MYCOMP ( SETUP
```

The following messages are produced:

```
VMFPSU2760I VMFPSU processing started
VMFPSU1071I There are 12 PTFs on the Recommended Service Upgrade for
PRODID 1VMVMC23%MYCOMP that are not currently
applied.
VMFPSU1072I There are 5 PTFs currently applied to PRODID
1VMVMC23%MYCOMP that need to be reapplied.
VMFPSU1076I There are 0 PTFs to be excluded from the Recommended Service
Upgrade.
VMFPSU1073I There are 3 parts with local modifications that need to
be reprocessed.
VMFPSU1078I Select data file HCP$PSU$ $SELECT was created or updated
to force the rebuild of local modifications.
VMFPSU1070I Creating 1VMVMC23 PSUPLAN file at service level 201-9401 for
component MYCOMP in PPF ESA.
VMFPSU2760I VMFPSU processing completed successfully
```

[Figure 159 on page 465](#) shows the *appid* PSUPLAN file created for MYCOMP during PSU processing.

```

*****
****          PPFNAME: ESA              COMPNAME: MYCOMP          **** 1
*****
****  PRODID: 1VMVMC23%MYCOMP          Service Level: 201-9401    ****
*****
****          Date: 07/01/22   Time: 13:31:36          ****
*****
VMFPSU1071I There are 12 PTFs on the Recommended Service Upgrade for 2
             PRODID 1VMVMC23%MYCOMP that are not currently
             applied.
VMFPSU1072I There are 5 PTFs currently applied to PRODID
             1VMVMC23%MYCOMP that need to be reapplied.
VMFPSU1076I There are 0 PTFs to be excluded from the Recommended Service
             Upgrade.
VMFPSU1073I There are 3 parts with local modifications that need to
             be reprocessed.
VMFPSU1078I Select data file HCP$PSU$ $SELECT was created or updated
             to force the rebuild of local modifications.
*****
****  PTFs TO BE APPLIED FOR PRODID 1VMVMC23%MYCOMP          ****
*****
             PTF.APAR          PTF.APAR          PTF.APAR          PTF.APAR          3
UM24491.VM55782  UM24493.VM55890  UM24508.VM56081  UM24508.VM56085
UM24512.VM56113  UM24555.VM24554  UM24636.VM55900  UM24646.VM55792
UM24663.VM56161  UM24698.VM56179  UM24918.VM54956  UM24984.VM55935
UM25038.VM56572
*****
****  PTFs TO BE REAPPLIED TO PRODID 1VMVMC23%MYCOMP          ****
*****
             PTF.APAR          PTF.APAR          PTF.APAR          PTF.APAR          4
UM25043.VM56291  UM45454.VM44445  UM89995.VM89991  UM89996.VM89992
UM89999.VM89998
*****
****  PTFs EXCLUDED FOR PRODID 1VMVMC23%MYCOMP          ****
*****
NONE 5
*****
****  LOCALMODS TO REPROCESS: 1VMVMC23 VVTLCL ON DISK 2C4(E)  6
*****
PART - HCPITT TXT 7
      PTF - UM25038.VM56572 8
      MOD - LCL1212.DL12 9
*****
****  LOCALMODS TO REPROCESS: 1VMVMC23 VVTLCL2 ON DISK 2C4(E) 10
*****
PART - HCPBLSAM EXC
      PTF - UM24508.VM56089  UM24508.VM56085  UM24555.VM24554
             UM24984.VM55935
      MOD - LCL0098          LCL0068
PART - HCPDTD TXT
      PTF - UM24663.VM56161
      MOD - LCL0005.DL0003DA  LCL0004.DL0004DA  LCL0003.DL3
             LCL0006.DL0006DA  LCL0007.LC0007DA
PART - HCPITT TXT 11
      PTF - UM25038.VM56572
      MOD - LCL0088.DL0088DA  LCL0044.DL0044DA  LCL0095.DL0095DA
             LCL0076.DL0076DA  LCL0017.LC0017DA

```

Figure 159. PSUPLAN File

In Figure 159 on page 465, the information that is most important to you is numbered for reference purposes.

Section 1 provides the name of the product parameter file (PPF), component (MYCOMP), and product (1VMVMC23), as well as the service level for which this PSUPLAN file was created. The date and time at which the file was created is also shown.

Section 2 shows you the messages issued by VMFPSU. You receive details on the specific PTFs and local modifications in the remainder of the PSUPLAN file. If there are any errors during VMFPSU processing, the last error message is shown here. To see additional error messages, refer to the console log.

Section 3 lists the PTFs from the Recommended Service Upgrade to be applied for the specified product. This section can contain:

- The PTFs (with associated APAR) that are to be applied from the RSU.

**Note:**

1. The number of entries shown in this section may not always match the number shown in message VMFPSU1071I. This section shows the PTF more than once if it affects more than one APAR.
  2. The PTFs are preapplied on the RSU so they will be displayed as "already applied" by VMFAPPLY.
- The word NONE if there are no PTFs to be applied from the RSU.

Section **4** shows the PTFs that need to be reapplied to your z/VM system. This section can contain:

- The PTFs (with associated APAR) you need to reapply.

**Note:** The number of entries shown in this section may not always match the number shown in message VMFPSU1072I. The PTF appears more than once if it affects more than one APAR.

- The word NONE if there are no PTFs to reapply.
- Question marks (???) if an error occurred during VMFPSU processing and the data could not be collected. If this is the case, check the console log or the message log for error messages.

Section **5** shows the PTFs excluded by you or IBM. This section can contain:

- The excluded PTFs.

**Note:** The PTFs shown are from the user and IBM exclude lists only.

- The word NONE if there were no excluded PTFs.
- Question marks (???) if an error occurred during VMFPSU processing and the data could not be collected. If this is the case, see the console log or the message log for error messages.

Section **6** tells you there are local modifications to reprocess in that particular version vector table on that particular disk.

**Note:**

1. If no local modifications are affected, the version vector table and disk address do not appear.
2. The total number of entries in the LOCALMODS sections (**6** and **10**) might not always match the number shown in message VMFPSU1073I. A part is shown more than once if it appears in more than one version vector table for a local modification.
3. The local modifications shown in the LOCALMODS sections (**6** and **10**) need to be reprocessed or reworked. There may be other local modifications that need to be rebuilt. All local modifications will be placed in the xxx\$PSU\$ \$SELECT file, if xxx\$PSU\$ was specified as a secondary apply ID on the :APPID tag in the PPF; or all local modifications will be placed in the default \$PSU\$ \$SELECT file, if no xxx\$PSU\$ file was specified on the :APPID tag in the PPF.

If a \$PSU\$ \$SELECT file was created, you must append \$PSU\$ \$SELECT to the top of the *appid* \$SELECT file to rebuild the local modifications before you issue the VMFBLD command

**7** tells you the name of the part that has the local modification. You might also see:

- The word NONE if there are no local modifications on your z/VM system.
- Question marks (???) if an error occurred during VMFPSU processing and the data could not be collected. If this is the case, see the console log or the message log for error messages.

**8** tells which of the PTFs to be applied affect this part.

**Note:** An at sign (@) preceding the **PTF -** indicates that at least one PTF is in the list of excluded PTFs. You do not need to reprocess this local modification when the PTF is excluded.

**9** tells you the local modification identifiers for this part.

Section **10** shows there was more than one version vector table affected by the PTFs to be applied.

**11** shows a part that appears in more than one local modification version vector table.

## Input and Output Files

### Input Files



**appid VVTlvld**

Version vector table specified by the control file that is pointed to by the :CNTRL tag in the product parameter file (PPF) or the CNTRL option.

**appid VVT\$PSU\$**

Version vector table from the Product Service Upgrade media.

**cntrlfn CNTRL**

The control file specified in the :CNTRL tag of the product parameter file (PPF) or the CNTRL option on the VMFPSU command.

**fn \$EXCLIST**

The user-supplied exclude list identified by the :EXCLIST tag in the PPF or the EXCLIST option on the VMFPSU command, the product-supplied exclude list identified by the :AXLIST tag in the product parameter file, or both.

**ppfname PPF**

The usable form product parameter file.

**prodid PRODPART**

Product parts file from the Product Service Upgrade (used to get the service upgrade level).

**Output Files****appid PSUPLAN**

Contains messages issued by VMFPSU, which includes a list of all PTFs new to the system, a list of all PTFs that would have to be reapplied, a list of PTFs excluded, and the parts with local modifications that are affected by the service. *appid* PSUPLAN is stored on the A-disk.

**\$VMFlogid \$MSGLOG**

The message log where the messages will be written to depending on the LOG option.

**xxx\$PSU\$ \$SELECT**

Updated select data file. \$PSU\$ \$SELECT is the default if xxx\$PSU\$ is not specified on the :APPID tag of the PPF.

**Temporary Files****appid VVTlvld**

A temporary version vector table used during VMFPSU processing, if one does not already exist on the APPLY disks for the same *appid*. This file is stored on your A-disk until processing is complete.

**PPF Tags Used****:APPID**

The identifier of the product used to find the version vector tables. Also identifies the xxx\$PSU\$ \$SELECT file to use.

**:AXLIST**

Identifies the file name of the product-supplied apply and exclude lists shipped in the service packages.

**:BLD**

Identifies build lists for the product.

**:CNTRL**

Identifies the file name of the control file to be used.

**:COMPNAME**

Defines the component in the product parameter file to be used.

**:DABBV**

Identifies file type abbreviations assigned by this product.

**:EXCLIST**

Identifies the file name of the user-supplied exclude list.

**:MDA**

Identifies the minidisk/directory assignments for the product to be used.

**:PRODID**

Defines the product in the product parameter file to be used.

**:RECID**

Identifies the product (used as file name for PRODPART file).

**:SETUP**

Controls whether the VMFSETUP EXEC is called to access minidisks/directories.

**:UPDTID**

Identifies the update level identifier used in the file types of AUX files and VVTs.

**Messages and Return Codes**

Appendix D, “Module Identifiers for VMSES/E Messages,” on page 749 shows the format of VMSES/E messages and lists the identifiers used by each VMSES/E EXEC.

For a complete explanation of any VMSES/E message, use the HELP Facility to view the message explanation online or see the appropriate messages documentation. To display information about a specific message - VMF002E, for example - enter:

```
help msg vmf002e
```

If you are not familiar with the z/VM HELP Facility, enter help to display the main HELP Menu. For information about the HELP command, enter:

```
help cms help
```

Appendix D, “Module Identifiers for VMSES/E Messages,” on page 749 shows the format of VMSES/E messages and lists the identifiers for each VMSES/E EXEC.

The VMFPSU EXEC issues the following return codes:

Return Code	Explanation
0	Command completed successfully. Check the <i>appid</i> PSUPLAN file and choose a service method.
1	Command completed successfully. No PTFs to be applied from PSU for product.
2	Command completed successfully. Recommends not installing RSU due to excluded PTFs.
4	Command completed with one or more warning conditions.
12	Command failed because of an internal error.
24	Command failed because of a command line syntax error.
28	Command failed because a required file was not found.
36	Command failed because a target disk or directory was not available.
100	Command failed because of an external error.

**Recovery Information**

The VMFPSU command can be restarted by reissuing the command.

## VMFQMDA EXEC

▶▶ VMFQMDA — *ppfname* — *compname* ▶▶

### Purpose

The VMFQMDA EXEC displays the current VMSES/E access order.

### Operands

#### *ppfname*

is the file name of the usable form product parameter file. The product parameter file must have a file type of PPF.

#### *compname*

is the name of the component as it is specified on the :COMPNAME tag in the product parameter file. *compname* is a 1-16 character alphanumeric identifier.

### Usage Notes

1. VMFQMDA uses the :MDA section of the product parameter file to obtain disk string definitions and issues the CMS QUERY ACCESSED command to obtain the current access order. It compares these two input sources to determine the VMSES/E access order and then displays it. [Figure 160 on page 469](#) shows the output from the VMFQMDA EXEC.

```

VMFUTL2205I Minidisk|Directory Assignments:
                String      Mode  Stat  Vdev  Label  (OwnerID 0dev : Cyl/%Used)
                -or-
                SFS Directory Name
VMFUTL2205I LOCALMOD  E    R/W   3C4   MNT3C4 (MAINT730 03C4 : 9/02)
VMFUTL2205I LOCALSAM  F    R/W   3C2   MNT3C2 (MAINT730 03C2 : 5/04)
VMFUTL2205I APPLY     G    R/W   3A6   MNT3A6 (MAINT730 03A6 : 6/01)
VMFUTL2205I          H    R/W   3A4   MNT3A4 (MAINT730 03A4 : 6/01)
VMFUTL2205I          I    R/W   3A2   MNT3A2 (MAINT730 03A2 : 6/01)
VMFUTL2205I DELTA     J    R/W   3D2   MNT3D2 (MAINT730 03D2 : 208/01)
VMFUTL2205I BUILD7    ---  ---   493   -----
VMFUTL2205I BUILD6    ---  ---   490   -----
VMFUTL2205I BUILD10   ---  ---   550   -----
VMFUTL2205I BUILD0    ---  ---   49E   -----
VMFUTL2205I BUILD8    ---  ---   400   -----
VMFUTL2205I BUILDA    ---  ---   890   -----
VMFUTL2205I BUILD4    ---  ---   49D   -----
VMFUTL2205I BUILDN    ---  ---   49D   -----
VMFUTL2205I BASE2     ---  ---   3B2   -----
VMFUTL2205I -----  A    R/W   191   MNT191 (MAINT730 0191 : 175/19)
VMFUTL2205I -----  B    R/W   5E6   MNT5E6 (MAINT730 05E6 : 9/81)
VMFUTL2205I -----  C    R/W   500   MNT500 (MAINT730 0500 : 900/60)
VMFUTL2205I -----  D    R/W   51D   MNT51D (MAINT730 051D : 26/45)
VMFUTL2205I -----  S    R/O   190   MNT190 (MAINT 0190 : 207/41)
VMFUTL2205I -----  Y/S  R/O   19E   MNT19E (MAINT 019E : 500/33)
VMFUTL2205I -----  Z    R/W   A191  MAC191 (MAINT730 A191 : 10/66)

```

Figure 160. VMFQMDA Sample Output

In [Figure 160 on page 469](#), the following information is provided:

#### String

is the name of a symbolic disk string, as defined in a product parameter file (PPF). If this field is left blank, the minidisk or Shared File System (SFS) directory cited is part of the same string as the

previously-listed minidisk or directory. If this field is filled with dashes, the minidisk or directory is not part of any defined disk string.

**Mode**

is the file mode letter at which the minidisk or directory is accessed. If this field is filled with dashes, the minidisk or directory is not accessed.

**Stat**

is the status of the minidisk or directory: R/O (read-only) or R/W (read/write). If this field is filled with dashes, the minidisk or directory is not accessed.

**Vdev**

is the virtual device number of a minidisk, or 'DIR', if the entry is an SFS directory. If the directory has the directory control (DIRCONTROL) attribute, 'DIRC' is displayed instead of 'DIR'.

**Label**

is the label assigned to a formatted CMS disk. If the entry is an OS or DOS disk, this is the volume label.

**(OwnerID Odev : Cyl/%Used)**

is additional disk information that is provided with minidisk entries only.

Disk information is presented using this format:

```
ownerID odev : nnnn/pp
```

where:

**ownerID**

identifies the user ID that owns the listed disk.

**odev**

is the owner's virtual device number for this disk.

**nnnn**

is the number of cylinders available on the disk.

**pp**

is the percentage of disk blocks in use.

**SFS Directory Name**

is the complete name of an SFS directory.

**Note:** Missing minidisks are indicated by hyphens (-) in the Mode and Status columns.

**Examples**

To use VMFQMDA to list the VMSES/E access order for CMS, enter:

```
VMFQMDA ESA CMS
```

**Input and Output Files****Input Files****ppfname PPF**

The usable form product parameter file.

**PPF Tags Used****:MDA**

Defines symbolic strings and the minidisks or SFS directories associated with them.

**Messages and Return Codes**

Appendix D, "Module Identifiers for VMSES/E Messages," on page 749 shows the format of VMSES/E messages and lists the identifiers used by each VMSES/E EXEC.

For a complete explanation of any VMSES/E message, use the HELP Facility to view the message explanation online or see the appropriate messages documentation. To display information on a specific message, for example VMF002E, enter:

```
help msg vmf002e
```

If you are unfamiliar with the z/VM HELP Facility, enter help to display the main HELP Menu. For information on the HELP command, enter:

```
help cms help
```

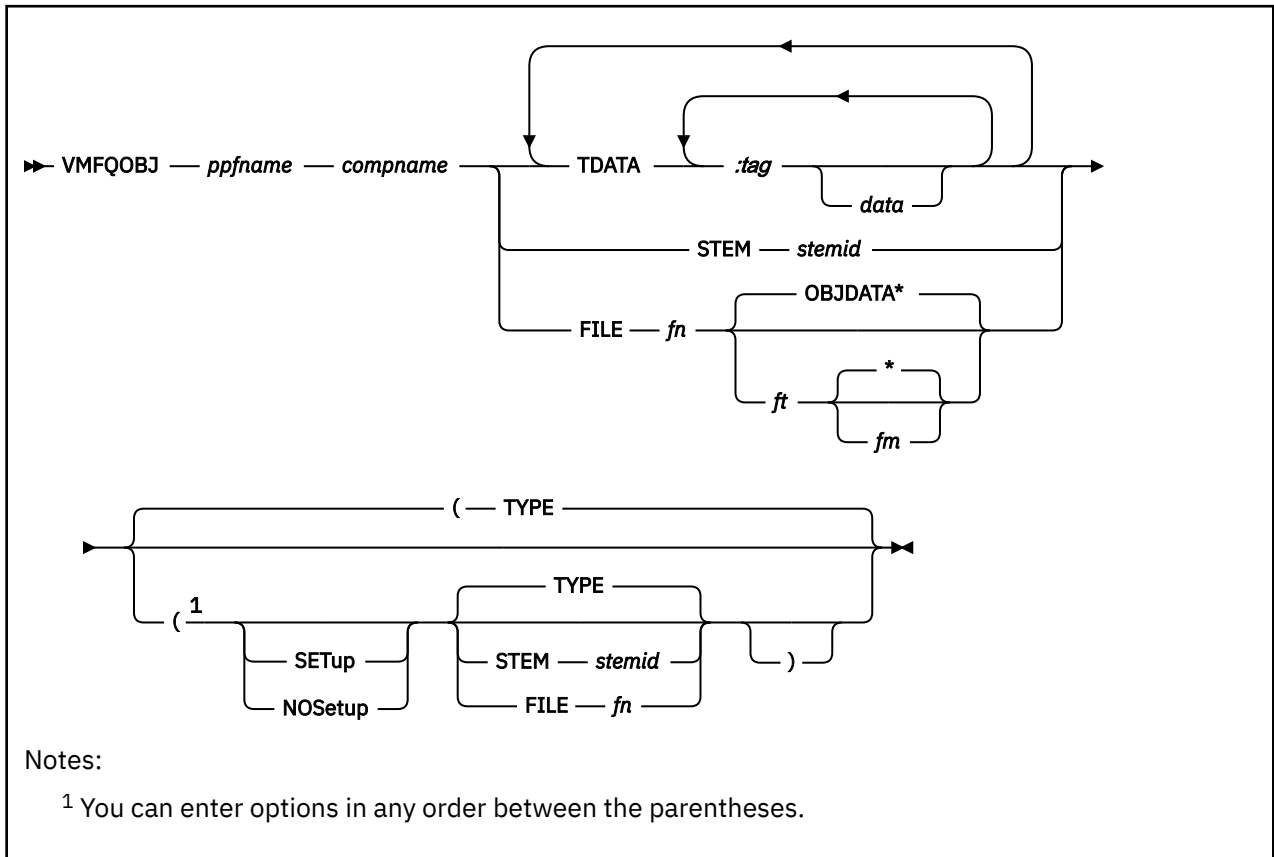
The VMFQMDA EXEC issues the following return codes:

Return Code	Explanation
0	Command completed successfully.
4	Command completed with one or more warning conditions.
24	Command failed because of a command line syntax error.
28	Command failed because a required file was not found.
100	Command failed because of an external error.

## Recovery Information

The VMFQMDA command can be restarted by reissuing the command.

## VMFQOBJ EXEC



### Purpose

VMFQOBJ returns information about objects defined in build lists.

You can use VMFQOBJ to identify any objects that must be rebuilt as a result of a local modification. You can also use VMFQOBJ when you are trying to identify and fix problems. VMFQOBJ allows you to easily identify the parts that are included in an object.

### Operands

#### *ppfname*

is the file name of the usable form product parameter file. The file type must be PPF.

#### *compname*

is the name of the component. *compname* is a 1-16 character alphanumeric identifier.

#### **TDATA**

identifies the search data.

#### **:tag**

is the name of the tag. If a tag is specified and it does not contain any *data*, it is treated as a return field. See [Table 21 on page 474](#) for a list of valid tags.

#### **data**

is the data for which you want to search.

#### **STEM**

identifies a REXX stem that lists TDATA.

***stemid***

is the name of a REXX stem. *stemid* must end with a period.

**FILE**

identifies a file that lists TDATA.

***fn***

is the file name of the file containing the TDATA.

**OBJDATA**

is the default file type for the file containing the TDATA.

***ft***

is the file type of the file containing the TDATA.

**\***

is the default file mode for the file containing the TDATA. If an asterisk (\*) is used as the file mode, VMFQOBJ uses the first file found in the search order that has the correct file name and file type.

***fm***

is the file mode of the file containing the TDATA. If no file mode is specified, VMFQOBJ uses the first file found in the search order that has the correct file name and file type.

**Options****SETup**

sets up a minidisk/directory access order according to entries in the :MDA section of the product parameter file.

**NOSetup**

does not set up a new access order.

**Note:** If you omit the SETUP and NOSETUP options, the VMFQOBJ EXEC uses the value of the :SETUP tag in the product parameter file to determine if a new access order should be set up.

**TYPE**

directs the output to the terminal. TYPE is the default.

**STEM**

directs the output to a REXX stem.

***stemid***

is the name of the REXX stem.

**FILE**

directs the output to a CMS file.

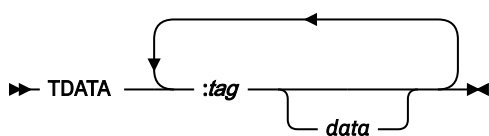
***fn***

is the file name of the file to use for the output. The output file always has a file type of OBJDATA and a file mode of A.

**Usage Notes**

1. VMFQOBJ uses tagged data (TDATA) statements as input. TDATA statements can be entered on the command line, from a file, or from a REXX stem.

Each set of tags and data to be processed begins with the keyword TDATA. The syntax for a TDATA statement is:



A tag must be specified following the TDATA keyword. You can specify data after the tag.

The following example shows a TDATA statement with two tags, one with and one without data:

TDATA :OBJECT BFNA.OBJA :STATUS

2. Input TDATA statements are processed according to these rules:
  - All tags specified in the TDATA statement are treated as return fields. That is, all data contained in the field is returned to the caller.
  - If a tag that corresponds to a parent field is specified as a return field, but none of its subfields are specified, all subfields are returned.
  - When data is entered with a tag in the TDATA statement, VMFQOBJ treats the data as a search argument. VMFQOBJ searches the corresponding fields in each object for the search arguments specified. If a match is found, the requested return fields for that object are returned to the caller.
  - VMFQOBJ treats statements with more than one search argument as if an "AND" condition were specified.
  - If data is specified with duplicate tags, VMFQOBJ treats the multiple search arguments as an "OR" condition; and either argument can result in a match.
  - Tags not defined for objects are ignored.
3. Output from VMFQOBJ can be returned to the terminal display, a file, or a REXX stem.
4. You can use the following tags with VMFQOBJ:

*Table 21. Valid Tags for VMFQOBJ*

Tag	Description
:OBJECT	is the key field; it identifies objects. An object is represented by a combination of the build list name and the object name, which are joined by a period (.). Any combination of the build list name, object file name, and object file type may be used as search criteria when specified without the periods. When you specify them with the periods, the entire object specification must be used. A build list name of UNKNOWN indicates a special build list that lists any parts that get serviced but are not included in any build lists. An object name of BLDLIST indicates the overall build list.
:STAT	provides status information from the service-level build status table. It lists the value on the :STAT tag. Any combination of the values of the object fields may be used as search criteria when you specify them without the periods. When you specify them with the periods, the entire field must be used.
:PARTID	is subordinate to the :STATUS tag. It lists the value of the :PARTID tag in the service-level build status table.
:LIBNAME	lists the library name for objects defined in format 3 build lists.
:BLDREQ	lists direct build requisites of an object. The objects are represented a combination of the build list name and the object name, which are joined by a period (.). Any combination of build list names, object file names, and object file types may be used as search criteria when you specify them without the periods. When specifying them with the periods, you must use entire requisite object specifications.
:BLDDEP	lists direct build dependents of an object. The objects are represented by a combination of the build list name and the object name, which are joined by a period (.). Any combination of build list names, object file names, and object file types may be used as search criteria when you specify them without the periods. When specifying them with the periods, you must use entire dependent object specifications.
:GLOBAL	lists GLOBALs for an object. The data that follows is a library type followed by library names.



Table 21. Valid Tags for VMFQOBJ (continued)

Tag	Description
:PARTHAND	lists the part handler for an object. This field lists a dash (-) if the build list is flagged to be bypassed in the PPF.
:TARGET	lists the target for an object.
:BLOPT	lists the build list options for an object.
:OBJPARG	lists the object parameters for an object.
:PART	lists a serviceable part that is included in an object. The data that follows is a part file name and one or more <i>ftabbrev</i> .
:PARTOPT	is subordinate to the :PART tag. It lists part options for a part included in an object.

5. All build lists defined in the product parameter file are read in their entirety regardless of the tagged data specified.

### Examples

- To display the serviceable parts included in object OFN1.OFT1 in build list BFN1, you enter:

```
VMFQOBJ ppfname compname TDATA :OBJECT BFN1.OFN1.OFT1 :PART
```

If you cannot remember the object file type, you can enter:

```
VMFQOBJ ppfname compname TDATA :OBJECT BFN1 OFN1 :PART
```

- To display the objects that include serviceable part DMSXYZ TXT, enter:

```
VMFQOBJ ppfname compname TDATA :PART DMSXYZ TXT
```

- To display the objects that require both TXTLIB TLB1 and MACLIB MLB1 to be made GLOBAL (which you can do using the CMS GLOBAL command), enter:

```
VMFQOBJ ppfname compname TDATA :OBJECT :GLOBAL TXTLIB TLB1
:GLOBAL MACLIB MLB1
```

- To display the objects that have build requisites of object OFN1.OFT1 in build list BFN1, enter:

```
VMFQOBJ ppfname compname TDATA :OBJECT :BLDREQ BFN1.OFN1.OFT1
```

If you cannot remember the requisite object file type, you can enter:

```
VMFQOBJ ppfname compname TDATA :OBJECT :BLDREQ BFN1 OFN1
```

If you cannot remember the requisite build list name either, enter:

```
VMFQOBJ ppfname compname TDATA :OBJECT :BLDREQ OFN1
```

## Input and Output Files

### Input Files

#### **ppfname PPF**

The usable form product parameter file.

#### **fn {EXEC|EXCnnnnn}**

Build lists.

#### **bldid SRVBLDS**

The service-level build status table.

***cntrlfn* CNTRL**

The control file.

***cntrlfn* CNTRLEXT**

The control file extension.

***appid* VVTlvlid**

The version vector tables.

**VMFNLS LANGLIST**

The language table.

**Input/Output Files**

***fn* {OBJDATA|ft}**

The file containing tagged data.

**Messages and Return Codes**

For a complete explanation of any VMSES/E message, use the HELP Facility to view the message explanation online or see the appropriate messages documentation. To display information on a specific message, for example VMF002E, enter:

```
help msg vmf002e
```

If you are unfamiliar with the z/VM HELP Facility, enter help to display the main HELP Menu. For information on the HELP command, enter:

```
help cms help
```

Appendix D, “Module Identifiers for VMSES/E Messages,” on page 749 shows the format of VMSES/E messages and lists the identifiers for each VMSES/E exec.

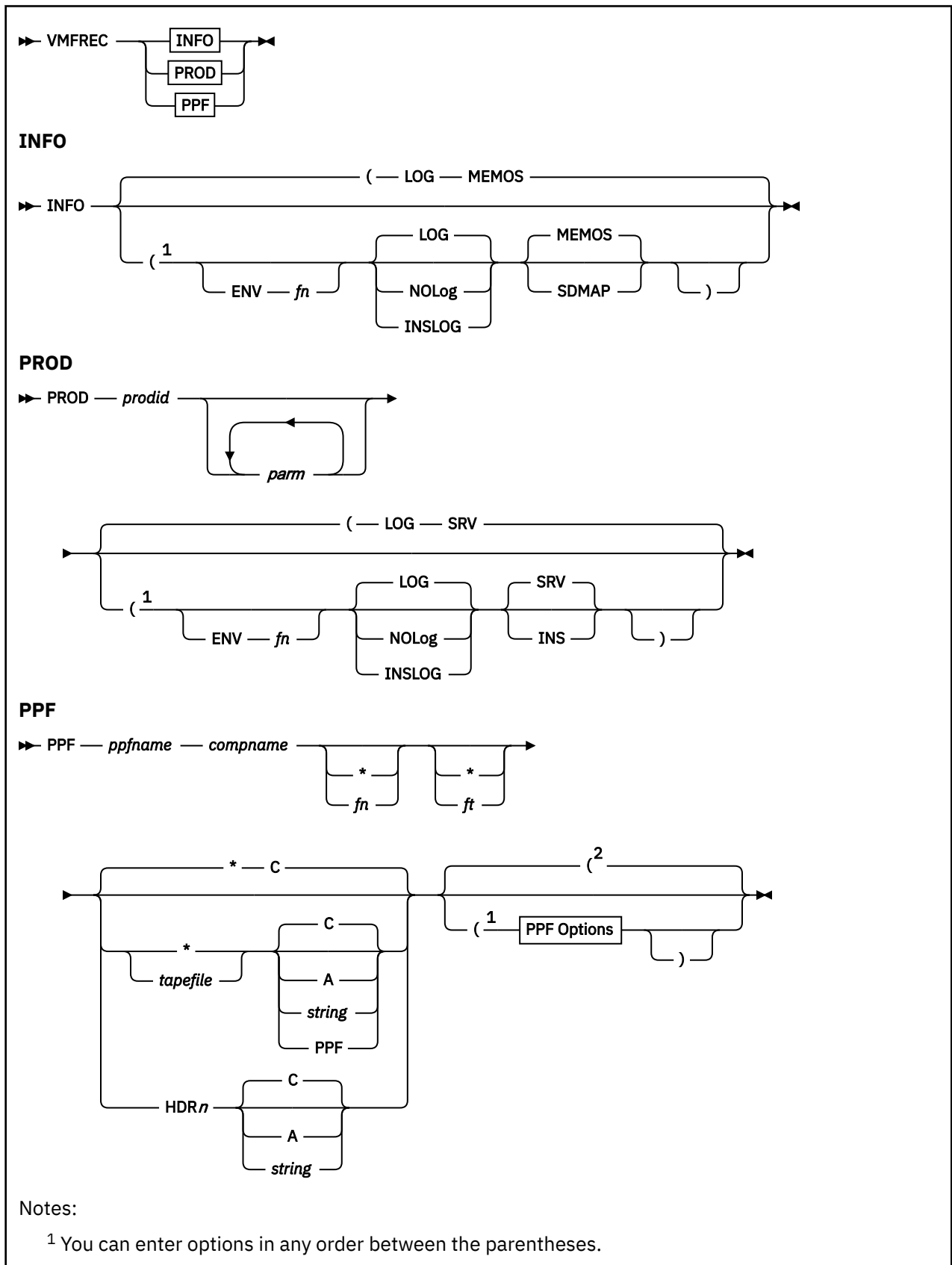
VMFQOBJ issues the following return codes:

<b>Return Code</b>	<b>Explanation</b>
0	Command completed successfully.
2	Command completed successfully but extraneous data was encountered.
4	Command completed with one or more warning conditions.
12	Command failed because of an internal error.
24	Command failed because of a command line syntax error.
28	Command failed because a required file was not found.
36	Command failed because a target disk or directory was not available.
100	Command failed because of an external error.

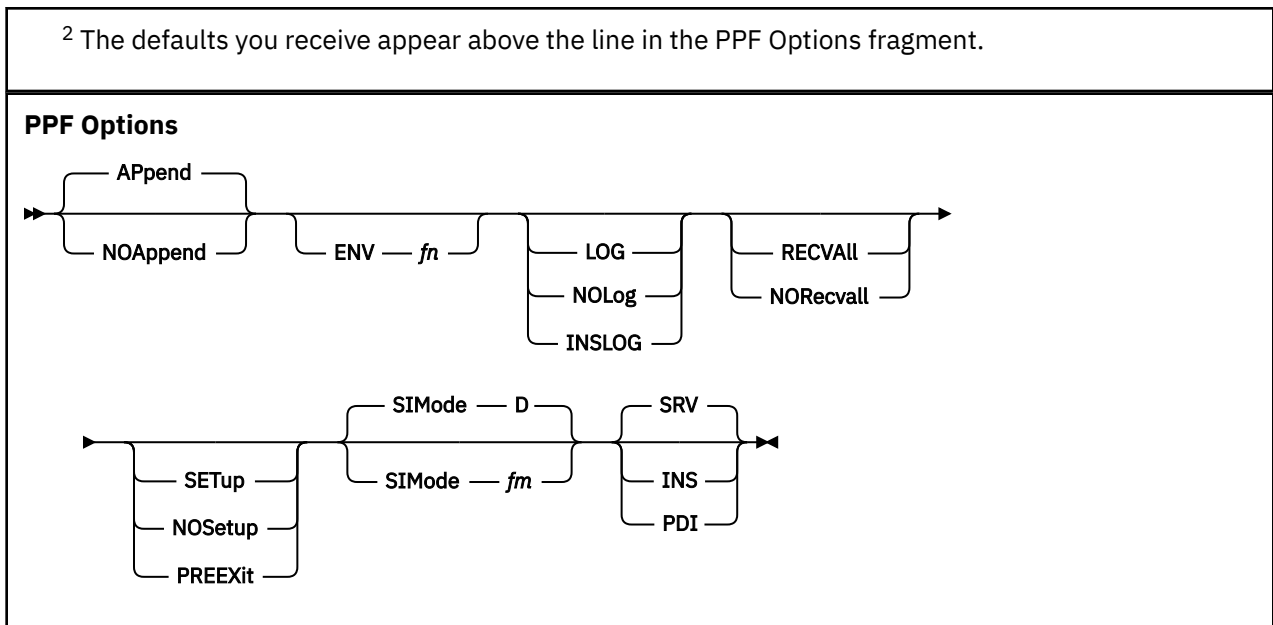
**Recovery Information**

The VMFQOBJ command can be restarted by reissuing the command.

# VMFREC EXEC



<sup>2</sup> The defaults you receive appear above the line in the PPF Options fragment.



## Purpose

VMFREC processes installation tapes, service tapes, and service envelopes.

- The INFO keyword is used to map the tape and plan the installation steps.
- The PROD keyword is used to receive products that are not fully supported by VMSES/E.
- The PPF keyword is used to receive products/components that are fully supported by VMSES/E.

When service is received for fully supported products/components, the Software Inventory is updated to reflect any new PTFs.

- The Receive Status table identifies any PTFs that have been received or committed.
- The Requisite table identifies any requisite relationships between PTFs.
- The Description table contains descriptions of each APAR.

## Operands

### INFO

maps the tape and gives you a list of the products on the tape. You can use this information to plan the installation of the service on the tape.

The INFO operand is only valid for service tapes.

### PROD

indicates the product to be received (*prodid*) is not fully supported by VMSES/E and a product-specific exec is to be used to receive it.

#### *prodid*

is the product identification number and the file name of the product-specific exec. The file type must be EXEC.

#### *parms*

are parameters you want to pass to the product-specific exec. You can enter a maximum of 27 parameters.

### PPF

indicates you want to use a specific product parameter file for receive processing.

#### *ppfname*

is the file name of the usable form product parameter file. The product parameter file must have a file type of PPF.

***compname***

is the name of the component as it is specified on the :COMPNAME tag in the product parameter file. *compname* is a 1-16 character alphanumeric identifier.

**Options****APpend**

appends the contents of any apply and exclude lists on the target disk to the apply and exclude lists on the tape or envelope being received. APPEND is the default.

**NOAppend**

overlays the contents of any apply and exclude lists on the target disk with the apply and exclude lists on the tape or envelope being received.

**ENV**

indicates an envelope rather than a tape is to be received. The envelope can reside on any accessed disk in the post-VMFSETUP access order.

***fn***

is the file name of the envelope. A file name must be specified. The file type of the envelope must be SERVLINK.

**LOG**

writes VMFREC messages into the receive message log (\$VMFREC \$MSGLOG).

No messages are logged until initial validation of the command is complete. The default is LOG.

**NOLog**

does not write VMFREC messages into the receive message log (\$VMFREC \$MSGLOG).

**Note:** If the LOG and NOLOG options are omitted, the VMFREC EXEC uses the value of the :LOG tag in the product parameter file to determine whether to log VMFREC messages into the receive message log.

**INSLOG**

logs messages in the \$VMFINS \$MSGLOG file.

**Note:** The INSLOG option is reserved for use by VMSES/E.

**MEMOS**

does everything the SDMAP option does, and in addition receives the tape document and memos.

The tape document and memos are received on your C-disk if it is accessed R/W, otherwise they are received on your A-disk. MEMOS is the default option for INFO.

**SDMAP**

receives the tape descriptor file, builds the SERVICE DISKMAP file, and lists the products on the tape.

**RECVALL**

requests that committed parts of previously-received PTFs are to be received.

**NOREcvall**

requests that missing parts of previously-received PTFs are not to be received.

**Note:** If the RECVALL and NORECVALL options are omitted, the VMFREC EXEC uses the value of the :RECVALL tag in the product parameter file to determine whether to receive missing parts of previously-received PTFs.

**SETup**

sets up a minidisk/directory access order for the receive function according to entries in the :MDA section of the product parameter file. If a user exit is specified in the product parameter file, setup will occur after the user exit is called.

**NOSetup**

does not set up a new access order.

### PREEXit

sets up a minidisk or SFS directory access order for the receive function according to entries in the :MDA section of the product parameter file. If a user exit is specified in the product parameter file, setup will occur before the user exit is called.

**Note:** If the SETUP, PREEXIT, and NOSETUP options are omitted, the VMFREC EXEC uses the value of the :SETUP tag in the product parameter file to determine whether to set up a new access order.

### SIMode

specifies the file mode for the system-level Software Inventory disk.

The SIMODE option should be used in conjunction with the PDI option. If you do not specify the PDI option, the SIMODE option is ignored.

#### **D**

is the default.

#### ***fm***

specifies the file mode for the system-level Software Inventory disk. If you are going to continue to use this file mode, create a product parameter file override to list it on the :RETAIN tag in the product parameter file so it is not changed by future invocations of the VMFSETUP command.

### SRV

receives a corrective service (COR) tape or program update tape (PUT), if one is mounted. SRV is the default.

### INS

receives an installation tape, if mounted.

### PDI

receives a Parameter Driven Installation tape, if one is mounted.

#### **Note:**

1. The PDI option is reserved for use by the VMFINS EXEC only. It is only valid with the PPF operand.
2. Usage notes [“3” on page 480](#) and [“4” on page 480](#) discuss how the VMFREC EXEC handles tape positioning with and without the PDI option.

## Usage Notes

1. VMFREC requires the alternate DELTA disk to be accessed as read-write.
2. VMFREC requires the entire DELTA string to be accessed.
3. The service tape can be positioned anywhere when you invoke VMFREC; you do not have to issue tape positioning commands. The VMFREC EXEC uses the multi-volume and program contents directories to determine where to position the tape to receive the specified product or service files.

On normal completion of receive processing, the tape is positioned at the first tape file following the product you just received. If receive processing ends with a failure condition, or if processing is interrupted due to a system failure, the position of the tape is unpredictable.

4. The PDI option is reserved for use by the VMFINS EXEC, although no checking is done to enforce this.

Because the VMFINS EXEC positions the tape to the first tape file following the header file for the specified product, the VMFREC EXEC does not check or position the tape when it is invoked with the PDI option.

When the PDI option is specified, messages are logged in the install message log (\$VMFINS \$MSGLOG).

5. Tape file names in the :RECINS and :RECSER sections of the product parameter file can only appear once per section.
6. You can receive a subset of the tape files that are listed in the product parameter file using the part handlers specified in the PPF (instead of the VMFRCALL part handler that is used when you invoke the VMFREC EXEC with the file specification options) and bypass tape files you do not wish to receive.

Simply add a hyphen (-) to the beginning of the part handler name in the tape file entry in the PPF for the tape files you do not want to receive.

If you choose to bypass tape files, it is up to you to ensure that you do not receive partial PTFs. When you apply PTFs, the VMFAPPLY EXEC notifies you if parts are missing.

#### PI

7. You can specify additional processing for tape files by using product processing exits. You identify these exits in the :RECINS and :RECSER sections of the product parameter file using the keyword PPEXIT. The VMFREC EXEC passes everything that follows the PPEXIT keyword, up to the first asterisk or end-of-line, to the EXEC processor.

The VMFREC EXEC executes product processing exits in the order they are encountered in the product parameter file. These exits are executed independently of any action taken by the VMFREC part handlers, that is, whether or not any tape files were loaded by the VMFREC EXEC prior to its encountering the PPEXIT keyword. You can sequentially specify more than one PPEXIT keyword in the product parameter file.

Product processing exit routines can be received as part of a tape file. If a specified exit cannot be found, or errors occur while the exit is processed; the VMFREC EXEC exits with a return code indicating the failure.

#### Note:

- a. If you run the VMFREC EXEC with any of the file-specification operands, the product processing exits are bypassed.
- b. This interface is intended for use by program products that make use of VMSES/E. IBM does not intend this interface for customer use.

#### PI end

### Examples

- To use the VMFREC mapping function using the IBM-supplied defaults, enter:

```
VMFREC INFO
```

- To run VMFREC using the IBM-supplied defaults for a fully-supported product or component, enter:

```
VMFREC PPF ppfname compname
```

- To run VMFREC using the IBM-supplied defaults for a non-fully-supported product, enter:

```
VMFREC PROD prodid parms
```

- To use VMFREC to receive a specific CMS file for a fully-supported product, enter:

```
VMFREC PPF ppfname compname fn ft * A
```

- To use VMFREC to receive a fully-supported product from an envelope, enter:

```
VMFREC PPF ppfname compname (ENV fn
```

## Input and Output Files

### Input Files

#### ***ppfname*** PPF

The usable form product parameter file.

#### **COR** *yddd*

The COR descriptor file.

#### **INS** *nnnn*

The INS descriptor file.

**PUT nnnn**

The PUT descriptor file.

**prodid \$CORymdd**

The COR product contents directory.

**prodid \$INSnnnn**

The INS product contents directory.

**prodid \$PUTnnnn**

The PUT product contents directory.

**VM SYSABRVT**

The file type abbreviation table.

**Input/Output Files****prodid SRVRECS**

The receive status table.

**Output Files****prodid SRVREQT**

The requisite table.

**prodid SRVDESCT**

The description table.

**SERVICE DISKMAP**

The tape or envelope map.

**VMPUT SERVICE**

Old service disk maps.

**VMSES PARTCAT**

The parts catalog table.

**\$VMFREC \$MSGLOG**

The receive message log.

**\$VMFINS \$MSGLOG**

The install message log.

**Temporary Files****axlist APxnnnn\$**

Saved copies of old tape-specific apply lists.

**axlist EXxnnnn\$**

Saved copies of old tape-specific exclude lists.

**PPF Tags Used****:AXLIST**

Defines the apply and exclude list names supplied by IBM on the service tape.

**:BCOMPNAME**

Identifies the base component name assigned to this product.

**:DABBV**

Defines file type abbreviations specific to a product or component and the real and base file types associated with them.

**:LOG**

Controls whether messages are logged.

**:MDA**

Defines symbolic strings and the minidisks or SFS directories associated with them.

**:PRODID**

The identifier of the product, component, release, version, and modification level.

**:PTFFPX**

Defines the 2 letter PTF prefix for the product.



**:RECID**

The identifier of the product being received as it appears on the tape.

**:RECINS**

Defines the tape files included on installation tapes and the part handlers and target strings associated with them.

**:RECSER**

Defines the tape files included on service tapes and the part handlers and target strings associated with them.

**:RECVALL**

Controls whether missing parts of previously received PTFs are received.

**:SETUP**

Controls whether the VMFSETUP EXEC is called to access minidisks and SFS directories.

**:USEREXIT**

Defines the file name of the user exit. If no value is specified no exit is invoked.

## Messages and Return Codes

Appendix D, “Module Identifiers for VMSES/E Messages,” on page 749 shows the format of VMSES/E messages and lists the identifiers used by each VMSES/E exec.

For a complete explanation of any VMSES/E message, use the HELP Facility to view the message explanation online or see the appropriate messages documentation. To display information on a specific message, for example VMF002E, enter:

```
help msg vmf002e
```

If you are unfamiliar with the z/VM HELP Facility, enter help to display the main HELP Menu. For information on the HELP command, enter:

```
help cms help
```

**PI**

These return codes may be returned to a user exit by VMFREC. For more information about user exits, see :USEREXIT..

The VMFREC EXEC issues the following return codes:

Return Code	Explanation
0	Command completed successfully.
4	Command completed with one or more warning conditions.
12	Command failed because of an internal error.
24	Command failed because of a command line syntax error.
28	Command failed because a required file was not found.
36	Command failed because a target disk or directory was not available.
100	Command failed because of an external error.
500	User terminated the command from a prompt.

**PI end**

## Recovery Information

The VMFREC command can be restarted by reissuing the command. If you determine there is an error in the packaging of the tape or envelope, report the problem to the IBM Support Center and order a new tape or envelope.

## Filespec Operands

These operands identify specific CMS files to be received from a tape or envelope. If specific CMS files are requested and none can be found that meet the specifications, the VMFREC EXEC terminates with a return code of 28.

**Note:** When invoked with the file specification operands, the VMFREC EXEC always uses the VMFRCALL part handler to receive the specified files.

### CMS File Specifications

**\***

indicates all files matching the specified file type are to be received.

***fn***

Specifies the file name of the file to be received.

**\***

indicates all files matching the specified file name are to be received.

***ft***

Specifies the file type of the file to be received.

### Tape File Specifications

**\***

indicates all tape files in the :RECINS or :RECSER section of the product parameter file are to be received.

***tapefile***

specifies the tape files from which the files are to be received. Tape files are defined in the :RECINS section of the product parameter file for installation tapes and in the :RECSER section of the product parameter file for service tapes. If an asterisk (\*) is specified for *tapefile*, all tape files in the :RECINS or :RECSER section of the product parameter file are received.

**HDR $n$**

indicates the  $n$ th product header file is to be received. HDR $n$  is a special tape file name that is not specified in the product parameter file.

### Target Specifications

**C**

indicates the C-disk is the target of the receive. If you do not specify a target with the file specification operands, the C-disk is the default target, if it is accessed in R/W mode. If the C-disk is not accessed in R/W mode or is not present, the A-disk is used as the target. The A-disk must be accessed in R/W mode.

**A**

indicates the A-disk is the target of the receive.

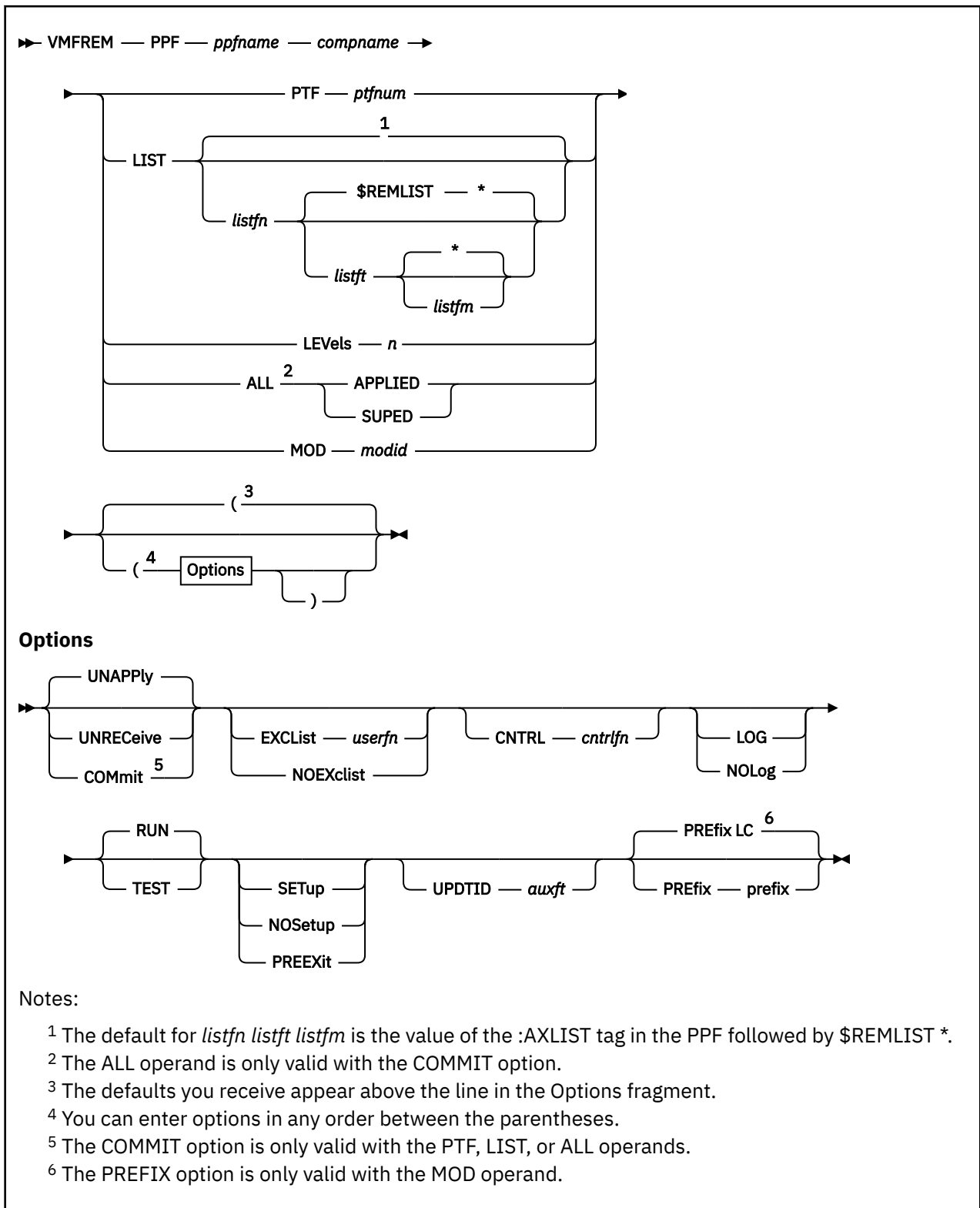
***string***

is an :MDA symbolic name specifying the target of the receive.

**PPF**

indicates you want to use the product parameter file entry for the specified tape file to determine the target of the receive.

# VMFREM EXEC



## Purpose

VMFREM removes individual PTFs by "un-applying" them from all service levels (apply disks) and optionally "un-receiving" them. "Un-apply" means the function previously performed by VMFAPPLY for the specified PTFs is removed. "Un-receive" means the function previously performed by VMFREC for the specified PTFs is removed.

VMFREM also removes complete service levels and optionally un-receives PTFs that are applied only to the removed levels. In addition, commit support is provided for individual PTFs that have been applied.

VMFREM removes local modifications from the VVT and AUX files and optionally erases the parts associated with the local modification.

## Operands

### PPF

indicates the specified product parameter file is to be used for remove processing.

### *ppfname*

is the file name of the usable form product parameter file. The file type must be PPF.

### *compname*

is the name of the component as it is specified on the :COMPNAME tag in the product parameter file. *compname* is a 1-16 character alphanumeric identifier.

### PTF

indicates the specified PTF is selected for processing.

### *ptfnum*

is the 7 character PTF number of the selected PTF.

### LIST

indicates all PTFs in the specified input file are to be selected for processing. The syntax for the input file is one PTF number per record. All data after the first word in a record is considered to be a comment. Any record that starts with an asterisk (\*) is a comment record.

### *listfn*

is the file name of the CMS file containing the list of selected PTFs. The default file name is the value of the :AXLIST tag in the PPF.

### *listft*

is the file type of the CMS file containing the list of selected PTFs. The default file type is \$REMLIST.

### *listfm*

is the file mode of the CMS file containing the list of selected PTFs. The default file mode is the \*.

### LEVELs

indicates complete service levels are to be removed. All PTFs applied to the specified service levels and not applied to lower service levels are selected for processing. A service level corresponds to one disk in the APPLY string in the :DCL section of the PPF.

### *n*

is the specific number of service levels to remove.

If *n* is equal to the total number of disks in the APPLY string, all service is removed.

**Note:** The LEVELS operand is not valid with the COMMIT option.

### ALL

indicates all PTFs with a specified apply status are selected for processing.

### APPLIED

indicates all PTFs that have been applied (status of 'APPLIED' or 'SUPED APPLIED') are selected.

### SUPED

indicates all PTFs that have been superseded after being applied (status of 'SUPED APPLIED') are selected.

**Note:** The ALL operand is valid only with the COMMIT option.

## **MOD**

indicates the specified local modification is selected for processing.

### ***modid***

is a local modification number. It should begin with L, followed by up to 4 alphanumeric characters. For example, LFIX2.

## **Options**

### **UNAPPLy**

indicates the selected PTFs are to be un-applied or selected local modification is to be removed.

**Note:** The UNAPPLY option is not valid with the ALL operand.

### **UNRECeive**

indicates the selected PTFs are to be un-received and un-applied or selected local modification is removed and its parts are erased.

**Note:** The UNRECEIVE option is not valid with the ALL operand.

### **COMmit**

indicates all selected PTFs are to be committed.

**Note:** The COMMIT option is not valid with the LEVELS operand.

### **EXCList**

defines the name of a user-supplied exclude list.

#### ***userfn***

is the file name of the exclude list. The file type of the exclude list is \$EXCLIST. All un-applied and un-received PTFs are added to this file.

### **NOEXclist**

indicates PTFs are not to be added to any exclude list.

**Note:** If neither the EXCLIST nor the NOEXCLIST option is specified, all un-applied and un-received PTFs are added to the IBM supplied exclude list. The file name of the IBM supplied exclude list is the value specified on the :AXLIST tag in the PPF.

### **CNTRL**

defines the name of the control file used to identify the AUX file and version vector table structure.

#### ***cntrlfn***

is the file name of the control file. The file type of the control file is CNTRL. This value overrides the value on the :CNTRL tag in the PPF.

### **LOG**

writes VMFREM messages into the remove message log (\$VMFREM \$MSGLOG). No messages are logged until initial validation of the command is complete.

### **NOLog**

does not write VMFREM messages into the remove message log (\$VMFREM \$MSGLOG).

**Note:** If the LOG and NOLOG options are omitted, the VMFREM EXEC uses the value of the :LOG tag in the PPF to determine whether to log messages into the remove message log.

### **RUN**

requests a complete run of the remove process. The Software Inventory files are updated when the remove process is completed and no errors are encountered. RUN is the default value.

### **TEST**

requests a dry run of the remove process. All remove processing steps are completed, but the Software Inventory files are not updated.

**SETup**

sets up a minidisk/directory access order for the VMFREM function according to entries in the :MDA section of the PPF. If a user exit is specified in the product parameter file, setup will occur after the user exit is called.

**NOSetup**

does not set up a new access order.

**PREEXit**

sets up a minidisk or SFS directory access order for the VMFREM function according to entries in the :MDA section of the product parameter file. If a user exit is specified in the product parameter file, setup will occur before the user exit is called.

**Note:** If the SETUP, PREEXIT, and NOSETUP options are omitted, the VMFREM EXEC uses the value of the :SETUP tag in the product parameter file to determine whether to set up a new access order.

**UPDTID**

defines the file type of the AUX files that are generated for parts that have the AUX option specified in the \$PTFPART files. It also defines the file type of the version vector tables that are updated by the VMFREM EXEC.

***auxft***

is the file type of the AUX files. This value overrides the value on the :UPDTID tag in the PPF.

**PREfix**

provides the local modification identification prefix which, combined with the modid, forms the local modification identification

***prefix***

is the two-character local modification identification prefix. The default prefix is LC.

**Usage Notes**

1. Backups of all the DELTA and APPLY disks (or directories) should be taken before running VMFREM.
2. VMFREM removes PTFs from the service software inventory tables and sets up for build processing. To complete the removal you must run VMFBLD (refer to the product's service instructions).
3. When you specify the UNRECEIVE option for an applied PTF, the PTF is first un-applied and then un-received.
4. Dependent PTFs in the same component or product of the PTF being removed are un-applied, but not un-received.
5. If you specify the UNAPPLY option for a PTF for which another component/product has a dependency, you must remove the dependent component/product or dependent component/product PTF. Run VMFREM with the TEST option to get a list of PTFs that will be un-applied. Use this list to determine whether these PTFs have any other component/product dependencies. Refer to the User Response section of message VMF2135I.
6. Removed PTFs are placed in the appropriate exclude list. To "re-apply" a removed PTF, you must remove it from the exclude list.
7. A PTF with a status of 'APPLIED' or 'SUPED APPLIED', in the service-level apply software inventory table (*appid* SRVAPPS), can be committed. If the status of a PTF is only 'SUPED', in *appid* SRVAPPS, it cannot be committed. If a 'SUPED' PTF has been received, its parts can be removed by using the UNRECEIVE option for the PTF.
8. When a PTF with a status of 'SUPED APPLIED', in the service-level apply software inventory table (*appid* SRVAPPS), is un-applied, only the 'APPLIED' status is removed. Because the 'SUPED' status is still active, no dependent PTFs need be removed.
9. VMFREM checks for the existence of the new highest level of each part of each PTF that will be un-applied. If such a part is missing, messages VMF2130W and VMF2131R are issued. You must respond by specifying QUIT, CONTINUE, or BYPASS. If you specify QUIT, VMFREM terminates with no changes to the software inventory. If you specify CONTINUE, VMFREM continues the remove processing, but you must restore the missing parts before running VMFBLD. If you specify BYPASS,

VMFREM continues the remove processing and the missing parts are not added to the select data (\$SELECT) file. This means that these parts will not trigger any build processing, and objects containing these parts will still contain the parts from the removed PTFs. The BYPASS response is only offered for base level parts. You should use the CONTINUE and BYPASS options with caution. Refer to the User Response section of message VMF2131R for more information on the responses.

10. When you use the COMMIT option for a PTF, its parts are erased and its status in the receive software inventory table (*recid* SRVRECS) is set to 'COMMITTED'.
11. If you specify the UNAPPLY option for a committed PTF, it will also be un-received.
12. To "re-receive" a committed PTF, first use the UNAPPLY option to un-apply and un-receive the PTF. Then remove the PTF from the exclude list, issue the VMFREC command, and issue the VMFAPPLY command.
13. Always review the remove message log (\$VMFREM \$MSGLOG) after using the VMFREM command.

## Examples

- To run VMFREM to do a dry run of un-applying a PTF, enter:

```
VMFREM PPF ppfname compname PTF ptfnum ( TEST
```

- To run VMFREM to un-apply a PTF, enter:

```
VMFREM PPF ppfname compname PTF ptfnum
```

- To run VMFREM to un-apply and un-receive a PTF, enter:

```
VMFREM PPF ppfname compname PTF ptfnum ( UNRECEIVE
```

- To run VMFREM to remove one service level (alternate level), enter:

```
VMFREM PPF ppfname compname LEVELS 1
```

- To run VMFREM to commit a PTF, enter:

```
VMFREM PPF ppfname compname PTF ptfnum ( COMMIT
```

- To run VMFREM to remove a local modification, enter:

```
VMFREM PPF ppfname compname MOD modid
```

## Input and Output Files

### Input Files

#### ***ppfname* PPF**

The useable form product parameter file.

#### ***cntrlfn* CNTRL**

The control file identified by the :CNTRL tag in the PPF or the CNTRL option on the VMFREM command.

#### ***ptfnum* \$PTFPARTS**

The PTF parts file.

#### **VM SYSABRVT**

The file type abbreviation table.

#### ***bldlist* EXCnnnnn**

The build list.

#### ***appid* \$APRCVRY**

The existence of this file on the APPLY disk string indicates VMFAPPLY was interrupted during critical processing on the last invocation of VMFAPPLY for the specified component (used for recovery).

**Input/Output Files*****appid* SRVAPPS**

The service-level apply status table.

***appid* VVTlvlid**

The version vector tables specified by the control file specified on the :CNTRL tag in the product parameter file.

***appid* \$SELECT**

The select data file.

***appid* \$RMRCVRY**

The existence of this file on the APPLY or DELTA disk string indicates VMFREM was interrupted during critical processing on the last invocation of VMFREM for the specified component (used for recovery).

***recid* SRVRECS**

The service-level receive status table.

***recid* SRVREQT**

The service-level requisite table.

***recid* SRVDESCT**

The service-level description table.

**Output Files*****fn* EXCLIST**

The exclude list, which identifies PTFs that are not to be processed by VMFAPPLY.

**VMSES PARTCAT**

The parts catalog table.

**\$VMFREM \$MSGLOG**

The remove message log.

***partid* AUXlvlid**

AUX files for parts that have the AUX option specified in the \$PTFPART file.

***partid* EAXlvlid**

Local AUX files that have become empty as a result of all entries being commented out by local modification removal processing.

**Temporary Files*****appid* \$SRAPPS**

The apply status table with a temporary file type (used for recovery).

***appid* \$SNAPPS**

The temporary file that indicates an empty apply status table (used for recovery).

***appid* \$VVLvlid**

The version vector tables with temporary file types (used for recovery).

***appid* \$VNLvlid**

The temporary files that indicate empty version vector tables (used for recovery).

***appid* \$\$SELECT**

The select data file with a temporary file type (used for recovery).

***appid* \$ASTATS**

This file contains the values of key variables that are required for un-apply recovery (used for recovery).

***recid* \$SRRECS**

The receive status table with a temporary file type (used for recovery).

***recid* \$SNRECS**

The temporary file that indicates an empty receive status table (used for recovery).

***recid* \$SRREQT**

The requisite table with a temporary file type (used for recovery).



**recid \$SNREQT**

The temporary file that indicates an empty requisite table (used for recovery).

**recid \$SRDESCT**

The description table with a temporary file type (used for recovery).

**recid \$SNDESCT**

The temporary file that indicates an empty description table (used for recovery).

**recid \$DSTATS**

This file contains the values of key variables that are required for un-receive recovery (used for recovery).

## Messages and Return Codes

Appendix D, “Module Identifiers for VMSES/E Messages,” on page 749 shows the format of VMSES/E messages and lists the identifiers used by each VMSES/E exec.

For a complete explanation of any VMSES/E message, use the HELP Facility to view the message explanation online or see the appropriate messages documentation. To display information on a specific message, for example VMF002E, enter:

```
help msg vmf002e
```

If you are unfamiliar with the z/VM HELP Facility, enter help to display the main HELP Menu. For information on the HELP command, enter:

```
help cms help
```

### PI

Return codes issued by the VMFREM EXEC may be returned to a user exit. For more information about user exits, see [:USEREXIT.](#)

The VMFREM EXEC issues the following return codes:

Return Code	Explanation
0	Command completed successfully.
4	Command completed with one or more warning conditions.
8	Command completed but at least one major process failed.
12	Command failed because of an internal error.
24	Command failed because of a command line syntax error.
28	Command failed because a required file was not found.
36	Command failed because a target disk or directory was not available.
100	Command failed because of an external error.
500	User terminated the command from a prompt.

**Note:** The Software Inventory is updated if the return code is less than or equal to 4.

### PI end

## Recovery Information

The VMFREM command can be restarted by reissuing the command. **Case 1:**

VMFREM is interrupted while processing PTFs.

While processing PTFs, all data is maintained in storage. No data is written to the APPLY or DELTA disks. The only external changes are in the \$VMFAPP \$MSGLOG. In terms of the APPLY disks (the maintenance level of the system), this is equivalent to the command not being issued. **Case 2:**

VMFREM is interrupted while updating the Software Inventory.

After the processing of PTFs is complete, the Software Inventory (the APPLY or DELTA disks) are updated. Because this is an interruptible process, a situation could exist where half the information is updated. To avoid this inconsistency, VMFREM:

- Updates the Software Inventory using temporary file types
- Saves key variables in temporary files
- Sets the recovery flag (writes a file named *appid* \$RMRCVRY to the APPLY disk or DELTA disk or both)
- Copies the Software Inventory files to the real file types
- Creates AUX files
- Updates the \$SELECT file
- Resets the recovery flag (erases the files named *appid* \$RMRCVRY)
- Erases all temporary files

If the process is restarted while the recovery flag (*appid* \$APRCVRY) is set, VMFREM detects this condition and completes the prior VMFREM invocation based on the response from the user. The prior VMFREM invocation is completed using the information saved in the temporary files.



## Purpose

The VMFREPL EXEC is used to support the local modification of replacement maintained parts. VMFREPL can be used to:

- Copy the highest level of a part
- Copy a specified part
- Update a Version Vector Table
- Update a Select Data file
- Display the highest levels of a part

## Operands

### *fn*

is the real file name of the replacement part.

### *ft*

is the real file type of the replacement part.

### *ppfname*

is the file name of a usable form Product Parameter File (PPF). It must have a file type of PPF. The name of the control file that is to be used to determine version vector table file types is obtained from this PPF.

### *compname*

is the name of the component (such as CP or CMS) as it is specified on the :COMPNAME tag in the PPF. *compname* is a 1-16 character alphanumeric identifier.

### *ifn*

is the file name of the input file. If *ifn* is specified as an equal sign (=), the value of *fn* is used. If *ifn* is not specified, the file ID of the input file is determined by using the version vector structure to locate the highest level of the replacement part.

### *ift*

is the file type of the input file. If *ift* is specified as an equal sign (=), the value of *ft* is used.

### *ifm*

is the file mode of the input file. If *ifm* is specified as an asterisk (\*), the first occurrence of the file is used.

## Options

### SETup

sets up a minidisk or SFS directory access order for the VMFREPL function according to entries in the :MDA section of the PPF. If a user exit is specified in the product parameter file, setup will occur after the user exit is called.

### NOSetup

does not set up a new access order.

### PREEXit

sets up a minidisk or SFS directory access order for the VMFREPL function according to entries in the :MDA section of the product parameter file. If a user exit is specified in the product parameter file, setup will occur before the user exit is called.

**Note:** If the SETUP, PREEXIT, and NOSETUP options are omitted, the VMFREPL EXEC uses the value of the :SETUP tag in the product parameter file to determine whether to set up a new access order.

### CNTRL

specifies the control file that is used to identify the version vector table structure.

### *cntrlfn*

is the file name of the control file that is used to identify the version vector table structure. The file type of the control file is CNTRL. This value overrides the value on the :CNTRL tag in the PPF.

**COPY**

copies the input part. COPY is the default.

**NOCopy**

bypasses copying the input part.

**NOQuery**

does not perform the query function. NOQUERY is the default.

**Query**

issues a message with the highest levels of IBM service and local modification for the replacement part.

**NO\$SElect**

does not update the *appid* \$SELECT file. NO\$SELECT is the default.

**\$SElect**

updates the *appid* \$SELECT file to indicate the replacement part has been changed. The first APPLY disk specified in the :MDA section of the product parameter file must be accessed Read/Write. The other APPLY disks must be accessed.

**LOGMOD**

updates a version vector table for the replacement part using the resolved file type abbreviation and the specified *modid*.

***modid1***

is the *modid* used to update the version vector table and to construct the file type of the copied file. *modid* is a local modification number. It should begin with L, followed by up to 4 alphanumeric characters. For example, LFIX2.

**VVTFT**

overrides the version vector table to be updated.

***VVTlvid***

is the file type of the version vector table to be updated.

**MODID**

provides the *modid* if the LOGMOD option is not specified or overrides the *modid* specified with the LOGMOD option.

***modid2***

is the *modid* used to construct the file type of the copied file and to update the version vector table if the LOGMOD option is specified. *modid* is a local modification number. It should begin with L, followed by up to 4 alphanumeric characters. For example, LFIX2.

**FTAbbr**

is used to name the file type for the output file that is created. This option overrides the three character abbreviation that is obtained from the VM SYSABRVT table.

***ftabbr***

is the three character abbreviation used with the file type. For example, a file type of SXE12345 has SXE as the *ftabbr*.

**FILEType**

indicates the file type for the output file that is created. This option overrides any naming from the VVT structure.

***out\_ft***

is the file type for the output file.

**OUTMode**

indicates the file mode for the replacement part that is created. This file mode must be accessed Read/Write.

**A**

indicates file mode A is the file mode for the replacement part. A is the default.

***fm***

is the file mode for the replacement part.

***mda\_string***

is the name of a symbolic string of disks from the :MDA section of the product parameter file. The replacement part is placed on the first disk specified in this string.

**PREfix**

provides the local modification identification prefix which, combined with the modid, forms the local modification identification

***prefix***

is the two-character local modification identification prefix. The default prefix is LC.

**MOD**

indicates the highest overall level of the specified part is used as the input part. MOD is the default.

**PTF**

indicates the highest service level of the specified part is used as the input part.

**LOG**

identifies the type of message logging to be done. Messages are logged in the specified message log as well as written to the terminal. The message log is not initialized.

No messages are logged until initial validation of the command is complete.

**NONE**

is the default. If NONE is specified, messages are written only to the terminal.

***logid***

is a three-character message log identifier, for example:

***logid*****Type of Log****APP**

The apply message log (\$VMFAPP \$MSGLOG A)

**BLD**

The build message log (\$VMFBLD \$MSGLOG A)

**XYZ**

The user message log (\$VMFXYZ \$MSGLOG A)

**ILOG**

identifies the type of message logging to be done. Messages are logged in the specified message log as well as written to the terminal. The message log is initialized before the first message is written.

No messages are logged until initial validation of the command is complete.

***logid***

is a three-character message log identifier, for example:

***logid*****Type of Log****APP**

The apply message log (\$VMFAPP \$MSGLOG A)

**BLD**

The build message log (\$VMFBLD \$MSGLOG A)

**XYZ**

The user message log (\$VMFXYZ \$MSGLOG A)

**LOGLVL**

identifies the level of message logging to be done.

**Note:** The LOGLVL option is ignored if the LOG option is NONE.

**I**

is the default. 'I' logs all informational messages, warning messages, error messages, severe error messages, and terminating errors.

***mlvl***

is the message severity level. Messages are logged in the specified message log if they have a severity level equal to or above the *mlvl* specified. The message levels are shown below, in order of severity. Messages of level 'R' have the lowest severity, and messages of level 'T' have the highest severity.

***mlvl*****Level of logging****R**

Response required

**I**

Informational Message

**W**

Warning Message

**E**

Error Message

**S**

Severe Error Message

**T**

Terminating Error

**Usage Notes**

1. When you specify the \$SELECT option, the select data file (*appid* \$SELECT) is updated with a record consisting of one of the following:
  - *fn* and the 3-character abbreviation for the file type of the output file
  - *fn* and the full file type (when you also specify the FILETYPE option)
  - *fn*, 3-character abbreviation for the file type of the output file, full file type of the previous level of the part, and "BASE-FILETYPE" if appropriate (when the part is a build list)

The select data file is used by VMFBLD to determine which objects need to be built using the replacement part.
2. When you create local modifications, you can use the \$SELECT, LOGMOD, and OUTMODE options to eliminate some manual steps, such as updating the *appid* \$SELECT file, updating local version vector tables, and saving the results on a LOCALMOD disk.
3. VMFBLD uses the version vector tables to determine the correct level of the part to use during build processing. If you do not specify the LOGMOD option, you must either manually update the version vector tables before you run VMFBLD or you must rerun VMFREPL and specify the LOGMOD option.
4. When you specify the LOGMOD option, a local version vector table is updated using the resolved file type abbreviation and the specified modid. If the part is currently found in a local version vector table, the same table is updated. If not, the highest local level version vector table defined in the control file is updated.
5. The file type of the copied part is determined by:
  - a. The FILETYPE option
  - b. The file type abbreviation and the MODID option
  - c. The file type abbreviation and the LOGMOD option
  - d. The file type abbreviation and the highest MOD level in the version vector structure
  - e. The base file type

## Examples

- To copy the highest level of CMSINST LSEG to the LOCALMOD disk as CMSINST SEGL0001, update the local version vector table, and update the \$SELECT file, enter:

```
vmfrepl cmsinst lseg servp2p cms ($select logmod 10001 outmode localmod
```

- To copy CMSINST TESTLSEG A to the LOCALMOD disk as CMSINST SEGL0002, update the local version vector table, and update the \$SELECT file, enter:

```
vmfrepl cmsinst lseg servp2p cms = testlseg a ($select
  logmod 10002 outmode localmod
```

- To copy TELL SXEL0003 F to TELL CEXL0003 F, update the local version vector table, and update the \$SELECT file, enter:

```
vmfrepl tell exec servp2p cms = sxel0003 f ($select ftabbr
  cex logmod 10003 outmode f
```

- To copy TELL SXEL0004 F to TELL CEXL0004 F and update the \$SELECT file, enter:

```
vmfrepl tell exec servp2p cms = sxel0004 f ($select
  ftabbr cex modid 10004 outmode f
```

- To update the local version vector table for TELL CEX with modid L0005 and update the \$SELECT file, enter:

```
vmfrepl tell exec servp2p cms ($select ftabbr cex logmod 10005 nocopy
```

- To update the \$SELECT file for TELL CEX, enter:

```
vmfrepl tell exec servp2p cms ($select ftabbr cex nocopy
```

- To update the \$SELECT file for FSOPEN MACRO, enter:

```
vmfrepl fsopen macro servp2p cms ($select nocopy filetype macro
```

- To update the \$SELECT file for SYSPROF EXC, enter:

```
vmfrepl sysprof exec servp2p cms ($select nocopy
```

- To display the highest service and local modification levels for SYSPROF EXEC, enter:

```
vmfrepl sysprof exec servp2p cms (query nocopy
```

## Input and Output Files

### Input Files

#### **ppfname** PPF

The usable form product parameter file.

#### **cntrlfn** CNTRL

The control file.

#### **appid** VVTlvid

The version vector table.

### Output Files

#### **fn out\_ft**

The replacement file, when the FILETYPE option is specified.

#### **fn xxxnnnnn**

The replacement file (xxx is the file type abbreviation; nnnnn is a modid).

**Note:** You receive only one of the above formats.



**appid VVTlvlid**

A version vector table when the LOGMOD option is specified.

**appid \$SELECT**

The list of build requirements when the \$SELECT option is specified.

**PPF Tags Used****:APPID**

The identifier of the product, which is used to name the version vector tables and the \$SELECT file.

**:CNTRL**

Defines the name of the control file.

**:MDA**

Defines symbolic strings and the minidisks or SFS directories associated with them.

**:SETUP**

Controls whether the VMFSETUP EXEC is called to access minidisks/directories.

**:USEREXIT**

Defines the file name of the user exit. If no value is specified no exit is invoked.

**Messages and Return Codes**

Appendix D, “Module Identifiers for VMSES/E Messages,” on page 749 shows the format of VMSES/E messages and lists the identifiers used by each VMSES/E exec.

For a complete explanation of any VMSES/E message, use the HELP Facility to view the message explanation online or see the appropriate messages documentation. To display information on a specific message, for example VMF002E, enter:

```
help msg vmf002e
```

If you are unfamiliar with the z/VM HELP Facility, enter help to display the main HELP Menu. For information on the HELP command, enter:

```
help cms help
```

**PI**

Return codes issued by the VMFREPL EXEC may be returned to a user exit. For more information about user exits, see [:USEREXIT..](#)

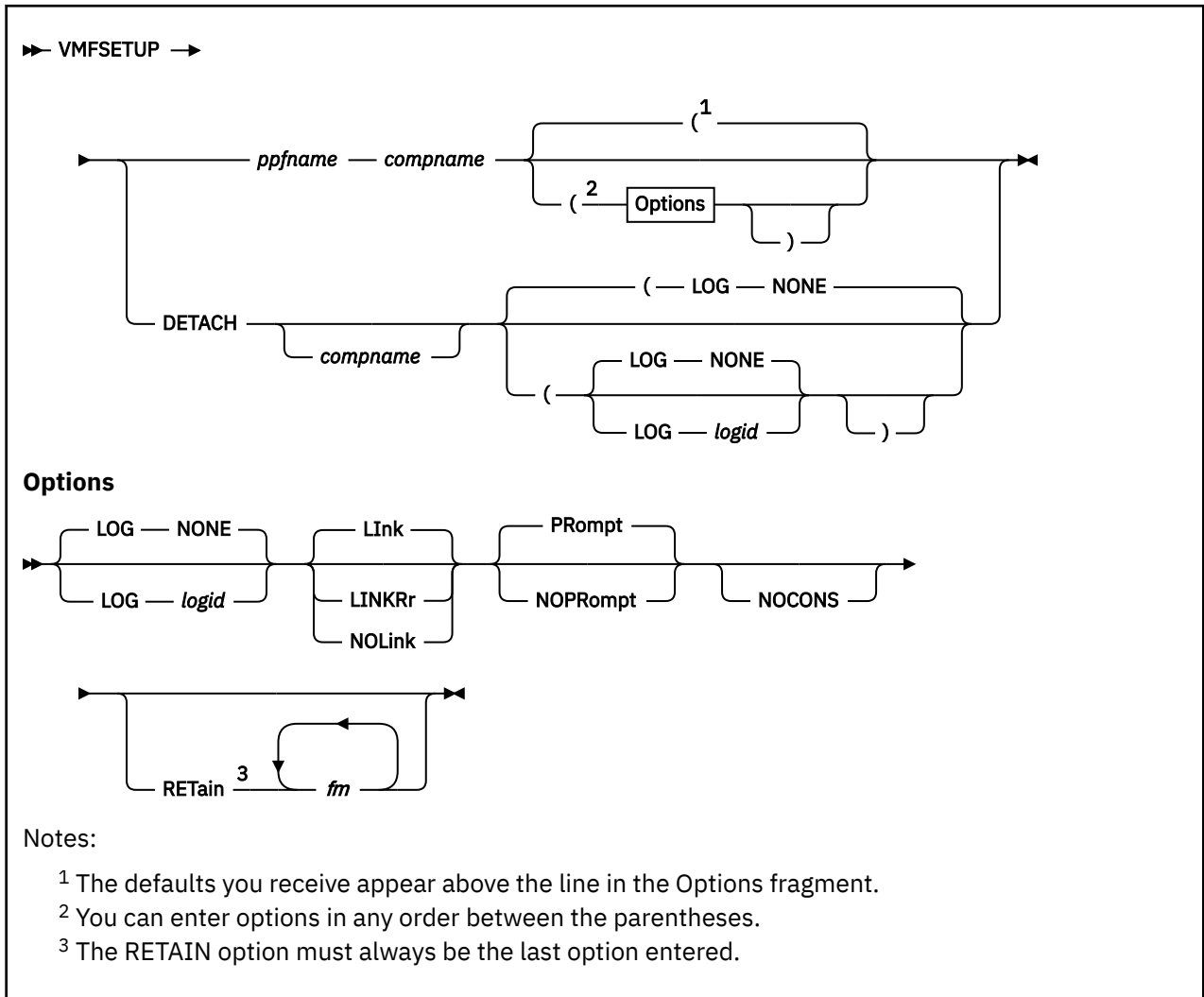
The VMFREPL EXEC issues the following return codes:

Return Code	Explanation
0	Command completed successfully.
4	Command completed with one or more warning conditions.
8	Command completed but at least one major process failed.
24	Command failed because of a command line syntax error.
28	Command failed because a required file was not found.
36	Command failed because a target disk or directory was not available.
100	Command failed because of an external error.

**PI end****Recovery Information**

The VMFREPL command can be restarted by reissuing the command.

## VMFSETUP EXEC



### Purpose

The VMFSETUP EXEC performs these functions:

- When the VMFSETUP EXEC is run without the DETACH keyword, it sets up a minidisk and SFS directory access order for the component being serviced, using the :MDA section of the product parameter file (PPF).
- When a linking option is specified (or used by default), VMFSETUP links to disks whose addresses appear in the :MDA section and are also defined in the :DCL section of the PPF. The LINKRR option causes all such disks to be linked and accessed with read-only status.
- When the VMFSETUP EXEC is run with the DETACH keyword, it detaches disks that were linked by previous invocations of VMFSETUP.

VMFSETUP does not release disks accessed as file mode A, B, C, D, S, or Y.

VMFSETUP can be called by other execs, for example the VMFAPPLY, VMFASM, VMFBLD, VMFMRDSK, and VMFREC execs, by using the SETUP option on the command invocation.

## Operands

### *ppfname*

is the file name of the usable form product parameter file. The file type must be PPF.

### *compname*

is the name of the component as it is specified on the :COMPNAME tag in the PPF. *compname* is a 1-16 character alphanumeric identifier.

### DETACH

detaches disks that were linked by previous invocations of VMFSETUP during the current logon session.

### *compname*

detaches only those disks associated with the component specified by *compname*. *compname* is a 1-16 character alphanumeric identifier.

## Options

### LOG

identifies the type of message logging to be done. Messages are logged in the specified message log as well as written to the terminal.

No messages are logged until initial validation of the command is complete.

**Note:** The LOG option is reserved for use by VMSES/E.

### NONE

is the default. If NONE is specified, messages are written only to the terminal.

### *logid*

is a three-character message log identifier, for example:

#### *logid*

#### Type of Log

#### APP

The apply message log (\$VMFAPP \$MSGLOG A)

#### BLD

The build message log (\$VMFBLD \$MSGLOG A)

#### XYZ

The user message log (\$VMFXYZ \$MSGLOG A)

### LIink

resolves any links specified in the :DCL section of the PPF using the CP LINK command. LINK is the default.

### LINKRr

resolves any links specified in the :DCL section of the PPF using the CP LINK command, and acquires these disks with read-only status.

### NOLink

does not resolve the :DCL. section of the PPF. If links are required and the NOLINK option is specified, VMFSETUP fails. Any required links must be established manually before using the VMFSETUP command.

### PRompt

prompts the user for the LINK password if the user ID does not have the necessary authority.

### NOPRompt

does not prompt the user for LINK passwords. If the user ID does not have the necessary authority, an error message is issued.

### NOCONS

prevents informational and warning messages (other than VMF2204I) from being displayed at the console.

**Note:** The NOCONS option is reserved for use by VMSES/E.

### RETain

lists file modes that VMFSETUP should not use. File modes specified in this list override any values on the :RETAIN. tag in the PPF. File modes in this list are not used and remain accessed.

Access order will be preserved. A disk listed on the :RETAIN tag will cause other disks listed in the :MDA section to be accessed after the next available file mode. If multiple modes are used by the :RETAIN tag and an out of sequence access order occurs (according to the access order in the :MDA section of the PPF) a warning message will be issued.

### *fm*

is the file mode. File modes must be separated by blanks.

**Note:** File modes A thru D, S, and Y are automatically retained.

## Usage Notes

1. The VMFSETUP EXEC accesses minidisks and directories by string name in the order they are specified in the :MDA section of the PPF. Within each string, the minidisks and directories are accessed from left to right. The first occurrence of the disk or directory establishes its place in the access order.
2. If you enter a forward slash (/) directly after the minidisk number in the :MDA section of the PPF, the VMFSETUP EXEC accesses it as read-only. This is the case when either the LINK or LINKRR linking option is used.

In the following example, the 2D4 and 2D2 disks are to be accessed as read-only by accessing them as extensions of themselves (for example, mode/mode) by the VMFSETUP EXEC.

```
DELTA 2D6 2D4/ 2D2/
```

- If a minidisk or directory appears more than once, its status is determined by the first specification.
  - If a minidisk or directory is already accessed read-write, it is released and reaccessed read-only.
  - If a minidisk is empty (no CMS files on it), it cannot be accessed as read-only.
    - When the LINK option is in effect and read-only access is requested for an empty disk, a warning message is issued and the disk is accessed as read/write. If the disk is linked read-only, an error message is issued and the disk is not accessed.
    - When the LINKRR option is in effect and read-only access is requested for an empty disk, a warning message is issued and the disk is not accessed.
3. If you enter a question mark (?) directly after the minidisk number in the :MDA section of the PPF, the VMFSETUP EXEC does not access the disk.

In the following example, 1DF is not to be accessed, but will be linked to if the LINK option is specified.

```
SYSTEM 1DF?
```

4. The DETACH operand will detach only minidisks that had been previously linked using the VMFSETUP command. Any disks that you have linked yourself will not be detached. The DETACH operand has no affect on SFS directories.
5. Some minidisks specified in the :DCL section of the PPF that are linked and then accessed by the processing of an :MDA section might not all be released when the DETACH operand is used. This is the case for minidisks which VMFSETUP determines to be "base" minidisks (that is, minidisks for which a link exists prior to the use of VMFSETUP). For information about restoring an original access order, see ["Recovery Information" on page 505](#).
6. If you enter a LINK, DEFINE, or DETACH command between invocations of the VMFSETUP EXEC, the results might be unexpected.

## Examples

- To run VMFSETUP for the access function and link to disks specified in the :MDA and :DCL sections without logging messages, and without retaining specific disks in your access order, enter:

```
VMFSETUP ppfname compname
```

- To run VMFSETUP for the access function and link to disks specified in the :MDA and :DCL sections, enter:

```
VMFSETUP ppfname compname (LINK
```

- To run VMFSETUP for the access function without linking any disks, enter:

```
VMFSETUP ppfname compname (NOLINK
```

- To run VMFSETUP for the access function, to issue links, and to retain modes H and R in your access order, enter:

```
VMFSETUP ppfname compname (LINK RETAIN H R
```

- To run VMFSETUP to detach any links done by VMFSETUP for a specific component name, enter:

```
VMFSETUP DETACH compname
```

- To run VMFSETUP to detach all the links done by previous invocations of VMFSETUP during this logon session, enter:

```
VMFSETUP DETACH
```

- To run VMFSETUP to detach all the links done by previous invocations of VMFSETUP during this logon session and log messages in the VMFAPPLY message log, enter:

```
VMFSETUP DETACH (LOG APP
```

- Figure 161 on page 503 shows an example of the output from the VMFSETUP EXEC.

```
ST:VMFSET2760I VMFSETUP processing started for SERVP2P CMS
ST:VMFUTL2205I Minidisk|Directory Assignments:
ST:          String      Mode  Stat  Vdev  Label  (OwnerID  Odev  :  Cyl/%Used)
              -or-
              SFS Directory Name
ST:VMFUTL2205I LOCALMOD  E     R/W  3C4  MNT3C4 (MAINT730 03C4 :   9/02)
ST:VMFUTL2205I LOCALSAM  F     R/W  3C2  MNT3C2 (MAINT730 03C2 :   5/04)
ST:VMFUTL2205I APPLY     G     R/W  3A6  MNT3A6 (MAINT730 03A6 :   6/01)
ST:VMFUTL2205I           H     R/W  3A4  MNT3A4 (MAINT730 03A4 :   6/01)
ST:VMFUTL2205I           I     R/W  3A2  MNT3A2 (MAINT730 03A2 :   6/01)
ST:VMFUTL2205I DELTA     J     R/W  3D2  MNT3D2 (MAINT730 03D2 : 208/01)
ST:VMFUTL2205I BUILD7    K     R/W  493  MNT493 (MAINT730 0493 : 250/53)
ST:VMFUTL2205I BUILD6    L     R/W  490  MNT490 (MAINT730 0490 : 207/41)
ST:VMFUTL2205I BUILD10   M     R/W  550  PMT550 (PMAINT  0550 :   20/02)
ST:VMFUTL2205I BUILD0    N     R/W  49E  MNT49E (7VMLEN30 049E : 250/61)
ST:VMFUTL2205I BUILD8    O     R/W  400  MNT400 (MAINT730 0400 : 275/03)
ST:VMFUTL2205I BUILDA    P     R/W  890  MNT890 (MAINT730 0890 :   50/01)
ST:VMFUTL2205I BUILD4    Q     R/W  49D  MNT49D (MAINT730 049D : 146/65)
ST:VMFUTL2205I BUILDN    Q     R/W  49D  MNT49D (MAINT730 049D : 146/65)
ST:VMFUTL2205I BASE2     R     R/W  3B2  MNT3B2 (MAINT730 03B2 : 375/70)
ST:VMFUTL2205I -----  A     R/W  191  MNT191 (MAINT730 0191 : 175/18)
ST:VMFUTL2205I -----  B     R/W  5E6  MNT5E6 (MAINT730 05E6 :   9/82)
ST:VMFUTL2205I -----  C     R/W  2CC  MNT2CC (PMAINT  02CC :  10/23)
ST:VMFUTL2205I -----  D     R/W  51D  MNT51D (MAINT730 051D :  26/45)
ST:VMFUTL2205I -----  S     R/O  190  MNT190 (MAINT  0190 : 207/41)
ST:VMFUTL2205I -----  Y/S   R/O  19E  MNT19E (MAINT  019E : 500/33)
ST:VMFSET2760I VMFSETUP processing completed successfully
```

Figure 161. VMFSETUP Sample Output

In Figure 161 on page 503, the following information is provided:

### String

is the name of a symbolic disk string, as defined in a product parameter file (PPF). If this field is left blank, the minidisk or Shared File System (SFS) directory cited is part of the same string as the previously-listed minidisk or directory. If this field is filled with dashes, the minidisk or directory is not part of any defined disk string.

### Mode

is the file mode letter at which the minidisk or directory is accessed. If this field is filled with dashes, the minidisk or directory is not accessed.

### Stat

is the status of the minidisk or directory: R/O (read-only) or R/W (read/write). If this field is filled with dashes, the minidisk or directory is not accessed.

### Vdev

is the virtual device number of a minidisk, or 'DIR', if the entry is an SFS directory. If the directory has the directory control (DIRCONTROL) attribute, 'DIRC' is displayed instead of 'DIR'.

### Label

is the label assigned to a formatted CMS disk. If the entry is an OS or DOS disk, this is the volume label.

### (OwnerID Odev : Cyl/%Used)

is additional disk information that is provided with minidisk entries only.

Disk information is presented using this format:

```
ownerID odev : nnnn/pp
```

where:

#### **ownerID**

identifies the user ID that owns the listed disk.

#### **odev**

is the owner's virtual device number for this disk.

#### **nnnn**

is the number of cylinders available on the disk.

#### **pp**

is the percentage of disk blocks in use.

### SFS Directory Name

is the complete name of an SFS directory.

## Input and Output Files

### Input Files

#### **ppfname PPF**

The usable form product parameter file.

### Input/Output Files

#### **SETUP \$LINKS**

The addresses linked by VMFSETUP.

### PPF Tags Used

#### **:DCL**

Used to identify the beginning of the Declaration section where the addresses to be linked are defined.

#### **:MDA**

Identifies the section that lists the disks or directories that need to be accessed.

## Messages and Return Codes

Appendix D, “Module Identifiers for VMSES/E Messages,” on page 749 shows the format of VMSES/E messages and lists the identifiers used by each VMSES/E EXEC.

For a complete explanation of any VMSES/E message, use the HELP Facility to view the message explanation online or see the appropriate messages documentation. To display information on a specific message, for example VMF002E, enter:

```
help msg vmf002e
```

If you are unfamiliar with the z/VM HELP Facility, enter help to display the main HELP Menu. For information on the HELP command, enter:

```
help cms help
```

The VMFSETUP EXEC issues the following return codes:

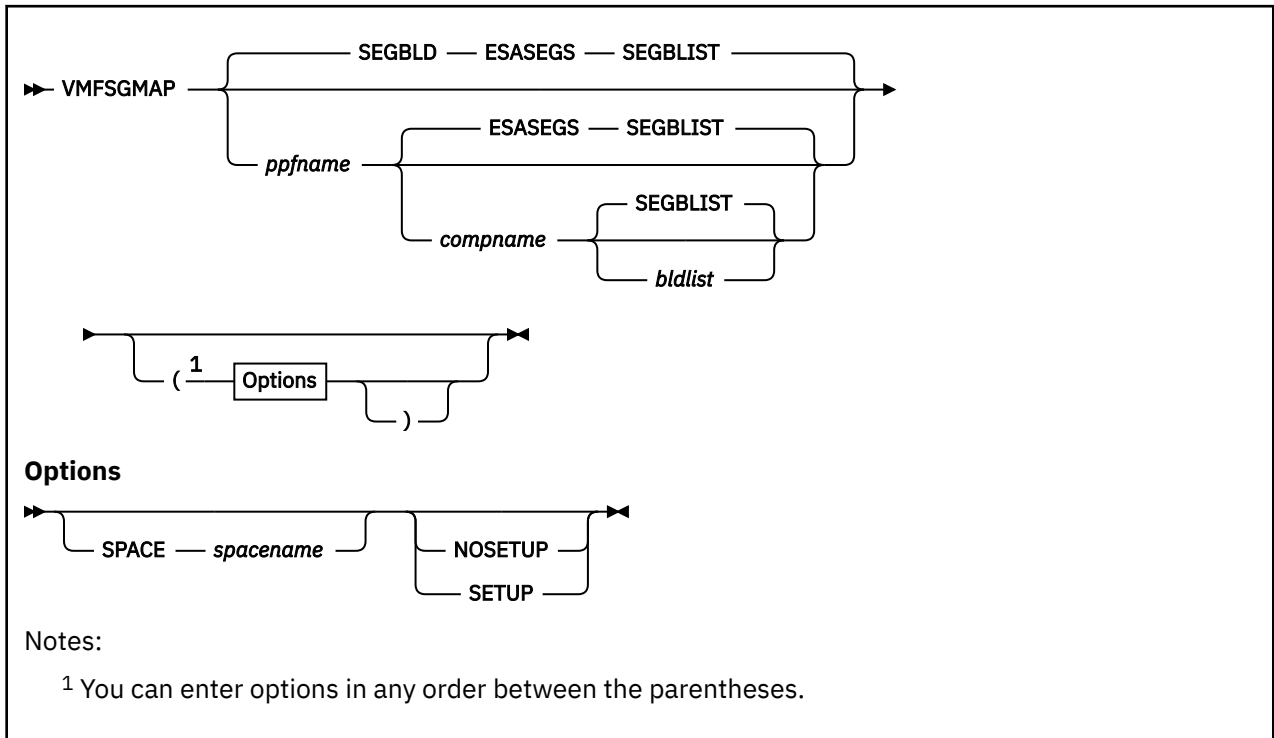
Return Code	Explanation
0	Command completed successfully.
4	Command completed with one or more warning conditions.
8	Command completed but at least one major process failed.
12	Command failed because of an internal error.
24	Command failed because of a command line syntax error.
28	Command failed because a required file was not found.
36	Command failed because a target disk or directory was not available.
100	Command failed because of an external error.

## Recovery Information

VMFSETUP can be restarted after correcting any error situation without doing any cleanup. However if you wish to return to your original access order, you can do so as follows:

- If the NOLINK option has been specified on all invocations of VMFSETUP, simply IPL CMS.
- If the LINK or LINKRR option has been specified (or LINK has been used by default) on a previous invocation of VMFSETUP, you need to detach the links. This can be done by invoking VMFSETUP and specifying the DETACH keyword. To bring back the original access order (from when you logged onto this session), IPL CMS. You can check the file SETUP \$LINKS to verify whether any links were done by VMFSETUP. If any component name is listed with a corresponding address, the address was linked by VMFSETUP. **\*\*BASE\*\*** indicates the link existed prior to invoking VMFSETUP.

## VMFSGMAP EXEC



### Purpose

Use the VMFSGMAP EXEC to process and display the saved segment information defined in the saved segment data file associated with a specified system saved segment build list. The display also includes information about saved segments and saved systems that are currently defined on the system (in system data files) but are not defined in the saved segment data file.

VMFSGMAP provides a panel interface that:

- Shows all segment spaces, member saved segments, discontinuous saved segments, and saved systems
- Shows gaps in segment spaces
- Shows overlapping member saved segments
- Detects ranges that are not valid in segment spaces
- Allows you to change, add, and delete saved segment definitions
- Saves the display in a file
- Updates the Software Inventory files

### Operands

#### SEGBLD

#### *ppfname*

is the file name of the product parameter file used for mapping and building saved segments. The default file name is SEGBLD. The file type must be PPF.

#### ESASEGS

#### *compname*

is the name of the system to be processed in the saved segment product parameter file. The default is ESASEGS.



**SEGBLIST*****bldlist***

is the file name of the system saved segment build list. This is also the file name of the saved segment data file to be processed. The default file name is SEGBLIST.

The file type of the build list must be EXC*nnnn*. (If the file is located on a base disk, the file type may be EXEC.) The file type of the data file must be SEGDATA. The default is SEGBLIST.

**Options****SPACE**

indicates you want to view a particular segment space. Only the specified space and its members are displayed, along with any other spaces in which members of the specified space also reside.

***spacename***

is the name of the segment space you want to view.

**SETUP**

sets up a minidisk/directory access order according to entries defined in the :MDA section of the product parameter file.

**NOSETUP**

does not set up a new access order.

**Usage Notes**

1. VMFSGMAP does not support segments defined above 2 GB.
2. Saved systems are not defined in the system saved segment build list or the SEGDATA file. Although VMFSGMAP displays saved systems currently defined on the system, the information is supplied for planning purposes only. You cannot use VMFSGMAP to add, change, or delete saved systems.
3. If the system saved segment build list does not exist, VMFSGMAP creates it on the build disk defined in the saved segment product parameter file.
4. VMFSGMAP processes the first available copy of the SEGDATA file. If the SEGDATA file does not exist, VMFSGMAP creates it on the same disk where the system saved segment build list resides.
5. If you specify the SPACE option with the name of a segment space that does not exist, VMFSGMAP displays the complete segment map.
6. If the SETUP/NOSETUP option is omitted, VMFSGMAP uses the value of the :SETUP tag in the product parameter file to determine whether to set up a new access order.
7. VMFSGMAP does not commit any changes until you press PF6 (Save) or PF5 (File) from the Segment Map panel.
8. VMFSGMAP provides several subcommands to help you use the VMFSGMAP panels. For example, you can use the VIEW subcommand to change what is being displayed on the panel.
9. VMFSGMAP creates a *segblst* DEL*lvlid* build list to identify the VMFBLD segments to delete. If you erase this file, VMFBLD does not detect any segments to be deleted. You can recreate this file by running VMFSGMAP and deleting the segments again.

**Panels**

VMFSGMAP displays the following types of panels:

- Segment Map panel
- Segment Definition panel
- Help panels

**Segment Map Panel**



**Note:** To see the system definition, place the cursor on the segment name and press PF2/14 (Chk Obj).

**E**

Error — The system definition or the SEGDATA definition is in error.

To view only the segment definitions in error, enter this subcommand on the command line:

```
view error
```

If a member saved segment is defined on the system but does not have an associated segment space, the following record is appended to the end of the map:

```
E segname MEM
```

If a segment space is defined on the system but has no members, the following record is appended to the end of the map:

```
E segname SPA NOMEMBERS
```

To see the system definition for a saved segment, place the cursor on the segment name and press:

```
PF2/14 (Chk Obj)
```

To purge a system data file causing an error, enter:

```
#cp purge nss spoolid
```

or:

```
#cp purge nss assoc spoolid
```

Use the second command if you are purging a member of a segment space or the entire page.

To see the SEGDATA definition for a saved segment, press PF4/16 from the Segment Map panel. Errors are highlighted on the Change Segment Definition panel. Correct the errors and return to the map by pressing PF5/17.

**M**

Mapped — The saved segment or saved system is defined on the system but not in the SEGDATA file.

**Note:** Saved systems are not defined in the SEGDATA file. Therefore, all saved systems have a status of 'M'.

**P**

Planned — The saved segment is defined in the SEGDATA file but not on the system.

**objname**

is the name of the saved segment or saved system.

**typ**

is an abbreviation for the type of saved segment or saved system. The possible types are:

**CPD**

CP system service saved segment

**DCS**

Discontiguous saved segment (DCSS)

**MEM**

Member of a segment space

**SPA**

Segment space

**SYS**

Saved system

**mmmmmmmm...**

is a 64-character storage mask that shows a 4 MB range of storage and the amount of storage used by the saved segment or saved system. Each mask character represents 64 KB of storage; 16 characters span a complete megabyte. The type of mask character indicates how the 64 KB is used. The possible storage mask characters are:

- Unused (if the first 64 KB in a megabyte range is unused, the dot (.) is replaced by a hexadecimal count character)
- R** Read-only pages
- W** Read/write pages
- N** Read/write pages, no data saved
- C** CP writable pages
- > Continued in another 4MB range
- = Range of a segment space actually used
- Padding (allocated but empty storage) to a megabyte boundary
- X** Overlapping member saved segment range

**0 1 2 3 ...**

are guide markers to help you locate the megabyte boundaries when there are many entries in a particular storage range.

**Map Record for a Deleted Saved Segment:** If a saved segment is deleted (marked 'DELETED' in the DEFPARMS field of the definition in the SEGDATA file), a record with the following format is appended to the end of the map:

```
s objname DCS DELETED
```

If the deleted saved segment is defined on the system, the status code in this record is 'D'. If the deleted saved segment is defined only in the SEGDATA file, the status code in this record is 'P'.

**Segment Map Panel with Spool File Classes**

You can also display spool file classes on the Segment Map panel by pressing PF12/24 (Class).

[Figure 163 on page 511](#) shows the format of the panel with spool file classes displayed.



Table 22. Program Function (PF) Keys on the VMFSGMAP EXEC Segment Map Panel (continued)

PF Keys	Function	Explanation
PF2/14	Chk Obj	<p>Uses the object name from the map record where the cursor is located to check the system definition against the definition in the SEGDATA file. If there are differences, a message is issued and the status code is changed to 'D'. PF2/14 also displays the Query NSS Map Window.</p> <p><b>Note:</b></p> <ol style="list-style-type: none"> <li>1. If PF2/14 is processing a segment space or a saved system, only the output from Query NSS MAP is displayed.</li> <li>2. If the cursor is not on an object name, all segments defined on the system are displayed.</li> </ol>
PF3/15	Exit	Exits from VMFSGMAP. If you made any changes since the last save (PF6/18), you receive a prompt. Enter 1 to discard the changes and exit from VMFSGMAP. Enter 0 to continue processing the Segment Map panel.
PF4/16	Chg Obj	<p>Displays the Change Segment Definition panel for the map record where the cursor is located.</p> <p><b>Note:</b></p> <ol style="list-style-type: none"> <li>1. You cannot use PF4/16 to select a saved system.</li> <li>2. If the selected object is a segment space all the members of the space are displayed.</li> <li>3. If the selected object has a status code of 'M' the status is changed in the refreshed map after you complete your changes to the definition. If you modify the DEFPARMS or SPACE information the status code is changed to 'D'. If you do not modify these fields the status code is changed to blank. When you save or file the map the definition is added to the SEGDATA file and an entry is added to the system saved segment build list.</li> </ol>
PF5/17	File	Records the changes in the SEGDATA file and exits from VMFSGMAP. If any definitions have been added, entries are also added to the build list.
PF6/18	Save	Records the changes in the SEGDATA file and remains in the map. If any saved segments have been added, they are added to the build list.
PF7/19	Bkwd	Scrolls up one page in the file.
PF8/20	Fwd	Scrolls down one page in the file.
PF9/21	Retrieve	Retrieves the last invoked command from the buffer.
PF10/22	Add Obj	<p>Displays the Add Segment Definition panel. If the cursor is on a map record when you press PF10/22, the panel contains the definition for that object. If the cursor is not on a map record when you press PF10/22, the panel contains a skeleton definition.</p> <p><b>Note:</b></p> <ol style="list-style-type: none"> <li>1. You cannot use PF10/22 to select a segment space or a saved system.</li> <li>2. When you complete the definition and return to the map, a record with a status code of 'P' is added to the refreshed map. When you save or file the map, the definition is added to the SEGDATA file and an entry is added to the system saved segment build list.</li> </ol>

Table 22. Program Function (PF) Keys on the VMFSGMAP EXEC Segment Map Panel (continued)

PF Keys	Function	Explanation
PF11/23	Del Obj	<p>Deletes the map record where the cursor is located. The panel is refreshed to show the new mapping.</p> <p><b>Note:</b></p> <ol style="list-style-type: none"> <li>1. You cannot use PF11/23 to delete a saved system.</li> <li>2. If the deleted object is a segment space the name of the space is deleted from the definition for each member of the space. If a member is not contained in any other space the member is also deleted.  If the VMFSGMAP command was entered with the SPACE option to view only this segment space and its members, the refreshed map shows all the saved segments and saved systems currently defined in the SEGDATA file and on the system.</li> <li>3. If the deleted object is a member of a segment space, the member is deleted only from the space that immediately precedes it in the map. The cursor then moves to the next occurrence of the member in the map. If the deleted member is the only member of a space, the space is deleted.</li> <li>4. The definition record for a deleted saved segment is not actually deleted from the SEGDATA file. The data in the DEFPARMS field of the definition is appended to the keyword 'DELETED'. The entry in the build list is removed.</li> <li>5. If a member saved segment or DCSS being deleted is defined only on the system (status code 'M'), the object is added to the SEGDATA file and the build list first, then marked 'DELETED' in the SEGDATA file.</li> <li>6. If every occurrence of a member saved segment is deleted, the member becomes a deleted DCSS.</li> <li>7. If the deleted member saved segment or DCSS is defined on the system, the following record is appended to the end of the map: <ul style="list-style-type: none"> <li>D <i>objname</i> DCS DELETED</li> </ul> </li> <li>8. If the deleted member saved segment or DCSS is defined only in the SEGDATA file, the following record is appended to the end of the map: <ul style="list-style-type: none"> <li>P <i>objname</i> DCS DELETED</li> </ul> </li> <li>9. You can retrieve a deleted saved segment by moving the cursor to the record at the end of the map, pressing PF4/16 or PF10/22 to view the definition, and removing the 'DELETED' keyword in the DEFPARMS field.</li> </ol>
PF12/24	Status/Class	Controls the display in column 1. You can either display the current status code or the current spool file class for the objects. PF12/24 is a toggle key. If the panel currently displays the status code, pressing PF12/24 changes the panel to display the spool file class. If the panel currently displays the spool file class, pressing PF12/24 changes the panel to display the status code.

### Segment Definition Panel

The Segment Definition panel displays one or more saved segment definition records. There are two versions of the panel, one to change a definition and one to add a definition. [Figure 164 on page 514](#) shows the format of the panel. Data is not required in all of the fields of the definition.

For information about the content and syntax of each field, see [Saved Segment Definition Record](#).

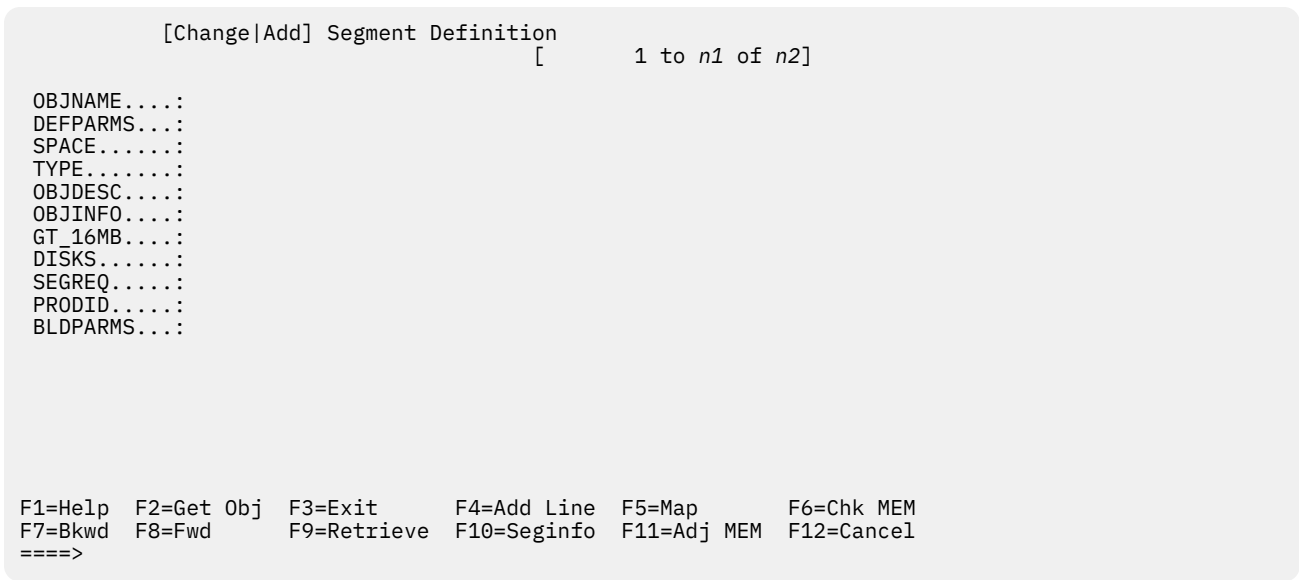


Figure 164. VMFSGMAP Segment Definition Panel

**Change Segment Definition Panel:** Pressing PF4/16 (Chg Obj) on the Segment Map panel when the cursor is located on a map record displays the Change Segment Definition panel. The panel contains the definition record for the selected saved segment. You can change any part of the definition except the name. If you want to change the name of a saved segment, you must first add a definition with the new name, then delete the definition with the old name.

If the selected map record has a status code of 'M', the saved segment is defined in a system data file but not in the SEGDATA file. Therefore, the definition panel contains only the information that can be obtained from the system data file.

**Add Segment Definition Panel:** Pressing PF10/22 (Add Obj) on the Segment Map panel displays the Add Segment Definition panel, which allows you to add a definition to the SEGDATA file. (When you save or file the map, an entry is also added to the system saved segment build list.) Depending on where you place your cursor in the map before you press PF10/22, you can use an existing saved segment as a template, or you can create an entirely new definition.

If the cursor is located on a map record when you press PF10/22, and the status code is not 'M', the displayed definition is from the SEGDATA file. If the status code is 'M', the definition contains only the information that can be obtained from the system data file. In either case, the OBJNAME field contains question marks instead of a name, because you cannot add a definition for a name already in use. If the cursor is not located on a map record when you press PF10/22, the displayed definition is a skeleton.

**Saved Segment Definition Record:** A saved segment definition record contains the following fields:

**OBJNAME....:**

contains the name of the saved segment.

▶▶ OBJNAME....: — *segname* ▶▶

***segname***

is the name of the saved segment.

If question marks are displayed, you must enter a name that is not already in use for another saved segment.

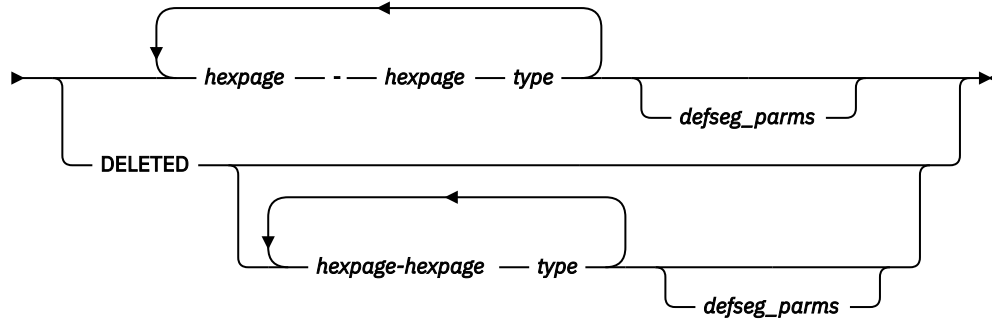
**DEFPARMS....:**

contains the storage range and other information the VMFBLD EXEC specifies on the DEFSEG command to define the saved segment to CP.

For information about the syntax and usage of the DEFSEG command, see [z/VM: CP Commands and Utilities Reference](#).



►► DEFPARMS...: →



**hexpage**

is the hexadecimal address of a page in storage.

**type**

is a page descriptor code that indicates the type of virtual machine access that is permitted to the page range.

**defseg\_parms**

are optional DEFSEG command operands. The following operands are permitted: LOADNSHR, RSTD, and SECURE.

**Note:** Do not specify the SPACE operand in this field. Segment spaces are identified in the SPACE field.

**DELETED**

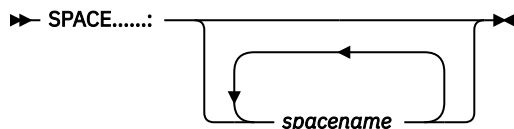
indicates the saved segment is to be deleted. No DEFSEG command is issued.

**Note:** If any DEFSEG data is specified in this field, it is saved in the SEGDATA file following the DELETED keyword.

**SPACE.....:**

lists the segment spaces in which the saved segment is a member.

**Note:** The VMFBLD EXEC uses the first name listed in this field as the primary segment space when issuing the DEFSEG command for the member.

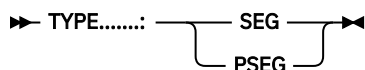


**spacename**

is the name of a segment space.

**TYPE.....:**

indicates whether the saved segment contains CMS logical saved segments.



**SEG**

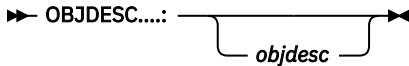
indicates the saved segment does not contain logical saved segments. One set of build parameters is allowed in the BLDPARMS field.

**PSEG**

indicates the saved segment is a physical saved segment that contains logical saved segments. Multiple sets of build parameters are allowed in the BLDPARMS field, each set defining one or more logical saved segments.

**OBJDESC.....:**

contains a description of the saved segment.

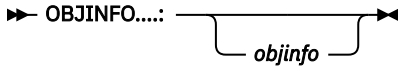


**objdesc**

is free-format text that describes the saved segment.

**OBJINFO.....:**

contains special installation information.

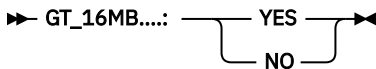


**objinfo**

is free-format text that describes any special requirements for installing the saved segment. This field is informational; its content does not affect the generation of the saved segment.

**GT\_16MB.....:**

indicates whether the saved segment can be loaded above the 16MB line.



**YES**

the saved segment can be loaded above 16MB.

**Note:** This does not mean the range specified in the DEFPARMS field actually defines the saved segment above 16MB, only that it can.

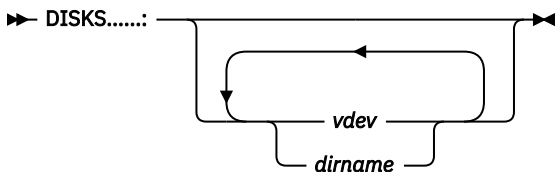
**NO**

the saved segment cannot be loaded above 16MB. The range specified in the DEFPARMS field must be less than 16MB.

**DISKS.....:**

lists the minidisks and SFS directories to be accessed by the VMFBLD EXEC when building the saved segment.

**Note:** If 'PPF' is specified in the BLDPARMS field, the minidisks and directories specified in this field are accessed before the minidisks and directories defined in the product parameter file. If 'UNKNOWN' is specified in the BLDPARMS field, the minidisks and directories specified in this field are not accessed.



**vdev**

is a minidisk virtual device number.

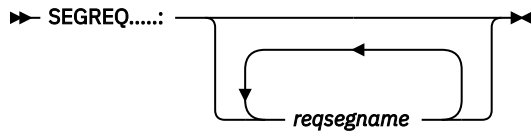
**dirname**

is a fully qualified SFS directory name.

**SEGREQ.....:**

lists the saved segments that must be built before this saved segment is built.

**Note:** All requisite saved segments must be defined in the same system saved segment build list and SEGDATA file as this saved segment.

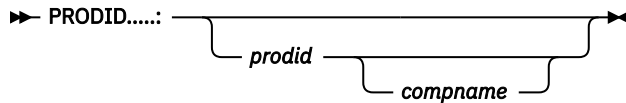


**reqsegname**

is the name of a requisite saved segment.

**PRODID.....:**

identifies the product parts (PRODPART) file that contains the default definition for the saved segment. This information is required for the SEGINFO function (PF10/22) on the Segment Definition panel.



**prodid**

is the 7 to 8 character alphanumeric identifier assigned to the product.

**compname**

is the 1 to 16 character component name.

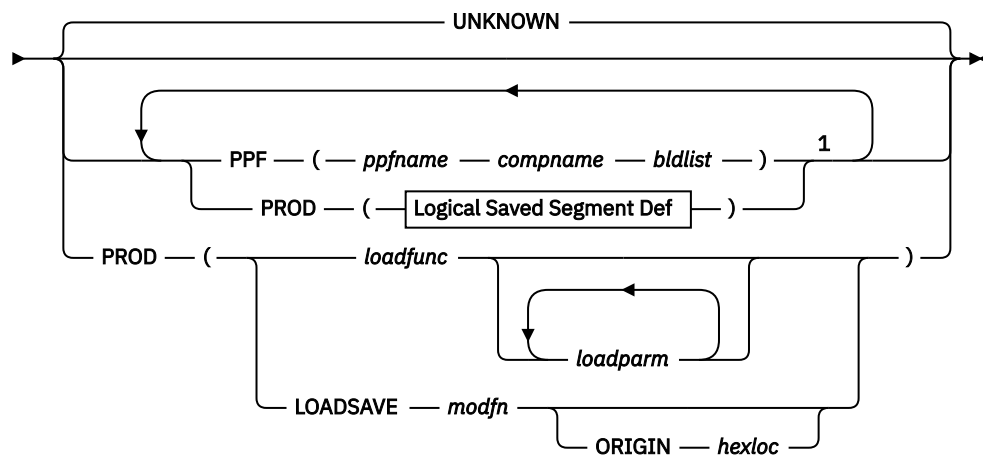
**Note:** The component name is necessary only if the PRODPART file contains saved segment definitions for more than one component. If the PRODPART file contains more than one component, and no component name is specified in this field, VMFSGMAP gets information from the first *segname* definition it finds. If you are using the SEGINFO function to obtain all the saved segment definitions for a product, and no component name is specified in this field, VMFSGMAP gets information from the first component that has saved segment definitions.

**BLDPARMS...:**

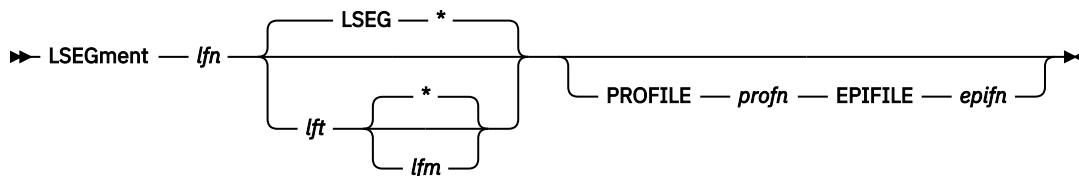
identifies the parameters the VMFBLD EXEC uses to build the saved segment.

If 'PSEG' is specified in the TYPE field, the VMFBLD EXEC creates a physical segment definition file (*segname* PSEG) that contains the logical segment records and then issues the SEGEN command to build the physical and logical saved segments. You can specify multiple sets of build parameters (for VMSES/E and non-VMSES/E products) in this field to identify the logical saved segments to be included in the physical saved segment.

▶▶ BLDPARMS...: →



**Logical Saved Segment Def**



Notes:

<sup>1</sup> This loop is valid only for a physical saved segment that contains CMS logical saved segments. 'PSEG' must be specified in the TYPE field.

**UNKNOWN**

indicates a build function for this saved segment cannot be issued by the VMFBLD EXEC. This forces VMFBLD to issue only the DEFSEG command to define the saved segment to CP. After VMFBLD has completed, you must issue the function that actually loads and saves the saved segment. This is the default.

**PPF**

indicates the build parameters are defined in the specified product parameter file and build list. Depending on the build list option specified (ACCESS, LINK, or NOACCESS), VMFSETUP might be called to link and access the disks specified in the product parameter file for the product when the segment is built.

**ppfname**

is the name of the product parameter file.

**compname**

is the name of the component section in the product parameter file.

**bldlist**

is the name of the product saved segment build list.

**PROD**

indicates the build parameters are not specified in a product parameter file. The VMFBLD EXEC uses the parameters specified in this field to build the saved segment (after issuing the DEFSEG command to define the saved segment to CP).

**LSEGment**

identifies a logical segment definition file, which defines a CMS logical saved segment to be included in the physical saved segment.

**Note:** 'PSEG' must be specified in the TYPE field.

**lfn**

is the file name of the logical segment definition file.

**lft**

is the file type of the logical segment definition file. If the file type is not specified in this definition, the SEGGEN command uses a default of LSEG.

**Note:** Do not use PROFILE or EPIFILE as the file type of the logical segment definition file.

**lfn**

is the file mode of the logical segment definition file. If the file mode is not specified in this definition, the SEGGEN command uses a default of \*.

**PROFILE**

indicates a profile routine is to be run before information is loaded into the logical saved segment. For information about using a profile when building a logical saved segment, see [z/VM: CP Planning and Administration](#).

**profn**

is the file name of the profile routine.

**EPIFILE**

indicates an epifile routine is to be run after information is loaded into the logical saved segment. For information about using an epifile when building a logical saved segment, see [z/VM: CP Planning and Administration](#).

**epifn**

is the file name of the epifile routine.

**loadfunc**

is the name of the routine the VMFBLD EXEC calls to load and save the saved segment.

**loadparm**

is a parameter to be passed to the *loadfunc* routine.

The following built-in variables are also available to indicate data to be passed to the *loadfunc* routine. When the saved segment is built, the VMFBLD EXEC resolves the variables as indicated:

**&RANGE**

Gets the hexadecimal page ranges, page descriptor codes, and optional DEFSEG operands from the DEFPARMS field

**&SPACE**

Gets the primary segment space name from the SPACE field

**&ORIGIN**

Gets the starting load address from the DEFPARMS field

**&SEGNAME**

Gets the name from the OBJNAME field

**LOADSAVE**

indicates the VMFBLD EXEC calls the built-in LOADSAVE function, which issues the LOADMOD command to load a specified relocatable module. LOADSAVE then issues the SAVESEG command to save the module as a saved segment.

See the usage notes on the LOADMOD command in [z/VM: CMS Commands and Utilities Reference](#).

**modfn**

is the file name of the relocatable module. The file type must be MODULE.

**ORIGIN**

specifies a load address for the module.

**hexloc**

is the hexadecimal storage address where the module is to be loaded.

**Note:** It is recommended that you specify the ORIGIN keyword and that you use the built-in variable &ORIGIN to indicate the storage location. When the saved segment is built, the VMFBLD EXEC resolves the &ORIGIN variable to get the load address from the DEFPARMS field. If the ORIGIN keyword is not specified, CMS selects any available storage location. If you specify an actual load address, it must be within the range defined in the DEFPARMS field.

**PF Keys on the Segment Definition Panel:** [Table 23 on page 519](#) describes the functions of the PF keys on the Segment Definition panel.

Table 23. Program Function (PF) Keys on the VMFSGMAP EXEC Segment Definition Panel

PF Keys	Function	Explanation
PF1/13	Help	Displays a help panel that describes the content and meaning of the Segment Definition panel, or a particular field. You can ask for help on a particular field by placing the cursor on the field and pressing PF1/13. When the cursor is located in any other area of the panel, pressing PF1/13 displays general help on the entire panel.
PF2/14	Get Obj	Gets the definition from the SEGDATA file and overlays the information on the panel and then gets information from the system data file to update the DEFPARMS and SPACE fields.

Table 23. Program Function (PF) Keys on the VMFSGMAP EXEC Segment Definition Panel (continued)

PF Keys	Function	Explanation
PF3/15	Exit	Exits from VMFSGMAP. If you made any changes since the last save, you receive a prompt. Enter 1 to discard the changes and exit from VMFSGMAP. Enter 0 to continue processing the Segment Definition panel.
PF4/16	Add Line	Adds a blank line following the line where the cursor is located. This allows you to add data to the field that precedes the blank line. <b>Note:</b> You cannot add lines to the OBJNAME, TYPE, GT_16MB, and PRODID fields.
PF5/17	Map	Returns to the segment map. All changes to the definition are kept, and the map is refreshed with the new data.
PF6/18	Chk Mem	Checks for overlapping members in all segment spaces that contain the member specified in the OBJNAME field. Definitions for all overlapping members are added to the panel.
PF7/19	Bkwd	Scrolls up one page in the file.
PF8/20	Fwd	Scrolls down one page in the file.
PF9/21	Retrieve	Retrieves the last invoked command from the buffer.
PF10/22	Seginfo	Gets default saved segment information from the PRODPART file identified in the PRODID field. If a component name is not specified in the PRODID field, information is obtained from the first component in the file that has saved segment definitions.  If the OBJNAME field contains a name when you press PF10/22, default information is obtained for that saved segment only and used to update the definition on the panel. If the OBJNAME field is empty or contains question marks when you press PF10/22, default information is obtained for all the saved segments defined for the component, and definitions for those saved segments are added to the panel.
PF11/23	Adj Mem	Checks for overlapping members in all segment spaces that contain the member specified in the OBJNAME field. The ranges of the members are adjusted where necessary to eliminate the overlaps. Definition records for all adjusted members are added to the panel. <b>Note:</b> Each member is adjusted downward or upward according to which direction requires the least total amount of movement.
PF12/24	Cancel	Returns to the segment map. If you made any changes on this panel, you receive a prompt. Enter 1 to discard the changes and return to the segment map panel. Enter 0 to continue processing the Segment Definition panel.

### Query NSS Map Window

The Query NSS Map window displays the current system definitions for a segment or segment space. The output is in the same format as the output returned by the CP QUERY NSS MAP command.

Figure 165 on page 521 shows the format of the Query NSS Map panel that is displayed when you press PF2/14 of the Segment Map panel for the CMSPIPES segment.



**VIEW**

Controls which segments are displayed in the Segment Map panel. Enter the VIEW subcommand on the command line from the VMFSGMAP Segment Map panels to change the set of segments displayed.

When you issue a VIEW subcommand, the same view is displayed until you issue a new VIEW subcommand or until there are no more segments in the specified view. For example, when you issue:

```
view error
```

and fix all of the segments in error, the view is reset to display all segments.

**ALL**

displays the segments defined in the SEGDATA file and the segments defined only on the system. ALL is the default.

**ERROR**

displays only segments with a status code of E.

**SEGDATA**

displays all segments defined in the SEGDATA file. Segments defined only on the system are not displayed in the map.

**Examples**

This section shows examples of the VMFSGMAP Segment Map panel and Change Segment Definition panel filled in with data. However, for guidance information on using the VMFSGMAP EXEC to do various saved segment management tasks, such as adding a saved segment, changing the range of a saved segment, and so on, see *z/VM: CP Planning and Administration*.

**Displaying the Segment Map:** To process system saved segment build list SEGBLIST EXC00000 and segment data file SEGBLIST SEGDATA, using control information defined in the ESASEGS section of saved segment product parameter file SEGBLD PPF, enter:

```
vmfsgmap segbld esasegs segblist
```

VMFSGMAP begins building the segment map, and issues the following message:

```
VMFSGM2034I Building segment map
```

When the build is complete, VMFSGMAP displays the Segment Map panel.

[Figure 166 on page 523](#) shows an example.



```

VMFSGMAP - Segment Map
More: + -
1 to 21 of 39

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

Meg          004-MB          005-MB          006-MB          007-MB
St Name      Typ 0123456789ABCDEF0123456789ABCDEF0123456789ABCDEF0123456789ABCDEF
M GCS        SYS RRRRNNNNNNNNNNNNNNNNNNNNNNNNNNNNNN6.....7.....

Meg          008-MB          009-MB          00A-MB          00B-MB
St Name      Typ 0123456789ABCDEF0123456789ABCDEF0123456789ABCDEF0123456789ABCDEF
M DOSBAM     SPA 8.....9.....A.....=====
M CMSDOS     MEM 8.....9.....A.....R.....
M CMSBAM     MEM 8.....9.....A.....RRRR.....
M DOSINST    DCS 8.....R-----A.....B.....

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

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 166. Example of the VMFSGMAP Segment Map Panel

The first map record in Figure 166 on page 523 is for the GCS saved segment. This record indicates GCS is mapped (defined on the system but not in the SEGDATA file) and that it is a named saved system located partly in the X'000'MB block of storage and in the X'004' to X'006'MB blocks of storage.

**Displaying a Segment Definition Record:** Moving the cursor to HELPINST in the map and pressing PF4/16 displays the definition record for HELPINST in the Change Segment Definition panel, as shown in Figure 167 on page 523.

```

Change Segment Definition

OBJNAME....: HELPINST
DEFPARMS...: C00-CFF SR
SPACE.....:
TYPE.....: PSEG
OBJDESC....: CMSINST AND HELP LSEGS
OBJINFO....:
GT_16MB....: NO
DISKS.....:
SEGREQ....:
PRODID.....: 1VMVMC23 MYCOMP
BLDPARMS...: PPF(SERV2P MYCOMP DMSSBINS) PPF(SERV2P MYCOMP DMSSBHLP)

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

```

Figure 167. Example of the VMFSGMAP Change Segment Definition Panel

The definition indicates the following additional information about HELPINST:

- It occupies pages X'C00-CFF' and the permitted access is shared read-only.
- It is not a member of any segment spaces.
- It is a physical saved segment that contains at least one logical saved segment.
- It may not be defined above the 16MB line.

- There are no additional minidisks or SFS directories that need to be accessed to build the saved segment.
- There are no requisite saved segments.
- The default information is defined in the MYCOMP section of the 1VMVMC23 PRODPART file.
- The build information is supplied in the MYCOMP section of the SERVP2P PPF file and in product saved segment build lists DMSSBINS and DMSSBHLP.

## **Input and Output Files**

### **Input Files**

#### ***ppfname* PPF**

Usable form product parameter file used for saved segment mapping and building.

#### ***prodid* PRODPART**

The PRODPART file shipped with the product (used only by the SEGINFO function on the Segment Definition panel).

### **Input/Output Files**

#### ***bldlist* EXCnnnnn**

The system saved segment build list.

#### ***bldlist* SEGDATA**

The saved segment data file.

### **Output Files**

#### ***appid* \$SELECT**

The file where VMFSGMAP indicates the saved segments that need to be built.

#### ***fn* SEGMAP**

Optional saved copy of the current segment map, generated by the SAVEMAP subcommand.

## **Messages and Return Codes**

For a complete explanation of any VMSES/E message, use the HELP Facility to view the message explanation online or see the appropriate messages documentation. To display information on a specific message, for example VMF002E, enter:

```
help msg vmf002e
```

If you are unfamiliar with the z/VM HELP Facility, enter help to display the main HELP Menu. For information on the HELP command, enter:

```
help cms help
```

Appendix D, “Module Identifiers for VMSES/E Messages,” on page 749 shows the format of VMSES/E messages and lists the identifier for each VMSES/E exec.

VMFSGMAP issues the following return codes:

<b>Return Code</b>	<b>Explanation</b>
0	Command completed successfully.
4	Command completed with one or more warning conditions.
100	Command failed because of an external error.

## **Recovery Information**

The VMFSGMAP command can be restarted by issuing the command again.

## VMFSIM EXEC

### Purpose

The VMFSIM EXEC is the interface to the Software Inventory. With the VMFSIM EXEC, you can:

- Compare the control/AUX file structure to the version vector tables (“[VMFSIM CHKLVL](#)” on page 531)
- Compare two Software Inventory tables (“[VMFSIM COMPTBL](#)” on page 538)
- Identify the latest level of a part (“[VMFSIM GETLVL](#)” on page 543)
- Initialize the Software Inventory (“[VMFSIM INIT](#)” on page 550)
- Support your local modification structure (“[VMFSIM LOGMOD](#)” on page 556)
- Update the Software Inventory (“[VMFSIM MODIFY](#)” on page 562)
- Query the Software Inventory (“[VMFSIM QUERY](#)” on page 567)
- List the dependent PTFs for a given PTF (“[VMFSIM SRVDEP](#)” on page 573)
- List the requisite PTFs for a given PTF (“[VMFSIM SRVREQ](#)” on page 579)
- List the dependent products for a given product (“[VMFSIM SYSDEP](#)” on page 585)
- List the requisite products for a given product (“[VMFSIM SYSREQ](#)” on page 591)

### Structure of the Data in the SI Tables

Before you use the VMFSIM EXEC, however, you need to be familiar with the structure of the data in the Software Inventory tables. Tables in the Software Inventory are made up of logical records. A logical record consists of all the fields between one key field and the next. A key field identifies the major grouping of information contained in each logical record. The data in each key field is unique throughout a table. A field consists of everything between one tag and the next, even if it spans multiple lines. The name of the field is the same as the tag in the field.

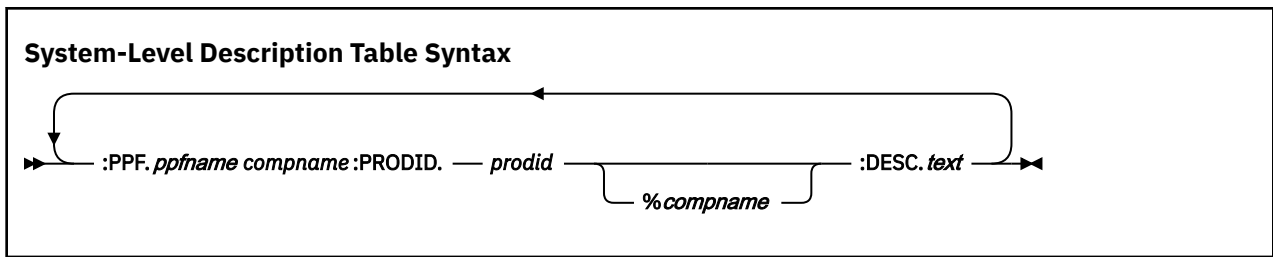
A tag is an alphanumeric string that starts with a colon (:), and ends with a period (.). The tag identifies the nature of the data following the tag.

[Figure 168 on page 525](#) illustrates the data structure.



*Figure 168. Software Inventory Data Structure*

For example, in the system-level description table, shown in “[System-Level Description Table Syntax](#)” on page 526, there are three fields, PPF, PRODID, and DESC. PPF is the key field, consisting of the tag :PPF and its data *ppfname compname*. The PRODID field consists of the tag :PRODID and its data *prodid%compname*. The percent sign (%) is a delimiter. The DESC field consists of the tag :DESC and its data *text*. Each logical record in the table begins with the key field PPF.



## Providing Input to VMFSIM

VMFSIM uses tagged data (TDATA) statements as input. There are three ways to enter TDATA statements for VMFSIM commands:

- From the command line
- In a REXX stem
- From a file

A TDATA statement begins with the keyword TDATA. The TDATA keyword is followed by one or more tags that correspond to fields in the Software Inventory tables and the data for the tags.

The syntax of a TDATA statement is:



The keyword TDATA precedes each set of tags and data to be processed. You must specify a tag after the TDATA keyword. You can also specify data after the tag.

## Receiving Output from VMFSIM

Output TDATA statements are returned from VMFSIM to the terminal display, a file, or a REXX stem.

## VMFSIM: Tagged Data (TDATA)

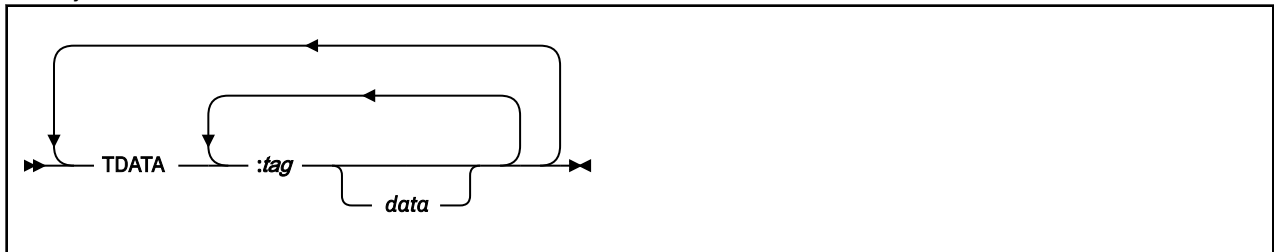
The VMFSIM functions process the tags and data that are entered on the command line or contained in a file or stem according to some basic rules. A tag is the name of a field that is defined for a table. Tags begin with a colon (:) prefix. The rules for processing tags and data are:

- All tags specified in the TDATA statement are treated as return fields. That is, all data contained on the tag is returned to the caller.
- If a tag that corresponds to a parent field is specified as a return field, but none of its sub fields are specified, all sub fields are returned.
- If a tag is specified with data in the TDATA statement, VMFSIM treats the data as search arguments. VMFSIM searches each record in the table for the specified search arguments. If a match is found, the record is processed by the appropriate VMFSIM function. For example, the QUERY function returns to the caller all fields specified in the TDATA statement for the record found.
- VMFSIM treats statements with more than one search argument as if an "AND" condition were specified, and all search arguments must match.
- Duplicate tags or tags not defined for the table being processed are ignored.

### Using File Input

The FILE parameter on the VMFSIM command allows you to input a collection of TDATA statements. The default file name for files containing a collection of TDATA statements is SIMDATA.

The syntax of a TDATA statement in a file is:



For example, the following file (Q1 SIMDATA A) contains two TDATA statements:

```
TDATA :PTF UV12345 :PREREQ
TDATA :PTF UV23456 :COREQ
```

Suppose you issue the following query against Q1 SIMDATA A:

```
VMFSIM QUERY 1VMVMC23 SRVREQT * FILE Q1
```

The QUERY function returns this response:

```
Results for TDATA statement
TDATA :PTF UV12345 :PREREQ

:PTF UV12345
  :PREREQ UV23456 UV56789

Results for TDATA statement
TDATA :PTF UV23456 :COREQ

:PTF UV23456
  :COREQ UV00001 UV00002
```

### Querying Multiple Tables Using the File Option

You can use the results of one query as input to a second query by specifying the FILE parameter and FILE option. The results are in the same format as the required input.

In the file, each group of records that is separated by the keyword TDATA is considered one collection of tags or data for the query.

The following example shows a multiple table query to find all of the PTFs associated with an APAR and the modules that were serviced by the PTFs.

- Step 1: Find all the PTFs that contain a given APAR and FILE the results.

Enter this command:

```
VMFSIM QUERY 1VMVMC23 SRVREQT TDATA :APARNUM VM12345 :PTF ( FILE Q1
```

The following is returned in Q1 SIMDATA A:

```
TDATA
:PTF UV00001
:APARNUM VM12345
TDATA
:PTF UV00002
:APARNUM VM12345
```

- Step 2: Find all the modules that contain those PTFs that were saved in file Q1 and save the results in a file Q2.

Enter this command:

```
VMFSIM QUERY 1VMVMC23 VVTVM FILE Q1 TDATA :PART ( FILE Q2
```

The following is returned in Q2 SIMDATA A:

```
TDATA
:PART HCPABC TXT
:PTF UV00001.VM12345.P12345DK UV01234.VM00234.P00234DK
TDATA
:PART HCPXXX TXT
:PTF UV00001.VM12345.P12345DK
TDATA
:PART HCPXXX TXA
:PTF UV00001.VM12345.P12345DK
TDATA
:PART HCPXXX TXM
:PTF UV00002.VM12345.P12345DK
```

In Step 1, the query function returned two PTFs that contained the APAR specified. The results were placed in a file, Q1 SIMDATA A.

In Step 2, a VMFSIM QUERY command was issued using the file that contained the results from the first query. The return field was specified as the :PART tag from the version vector table (VVTVM). The results of the second query were placed in the file Q2 SIMDATA A.

**Note:** Because the :APARNUM tag is not a valid tag in version vector tables, it is ignored by the second query.

Using this process, where the output of the first query is input for a second query, it is possible to perform many complicated queries against the Software Inventory.

## Using the STEM Variable

### PI

The stem variable allows you to develop REXX applications that use the query function without the overhead of file input and output. The VMFSIM command supports two types of stem input and output, simple and associative stems.

## Simple Stems

Simple stem input and output uses the same format as file input and output. Each element in the stem is equivalent to a record in a file. The zero element of the stem indicates the total number of other elements in the stem.

The following example shows the use of a simple stem as input and output in a REXX exec:

```
/* REXX EXEC using VMFSIM STEM input and output */
X.0 = 2
X.1 = 'TDATA :PTF UV12345 :PREREQ'
X.2 = 'TDATA :PTF UV23456 :COREQ'
'EXEC VMFSIM QUERY 1VMVMC23 SRVREQT * STEM X. (STEM Q.)'
Do j = 1 to Q.0
  Say Q.j
End
Exit
```

When you run the exec, the following is displayed:

```
TDATA
:PTF UV12345
:PREREQ UV23456 UV56789
TDATA
:PTF UV23456
:COREQ UV00001 UV00002
```

## Associative Stems

The second type of stem input and output is called associative stem input and output. This format returns the data in a tree structure, which minimizes the amount of necessary searching when data is processed. The data is processed by VMFSIM in the same way as simple stem input and output is processed.

The following example shows the use of an associative stem input and output:

```
X.0 - is the root node of the tree. It contains:
  X.0 = nodes - the number of nodes in the tree
  X.0.TAGS = tag ptr ... - ordered pairs of key tags and pointers
                    to the keys and data entries
  X.0.0 = 0 - no data in root node

X.ptr - is a node in the tree. It contains:
  X.ptr = tag - the name of the tag at this node
  X.ptr.TAGS = tag ptr ... - ordered pairs of fields and pointers
                    to the entries associated with this TAG
  X.ptr.0 = n - number of records containing data
                    for this TAG
  X.ptr.n = data - data record for the TAG
```

When you use this query from an exec:

```
VMFSIM QUERY 1VMVMC23 SRVREQT TDATA :PTF UV00100 (ASTEM X.)
```

the results are in the ASTEM variable "X.", as you see below:

```
X.0 = 4
X.0.0 = 0
X.0.TAGS = PTF ptr1

X.ptr1 = PTF (tag name)
X.ptr1.0 = 1 (number of data)
X.ptr1.1 = UV00100 (data)
X.ptr1.TAGS = APARNUM ptr2 PREREQ ptr3 COREQ ptr4 (tag pointer)

X.ptr2 = APARNUM
X.ptr2.0 = 1
X.ptr2.1 = VM11111
X.ptr2.TAGS = ''

X.ptr3 = PREREQ
X.ptr3.0 = 1
X.ptr3.1 = UV00090
```

```
X.ptr3.TAGS = ''
X.ptr4      = COREQ
X.ptr4.0    = 1
X.ptr4.1    = UV00099
X.ptr4.TAGS = ''
```

To access the required :COREQ data, the following parsing commands are required:

```
Parse var X.0.TAGS . 'PTF' inx1 .
Parse var X.inx1.TAGS . 'COREQ' inx2 .
level = x.inx2.1
```

Figure 169 on page 530 shows the tree structure of ASTEM X.

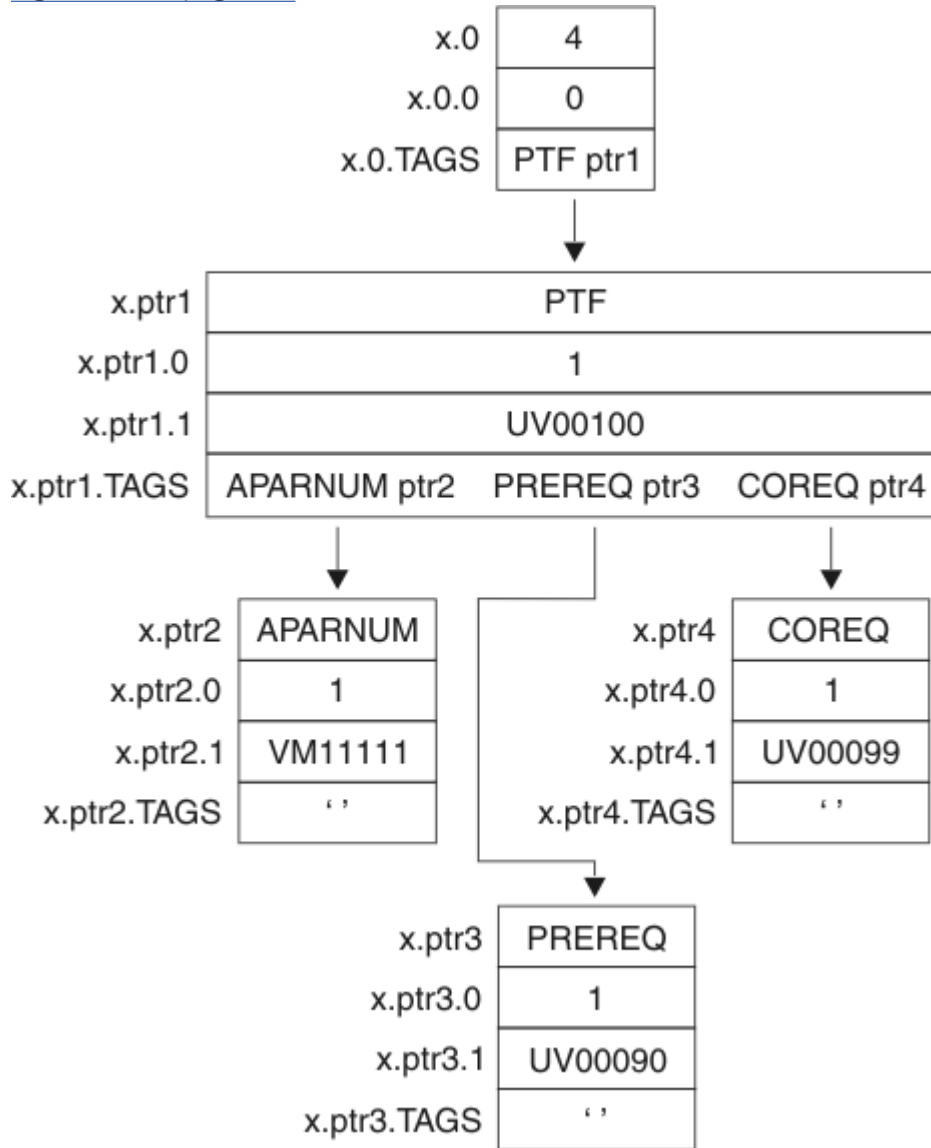


Figure 169. Tree Structure Format

## Considerations for Using Stem Input and Output

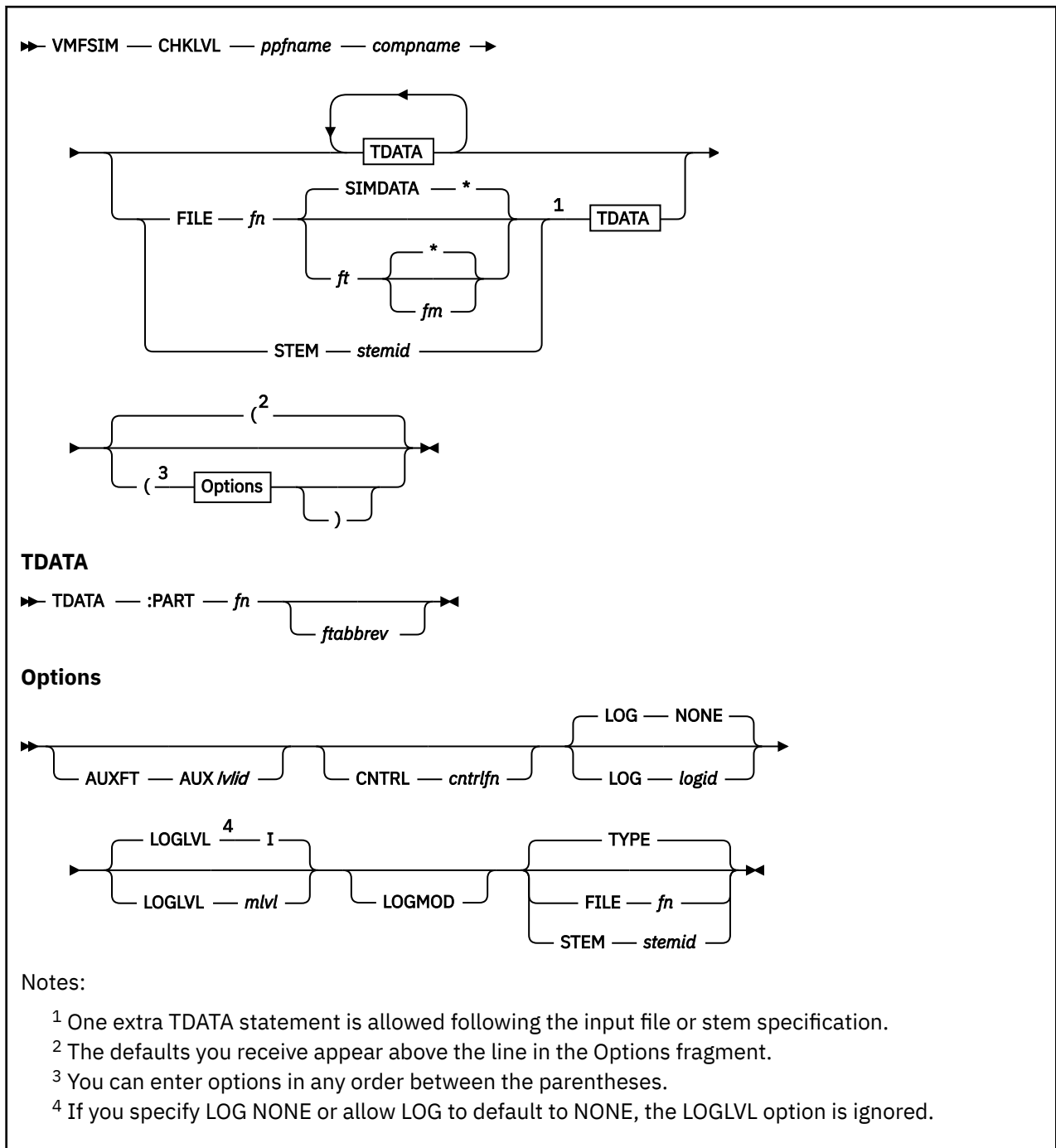
When you develop applications using stem input and output, the calling application **must**:

- Be a REXX exec
- Invoke the VMFSIM EXEC with an EXEC statement

**PI end**



# VMFSIM CHKLVL



## Purpose

The CHKLVL function verifies the version vector structure matches the corresponding AUX structure for one or more parts.

Beginning at the highest level in the control file, each version vector table is compared to the corresponding AUX file until either:

- A mismatch is found.

- All levels have been checked.

VMFSIM CHKLVL reports mismatches by providing:

- The file IDs of the files in error
- The records in error

Parts not supported by AUX files are also reported. To determine if a part is supported by AUX files, VMFSIM checks to see if there are any update files identified for that part in the version vector tables.

**Note:** If a specific AUX file is entered on the command, only that AUX file and its corresponding version vector table, are checked.

## Operands

### *ppfname*

is the file name of the usable form product parameter file. The product parameter file must have a file type of PPF.

### *compname*

is the name of the component, for example CP or CMS, as it is specified on the :COMPNAME tag in the product parameter file. *compname* is a 1-16 character alphanumeric identifier.

### TDATA

identifies the parts to be validated. You can specify additional tags, but they are ignored.

#### **:PART**

identifies the part to be validated.

You can specify additional tags, but they are ignored. A detailed description on how tag data is processed is contained in the section [“VMFSIM: Tagged Data \(TDATA\)”](#) on page 527. The only difference between how QUERY and CHKLVL process tag data is that CHKLVL does not allow return tags.

#### *fn*

is the file name of the part in the version vector table that is to be validated.

#### *ftabbrev*

is the file type of the part in the version vector table that is to be validated. The *ftabbrev* must be the 3-character PTF abbreviation or the real CMS file type for parts that are not serviced by replacement.

If you omit *ftabbrev*, all parts with the specified file name are validated.

**Note:** If you specify a TDATA statement following the FILE, STEM, or ASTEM operand, it will be added to the end of the list of PARTs or products in the file or stem defined by the FILE, STEM, or ASTEM operand.

### FILE

identifies a CMS file that lists the parts to be validated.

#### *fn*

is the file name of the CMS file containing the parts.

#### SIMDATA

is the default file type.

#### *ft*

is the file name of the CMS file containing the parts.

#### \*

indicates the first file found in the search order with the correct file name and file type should be used. Asterisk (\*) is the default.

#### *fm*

is the file mode of the file. If a file mode is specified, it must be accessed.

**CMS File Contents:** The CMS file can contain one of the following:

1. A collection of TDATA statements that identify parts to be validated. For example, a file could contain this:

```
TDATA :PART VMFAAA EXC
TDATA :PART VMFBBB XED
TDATA :PART VMFCCC
```

2. A collection of file names and, optionally, file type abbreviations of the parts to be validated. The format of the file can be that of a CMS exec generated from the LISTFILE command using the EXEC option, because any EXEC 2 statements (for example, &TRACE OFF) and dummy arguments (for example, &1) are ignored.

For example, a file could contain this:

```
&1 &2 VMFAAA EXC
&1 &2 VMFBBB XED
&1 &2 VMFCCC
```

For more information, see [“Using File Input” on page 527](#).

### STEM

identifies a simple REXX stem variable that lists the parts to be validated.

#### *stemid*

is the name of the REXX stem containing the parts.

### PI

**Stem Contents:** The stem can contain one of the following:

1. A collection of TDATA statements that identify parts to be validated. For example, a stem could contain:

```
x.0 = 3
x.1 = 'TDATA :PART VMFAAA EXC'
x.2 = 'TDATA :PART VMFBBB XED'
x.3 = 'TDATA :PART VMFCCC'
```

2. A collection of file names and, optionally, the file types of the parts in the version vector table to validate against corresponding AUX files. The format of the file can be that of a CMS exec generated from the LISTFILE command using the EXEC option, because any EXEC 2 statements (for example, &TRACE OFF) and dummy arguments (for example, &1) are ignored. For example, a stem could contain:

```
x.0 = 3
x.1 = 'VMFAAA EXC'
x.2 = 'VMFBBB XED'
x.3 = 'VMFCCC'
```

The zero element of the stem identifies the number of other elements in the stem.

For more information, see [“Using the STEM Variable” on page 528](#).

### PI end

## Options

### AUXFT

identifies the file type of an auxiliary (AUX) control file, specified in the control file, at which the validation should occur. If this option is not specified on the command, all AUX levels identified in the control file are validated against the corresponding version vector table for that level.

#### *AUXlvld*

is the file type of an AUX file.

### CNTRL

indicates a control file is used to identify the AUX file structure.

***cntrlfn***

is the file name of the control file that is used to identify the AUX file structure. The file type of the control file is CNTRL. This value overrides the value on the :CNTRL tag in the PPF.

**LOG**

identifies the type of message logging to be done. Messages are logged in the specified message log, as well as written to the terminal.

No messages are logged until initial validation of the command is complete.

**Note:** The LOG option is reserved for use by VMSES/E.

**NONE**

is the default. If NONE is specified, messages are written only to the terminal.

***logid***

is a three-character message log identifier, for example:

***logid*****Type of Log****APP**

The apply message log (\$VMFAPP \$MSGLOG A)

**BLD**

The build message log (\$VMFBLD \$MSGLOG A)

**XYZ**

The user message log (\$VMFXYZ \$MSGLOG A)

**LOGLVL**

identifies the level of message logging to be done.

**Note:** The LOGLVL option is ignored if the LOG option is NONE.

**I**

is the default. 'I' logs all informational messages, warning messages, error messages, severe error messages, and terminating errors.

***mlvl***

is the message severity level. Messages are logged in the specified message log if they have a severity level equal to or above the *mlvl* specified. The message levels are shown below, in order of severity. Messages of level 'R' have the lowest severity, and messages of level 'T' have the highest severity.

***mlvl*****Level of logging****R**

Response required

**I**

Informational Message

**W**

Warning Message

**E**

Error Message

**S**

Severe Error Message

**T**

Terminating Error

**LOGMOD**

automatically logs information in the version vector tables based on information in the AUX files. VMFSIM CHKLVL compares the AUX files for all AUX file levels above the level specified on the :UPDTID tag in the product parameter file to the corresponding version vector tables. If a

mismatch is detected, the information from the AUX file is used to replace the existing version vector table entry. The first token in the AUX file is the update file type, the second token is ignored (however, LCL is recommended), and the third token is *modid*. All LOCAL disks must be accessed as Read/Write?

When you use the LOGMOD option:

- If a version vector table does not exist, it is created on the first disk in the LOCAL string.
- If the *modid* is not specified in the AUX file, VMFSIM CHKLVL uses the *levelid*, which has been appended to LC, from the control file.
- If the AUX file for a part is not found, the :PART entry is deleted from the version vector table.
- If the AUX file for a part is empty, the :MOD data is deleted from the version vector table for that part. The :PART entry is not deleted from the version vector table.
- All LOCAL disks must be accessed as read-write.

**TYPE**

directs the output to the terminal. TYPE is the default.

**FILE**

directs the output to a CMS file.

*fn*

is the file name of the CMS file containing the output from CHKLVL. The file type is SIMDATA, and the file mode is A.

**STEM**

identifies a simple REXX stem variable that will contain the output of CHKLVL. The structure and content of the stem are the same as if an EXECIO was performed on the output file.

*stemid*

is the name of the REXX stem that will contain the output from CHKLVL.

**Examples**

To use VMFSIM CHKLVL to validate the AUX files for a specific part match the corresponding version vector table entries for that part, enter:

```
VMFSIM CHKLVL ESA MYCOMP TDATA :PART DMSABC
```

This command checks the levels of the AUX files for DMSABC against the version vector table entries. This command checks the AUX files for DMSABC against the version vector table entries. In the output returned, the "X" shows the first mismatch detected.

```
=====DMSABC TXT=====1VMVMC23
VVTVM1 E.....| DMSABC AUXVM1 E
PTF
-----|-----|-----|
UM00004.VM40000.H40000DS...| H40000DS PTF UM00004 * Fix to problem 4
UM00003.VM30000.H30000DS...| H30000DS PTF UM00003 * Fix to problem 3
UM00002.VM20000.H20000DS...|X|HXXXXXDS PTF UM00002 * Fix to problem 2
UM00001.VM10000.H10000DS...| H10000DS PTF UM00001 * Fix to problem 1
=====
```

**Input and Output Files**

**Input Files**

*cntrlfn* CNTRL

The control file.

*cntrlfn* CNTRLEXT

The control file extension table.

*fn* SIMDATA

An optional input file for the data.

**appid VVTlvlid**

The version vector table.

**partid AUXlvlid**

The AUX file.

**ppfname PPF**

The usable form product parameter file.

**VM SYSABRVT**

The table that contains the file type abbreviations (*ftabbrev*).

**Output Files**

**fn SIMDATA**

An optional output file for the returned data.

**fn OLDDATA**

The previous level of the *fn SIMDATA* output file.

**appid VVTlvlid**

A version vector table.

**Temporary Files**

**\$VMFSIM CNTRL**

A control file used with the LOGMOD option.

**fn AUX\$\$\$\$**

An AUX file used with the LOGMOD option.

**PPF Tags Used**

**:APPID**

Identifies the product, which is used as the file names of the version vector tables.

**:CNTRL**

Defines the name of the control file, which is used to identify the version vector structure.

**:DABBV**

Defines the file type abbreviation specific to a product and the real and base file types associated with it.

**Messages and Return Codes**

Appendix D, “Module Identifiers for VMSES/E Messages,” on page 749 shows the format of VMSES/E messages and lists the identifiers used by each VMSES/E exec.

For a complete explanation of any VMSES/E message, use the HELP Facility to view the message explanation online or see the appropriate messages documentation. To display information on a specific message, for example VMF002E, enter:

```
help msg vmf002e
```

If you are unfamiliar with the z/VM HELP Facility, enter `help` to display the main HELP Menu. For information on the HELP command, enter:

```
help cms help
```

The VMFSIM CHKLVL EXEC issues the following return codes:

Return Code	Explanation
0	Command completed successfully.
4	Command completed with one or more warning conditions.
8	Command completed but one or more parts were not supported by AUX files or there were mismatches between AUX files and version vector tables.

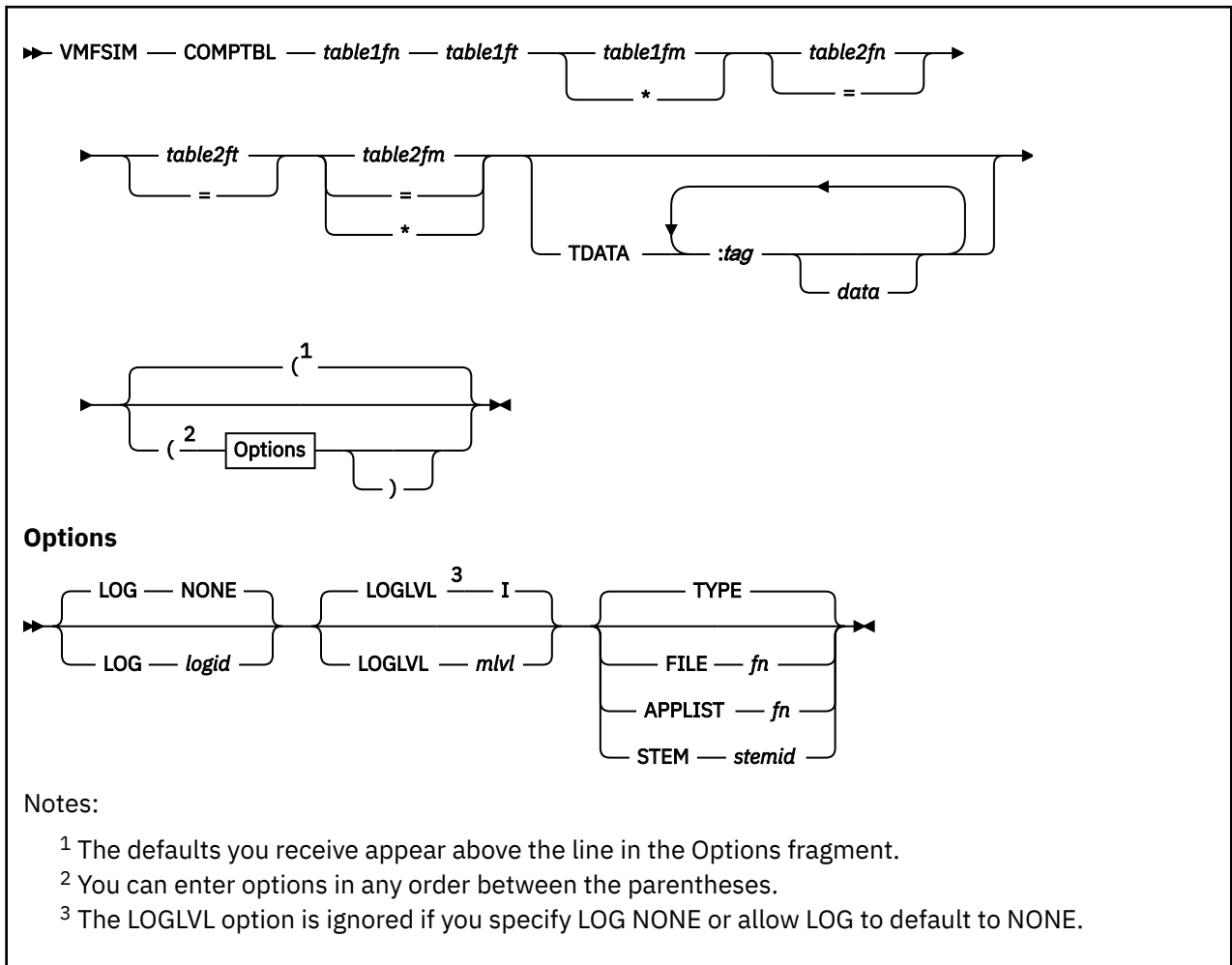
<b>Return Code</b>	<b>Explanation</b>
12	Not all input data was processed successfully due to syntax errors in the data.
24	Command failed because of a command line syntax error.
28	Command failed because a required file was not found.
36	Command failed because a target disk or directory was not available.
100	Command failed because of an external error.

### **Recovery Information**

The VMFSIM CHKLVL function saves a backup copy of the SIMDATA file that is being updated if it is found on the output disk. The file name of the file is the same as the original file, and the file type is OLDDATA. You can use this file to recover the SIMDATA file to the level of data it contained before the command was run.

After the errors have been corrected, rerun the command to regenerate the SIMDATA file.

## VMFSIM COMPTBL



### Purpose

The VMFSIM COMPTBL command can be used to verify that the first table that has been entered on the command is a subset of the second table. The tables being compared must have the same key and one or more of the same fields.

### Operands

#### **table1fn**

is the file name of the table containing the fields that are to be compared to the second table.

#### **table1ft**

is the file type of the table containing the fields that are to be compared to the second table.

#### **table1fm**

is the file mode of the table containing the fields that are to be compared to the second table. If an asterisk (\*) is specified, the first table in the search order that matches the file name and file type specifications for table 1 is used. If a file mode is specified, it must be accessed.

#### **table2fn**

is the file name of the table containing the fields the first table compares to. If '=' is specified, the same file name that was specified for the first table is used.



**table2ft**

is the file type of the table containing the fields the first table compares to. If '=' is specified, the same file type that was specified for the first table is used.

**table2fm**

is the file mode of the table containing the fields that are to be compared to the first table. If a file mode is specified, it must be accessed. If '=' is specified, the same file mode that was specified for the first table is used. If an asterisk (\*) is specified, the first table in the search order that matches the file name and file type specifications for table 2 is used.

**TDATA**

identifies the fields to be compared in the tables. If TDATA is not specified, all fields defined in the first table are compared to the second table. If TDATA is specified and a key is not specified on the command, it is added to the list of tags specified for the comparison.

**:tag**

defines which entries to compare.

**data**

specifies the search arguments to be used in the table search.

**Options****LOG**

identifies the type of message logging to be done. Messages are logged in the specified message log as well as written to the terminal.

No messages are logged until initial validation of the command is complete.

**Note:** The LOG option is reserved for use by VMSES/E.

**NONE**

is the default. If NONE is specified, messages are written only to the terminal.

**logid**

is a 3-character message log identifier, for example:

**logid****Type of Log****APP**

The apply message log (\$VMFAPP \$MSGLOG A)

**BLD**

The build message log (\$VMFBLD \$MSGLOG A)

**XYZ**

The user message log (\$VMFXYZ \$MSGLOG A)

**LOGLVL**

identifies the level of message logging to be done.

**Note:** The LOGLVL option is ignored if the LOG option is NONE.

**I**

is the default. 'I', logs all informational messages, warning messages, error messages, severe error messages, and terminating errors.

**mlvl**

is the message severity level. Messages are logged in the specified message log if they have a severity level equal to or above the *mlvl* specified. The message levels are shown below, in order of severity. Messages of level 'R' have the lowest severity, and messages of level 'T' have the highest severity.

**mlvl****Level of logging**

- R** Response required
- I** Informational Message
- W** Warning Message
- E** Error Message
- S** Severe Error Message
- T** Terminating Error

**TYPE**

directs the output to the terminal. TYPE is the default.

**FILE**

directs the output of COMPTBL to a CMS file.

*fn*

is the file name of the CMS file that will store the output from COMPTBL. The file type is SIMDATA, and the file mode is A.

**APPLIST**

directs the output of COMPTBL to an apply list that can later be used by VMFAPPLY. This function can only be used when you are comparing tables that have :PTF as their key.

*fn*

is the file name of the apply list. The file type is \$APPLIST, and the file mode is A.

**STEM**

identifies a simple REXX stem variable that will contain the output of COMPTBL. The structure and content of the stem is the same as if an EXECIO was performed on the output file.

*stemid*

is the name of the REXX stem variable.

**Examples**

- To use VMFSIM COMPTBL to compare PTFs in different levels of the apply status tables, enter:

```
VMFSIM COMPTBL 1VMVMC23 SRVAPPS D = = E TDATA :PTF
```

This command returns a list of all PTFs that are in the first table and not in the second table. The format of the output data follows.

```
===== 1VMVMC23 SRVAPPS D2 =====
:PTF UV45678
----- 1VMVMC23 SRVAPPS E2
:PTF **NOT FOUND**
===== 1VMVMC23 SRVAPPS D2 =====
:PTF UV00001
----- 1VMVMC23 SRVAPPS E2
:PTF **NOT FOUND**
=====
```

- To use VMFSIM to compare PTFs and status in different apply status tables, enter:

```
VMFSIM COMPTBL 1VMVMC23 SRVAPPS D = = E TDATA :PTF :STAT
```

This command returns a list of all PTFs that are in the first table and not in the second table, along with their status. The format of the output data is:

```
===== 1VMVMC23 SRVAPPS D2 =====
:PTF UV12345 :STAT APPLIED.01/03/22.11:11:11.JONES
```

```

----- 1VMVMC23 SRVAPPS E2
:PTF **NOT FOUND**
===== 1VMVMC23 SRVAPPS D2 =====
:PTF UV00007 :STAT SUPED.02/04/22.12:12:12.MAINT
----- 1VMVMC23 SRVAPPS E2
:PTF UV00007 :STAT APPLIED.01/04/22.22:22:22.MAINT
=====

```

## Input and Output Files

### Input Files

#### **table1fn table1ft**

The first table specified.

#### **table2fn table2ft**

The second table specified.

### Output Files

#### **fn \$APPLIST**

An apply list.

#### **fn SIMDATA**

An optional output file for the returned data.

#### **fn OLDDATA**

The previous level of the *fn* SIMDATA output file.

## Messages and Return Codes

Appendix D, “Module Identifiers for VMSES/E Messages,” on page 749 shows the format of VMSES/E messages and lists the identifiers used by each VMSES/E EXEC.

For a complete explanation of any VMSES/E message, use the HELP Facility to view the message explanation online or see the appropriate messages documentation. To display information about a specific message - VMF002E, for example - enter:

```
help msg vmf002e
```

If you are not familiar with the z/VM HELP Facility, enter `help` to display the main HELP Menu. For information about the HELP command, enter:

```
help cms help
```

Appendix D, “Module Identifiers for VMSES/E Messages,” on page 749 shows the format of VMSES/E messages and lists the identifiers for each VMSES/E EXEC.

The VMFSIM COMPTBL EXEC issues the following return codes:

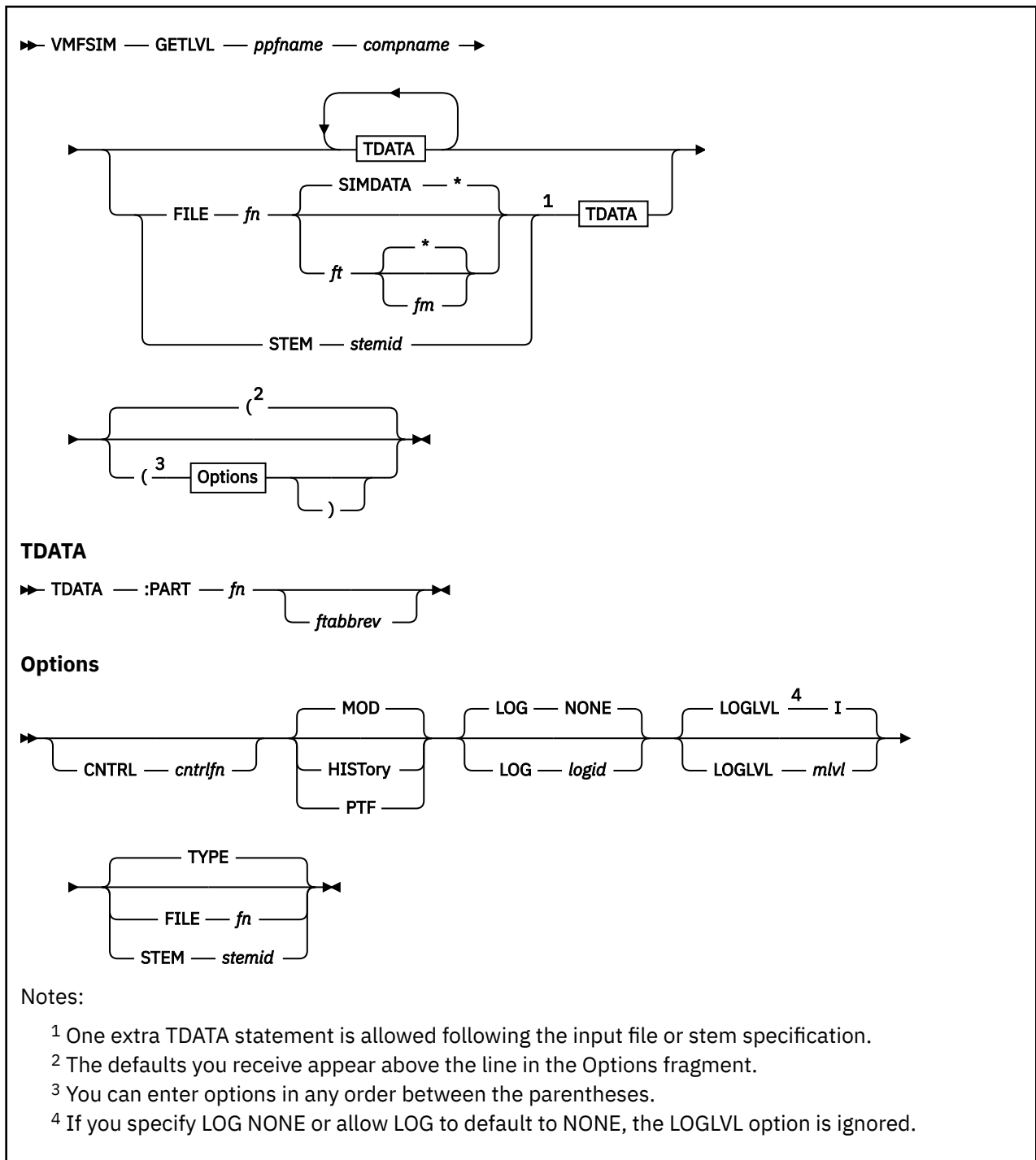
Return Code	Explanation
0	Command completed successfully.
2	Command completed successfully but extraneous data was encountered.
4	Command completed, the first table was not a subset of the second table.
12	Not all input data was processed successfully due to syntax errors in the data.
24	Command failed because of a command line syntax error.
28	Command failed because a required file was not found.
36	Command failed because a target disk or directory was not available.
100	Command failed because of an external error.

## **Recovery Information**

The VMFSIM COMPTBL function saves a backup copy of the SIMDATA file that is being updated if it is found on the output disk. The file name of the file is the same as the original file, and the file type is OLDDATA. You can use this file to recover the SIMDATA file to the level of data it contained before the command was run.

After the errors have been corrected, rerun the command to regenerate the SIMDATA file.

# VMFSIM GETLVL



## Purpose

The GETLVL function returns the file type of the highest level of a part, using the control file with the version vector tables.

## Operands

### *ppfname*

is the file name of the usable form product parameter file. It must have a file type of PPF.

### *compname*

is the name of the component, for example CP or CMS, as it is specified on the :COMPNAME tag in the product parameter file. *compname* is a 1-16 character alphanumeric identifier.

### TDATA

identifies the parts to be located. The TDATA statement must contain the tag :PART followed by the file name of a part to locate. You can specify additional tags, but they are ignored.

#### **:PART**

identifies a part to be located.

#### *fn*

is the file name of the part to be located.

#### *ftabbrev*

is the file type abbreviation for the part to be located. The *ftabbrev* must be the 3-character PTF abbreviation or the real CMS file type for parts that are not serviced by replacement.

If you do not specify *ftabbrev*, VMFSIM GETLVL locates all parts with the specified file name.

**Note:** If you specify a TDATA statement following the FILE, STEM, or ASTEM operand, it will be added to the end of the list of PARTs or products in the file or stem defined by the FILE, STEM, or ASTEM operand.

### FILE

identifies a CMS file that lists the parts to be located.

#### *fn*

is the file name of the file that lists the parts.

#### **SIMDATA**

is the default file type.

#### *ft*

is the file type of the file that lists the parts.

#### **\***

indicates the first file found in the search order with the correct file name and file type should be used. Asterisk (\*) is the default.

#### *fm*

is the file mode of the file that lists the parts. If a file mode is specified, it must be accessed.

**CMS File Contents:** The contents of the file can be one of the following:

1. A collection of TDATA statements that identify parts to be located. For example, the file could contain this:

```
TDATA :PART VMFAAA EXC
TDATA :PART VMFBBB XED
TDATA :PART VMFCCC
```

2. A collection of file names and, optionally, file types of the parts in the Version Vector table to be processed. The format of the file can be that of a CMS exec generated from the LISTFILE command using the EXEC option, because any EXEC 2 statements (for example, &TRACE OFF) and dummy arguments (for example, &1) are ignored. For example, the file could contain this:

```
&1 &2 VMFAAA EXC
&1 &2 VMFBBB XED
&1 &2 VMFCCC
```

For more information, see [“Using File Input” on page 527](#).

**STEM**

identifies a simple REXX stem variable that lists the parts to be located.

***stemid***

is the name of a simple REXX stem.

**PI**

**Stem Contents:** The contents of the stem can be one of the following:

1. A collection of TDATA statements that identify parts to be located. For example, a stem could contain this:

```
x.0 = 3
x.1 = 'TDATA :PART VMFAAA EXC'
x.2 = 'TDATA :PART VMFBBB XED'
x.3 = 'TDATA :PART VMFCCC'
```

2. A collection of file names and, optionally, the file types of the parts in the Version Vector table to be processed. The format of the file can be that of a CMS exec generated from the LISTFILE command using the EXEC option, because any EXEC 2 statements (for example, &TRACE OFF) and dummy arguments (for example, &1) are ignored. For example, a stem could contain this:

```
x.0 = 3
x.1 = 'VMFAAA EXC'
x.2 = 'VMFBBB XED'
x.3 = 'VMFCCC'
```

The zero element of the stem identifies the number of other elements in the stem.

For more information, see [“Using the STEM Variable” on page 528](#).

**PI end****Options****CNTRL**

indicates a control file is used to identify the AUX file structure.

***cntrlfn***

is the file name of the control file that is used to identify the AUX file structure. The file type of the control file is CNTRL. This value overrides the value on the :CNTRL tag in the PPF.

**MOD**

displays the file type of the highest overall level of a part. MOD is the default.

**HISTory**

displays the service history for a part.

**PTF**

displays the file type of the highest service level of a part.

**LOG**

identifies the type of message logging to be done. Messages are logged in the specified message log, as well as written to the terminal.

No messages are logged until initial validation of the command is complete.

**Note:** The LOG option is reserved for use by VMSES/E.

**NONE**

is the default. If NONE is specified, messages are written only to the terminal.

***logid***

is a three-character message log identifier, for example:

***logid***

**Type of Log**

### **APP**

The apply message log (\$VMFAPP \$MSGLOG A)

### **BLD**

The build message log (\$VMFBLD \$MSGLOG A)

### **XYZ**

The user message log (\$VMFXYZ \$MSGLOG A)

## **LOGLVL**

identifies the level of message logging to be done.

**Note:** The LOGLVL option is ignored if the LOG option is NONE.

### **I**

is the default. 'I' logs all informational messages, warning messages, error messages, severe error messages, and terminating errors.

### ***mlvl***

is the message severity level. Messages are logged in the specified message log if they have a severity level equal to or above the *mlvl* specified. The message levels are shown below, in order of severity. Messages of level 'R' have the lowest severity, and messages of level 'T' have the highest severity.

### ***mlvl***

#### **Level of logging**

### **R**

Response required

### **I**

Informational Message

### **W**

Warning Message

### **E**

Error Message

### **S**

Severe Error Message

### **T**

Terminating Error

## **TYPE**

directs the output to the terminal. TYPE is the default.

## **FILE**

directs the output from GETLVL to a CMS file.

### ***fn***

is the file name of the CMS file containing the output from GETLVL. The file type is SIMDATA, and the file mode is A.

## **STEM**

identifies a simple REXX stem variable that will contain the output of GETLVL. The structure and content of the stem are the same as if an EXECIO was performed on the output file.

### ***stemid***

is the name of the REXX stem variable.

## **Examples**

- To use VMFSIM GETLVL to search for the current level of a part using the corresponding version vector table entries for the part, enter:

```
vmfsim getlvl esa cp tdata :part receive
```



This command checks the levels of the version vector tables for the part 'RECEIVE' and returns the highest PTF-numbered levels found. The format of the output data is shown below.

```
RECEIVE EXC12345
RECEIVE XED00000 BASE-FILETYPE
RECEIVE MSG72345
```

- To display the service history for several parts, enter:

```
vmfsim getlvl esa cp tdata :part simsmpl txt
      tdata :part simrepl txt tdata :part simupdt txt (history
```

The output from VMFSIM GETLVL is:

```
:PART SIMSMPL TXT06009
:VVT VVTVM2
:PTF
    UM06009
    UM06005
    UM06004
    UM06002
    UM06001
:PART SIMREPL TXT06009
:VVT VVTVM2
:PTF
    UM06009.VM99999
    UM06004.VM44444
    UM06001.VM11111
:PART SIMUPDT TXT06009
:VVT VVTVM2
:PTF
    UM06009.VM99999.H99999HP
    UM06005.VM55555.H55555HP
    UM06004.VM44444.H44444HP
    UM06001.VM11111.H11111HP
```

To display the service history of several parts with local service, enter:

```
vmfsim getlvl esa cp tdata :part simsmpl txt
      tdata :part simrepl txt tdata :part simupdt txt (history
```

VMFSIM GETLVL displays:

```
:PART SIMSMPL TXTL0005
:VVT VVTLCL
:MOD
    LCL0005
    LCL0004
:VVT VVTVM2
:PTF
    UM06009
    UM06005
    UM06004
    UM06002
    UM06001
:PART SIMREPL TXTL0008
:VVT VVTLCL
:MOD
    LCL0008
    LCL0004
:VVT VVTVM2
:PTF
    UM06009.VM99999
    UM06004.VM44444
    UM06001.VM11111
:PART SIMUPDT TXTL0008
:VVT VVTLCL
:MOD
    LCL0008
    LCL0003
:VVT VVTVM2
:PTF
    UM06009.VM99999.H99999HP
    UM06005.VM55555.H55555HP
    UM06004.VM44444.H44444HP
    UM06001.VM11111.H11111HP
```

## Input and Output Files

### Input Files

#### *cntrlfn* CNTRL

The control file.

#### *cntrlfn* CNTRLEXT

The control file extension table.

#### *appid* VVTlvlid

The version vector tables.

#### VM SYSABRVT

The table that contains the file type abbreviations (*ftabbrev*).

#### *ppfname* PPF

The usable form product parameter file.

### Output Files

#### *fn* SIMDATA

An optional output file for the returned data.

#### *fn* OLDDATA

The previous level of the *fn* SIMDATA output file.

### PPF Tags Used

#### :APPID

Identifies the file names of the version vector tables.

#### :CNTRL

Defines the name of the control file, which is used to identify the version vector structure.

#### :DABBV

Defines the file type abbreviation specific to a product and the real and base file types associated with it.

## Messages and Return Codes

For a complete explanation of any VMSES/E message, use the HELP Facility to view the message explanation online or see the appropriate messages documentation. To display information on a specific message, for example VMF002E, enter:

```
help msg vmf002e
```

If you are unfamiliar with the z/VM HELP Facility, enter `help` to display the main HELP Menu. For information on the HELP command, enter:

```
help cms help
```

The VMFSIM GETLVL EXEC issues the following return codes:

Return Code	Explanation
0	Command completed successfully.
2	The AUX option was specified but the part is not supported by AUX files.
4	Command completed, but no serviced level was found for the part.
12	Not all input data was processed successfully due to syntax errors in the data.
24	Command failed because of a command line syntax error.
28	Command failed because a required file was not found.
36	Command failed because a target disk or directory was not available.

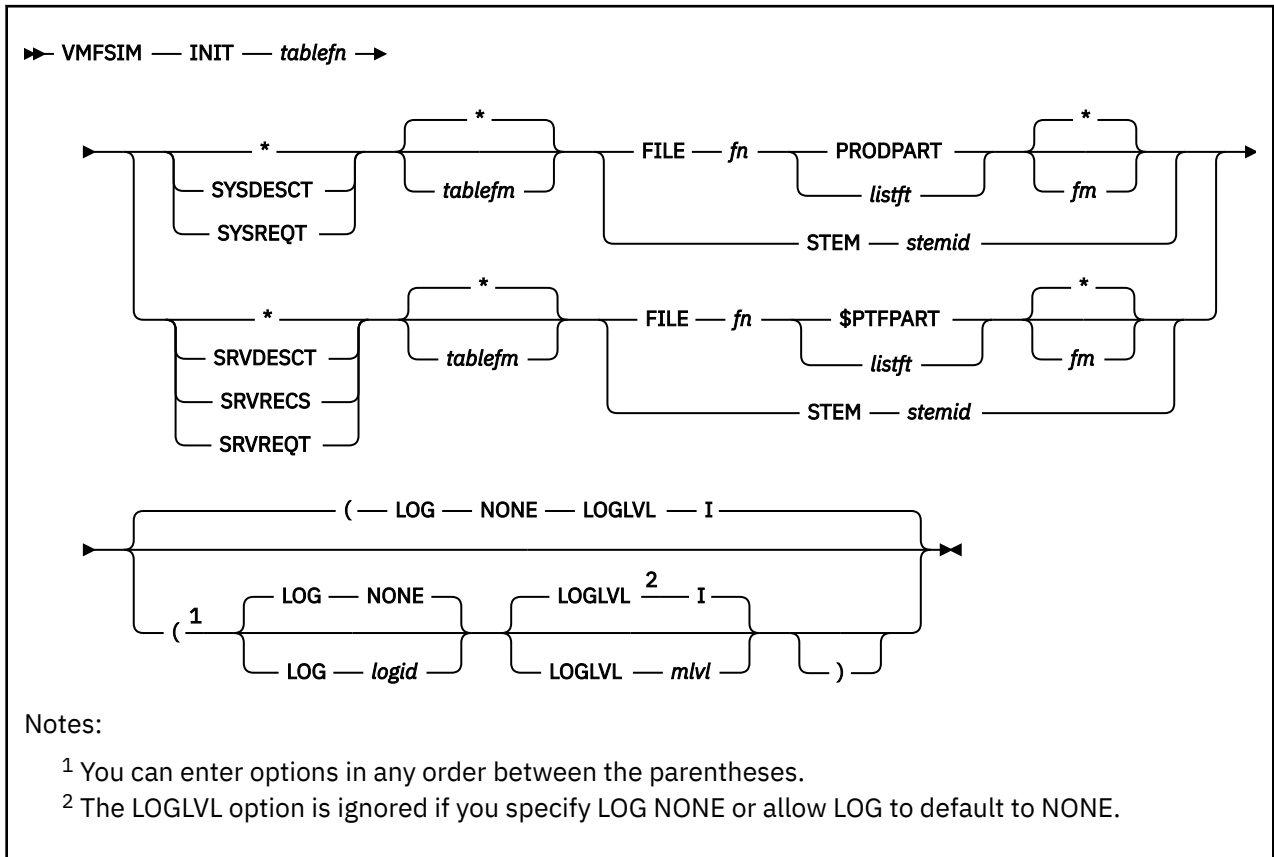
Return Code	Explanation
100	Command failed because of an external error.

### Recovery Information

The VMFSIM GETLVL function saves a backup copy of the SIMDATA file that is being updated, if it is found on the output disk. The file name of the backup file is the same as the original file, and the file type is OLDDATA. You can use this file to return to the same level of data that was in the SIMDATA file before the command was run.

After the errors have been corrected, rerun the command to regenerate the SIMDATA file.

## VMFSIM INIT



### Purpose

The INIT function either creates the specified tables or updates them based on the tags and data being processed from the \$PTFPART or PRODPART file.

The INIT function reads the specified files and adds the data in them to the Software Inventory requisite, description, and receive status tables.

### Operands

#### *tablefn*

is the file name of the table to be initialized. If \$PTFPART files are being processed to create or update service-level Software Inventory tables, the file name must be the same as the *prodid* identifier specified on the :RECID tag in the PPF file. If PRODPART files are being processed to create or update system-level Software Inventory tables, the file name should be the same as the file name of your system-level Software Inventory tables. VM is the default file name for the system-level Software Inventory tables.

\*

indicates all applicable Software Inventory tables should be updated. If the input is made up of PRODPART files, the SYSDESCT and SYSREQT tables are updated. If the input is made up of \$PTFPART files, the SRVDESCT, SRVRECS, and SRVREQT tables are updated.

If you specify an asterisk (\*) for the table file type (*tableft*) and you select PRODPART as the file type with the FILE operand, the SYSREQT and SYSDESCT tables are updated. If you specify an asterisk (\*) for the file type and you select \$PTFPART as the file type with the FILE operand, the SRVREQT, SRVDESCT and SRVRECS tables are updated.

**Note:** If the SRVRECS table is being updated, the :STAT field is updated with a status of RECEIVED along with the date, time, and user ID performing the update.

**SYSDESCT**

indicates the system-level description table is to be created or updated.

**SYSREQT**

indicates the system-level requisite table is to be created or updated.

**SRVDESCT**

indicates the service-level requisite table is to be created or updated.

**SRVRECS**

indicates the service-level receive status table is to be created or updated.

**SRVREQT**

indicates the service-level requisite table is to be created or updated.

**\***

indicates the first tables found in the CMS search order on a read/write accessed disk with the correct file names and file types should be updated. If the tables are found on a disk that is not accessed read/write, you receive an error message. If the tables are not found anywhere in the search order, they are created on your A-disk. If your A-disk is not in read/write mode, you receive an error message. Asterisk (\*) is the default.

***tablefm***

is the file mode of the minidisk or directory containing the tables. The file mode must be accessed read-write. If the tables are not found on the file mode specified, they are created on that mode. If the file mode is not accessed, an error message is displayed; and the table is not created or updated.

**FILE**

identifies the \$PTFPART or PRODPART files that are to be used as input to update the Software Inventory. The file can contain the file name of a single PRODPART or \$PTFPART file, or it can contain a list of PRODPART or \$PTFPART files. For more information, see [“Using File Input” on page 527](#).

***fn***

is the file name of the file. The file name for \$PTFPART files is the PTF number. The file name for PRODPART files is the product identifier (*prodid*) for the product. The file name for a file that contains a list of files can be any valid CMS file name.

**PRODPART**

is the file type for a PRODPART file. PRODPART files are used to update the system-level description table and requisite table.

**\$PTFPART**

is the file type for a \$PTFPART file. \$PTFPART files are used to update the service-level description table, requisite table, and receive status table.

***listft***

is the file type of a CMS file that contains a **collection** of either \$PTFPART or PRODPART files. The list must contain either all PRODPART or all \$PTFPART files; they cannot be mixed.

If you specify a system-level Software Inventory table on the command, the list must contain all PRODPART files. If you specify a service-level Software Inventory table on the command, the list must contain all \$PTFPART files. If you specify an asterisk (\*) for the table file type on the command, the file type of the first entry in the list that passes validation determines the set of tables to be updated.

The format of the file can be that of a CMS exec generated from the LISTFILE command using the EXEC option.

**\***

indicates the first file found in the search order with the correct file name and file type should be used. Asterisk (\*) is the default.

***fm***

is the file mode of the file. If a file mode is specified, it must be accessed.

**File Contents:** The file contains a collection of file names, file types, and, optionally, file modes of the \$PTFPART or PRODPART files to be processed. The format of the file can be that of a CMS exec generated from the LISTFILE command using the EXEC option, because any EXEC 2 statements (for example, &TRACE OFF) and dummy arguments (for example, &1) are ignored.

For example, the file could contain:

```
&1 &2 UM00001 $PTFPART E
&1 &2 UM00002 $PTFPART E
&1 &2 UM00003 $PTFPART E
```

Each file in the list is checked to make sure it has the correct file type and it exists on a minidisk or SFS directory in the CMS access order. If it does not have the correct file type or it cannot be found, an error message is issued, it is removed from the list, and the next file in the list is checked.

## STEM

identifies a REXX stem variable that contains a collection of either \$PTFPART or PRODPART files that are to be used as input to update the Software Inventory.

### *stemid*

is the name of the REXX stem.

If you specify a system-level Software Inventory table on the command, the stem must contain all PRODPART files. If you specify a service-level Software Inventory table on the command, the stem must contain all \$PTFPART files. The stem must contain either all \$PTFPART files or all PRODPART files; they cannot be mixed. If you specify an asterisk (\*) for the table file type (*tableft*) on the command, the file type of the first entry in the stem that passes validation determines the set of tables to be updated.

## PI

**Stem Contents:** The stem contains a collection of file names, file types, and, optionally, file modes of the \$PTFPART or PRODPART files to be processed. The format of the stem can be that of a CMS exec generated from the LISTFILE command using the EXEC option, because any EXEC 2 statements (for example, &TRACE OFF) and dummy arguments (for example, &1) are ignored.

For example, the stem could contain:

```
x.0 = 3
x.1 = 'UM00001 $PTFPART'
x.2 = 'UM00002 $PTFPART'
x.3 = 'UM00003 $PTFPART'
```

The zero element of the stem identifies the number of other elements in the stem.

Each file in the stem is checked to make sure it has the correct file type and it exists on a minidisk or SFS directory in the CMS access order. If it does not have the correct file type or it cannot be found, an error message is issued, it is removed from the stem, and the next file is checked.

For more information, see [“Using the STEM Variable” on page 528](#).

## PI end

## Options

### LOG

identifies the type of message logging to be done. Messages are logged in the specified message log as well as written to the terminal.

No messages are logged until initial validation of the command is complete.

**Note:** The LOG option is reserved for use by VMSES/E.

### NONE

is the default. If NONE is specified, messages are written only to the terminal.

**logid**

is a three-character message log identifier, for example:

**logid****Type of Log****APP**

The apply message log (\$VMFAPP \$MSGLOG A)

**BLD**

The build message log (\$VMFBLD \$MSGLOG A)

**XYZ**

The user message log (\$VMFXYZ \$MSGLOG A)

**LOGLVL**

identifies the level of message logging to be done.

**Note:** The LOGLVL option is ignored if the LOG option is NONE.

**I**

is the default. 'I' logs all informational messages, warning messages, error messages, severe error messages, and terminating errors.

**mlvl**

Messages are logged in the specified message log if they have a severity level equal to or above the *mlvl* specified. The message levels are shown below, in order of severity. Messages of level 'R' have the lowest severity, and messages of level 'T' have the highest severity.

**mlvl****Level of logging****R**

Response required

**I**

Informational Message

**W**

Warning Message

**E**

Error Message

**S**

Severe Error Message

**T**

Terminating Error

**Examples**

- To use VMFSIM INIT to update the SYSREQT and SYSDESCT tables with the data contained in the 1VMVMC23 PRODPART file, enter:

```
VMFSIM INIT VM * D FILE 1VMVMC23 PRODPART
```

As a result of this command, the following files are updated:

VM SYSREQT D is updated with requisite data from the PRODPART file.

VM SYSDESCT D is updated with product description data from the PRODPART file.

- To use VMFSIM INIT to update the SRVREQT, SRVDESCT, and SRVRECS tables with the data contained in the UM12345 \$PTFPART file, enter:

```
VMFSIM INIT 1VMVMC23 * E FILE UM12345 $PTFPART
```

As a result of this command, the following files are updated:

1VMVMC23 SRVREQT E is updated with requisite data from the \$PTFPART file.  
 1VMVMC23 SRVDESCT E is updated with APAR description data from the \$PTFPART file.  
 1VMVMC23 SRVRECS E is updated with a status of RECEIVED for PTF UM12345.

## Input and Output Files

### Input Files

***fn listft***

An optional input file.

***prodid PRODPART***

The PRODPART file.

***fn \$PTFPART***

The \$PTFPART file.

### Output Files

***sysid SYSDESCT***

The system-level description table.

***sysid SYSREQT***

The system-level requisite table.

***prodid SRVDESCT***

The service-level description table.

***prodid SRVRECS***

The service-level receive status table.

***prodid SRVREQT***

The service-level requisite table.

## Messages and Return Codes

Appendix D, “Module Identifiers for VMSES/E Messages,” on page 749 shows the format of VMSES/E messages and lists the identifiers used by each VMSES/E exec.

For a complete explanation of any VMSES/E message, use the HELP Facility to view the message explanation online or see the appropriate messages documentation. To display information on a specific message, for example VMF002E, enter:

```
help msg vmf002e
```

If you are unfamiliar with the z/VM HELP Facility, enter help to display the main HELP Menu. For information on the HELP command, enter:

```
help cms help
```

The VMFSIM INIT EXEC issues the following return codes:

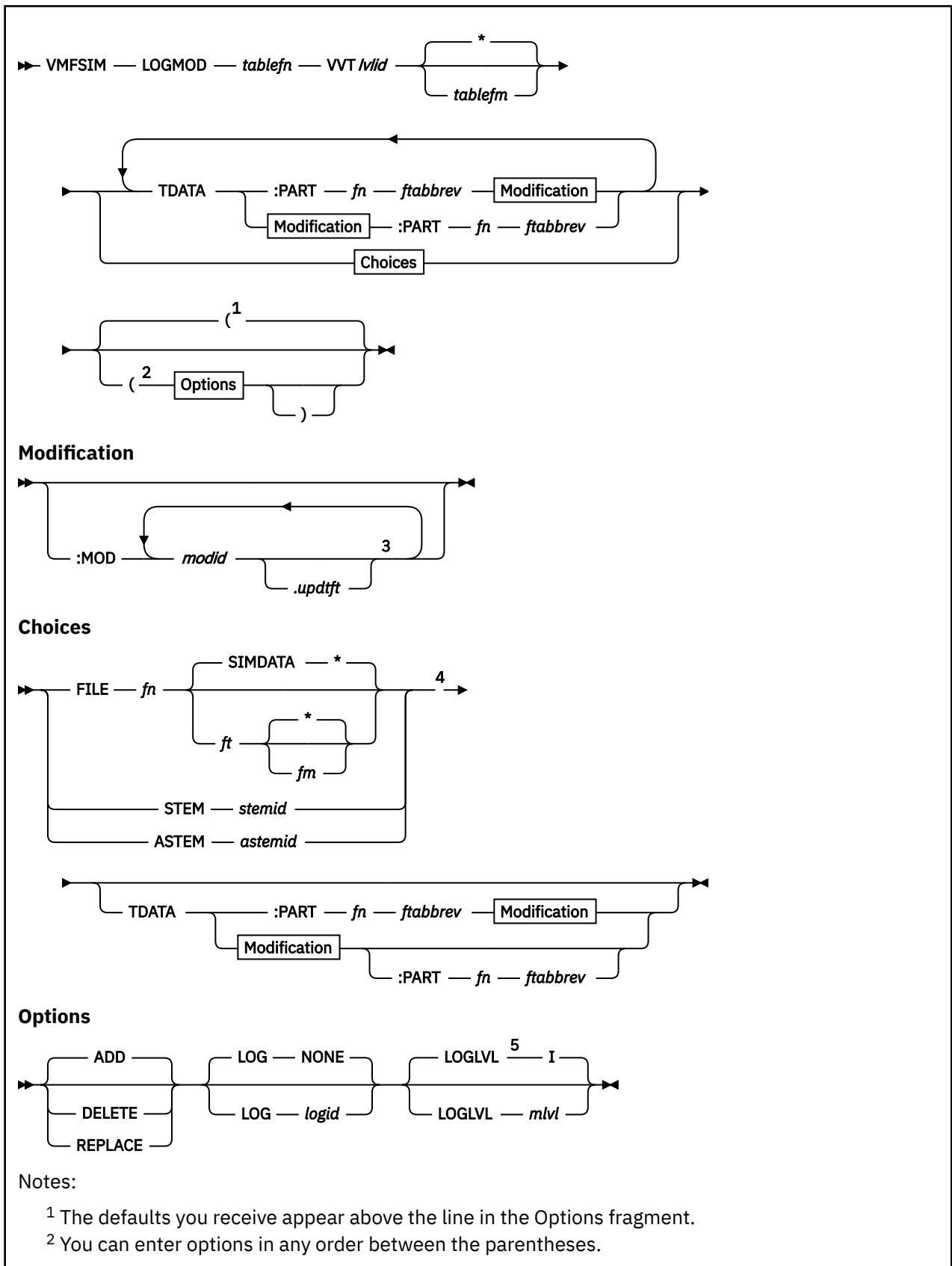
Return Code	Explanation
0	Command completed successfully.
2	Command completed successfully but extraneous data was encountered.
12	Not all input data was processed successfully due to syntax errors in the data.
24	Command failed because of a command line syntax error.
28	Command failed because a required file was not found.
36	Command failed because a target disk or directory was not available.
100	Command failed because of an external error.



## Recovery Information

The VMFSIM INIT command can be restarted by reissuing the command.

# VMFSIM LOGMOD



<sup>3</sup> You can string multiple *modid* entries together as long as they relate to the same file listed on the PART tag.

<sup>4</sup> One extra TDATA statement is allowed following the input file or stem specification.

<sup>5</sup> The LOGLVL option is ignored if you specify LOG NONE or allow LOG to default to NONE.

## Purpose

The LOGMOD function adds, replaces, or deletes local modification data in the specified version vector table.

## Operands

### *tablefn*

is the file name of the version vector table to be modified. This file name should be in the APPID string in the PPF.

### VVTlvlid

is the file type of the version vector table to be modified.

\*

indicates the first table found in the CMS search order on read/write accessed disk with the correct file name and file type should be updated. If the table is found on a disk that is not accessed read/write, you receive an error message. If the table is not found anywhere in the search order, it is created on your A-disk. If your A-disk is not in read/write mode, you receive an error message. Asterisk (\*) is the default.

### *tablefm*

is the file mode of the minidisk or directory containing the table. The file mode must be accessed as read/write. If the table is not found on that file mode, it is created on that mode. If the file mode is not accessed, an error message is displayed; and the table is not created or updated.

## TDATA

identifies the search data, a part for which a local modification is to be added, deleted, or replaced. You can specify additional tags, but they are ignored.

### :PART

indicates a part for which a local modification is to be added, deleted, or replaced.

### *fn*

is the file name of the part for which a local modification is to be added, deleted, or replaced.

### *ftabbrev*

is the file type abbreviation for the part for which a local modification is to be added, deleted, or replaced. The file type must be the 3-character PTF abbreviation or the real CMS file type for parts that are not serviced by replacement.

### :MOD

indicates a local modification.

### *modid*

is a 7-character local tracking number that identifies the local modification to be added, deleted, or replaced in the specified version vector table. Multiple *modid* entries can be strung together as long as they relate to the same file listed on the PART tag.

The first two characters indicate a local tracking number follows. It is recommended the first two characters of the local tracking number be LC. Characters 3-7 are a 5-character identifier for your local modification, which you create according to your own tracking scheme. It is recommended that the first character be an L. Characters 3-7 of *modid* are concatenated to the *ftabbrev* to form the file type of the serviceable part that is associated with this modification level of the part. For example, LCL1234 is a local modification tracking number. In this example, L1234 would be concatenated to the *ftabbrev* to form the file type of the serviceable part.

We recommend you start the local tracking number with LCL to ensure it does not interfere with service delivered by IBM. If you use characters other than LCL, make sure they are unique for your product.

**.updtft**

is the file type of the source update file that contains the change for the local modification identified.

**Note:** If you specify a TDATA statement following the FILE, STEM, or ASTEM operands, it will either be:

- Added to the end of the list of modifications in the file or stem defined by the FILE, STEM, or ASTEM operand, if you specified both the :PART tag and the :MOD tag.
- Logically added to the list of modifications in the file or stem defined by the FILE, STEM, or ASTEM operand, if you did not specify either the :PART tag or the :MOD tag. This means the tags and data on the TDATA statement are added to the tags and data for each of the modifications listed in the file or stem.

**FILE**

identifies a CMS file that contains a collection of parts for which local modifications are to be added, deleted, or replaced. For more information, see [“Using File Input” on page 527](#).

**fn**

is the file name of the file.

**SIMDATA**

is the default file type for the file.

**ft**

is the file type of the file.

**\***

indicates the first file found in the search order with the correct file name and file type should be used. Asterisk (\*) is the default.

**fm**

is the file mode of the file. If a file mode is specified, it must be accessed.

**File Contents:** For example, a file could contain:

```
TDATA :PART VMFAAA EXC :MOD LCL0001.LCL0001L
TDATA :PART VMFAAA XED :MOD LCL0001.LCL0001L
TDATA :PART VMFAAA TXT :MOD LCL0001.LCL0001L
```

**STEM**

identifies a simple REXX stem variable that contains a collection of parts for which local modifications are to be added, deleted, or replaced.

**stemid**

is the name of a REXX stem.

**PI**

**Stem Contents:** For example, the stem could contain:

```
x.0 = 3
x.1 = 'TDATA :PART VMFAAA EXC :MOD LCL0001.LCL0001L '
x.2 = 'TDATA :PART VMFAAA XED :MOD LCL0001.LCL0001L '
x.3 = 'TDATA :PART VMFAAA TXT :MOD LCL0001.LCL0001L '
```

For more information, see [“Using the STEM Variable” on page 528](#).

**PI end****ASTEM**

identifies an associative REXX stem variable that contains a collection of parts for which local modifications are to be added, deleted, or replaced.

***astemid***

is the name of an associative REXX stem.

**PI**

**Astem Contents:** For example, an associate stem could contain:

```
x.0 = 4
x.0.0 = 0
x.0.tags = 'PART 1 PART 2 PART 3'

x.1 = 'PART'
x.1.0 = 1
x.1.1 = 'VMFAAA EXC'
x.1.tags = 'MOD 4'

x.2 = 'PART'
x.2.0 = 1
x.2.1 = 'VMFBBB XED'
x.2.tags = 'MOD 4'

x.3 = 'PART'
x.3.0 = 1
x.3.1 = 'VMFCCC TXT'
x.3.tags = 'MOD 4'

x.4 = 'MOD'
x.4.0 = 1
x.4.1 = 'LCL0001.LCL0001L'
x.4.tags = ''
```

For more information, see [“Associative Stems” on page 529](#).

**PI end****Options****ADD**

Specifies the tags and data on the command are to be added to the table if they do not already exist. ADD is the default.

**DELETE**

Specifies the tags and data on the command are to be deleted from the table.

**REPLACE**

Specifies the tags and data on the command are to replace any existing fields in the table. If you are replacing multiple entries for the same PART entry, string the *modid* entries together on one TDATA statement. If you use duplicate TDATA statements for each *modid*, each *modid* is overlaid on top of the previous one.

**LOG**

identifies the type of message logging to be done. Messages are logged in the specified message log as well as written to the terminal.

No messages are logged until initial validation of the command is complete.

**Note:** The LOG option is reserved for use by VMSES/E.

**NONE**

is the default. If NONE is specified, messages are written only to the terminal.

***logid***

is a three-character message log identifier, for example:

***logid*****Type of Log****APP**

The apply message log (\$VMFAPP \$MSGLOG A)

**BLD**

The build message log (\$VMFBLD \$MSGLOG A)

**XYZ**

The user message log (\$VMFXYZ \$MSGLOG A)

**LOGLVL**

identifies the level of message logging to be done.

**Note:** The LOGLVL option is ignored if the LOG option is NONE.

**I**

is the default. 'I' logs all informational messages, warning messages, error messages, severe error messages, and terminating errors.

**mlvl**

is the message severity level. Messages are logged in the specified message log if they have a severity level equal to or above the *mlvl* specified. The message levels are shown below, in order of severity. Messages of level 'R' have the lowest severity, and messages of level 'T' have the highest severity.

**mlvl**

**Level of logging**

**R**

Response required

**I**

Informational Message

**W**

Warning Message

**E**

Error Message

**S**

Severe Error Message

**T**

Terminating Error

**Usage Notes**

1. For more information on local modifications, see [Chapter 11, "Installing Local Service and Modifications,"](#) on page 101.

**Examples**

- To use VMFSIM LOGMOD to identify a local modification added to a part that is serviced by replacement, enter:

```
VMFSIM LOGMOD 1VMVMC23 VVTLCL B TDATA :PART HCPXXX EXC :MOD LCLTST1
```

- To use VMFSIM LOGMOD to identify a local modification added to a part that is modified using a source update, enter:

```
VMFSIM LOGMOD 1VMVMC23 VVTLCL B TDATA :PART HCPSNT TXT
:MOD LCLTST1.LCLMOD1
```

**Note:** LCLTST1 is the name of the local tracking number, and LCLMOD1 is the file type of the source update file.

- To use VMFSIM LOGMOD with the REPLACE option to identify multiple local modifications and update the version vector table, enter:

```
VMFSIM LOGMOD 1VMVMC23 VVTLCL B TDATA :PART HCPSNT TXT
:MOD LCLTEST1.LCLMOD1 LCLTEST2.LCLMOD2 LCLTEST3.LCLMOD3
```

## Input and Output Files

### Input Files

#### *fn SIMDATA | ft*

An optional input file.

### Input/Output Files

#### *fn VVTlvlid*

The version vector table.

## Messages and Return Codes

Appendix D, “Module Identifiers for VMSES/E Messages,” on page 749 shows the format of VMSES/E messages and lists the identifiers used by each VMSES/E exec.

For a complete explanation of any VMSES/E message, use the HELP Facility to view the message explanation online or see the appropriate messages documentation. To display information on a specific message, for example VMF002E, enter:

```
help msg vmf002e
```

If you are unfamiliar with the z/VM HELP Facility, enter `help` to display the main HELP Menu. For information on the HELP command, enter:

```
help cms help
```

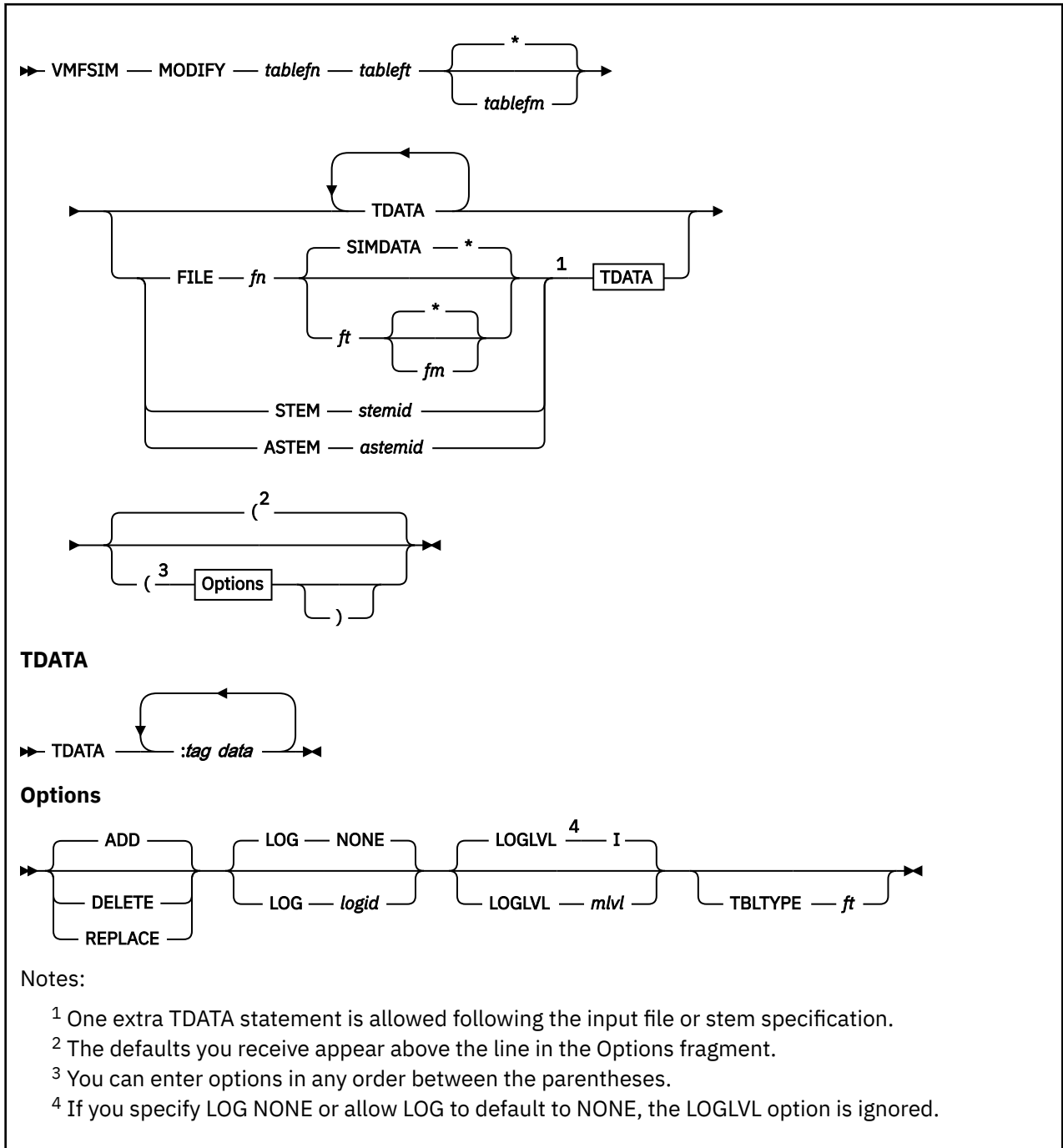
The VMFSIM LOGMOD EXEC issues the following return codes:

Return Code	Explanation
0	Command completed successfully.
2	Command completed successfully but extraneous data was encountered.
12	Not all input data was processed successfully due to syntax errors in the data.
24	Command failed because of a command line syntax error.
28	Command failed because a required file was not found.
36	Command failed because a target disk or directory was not available.
100	Command failed because of an external error.

## Recovery Information

The VMFSIM LOGMOD command can be restarted by reissuing the command.

# VMFSIM MODIFY



## Purpose

The MODIFY function adds, replaces, or deletes specific data from the Software Inventory.

**Note:** The MODIFY function does not support the PPF, \$PPFTMP, PRODPART, or \$PTFPART files.



## Operands

### ***tablefn***

is the file name of the table to be modified.

### ***tableft***

is the file type of the table to be modified.

**\***

indicates the first table found in the CMS search order on a read/write accessed disk with the correct file name and file type should be modified. If the table is found on a disk that is not accessed read/write, you receive an error message. If the table is not found anywhere in the search order, it is created on your A-disk. If your A-disk is not in read/write mode, you receive an error message. Asterisk (\*) is the default.

### ***tablefm***

is the file mode of the minidisk or directory containing the table. The file mode must be accessed as read-write. If the table is not found on the file mode, it is created on that mode. If the file mode is not accessed, an error message is displayed; and the table is not created or updated.

## **TDATA**

identifies modification data.

You must specify the key field and a search value, or the modify function exits with a nonzero return code. If only the key field and data are specified and you choose the DELETE option, the entire record is deleted (if it is found).

### ***:tag***

are the tags to be added, deleted, or replaced.

### ***data***

is the data to be added, deleted, or replaced.

**Note:** If you specify a TDATA statement following the FILE, STEM, or ASTEM operand, it will either be:

- Added to the end of the list of modifications in the file or stem defined by the FILE, STEM, or ASTEM operand, if you specified the key field.
- Logically added to the list of modifications in the file or stem defined by the FILE, STEM, or ASTEM operand, if you did not specify the key field. This means the tags and data on the TDATA statement are added to the tags and data for each of the modifications listed in the file or stem.

Each final TDATA statement, whether it is specified on the command line or in a file or stem, must include the key field for the table being modified. If you use the DELETE option and only specify the key field, the entire record is deleted (if it is found).

## **FILE**

identifies a CMS file that lists TDATA statements that contain modification data. The CMS file has a collection of tags that contain the search data and tags to be edited. For more information, see [“Using File Input” on page 527](#).

### ***fn***

is the file name of the file.

### **SIMDATA**

is the default file type.

### ***ft***

is the file type of the CMS file that contains the tags and data.

**\***

indicates the first file found in the search order with the correct file name and file type should be used. Asterisk (\*) is the default.

### ***fm***

is the file mode of the file. If a file mode is specified, it must be accessed.

**STEM**

identifies a simple REXX stem variable that lists TDATA statements that contain modification data. For more information, see [“Using the STEM Variable”](#) on page 528.

***stemid***

is the name of the REXX stem.

**ASTEM**

identifies an associative REXX stem variable that contains modification data. For more information, see [“Associative Stems”](#) on page 529.

***astemid***

is the name of the associate REXX stem.

**Options****ADD**

specifies the tags and data on the command are to be added to the table if they do not already exist. If the data already exists in the table, it is ignored. ADD is the default.

**DELETE**

specifies the tags and data on the command are to be deleted from the table.

**REPLACE**

specifies the tags and data on the command are to replace any existing fields in the table. If the entry does not already exist in the table, it is added.

**Note:** The existence of an entry is determined solely by the key field.

**LOG**

identifies the type of message logging to be done. Messages are logged in the specified message log as well as written to the terminal.

No messages are logged until initial validation of the command is complete.

**Note:** The LOG option is reserved for use by VMSES/E.

**NONE**

is the default. If NONE is specified, messages are written only to the terminal.

***logid***

is a three-character message log identifier, for example:

***logid*****Type of Log****APP**

The apply message log (\$VMFAPP \$MSGLOG A)

**BLD**

The build message log (\$VMFBLD \$MSGLOG A)

**XYZ**

The user message log (\$VMFXYZ \$MSGLOG A)

**LOGLVL**

identifies the level of message logging to be done.

**Note:** The LOGLVL option is ignored when the LOG option is NONE.

**I**

is the default. ‘I’ logs all informational messages, warning messages, error messages, severe error messages, and terminating errors.

***mlvl***

is a severity level. Messages are logged in the specified message log if they have a severity level equal to or above the *mlvl* specified. The message levels are shown below, in order of severity. Messages of level ‘R’ have the lowest severity, and messages of level ‘T’ have the highest severity.

***mlvl*****Level of logging****R**

Response required

**I**

Informational Message

**W**

Warning Message

**E**

Error Message

**S**

Severe Error Message

**T**

Terminating Error

**TBLTYPE**

identifies the file type or definition of a table in the Software Inventory that is to be used to process the table that is specified in the command. When you specify the TBLTYPE option, VMFSIM can process Software Inventory tables with file types other than those usually associated with a Software Inventory table, for example, NEWFILE.

If this option is not specified, the VMFSIM MODIFY function assumes the file type of the table specified on the command, *tableft*, is defined in the Software Inventory.

***ft***

is the file type to use when processing the table that is specified on the command. For example, if you specify TBLTYPE SYSDESCT, you tell VMFSIM the file is really a system-level description table.

**Examples**

- To use VMFSIM MODIFY to add a prerequisite to the SRVREQT table for a specific PTF, enter:

```
VMFSIM MODIFY 1VMVMC23 SRVREQT TDATA :PTF UV12345 :PREREQ UV56789
```

This command adds the prerequisite UV56789 to the PTF UV12345. The format of the output data is shown below.

```
:PTF UV12345
:PREREQ UV56789 UV23456
```

**Input and Output Files****Input Files*****fn SIMDATA | ft***

An optional input file.

**Input/Output Files*****tablefn tableft***

The Software Inventory table.

**Messages and Return Codes**

Appendix D, “Module Identifiers for VMSES/E Messages,” on page 749 shows the format of VMSES/E messages and lists the identifiers used by each VMSES/E exec.

## VMFSIM MODIFY

For a complete explanation of any VMSES/E message, use the HELP Facility to view the message explanation online or see the appropriate messages documentation. To display information on a specific message, for example VMF002E, enter:

```
help msg vmf002e
```

If you are unfamiliar with the z/VM HELP Facility, enter help to display the main HELP Menu. For information on the HELP command, enter:

```
help cms help
```

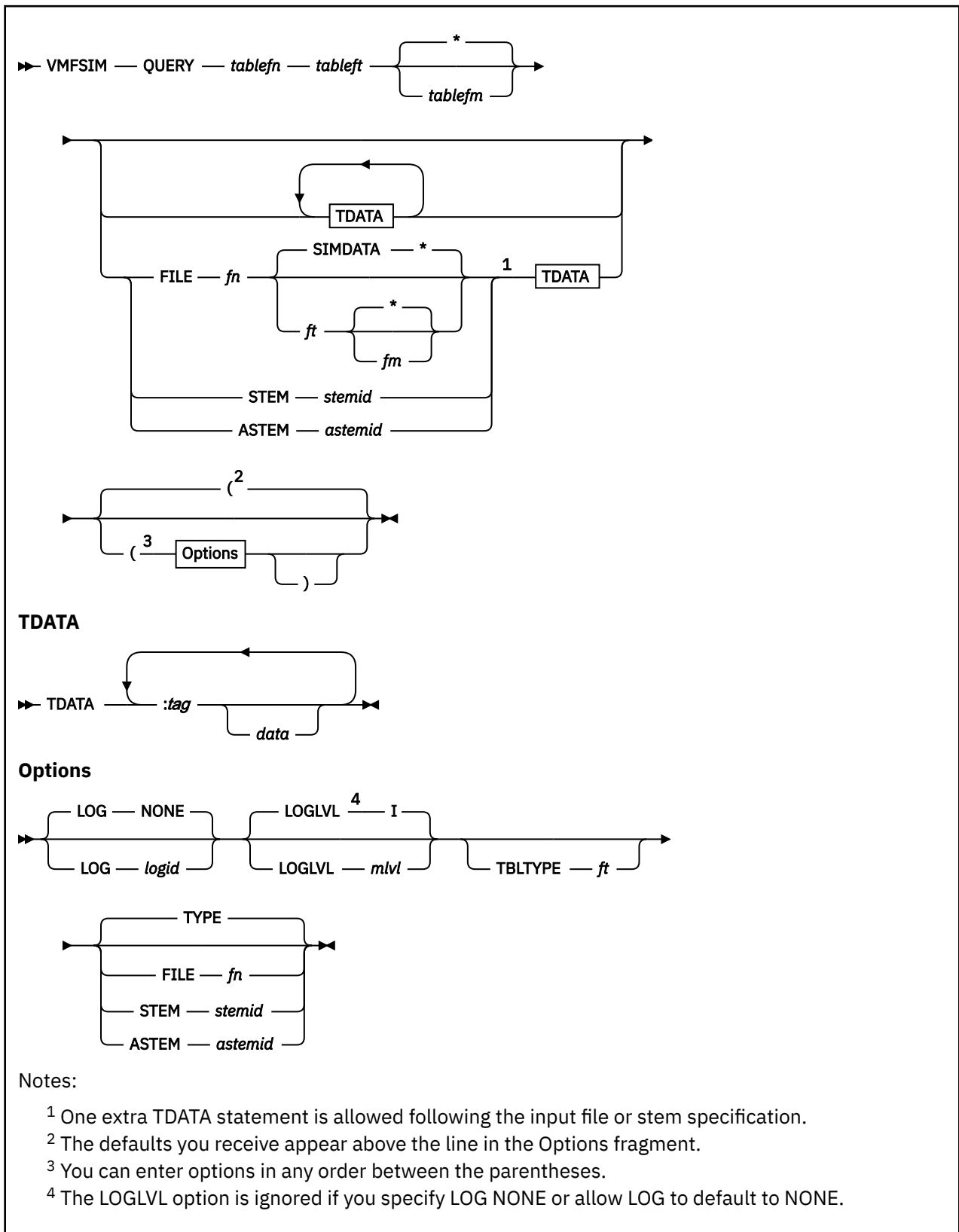
The VMFSIM MODIFY EXEC issues the following return codes:

Return Code	Explanation
0	Command completed successfully.
2	Command completed successfully but extraneous data was encountered.
12	Not all input data was processed successfully due to syntax errors in the data.
24	Command failed because of a command line syntax error.
28	Command failed because a required file was not found.
36	Command failed because a target disk or directory was not available.
100	Command failed because of an external error.

### Recovery Information

The VMFSIM MODIFY command can be restarted by reissuing the command.

# VMFSIM QUERY



## Purpose

The QUERY function accesses the specified table and searches for the specified tag and its corresponding value.

## Operands

### *tablefn*

is the file name of the table to be queried.

### *tableft*

is the file type of the table to be queried.

**\***

is the default file mode. If an asterisk is used as the file mode, the query function processes the first table found in the search order that matches the specified file name and file type.

### *tablefm*

is the file mode of the minidisk or directory containing the table. If a mode is specified, it must be accessed.

**Note:** If you do not specify TDATA, FILE, STEM, or ASTEM, all fields that are applicable to the table are displayed to show you which fields may be entered.

## TDATA

identifies the search data.

### *:tag*

is the tag to search for. If a tag is specified and it does not contain any data, it is treated as a return field.

### *data*

is the data to search for. The format of the data returned is dependent on the options specified on the command.

**Note:** If you specify a TDATA statement **following** the FILE, STEM, or ASTEM operand, it will be one of the following:

- Added to the end of the list of search data in the file or stem defined by the FILE, STEM, or ASTEM operand, if you specified the key field.
- Logically added to the list of search data in the file or stem defined by the FILE, STEM, or ASTEM operand, if you did not specify the key field. This means the tags and data on the TDATA statement are added to the tags and data for the search data listed in the file or stem.

## FILE

identifies a CMS file that lists TDATA statements that contain search data. For more information, see [“Using File Input” on page 527](#).

### *fn*

is the file name of the file.

### **SIMDATA**

is the default file type.

### *ft*

is the file type of the file.

**\***

indicates the first file found in the search order with the correct file name and file type should be used. Asterisk (\*) is the default.

### *fm*

is the file mode of the CMS file. If a file mode is specified, it must be accessed.

## STEM

identifies a simple REXX stem variable that lists TDATA statements that contain search data. For more information, see [“Using the STEM Variable” on page 528](#).

***stemid***

is the name of a REXX stem.

**ASTEM**

identifies an associative REXX stem variable that contains search data. For more information, see [“Associative Stems” on page 529](#).

***astemid***

is the name of an associate REXX stem.

**Options****LOG**

identifies the type of message logging to be done. Messages are logged in the specified message log as well as written to the terminal.

No messages are logged until initial validation of the command is complete.

**Note:** The LOG option is reserved for use by VMSES/E.

***NONE***

is the default. If NONE is specified, messages are written only to the terminal.

***logid***

is a 3-character message log identifier, for example:

***logid*****Type of Log****APP**

The apply message log (\$VMFAPP \$MSGLOG A)

**BLD**

The build message log (\$VMFBLD \$MSGLOG A)

**XYZ**

The user message log (\$VMFXYZ \$MSGLOG A)

**LOGLVL**

identifies the level of message logging to be done.

**Note:** The LOGLVL option is ignored if the LOG option is NONE.

**I**

is the default. ‘I’ logs all informational messages, warning messages, error messages, severe error messages, and terminating errors.

***mlvl***

is the message severity level. Messages are logged in the specified message log if they have a severity level equal to or above the *mlvl* specified. The message levels are shown below, in order of severity. Messages of level ‘R’ have the lowest severity, and messages of level ‘T’ have the highest severity.

***mlvl*****Level of logging****R**

Response required

**I**

Informational Message

**W**

Warning Message

**E**

Error Message

**S**

Severe Error Message

**T**

Terminating Error

**TBLTYPE**

identifies the file type or definition of a table in the Software Inventory that is to be used to process the table that is specified in the command. When you specify the TBLTYPE option, VMFSIM QUERY can process Software Inventory tables with file types other than those usually associated with a Software Inventory table, for example NEWFILE.

If this option is not specified, the VMFSIM QUERY function assumes the file type of the table specified on the command, *tableft*, is defined in the Software Inventory.

***ft***

is the file type to use when processing the table that is specified on the command. For example, if you specify TBLTYPE SYSDESCT, you tell VMFSIM the file is really a system-level description table.

**TYPE**

directs the output to the terminal. TYPE is the default.

**FILE *fn***

directs the output from the query to a CMS file.

***fn***

is the file name of the CMS file containing the output. The file type is SIMDATA, and the file mode is A.

**STEM**

identifies a simple REXX stem variable that will contain the output from the query.

***stemid***

is the name of the REXX stem that contains the output.

**ASTEM**

identifies an associative REXX stem variable that will contain the output from the query.

***astemid***

is the name of the associative REXX stem that contains the output.

**Usage Notes**

1. You can also use the VMFINFO EXEC to query the Software Inventory tables. VMFINFO provides an easy-to-use panel interface. For more information, see [Chapter 17, "Using the VMFINFO Panels," on page 199.](#)

**Examples**

- To use VMFSIM QUERY to query the fields defined for the SRVREQT table, enter:

```
VMFSIM QUERY 1VMVMC23 SRVREQT
```

This command returns all the fields defined for the SRVREQT table. The format of the output follows.

```
:PTF - PTF Number field (KEY)
:APARNUM - APAR(s) contained in the PTF (FIELD)
:PREREQ - Pre-requisite PTF(s) (FIELD)
:COREQ - Co-requisite PTF(s) (FIELD)
:IFREQ - If-requisite PTF(s) (FIELD)
:SUP - Superseded PTF(s) (FIELD)
:HARDREQ - Logical requisite (FIELD)
```

- To use VMFSIM QUERY to return all the pre-requisites for a specific PTF, enter:

```
VMFSIM QUERY 1VMVMC23 SRVREQT TDATA :PTF UV12345 :PREREQ
```

This command returns the pre-requisites for PTF UV12345. The format of the output follows.



```
:PTF UV12345
:PREREQ UV23456 UV56789
```

- To use VMFSIM QUERY to return the status for a specific PTF, enter:

```
VMFSIM QUERY 1VMVMC23 SRVAPPS TDATA :PTF UV12345 :STAT
```

This command returns the status for PTF UV12345. The format of the output follows.

```
:PTF UV12345
:STAT SUPED.09/10/21.MAINT APPLIED.08/01/22.JONES
```

## Input and Output Files

### Input Files

#### *tablefn tableft*

The Software Inventory table.

#### *fn SIMDATA*

An optional input file.

### Output Files

#### *fn SIMDATA*

An optional output file for the returned data.

#### *fn OLDDATA*

The previous level of the *fn SIMDATA* output file.

## Messages and Return Codes

Appendix D, “Module Identifiers for VMSES/E Messages,” on page 749 shows the format of VMSES/E messages and lists the identifiers used by each VMSES/E EXEC.

For a complete explanation of any VMSES/E message, use the HELP Facility to view the message explanation online or see the appropriate messages documentation. To display information about a specific message - VMF002E, for example - enter:

```
help msg vmf002e
```

If you are not familiar with the z/VM HELP Facility, enter `help` to display the main HELP Menu. For information about the HELP command, enter:

```
help cms help
```

Appendix D, “Module Identifiers for VMSES/E Messages,” on page 749 shows the format of VMSES/E messages and lists the identifiers for each VMSES/E EXEC.

The VMFSIM QUERY EXEC issues the following return codes:

Return Code	Explanation
0	Command completed successfully.
2	Command completed successfully but extraneous data was encountered.
4	Command completed successfully but no entries matched the search arguments.
12	Not all input data was processed successfully due to syntax errors in the data.
24	Command failed because of a command line syntax error.
28	Command failed because a required file was not found.

<b>Return Code</b>	<b>Explanation</b>
36	Command failed because a target disk or directory was not available.
100	Command failed because of an external error.

### **Recovery Information**

The VMFSIM QUERY function saves a backup copy of the SIMDATA file that is being updated if it is found on the output disk. The file name of the file is the same as the original file, and the file type is OLDDATA. You can use this file to recover the SIMDATA file to the level of data it contained before the command was run.

After the errors have been corrected, rerun the command to regenerate the SIMDATA file.



**Operands*****reqtablefn***

is the file name of the requisite table to search.

**SRVREQT**

is the file type of the service-level requisite table.

***reqtableft***

is the file type of the requisite table to search.

***reqtablefm***

is the file mode of the requisite table to search. If a file mode is specified, it must be accessed.

**\***

indicates you want to search all file modes. If you specify an asterisk (\*) as the file mode, VMFSIM uses the first file in the search order that matches the specified file name and file type.

***apptablefn***

is the file name of the apply status table to use to determine the status of the PTFs that will be processed.

**=**

tells VMFSIM to use the name that was entered for the requisite table. If an equal sign (=) is used as the file name, the same file name is used for both the requisite table and the apply status table.

**SRVAPPS**

is the file type of the service-level apply status table.

***apptableft***

is the file type of the service-level apply status table to use to determine the status of the PTFs that will be processed.

***apptablefm***

is the file mode of the service-level apply status table to use to determine the status of the PTFs that will be processed. If a file mode is specified, it must be accessed.

**\***

searches all file modes. If an asterisk (\*) is used as the file mode, the first file in the search order that matches the specified file name and file type is used.

**=**

uses the file mode specified for the service-level requisite table. If an equal sign (=) is used as the file mode, the same file mode is used for both the requisite table and the apply status table.

**TDATA**

identifies the search data. You can specify additional tags, but they are ignored.

***:PTF***

is the tag to search for. You must use *:PTF* when you enter a VMFSIM SRVDEP command.

***ptfnum***

is the PTF number to search for. You must enter a PTF number (*ptfnum*) when you enter a VMFSIM SRVDEP command.

**Note:** If you specify a TDATA statement following the FILE, STEM, or ASTEM operand, the tags and data specified are logically added to each TDATA statement contained in the file or stem defined by the FILE, STEM, or ASTEM operand.

**FILE**

identifies a CMS file that lists TDATA statements that contain search data. For more information, see [“Using File Input” on page 527](#).

***fn***

is the file name of the file.

**SIMDATA**

is the default file type.

***ft***

is the file type of the CMS file.

**\***

indicates the first file found in the search order with the correct file name and file type should be used. Asterisk (\*) is the default.

***fm***

is the file mode of the file. If a file mode is specified, it must be accessed.

**STEM**

identifies a simple REXX stem variable that lists TDATA statements that contain search data. For more information, see [“Using the STEM Variable” on page 528](#).

***stemid***

is the name of a REXX stem.

**ASTEM**

identifies an associative REXX stem variable that contains search data. For more information, see [“Associative Stems” on page 529](#).

***astemid***

is the name of an associate REXX stem.

**Options****LASTAPP**

returns all dependents that have a status of APPLIED. The default is LASTAPP.

**BASELVL**

returns all dependents regardless of status.

**LOG**

identifies the type of message logging to be done. Messages are logged in the specified message log as well as written to the terminal.

No messages are logged until initial validation of the command is complete.

**Note:** The LOG option is reserved for use by VMSES/E.

**NONE**

is the default. If NONE is specified, messages are written only to the terminal.

***logid***

is a three-character message log identifier, for example:

***logid*****Type of Log****APP**

The apply message log (\$VMFAPP \$MSGLOG A)

**BLD**

The build message log (\$VMFBLD \$MSGLOG A)

**XYZ**

The user message log (\$VMFXYZ \$MSGLOG A)

**LOGLVL**

identifies the level of message logging to be done.

**Note:** The LOGLVL option is ignored if the LOG option is NONE.

**I**

is the default. ‘I’ logs all informational messages, warning messages, error messages, severe error messages, and terminating errors.

***mlvl***

is the message severity level. Messages are logged in the specified message log if they have a severity level equal to or above the *mlvl* specified. The message levels are shown below, in order

of severity. Messages of level 'R' have the lowest severity, and messages of level 'T' have the highest severity.

***mlvl***

**Level of logging**

**R**

Response required

**I**

Informational Message

**W**

Warning Message

**E**

Error Message

**S**

Severe Error Message

**T**

Terminating Error

**TYPE**

directs the output to the terminal. TYPE is the default.

**APPLIST**

directs the output to a CMS file in the format required by VMFAPPLY.

**Note:** Only PTF numbers defined on :PTF tags are added to the apply list.

***fn***

is the file name of the CMS file. The file type is \$APPLIST, and the file mode is A.

**FILE**

directs the output to a CMS file.

***fn***

is the file name of the CMS file. The file type is SIMDATA, and the file mode is A.

**STEM**

identifies a simple REXX stem variable that will contain the output.

***stemid***

is the name of a REXX stem.

**Usage Notes**

1. If the specified apply status table does not exist, VMFSIM assumes nothing has been applied.

**Examples**

- Determining Dependent PTFs

You can use the VMFSIM SRVDEP function with the service-level requisite table and service-level apply status table to return the dependents for the PTFs specified. By definition, a dependent PTF is applied. To determine which PTFs are dependent on a specific PTF that you want to remove from the product, enter:

```
vmfsim srvdep 5700abcd srvreqt * = srvapps * tdata :ptf um15010
```

You receive this response:

```
VMFSPC2480I Results for TDATA :PTF UM15010
:PTF UM15010
:DEPS UM15020
:SUPBY *NONE*
:OUTREQS *NONE*
```

The response includes:

**:PTF**

the PTF number being processed.

**:DEPS**

a list of the PTFs that are dependent on the PTF specified.

**:SUPBY**

a list of the PTFs that supersede this PTF.

**:OUTREQS**

a list of the PTFs in another product that are dependent on the PTF specified.

## Input and Output Files

### Input Files

***reqtablefn reqtableft***

The requisite table.

***prodid SRVREQT***

The service-level requisite table.

***apptablefn apptableft***

The system-level apply status table.

***appid SRVAPPS***

The service-level apply status table.

***fn SIMDATA | ft***

An optional input file.

### Output Files

***fn \$APPLIST***

An apply list.

***fn SIMDATA***

An optional output file for the returned data.

***fn OLDDATA***

The previous level of the *fn SIMDATA* output file.

## Messages and Return Codes

Appendix D, “Module Identifiers for VMSES/E Messages,” on page 749 shows the format of VMSES/E messages and lists the identifiers used by each VMSES/E exec.

For a complete explanation of any VMSES/E message, use the HELP Facility to view the message explanation online or see the appropriate messages documentation. To display information on a specific message, for example VMF002E, enter:

```
help msg vmf002e
```

If you are unfamiliar with the z/VM HELP Facility, enter `help` to display the main HELP Menu. For information on the HELP command, enter:

```
help cms help
```

The VMFSIM SRVDEP EXEC issues the following return codes:

Return Code	Explanation
0	Command completed successfully.
2	Command completed successfully but extraneous data was encountered.
4	PTF or PRODID was not found in tables.

<b>Return Code</b>	<b>Explanation</b>
12	Not all input data was processed successfully due to syntax errors in the data.
24	Command failed because of a command line syntax error.
28	Command failed because a required file was not found.
36	Command failed because a target disk or directory was not available.
100	Command failed because of an external error.

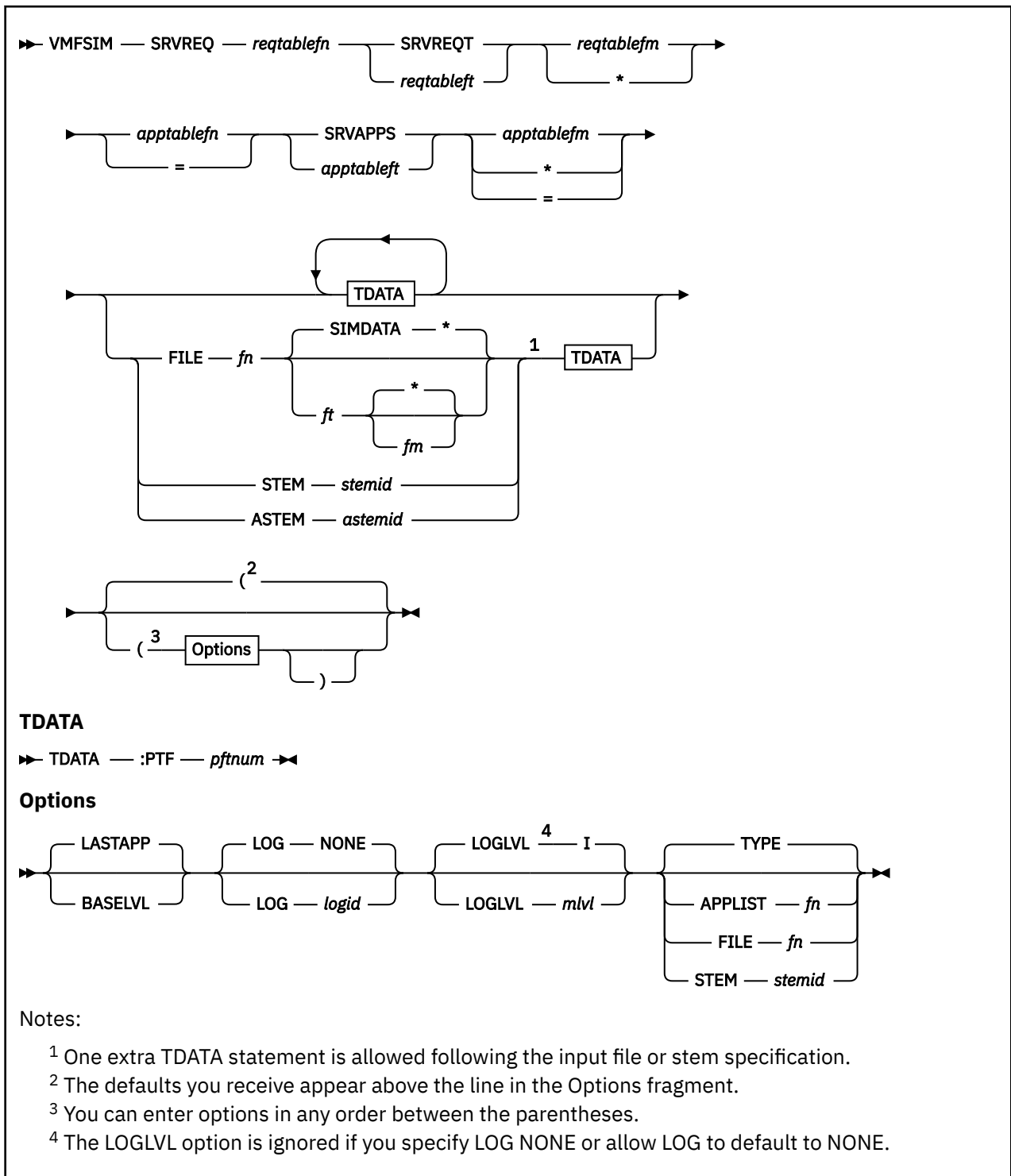
### **Recovery Information**

The VMFSIM SRVDEP function saves a backup copy of the SIMDATA file that is being updated if it is found on the output disk. The file name of the file is the same as the original file, and the file type is OLDDATA. You can use this file to recover the SIMDATA file to the level of data it contained before the command was run.

After the errors have been corrected, rerun the command to regenerate the SIMDATA file.



# VMFSIM SRVREQ



## Purpose

The VMFSIM SRVREQ function accesses the service-level requisite table and the service-level apply status table and returns the requisites for the specified PTFs. You can use this command to determine if you have received and applied all the PTFs that are required by a PTF that you want to install.

**Operands*****reqtablefn***

is the file name of the requisite table to search.

**SRVREQT**

is the file type of the service-level requisite table.

***reqtableft***

is the file type of the requisite table to search.

***reqtablefm***

is the file mode of the requisite table to search. If a file mode is specified, it must be accessed.

**\***

searches all file modes. If you enter an asterisk (\*) as the file mode, VMFSIM uses the first file in the search order that matches the specified file name and file type.

***apptablefn***

is the file name of the service-level apply status table to use to determine the status of the PTFs that will be processed.

**=**

tells VMFSIM to use the name that was entered for the service-level requisite table. If an equal sign (=) is used as the file name, the same file name is used for both the requisite table and the apply status table.

**SRVAPPS**

is the file type of the service-level apply status table.

***apptableft***

is the file type of the service-level apply status table to use to determine the status of the PTFs that will be processed.

***apptablefm***

is the file mode of the service-level apply status table to use to determine the status of the PTFs that will be processed. If a file mode is specified, it must be accessed.

**\***

searches all file modes. If an asterisk (\*) is used as the file mode, VMFSIM uses the first file in the search order that matches the specified file name and file type.

**=**

uses the file mode specified for the requisite table. If an equal sign (=) is used as the file mode, the same file mode is used for both the requisite table and the apply status table.

**TDATA**

identifies the search data. You can specify additional tags, but they are ignored.

**:PTF**

is the tag to search for. You must use :PTF when you enter a VMFSIM SRVREQ command.

***ptfnum***

is the PTF number to search for. You must enter a PTF number (*ptfnum*) when you enter a VMFSIM SRVREQ command.

**Note:** If you specify a TDATA statement following the FILE, STEM, or ASTEM operand, it will be added to the end of the list of PTFs or products in the file or stem defined by the FILE, STEM, or ASTEM operand.

**FILE**

identifies a CMS file that lists TDATA statements that contain search data. For more information, see [“Using File Input” on page 527](#).

***fn***

is the file name of the file.

**SIMDATA**

is the default file type.

***ft***

is the file type of the CMS file that contains the tags and data.

**\***

indicates the first file found in the search order with the correct file name and file type should be used. Asterisk (\*) is the default.

***fm***

is the file mode of the file. If a file mode is specified, it must be accessed.

**STEM**

identifies a simple REXX stem variable that lists TDATA statements that contain search data. For more information, see [“Using the STEM Variable” on page 528](#).

***stemid***

is the name of a REXX stem.

**ASTEM**

identifies an associative REXX stem variable that contains search data. For more information, see [“Associative Stems” on page 529](#).

***astemid***

is the name of an associative REXX stem.

**Options****LASTAPP**

indicates the requisites search should stop when a requisite has a status of APPLIED or SUPED. LASTAPP is the default.

**BASELVL**

indicates the requisites search should continue when a requisite has a status of APPLIED or SUPED. This causes all the requisites to be collected back to the base level of the product.

**LOG**

identifies the type of message logging to be done. Messages are logged in the specified message log as well as written to the terminal.

No messages are logged until initial validation of the command is complete.

**Note:** The LOG option is reserved for use by VMSES/E.

**NONE**

is the default. If NONE is specified, messages are written only to the terminal.

***logid***

is a three-character message log identifier, for example:

***logid*****Type of Log****APP**

The apply message log (\$VMFAPP \$MSGLOG A)

**BLD**

The build message log (\$VMFBLD \$MSGLOG A)

**XYZ**

The user message log (\$VMFXYZ \$MSGLOG A)

**LOGLVL**

identifies the level of message logging to be done.

**Note:** The LOGLVL option is ignored if the LOG option is NONE.

**I**

is the default. ‘I’ logs all informational messages, warning messages, error messages, severe error messages, and terminating errors.

***mlvl***

is the message severity level. Messages are logged in the specified message log if they have a severity level equal to or above the *mlvl* specified. The message levels are shown below, in order of severity. Messages of level 'R' have the lowest severity, and messages of level 'T' have the highest severity.

***mlvl***

**Level of logging**

**R**

Response required

**I**

Informational Message

**W**

Warning Message

**E**

Error Message

**S**

Severe Error Message

**T**

Terminating Error

**TYPE**

directs the output to the terminal. TYPE is the default.

**APPLIST**

directs the output to an apply list that can later be used by VMFAPPLY.

**Note:** Only PTF numbers defined on :PTF tags are added to the apply list.

***fn***

is the file name of the apply list. The file type is \$APPLIST, and the file mode is A.

**FILE**

directs the output to a CMS file.

***fn***

is the file name of the CMS file. The file type is SIMDATA, and the file mode is A.

**STEM**

identifies a simple REXX stem variable that will contain the output.

***stemid***

is the name of a REXX stem.

**Usage Notes**

1. If the specified apply status table does not exist, VMFSIM assumes nothing has been applied.

**Examples**

- Determining Requisite PTFs

You can use the VMFSIM SRVREQ function with the service-level requisite table and service-level apply status table to return the requisites for the PTFs specified.

To determine if you have all the PTFs (received and applied) that are required by a PTF you want to install, enter:

```
vmfsim srvreq 5700abcd srvreqt * = srvapps * tdata :ptf um18135
```

You receive this response:

```

VMFSIP2480I RESULTS FOR
          TDATA :PTF UM18135
:PTF UM18135
  :PREREQ UM18109
  :HARDREQ VM47104
  :SUBREQ UM18082
  :SUBIF * NONE *
  :SUBHARDREQ * NONE *

```

The response can include:

**:PTF**

the PTF number being processed.

**:PREREQ**

a list of the PTFs in this product or another product that must be installed before installing this PTF. PREREQs satisfy the service chain.

**:COREQ**

a list of the PTFs in this product or another product that must be installed before running this product with the PTF installed. COREQs satisfy the service chain.

**:SUP**

a list of the PTFs this PTF supersedes, or is a functional replacement of.

**:IFREQ**

a list of the PTFs in another product that are required if the product is installed.

**:HARDREQ**

a list of the APARs that are required to be installed for this PTF to function. These are a subset of the PREREQs that have real code intersections or functional dependencies. HARDREQs satisfy functional requisites.

**:SUBREQ**

a list of the PTFs in this product or another product that are required by the requisites identified above.

**:SUBIF**

a list of the PTFs in another product that are required by the requisites identified above if the product identified in the condition is installed.

**:SUBHARDREQ**

a list of the APARs that are required to be installed by the PTFs listed above for them to function correctly.

**Note:** The fields :SUBREQ, :SUBIF, and :SUBHARDREQ are not in the actual table, they are output only from the VMFSIM SRVREQ command.

## Input and Output Files

### Input Files

***reqtablefn reqtableft***

The requisite table.

***prodid SRVREQT***

The service-level requisite table.

***apptablefn apptableft***

The apply status table.

***appid SRVAPPS***

The service-level apply status table.

***fn SIMDATA | ft***

An optional input file.

### Output Files

***fn \$APPLIST***

An apply list.

***fn SIMDATA***

An optional output file for the returned data.

***fn OLDDATA***

The previous level of the *fn SIMDATA* output file.

**Messages and Return Codes**

Appendix D, “Module Identifiers for VMSES/E Messages,” on page 749 shows the format of VMSES/E messages and lists the identifiers used by each VMSES/E exec.

For a complete explanation of any VMSES/E message, use the HELP Facility to view the message explanation online or see the appropriate messages documentation.

To display information on a specific message, for example VMF002E, enter:

```
help msg vmf002e
```

If you are unfamiliar with the z/VM HELP Facility, enter help to display the main HELP Menu. For information on the HELP command, enter:

```
help cms help
```

The VMFSIM SRVREQ EXEC issues the following return codes:

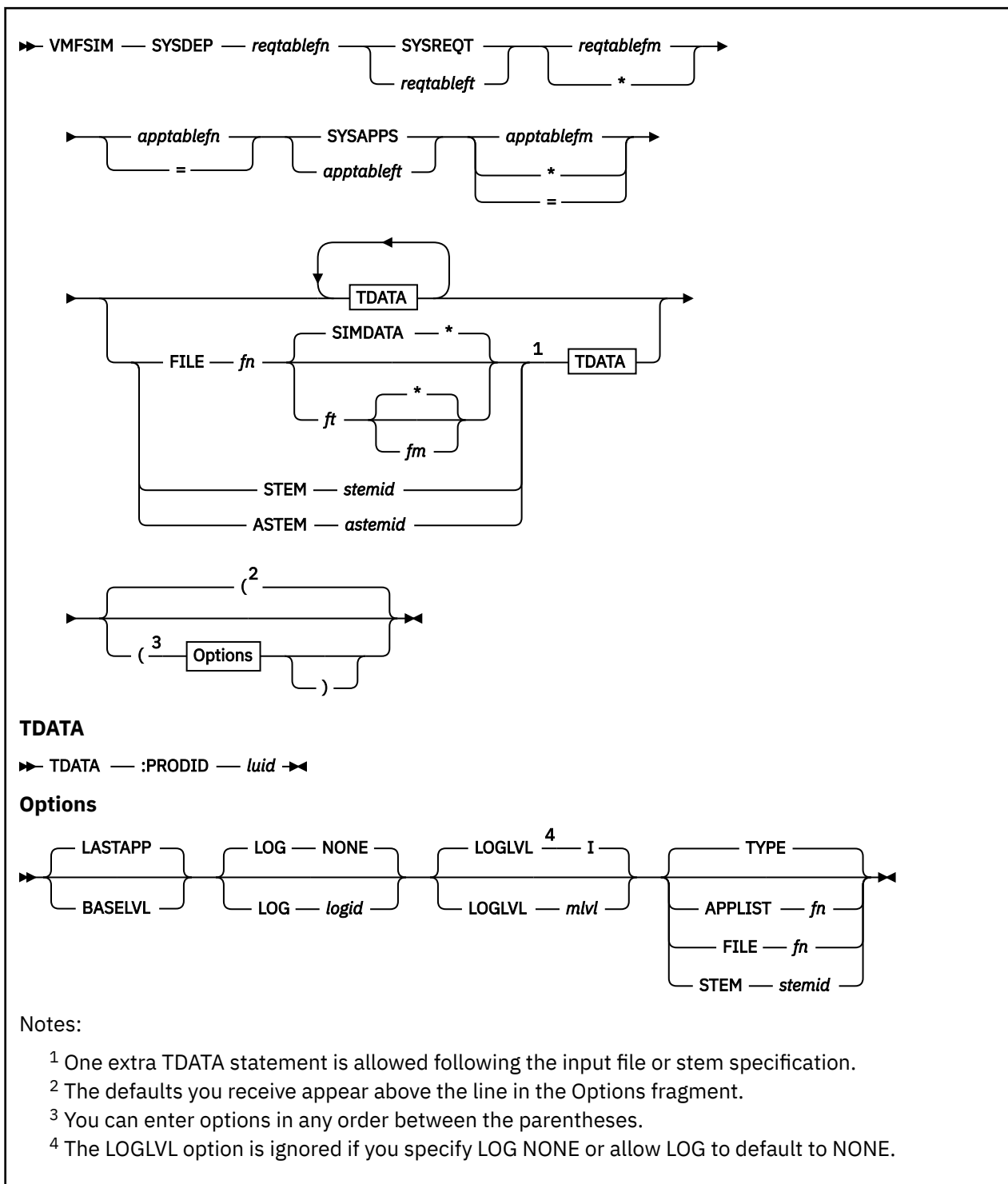
Return Code	Explanation
0	Command completed successfully.
2	Command completed successfully but extraneous data was encountered.
4	PTF or PRODID was not found in tables.
12	Not all input data was processed successfully due to syntax errors in the data.
24	Command failed because of a command line syntax error.
28	Command failed because a required file was not found.
36	Command failed because a target disk or directory was not available.
100	Command failed because of an external error.

**Recovery Information**

The VMFSIM SRVREQ function saves a backup copy of the SIMDATA file that is being updated if it is found on the output disk. The file name of the file is the same as the original file, and the file type is OLDDATA. You can use this file to recover the SIMDATA file to the level of data it contained before the command was run.

After the errors have been corrected, rerun the command to regenerate the SIMDATA file.

# VMFSIM SYSDEP



## Purpose

The VMFSIM SYSDEP command accesses the the system-level requisite and apply status tables and returns all the dependents that have been applied.

You can use this command to determine which products depend on a specific product before you remove it from the system.

## Operands

### ***reqtablefn***

is the file name of the requisite table to search.

### **SYSREQT**

is the file type of the system-level requisite table.

### ***reqtableft***

is the file type of the requisite table to search.

### ***reqtablefm***

is the file mode of the requisite table to search. If a file mode is specified, it must be accessed.

**\***

indicates you want to search all file modes. If you specify an asterisk (\*) as the file mode, VMFSIM uses the first file in the search order that matches the specified file name and file type.

### ***apptablefn***

is the file name of the apply status table to use to determine the status of the products that will be processed.

**=**

tells VMFSIM to use the name that was entered for the requisite table. If an equal sign (=) is used as the file name, the same file name is used for both the requisite table and the apply status table.

### **SYSAPPS**

is the file type of the system-level apply status table.

### ***apptableft***

is the file type of the system-level apply status table to use to determine the status of the products that will be processed.

### ***apptablefm***

is the file mode of the system-level apply status table to use to determine the status of the products that will be processed. If a file mode is specified, it must be accessed.

**\***

searches all file modes. If an asterisk (\*) is used as the file mode, the first file in the search order that matches the specified file name and file type is used.

**=**

uses the file mode specified for the system-level requisite table. If an equal sign (=) is used as the file mode, the same file mode is used for both the requisite table and the apply status table.

## **TDATA**

identifies the search data. You can specify additional tags, but they are ignored.

### **:PRODID**

is the tag to search for. You must use :PRODID when you enter a VMFSIM SYSDEP command.

### ***luid***

is the loadable unit identifier to search for. You must enter a loadable unit identifier (*luid*) when you enter a VMFSIM SYSDEP command. The format of *luid* is *prodid%compname*.

**Note:** If you specify a TDATA statement following the FILE, STEM, or ASTEM operand, the tags and data specified are logically added to each TDATA statement contained in the file or stem defined by the FILE, STEM, or ASTEM operand.

## **FILE**

identifies a CMS file that lists TDATA statements that contain search data. For more information, see [“Using File Input” on page 527](#).

### ***fn***

is the file name of the file.



**SIMDATA**

is the default file type.

***ft***

is the file type of the CMS file.

**\***

indicates the first file found in the search order with the correct file name and file type should be used. Asterisk (\*) is the default.

***fm***

is the file mode of the file. If a file mode is specified, it must be accessed.

**STEM**

identifies a simple REXX stem variable that lists TDATA statements that contain search data. For more information, see [“Using the STEM Variable” on page 528](#).

***stemid***

is the name of a REXX stem.

**ASTEM**

identifies an associative REXX stem variable that contains search data. For more information, see [“Associative Stems” on page 529](#).

***astemid***

is the name of an associate REXX stem.

**Options****LASTAPP**

returns all dependents that have a status of APPLIED. The default is LASTAPP.

**BASELVL**

returns all dependents regardless of status.

**LOG**

identifies the type of message logging to be done. Messages are logged in the specified message log as well as written to the terminal.

No messages are logged until initial validation of the command is complete.

**Note:** The LOG option is reserved for use by VMSES/E.

**NONE**

is the default. If NONE is specified, messages are written only to the terminal.

***logid***

is a three-character message log identifier, for example:

***logid*****Type of Log****APP**

The apply message log (\$VMFAPP \$MSGLOG A)

**BLD**

The build message log (\$VMFBLD \$MSGLOG A)

**XYZ**

The user message log (\$VMFXYZ \$MSGLOG A)

**LOGLVL**

identifies the level of message logging to be done.

**Note:** The LOGLVL option is ignored if the LOG option is NONE.

**I**

is the default. ‘I’ logs all informational messages, warning messages, error messages, severe error messages, and terminating errors.

***mlvl***

is the message severity level. Messages are logged in the specified message log if they have a severity level equal to or above the *mlvl* specified. The message levels are shown below, in order of severity. Messages of level 'R' have the lowest severity, and messages of level 'T' have the highest severity.

***mlvl***

**Level of logging**

**R**

Response required

**I**

Informational Message

**W**

Warning Message

**E**

Error Message

**S**

Severe Error Message

**T**

Terminating Error

**TYPE**

directs the output to the terminal. TYPE is the default.

**APPLIST**

directs the output to a CMS file in the format required by VMFINS INSTALL LIST.

***fn***

is the file name of the CMS file. The file type is \$APPLIST, and the file mode is A.

**FILE**

directs the output to a CMS file.

***fn***

is the file name of the CMS file. The file type is SIMDATA, and the file mode is A.

**STEM**

identifies a simple REXX stem variable that will contain the output.

***stemid***

is the name of a REXX stem.

**Usage Notes**

1. If the specified apply status table does not exist, VMFSIM assumes nothing has been applied.

**Examples**

Determining Dependent Products

You can use the VMFSIM SYSDEP command with the system-level requisite table and system-level apply status table to return the dependents for the specified products.

To determine which products are dependent on a specific product that you want to remove from the system, enter:

```
vmfsim sysdep vm sysreqt * = sysapps * tdata :prodid 1VMVMC23%MYCOMP
```

You receive this response:

```
VMFSIP2480I RESULTS FOR
          TDATA :PRODID 1VMVMC23%MYCOMP
:PRODID 1VMVMC23%MYCOMP
```

```
:DEPS prod1, prod2, prod3, prod4
      prod5, prod6
:DREQDEPS prod7, prod8
:SUPBY *NONE*
```

The response includes:

**:PRODID**

the name of the product that was processed.

**:DEPS**

a list of the products that are dependent on the product specified.

**:DREQDEPS**

a list of the products that are dependent features of the product specified on the command.

**:SUPBY**

the name of a product that supersedes this product.

## Input and Output Files

### Input Files

***reqtablefn reqtableft***

The requisite table.

***sysid SYSREQT***

The system-level requisite table.

***apptablefn apptableft***

The apply status table.

***sysid SYSAPPS***

The system-level apply status table.

***fn SIMDATA | ft***

An optional input file.

### Output Files

***fn \$APPLIST***

An apply list.

***fn SIMDATA***

An optional output file for the returned data.

***fn OLDDATA***

The previous level of the *fn SIMDATA* output file.

## Messages and Return Codes

Appendix D, “Module Identifiers for VMSES/E Messages,” on page 749 shows the format of VMSES/E messages and lists the identifiers used by each VMSES/E exec.

For a complete explanation of any VMSES/E message, use the HELP Facility to view the message explanation online or see the appropriate messages documentation. To display information on a specific message, for example VMF002E, enter:

```
help msg vmf002e
```

If you are unfamiliar with the z/VM HELP Facility, enter `help` to display the main HELP Menu. For information on the HELP command, enter:

```
help cms help
```

The VMFSIM SYSDEP EXEC issues the following return codes:

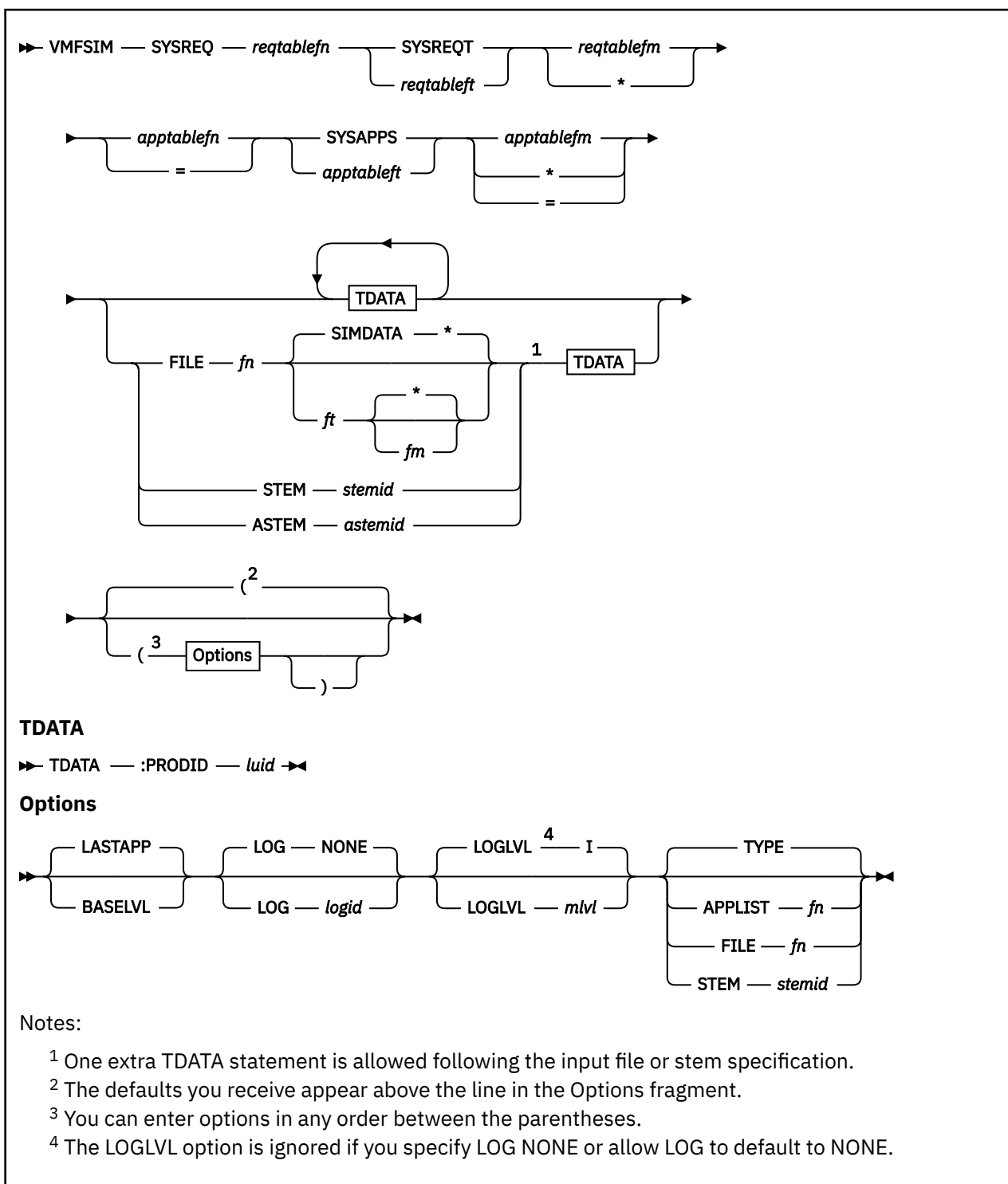
Return Code	Explanation
0	Command completed successfully.
2	Command completed successfully but extraneous data was encountered.
4	PTF or PRODID was not found in tables.
12	Not all input data was processed successfully due to syntax errors in the data.
24	Command failed because of a command line syntax error.
28	Command failed because a required file was not found.
36	Command failed because a target disk or directory was not available.
100	Command failed because of an external error.

### Recovery Information

The VMFSIM SYSDEP function saves a backup copy of the SIMDATA file that is being updated if it is found on the output disk. The file name of the file is the same as the original file, and the file type is OLDDATA. You can use this file to recover the SIMDATA file to the level of data it contained before the command was run.

After the errors have been corrected, rerun the command to regenerate the SIMDATA file.

# VMFSIM SYSREQ



## Purpose

The VMFSIM SYSREQ function accesses the the system-level requisite table and the system-level apply status table and returns the requisites for the specified products. You can use this command to determine if you have all the products installed that are required by another product that you want to install.

## Operands

### ***reqtablefn***

is the file name of the requisite table to search.

### **SYSREQT**

is the file type of the system-level requisite table.

### ***reqtableft***

is the file type of the system-level requisite table to search.

### ***reqtablefm***

is the file mode of the system-level requisite table to search. If a file mode is specified, it must be accessed.

\*

searches all file modes. If you enter an asterisk (\*) as the file mode, VMFSIM uses the first file in the search order that matches the specified file name and file type.

### ***apptablefn***

file name of the apply status table to use to determine the status of the products that will be processed.

=

tells VMFSIM to use the name that was entered for the system-level requisite table. If an equal sign (=) is used as the file name, the same file name is used for both the requisite table and the apply status table.

### **SYSAPPS**

is the file type of the system-level apply status table.

### ***apptableft***

file type of the system-level apply status table to use to determine the status of the products that will be processed.

### ***apptablefm***

file mode of the system-level apply status table to use to determine the status of the products that will be processed. If a file mode is specified, it must be accessed.

\*

searches all file modes. If an asterisk (\*) is used as the file mode, VMFSIM uses the first file in the search order that matches the specified file name and file type.

=

uses the file mode specified for the system-level requisite table. If an equal sign (=) is used as the file mode, the same file mode is used for both the requisite table and the apply status table.

### **TDATA**

identifies the search data. You can specify additional tags, but they are ignored.

#### **:PRODID**

is the tag to search for. You must use :PRODID when you enter a VMFSIM SYSREQ command.

#### ***luid***

is the loadable unit identifier to search for. You must enter a loadable unit identifier (*luid*) when you enter a VMFSIM SYSREQ command. The format of *luid* is *prodid%compname*.

**Note:** If you specify a TDATA statement following the FILE, STEM, or ASTEM operand, it will be added to the end of the list of PTFs or products in the file or stem defined by the FILE, STEM, or ASTEM operand.

### **FILE**

identifies a CMS file that lists TDATA statements that contain search data. For more information, see [“Using File Input” on page 527](#).

#### ***fn***

is the file name of the file.

**SIMDATA**

is the default file type.

***ft***

is the file type of the CMS file that contains the tags and data.

**\***

indicates the first file found in the search order with the correct file name and file type should be used. Asterisk (\*) is the default.

***fm***

is the file mode of the file. If a file mode is specified, it must be accessed.

**STEM**

identifies a simple REXX stem variable that lists TDATA statements that contain search data. For more information, see [“Using the STEM Variable” on page 528](#).

***stemid***

is the name of a REXX stem.

**ASTEM**

identifies an associative REXX stem variable that contains search data. For more information, see [“Associative Stems” on page 529](#).

***astemid***

is the name of an associative REXX stem.

**Options****LASTAPP**

indicates the requisites search should stop when a requisite has a status of APPLIED or SUPED. LASTAPP is the default.

**BASELVL**

indicates the requisites search should continue when a requisite has a status of APPLIED or SUPED. This causes all the requisites to be collected back to the base level of the product.

**LOG**

identifies the type of message logging to be done. Messages are logged in the specified message log as well as written to the terminal.

No messages are logged until initial validation of the command is complete.

**Note:** The LOG option is reserved for use by VMSES/E.

**NONE**

is the default. If NONE is specified, messages are written only to the terminal.

***logid***

is a three-character message log identifier, for example:

***logid*****Type of Log****APP**

The apply message log (\$VMFAPP \$MSGLOG A)

**BLD**

The build message log (\$VMFBLD \$MSGLOG A)

**XYZ**

The user message log (\$VMFXYZ \$MSGLOG A)

**LOGLVL**

identifies the level of message logging to be done.

**Note:** The LOGLVL option is ignored if the LOG option is NONE.

**I**

is the default. 'I' logs all informational messages, warning messages, error messages, severe error messages, and terminating errors.

***mlvl***

is the message severity level. Messages are logged in the specified message log if they have a severity level equal to or above the *mlvl* specified. The message levels are shown below, in order of severity. Messages of level 'R' have the lowest severity, and messages of level 'T' have the highest severity.

***mlvl***

**Level of logging**

**R**

Response required

**I**

Informational Message

**W**

Warning Message

**E**

Error Message

**S**

Severe Error Message

**T**

Terminating Error

**TYPE**

directs the output to the terminal. TYPE is the default.

**APPLIST**

directs the output to an apply list that can later be used by VMFINS INSTALL LIST.

***fn***

is the file name of the apply list. The file type is \$APPLIST, and the file mode is A.

**FILE**

directs the output to a CMS file.

***fn***

is the file name of the CMS file. The file type is SIMDATA, and the file mode is A.

**STEM**

identifies a simple REXX stem variable that will contain the output.

***stemid***

is the name of a REXX stem.

**Usage Notes**

1. If the specified apply status table does not exist, VMFSIM assumes nothing has been applied.

**Examples**

Determining Product Requisites

You can use the VMFSIM SYSREQ command with the system-level requisite table and system-level apply status table to return the requisites for specified products.

To determine if you have all the products installed on your system that are required by a product you want to install, enter:

```
vmfsim sysreq vm sysreqt * = sysapps * tdata :prodid 1VMVMC23%MYCOMP
```

You receive this response:



```

VMFSIP2480I RESULTS FOR
          TDATA :PRODID 1VMVMC23%MYCOMP
:PRODID 1VMVMC23%MYCOMP
:PREREQ 1VMVMP11
:REQ *NONE*
:DREQ 1VMVME10
:SUP *NONE*
:IFREQ *NONE*
:NPRE *NONE*
:SUBREQ 1VMVMS10
:SUBIF *NONE*
:PTFREQS *NONE*

```

The response includes:

**:PRODID**

the name of the product that was processed.

**:PREREQ**

a list of the products that must be installed before installing this product.

**:REQ**

a list of the products that must be installed before running this product.

**:DREQ**

a list of the products that this product is a dependent feature of.

**:SUP**

a list of the products that this product supersedes or is a functional replacement of.

**:IFREQ**

a list of the products that are required, if the product identified in the condition is installed.

**:NPRE**

a list of the products that cannot be installed if this product is installed.

**:SUBREQ**

a list of the products that are required to be installed by the requisites identified above.

**:SUBIF**

a list of the products that are required to be installed by the requisites identified above if the product identified in the condition is installed.

**:PTFREQS**

PTFs that must be installed to products that are requisites.

## Input and Output Files

### Input Files

***reqtablefn reqtableft***

The requisite table.

***sysid SYSREQT***

The system-level requisite table.

***apptablefn apptableft***

The apply status table.

***sysid SYSAPPS***

The system-level apply status table.

***fn SIMDATA | ft***

An optional input file.

### Output Files

***fn \$APPLIST***

An apply list.

***fn SIMDATA***

An optional output file for the returned data.

***fn* OLDDATA**

The previous level of the *fn* SIMDATA output file.

**Messages and Return Codes**

Appendix D, “Module Identifiers for VMSES/E Messages,” on page 749 shows the format of VMSES/E messages and lists the identifiers used by each VMSES/E exec.

For a complete explanation of any VMSES/E message, use the HELP Facility to view the message explanation online or see the appropriate messages documentation.

To display information on a specific message, for example VMF002E, enter:

```
help msg vmf002e
```

If you are unfamiliar with the z/VM HELP Facility, enter `help` to display the main HELP Menu. For information on the HELP command, enter:

```
help cms help
```

The VMFSIM SYSREQ EXEC issues the following return codes:

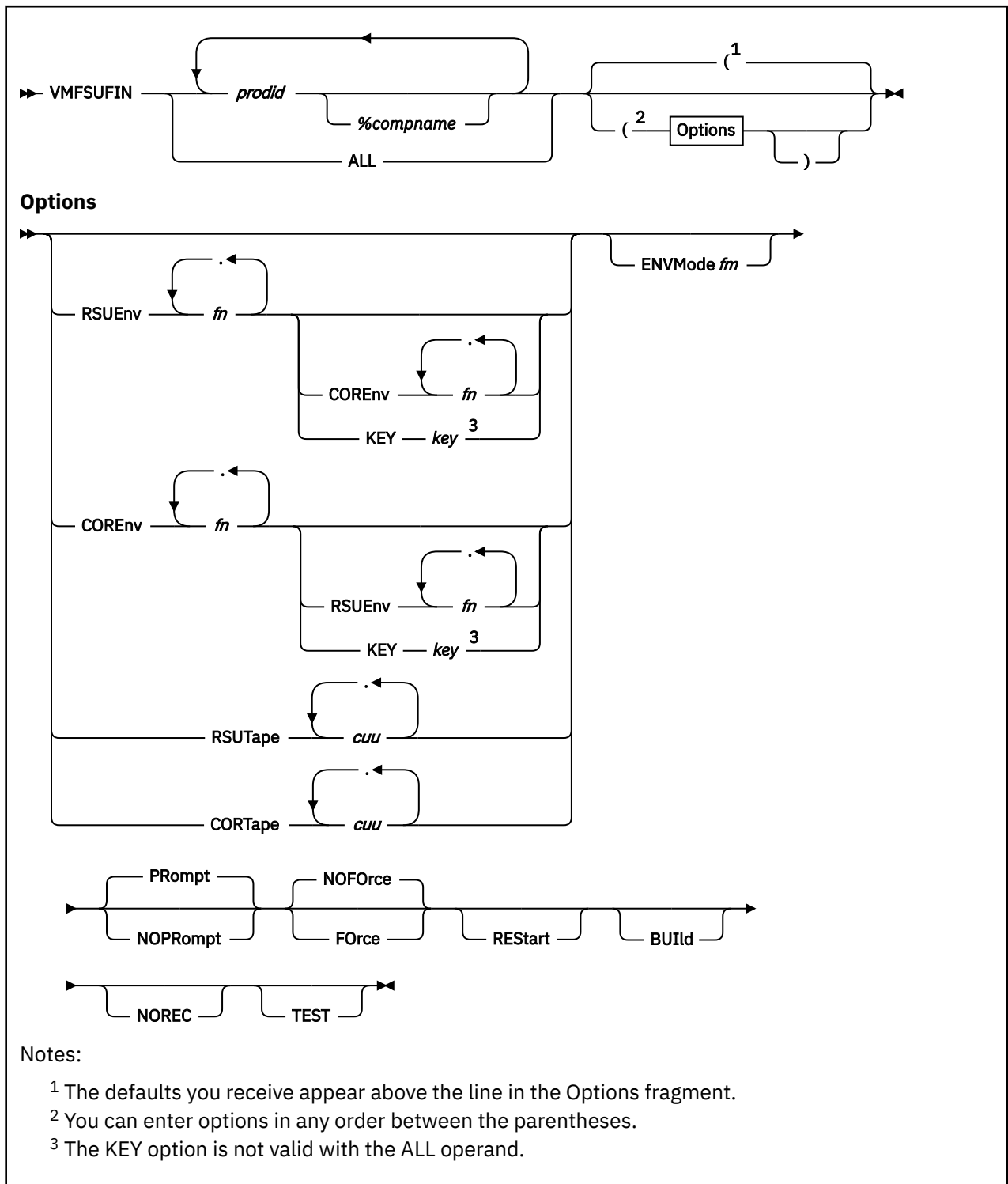
Return Code	Explanation
0	Command completed successfully.
2	Command completed successfully but extraneous data was encountered.
4	PTF or PRODID was not found in tables.
12	Not all input data was processed successfully due to syntax errors in the data.
24	Command failed because of a command line syntax error.
28	Command failed because a required file was not found.
36	Command failed because a target disk or directory was not available.
100	Command failed because of an external error.

**Recovery Information**

The VMFSIM SYSREQ function saves a backup copy of the SIMDATA file that is being updated if it is found on the output disk. The file name of the file is the same as the original file, and the file type is OLDDATA. You can use this file to recover the SIMDATA file to the level of data it contained before the command was run.

After the errors have been corrected, rerun the command to regenerate the SIMDATA file.

# VMFSUFIN EXEC



## Purpose

The VMFSUFIN EXEC installs service from RSU service envelope files, COR service envelope files, RSU service tapes, or COR service tapes. If you use service envelope files, you can install RSU and COR service with one invocation of VMFSUFIN.

## Operands

### *prodid*

is the 7-8 character alphanumeric identifier assigned to the product by IBM (for example, 1VMVMC23).

### *%compname*

is the component name preceded by a percent sign (%), for example %CMS. *compname* is a 1-16 character alphanumeric identifier.

### **ALL**

indicates service is to be installed for all products on the selected RSU (or COR, if no RSU is selected).

## Options

### **RSUEnv**

indicates an RSU envelope is to be installed.

### *fn*

is the file name of an RSU envelope. The file type must be SERVLINK.

.

joins multiple file names (no blanks) for multivolume RSU envelopes (for example, RPTF1234.RPTF1235).

### **COREnv**

indicates a COR envelope is to be installed.

### *fn*

is the file name of a COR envelope. The file type must be SERVLINK.

.

joins multiple file names (no blanks) for multivolume COR envelopes (for example, VPTF5678.VPTF5679).

### **ENVMode**

indicates a disk must be retained when the envelopes are installed.

### *fm*

is the file mode of the disk that contains the envelopes.

### **RSUTape**

indicates an RSU tape is to be installed.

### *cuu*

is the address of the tape drive where the RSU tape is to be mounted

.

joins multiple addresses (no blanks) for multivolume RSU tapes (for example, 181.182).

### **CORTape**

indicates a COR tape is to be installed.

### *cuu*

is the address of the tape drive where the COR tape is to be mounted

.

joins multiple addresses (no blanks) for multivolume COR tapes (for example, 181.181).

### **KEY**

indicates this call is one of two that pass the data required to install the service for these products.

### *key*

is the character string that identifies the two calls required to pass the data for these products.

**Prompt**

calls VMFSETUP with the PROMPT option. Therefore, the user will be prompted for link passwords, if necessary.

**NOPrompt**

calls VMFSETUP with the NOPROMPT option. Therefore, the user will not be prompted for link passwords. If a link password is required, VMFSUFIN fails.

**NOForce**

may install or not install service based on the setting of the :INSTALL tag in the VM SYSSUF table.

**Force**

installs service regardless of the setting of the :INSTALL tag in the VM SYSSUF table.

**REstart**

restarts a previous call to VMFSUFIN. The ENVMODE and KEY options, as well as any specified envelope names, are ignored when RESTART is specified.

**BUILD**

builds serviced parts by invoking VMFBLD with the SERVICED option.

**NOREC**

skips the receive step and invokes VMFAPPLY and VMFBLD.

**TEST**

invokes VMFAPPLY with the TEST option and skips VMFBLD.

**Usage Notes**

1. To run VMFSUFIN manually, you must first invoke the VMFSUFTB command to build the SYSSUF table. You can then customize the SYSSUF table using the VMFSIM MODIFY or the VMFUPDAT SYSSUF command.
2. VMFSUFIN uses the :INSTALL and :BUILD tags in the SYSSUF table to determine whether to install service for a product and whether to build the installed service.
3. VMFSUFIN uses the :INSPPF and :BLDPPF tags in the SYSSUF table to obtain the PPF names and components names used to install service for a product and to build the installed service.
4. After using VMFSUFIN to install and build service, you must refer to the product's service instructions to complete the service process. VMFSUFIN automatically completes the service procedures by issuing the VMFBLD command with the SERVICED option. Any further service procedures, such as placing into production or customization, must be performed manually.
5. If the NOPROMPT option is specified, you must have the appropriate LINK authority to all disks listed in the :MDA section of the PPF for all products specified on the command. The :DCL section of the PPF defines the authority for all the disks. If the NOPROMPT option is not specified, you will be prompted for the password of any disk for which you do not have the appropriate authority.
6. You can not initiate VMFSUFIN without the RESTART option if a restart record exists in the SYSREST table for any of the specified products or the service package. If you do not intend to restart the interrupted invocation, you must delete the restart record using the VMFSIM MODIFY command.
7. To process a multivolume service tape, a device address must be specified for each tape volume (for example, to process a 3-volume RSU tape using tape addresses 181, 182, and 183, specify RSUTAPE 181.182.183). If you use a single tape device, specify the same address multiple times (for example, to process a 2-volume RSU tape using tape address 181, specify RSUTAPE 181.181).
8. You can process a single volume of a multivolume service tape by specifying one tape address and mounting the appropriate volume. If you use the ALL operand, only products on the tape volume that is mounted are processed.
9. If the BUILD option is specified, only the first product specified is processed and the ALL operand is invalid.
10. If the BUILD option is specified, the RSUENV, CORENV, ENVMODE, RSUTAPE, CORTAPE, KEY, and RESTART options are invalid.

**Examples**

- To install service to MYCOMP and MYCOMP2 from RSU envelope RSU9802 SERVLINK on the L-disk, enter:

```
VMFSUFIN 1VMVMC23%MYCOMP 1VMVMP11%MYCOMP2 (RSUENV RSU9802 ENVMODE L
```

- To restart the previous example after an error condition has been corrected, enter:

```
VMFSUFIN 1VMVMC23%MYCOMP (RESTART
```

- To install service to all the products on RSU envelope RPTF1234 along with any COR service for those products on COR envelope VPTF5678, enter:

```
VMFSUFIN ALL ( RSUENV RPTF1234 CORENV VPTF5678
```

- To install service to all the products on a 2-volume RSU tape using a single tape drive at address 181, enter:

```
VMFSUFIN ALL ( RSUTAPE 181.181
```

**Input and Output Files**

VMFSUFIN calls the VMFSETUP, VMFMRDSK, VMFINS, VMFPSU, VMFAPPLY, VMFREC, and VMFBLD execs; therefore, VMFSUFIN uses all of the input and output files for those execs. The input and output files unique to VMFSUFIN are:

**Input Files****sysid SYSSUF**

The system-level Service Update Facility table. *sysid* is the value assigned to the SYSTEM option in the VMFINS DEFAULTS file.

**fn SERVLINK**

RSU and COR service envelopes.

**Input/Output Files****sysid SYSREST**

The system-level restart table. *sysid* is the value assigned to the SYSTEM option in the VMFINS DEFAULTS file.

**SERVICE \$PTFS**

Applied PTFs by component.

**Messages and Return Codes**

Appendix D, “Module Identifiers for VMSES/E Messages,” on page 749 shows the format of VMSES/E messages and lists the identifiers used by each VMSES/E exec.

For a complete explanation of any VMSES/E message, use the HELP Facility to view the message explanation online or see the appropriate messages documentation.

To display information on a specific message, for example VMF002E, enter:

```
help msg vmf002e
```

If you are unfamiliar with the z/VM HELP Facility, enter help to display the main HELP Menu. For information on the HELP command, enter:

```
help cms help
```

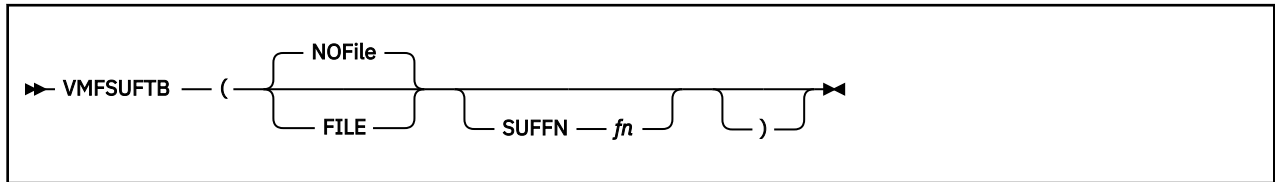
The VMFSUFIN EXEC issues the following return codes:

Return Code	Explanation
0	Command completed successfully.
4	Command completed with one or more warning conditions.
5	Command not complete because service that affects core VMSES/E components has been identified.
6	Command not completed because local modifications were found.
24	Command failed because of a command line syntax error.
28	Command failed because a required file was not found.
36	Command failed because a target disk or directory was not available.
100	Command failed because of an external error.

### Recovery Information

VMFSUFIN stores recovery information in the SYSREST table. You can restart an interrupted invocation of VMFSUFIN by entering the VMFSUFIN command with the RESTART option. VMFSUFIN uses the first product specified (and ignores any other products) and locates a restart record in the SYSREST table that contains that product. If you specify the ALL operand instead of a list of products, VMFSUFIN locates a restart record that contains the ALL keyword. This restart record is used to complete the interrupted invocation by finishing the service install for the interrupted product and processing the remaining products that were not previously processed. The ENVMODE and KEY options are ignored when RESTART is specified. If the restart record indicates envelope service, the envelope names are taken from the restart record. If the restart record indicates tape service, the tape addresses must be specified on the command.

## VMFSUFTB EXEC



### Purpose

The VMFSUFTB EXEC creates and updates the system-level service update facility table (VM SYSSUF), that contains a list of all installed products and related data needed by the automated service commands.

### Options

#### FILE

creates an output file containing a subset of data from the VM SYSSUF table.

#### NOFile

does not create the output file.

#### SUFFN *fn*

indicates which system-level service update facility file to update. *fn* is the file name of the system-level service update facility file. The default file name is defined by the SYSTEM option in the VMFINS DEFAULTS file. This option is used by the VMFUPDAT command.

### Usage Notes

1. VMFSUFTB builds records in the VM SYSSUF table from the SYSAPPS, SYSRECS, and SYSDESCT tables. VMFSUFTB only updates the :SERVLEV field in an existing record.
2. If the PRODID in the SYSRECS table does not currently appear in the VM SYSSUF table, the entry with the latest date in the :STAT field is selected. If the PRODID in the SYSRECS table does appear in the VM SYSSUF table, an entry is selected if its :PPF field matches the :INSPPF field in the VM SYSSUF table. If no match is found, no entry is selected.
3. You can modify the :P2PPPF, :INCLUDE, :INSTALL, :INSPPF, :BUILD, and :BLDPPF fields in the SYSSUF table using the VMFSIM MODIFY or the VMFUPDAT command. Before making any updates, you should invoke VMFSUFTB to update the SYSSUF table to the latest level. For more information on the SYSSUF table, see [“The System-Level Service Update Facility Table \(VM SYSSUF\)”](#) on page 169.

### Examples

To update the VM SYSSUF table to the latest level, enter:

```
VMFSUFTB
```

### Input and Output Files

#### Input Files

##### *sysid* SYSAPPS

The system-level apply status table. *sysid* is the value assigned to the SYSTEM option in the VMFINS DEFAULTS file.

##### *sysid* SYSRECS

The system-level receive status table. *sysid* is the value assigned to the SYSTEM option in the VMFINS DEFAULTS file.



**sysid SYDESCT**

The system-level description table. *sysid* is the value assigned to the SYSTEM option in the VMFINS DEFAULTS file.

**Input/Output Files****sysid SYSSUF**

The system-level service update facility table. *sysid* is the value assigned to the SYSTEM option in the VMFINS DEFAULTS file.

**Output Files****SUF OUT**

The service update facility output file.

**Messages and Return Codes**

Appendix D, “Module Identifiers for VMSES/E Messages,” on page 749 shows the format of VMSES/E messages and lists the identifiers used by each VMSES/E exec.

For a complete explanation of any VMSES/E message, use the HELP Facility to view the message explanation online or see the appropriate messages documentation.

To display information on a specific message, for example VMF002E, enter:

```
help msg vmf002e
```

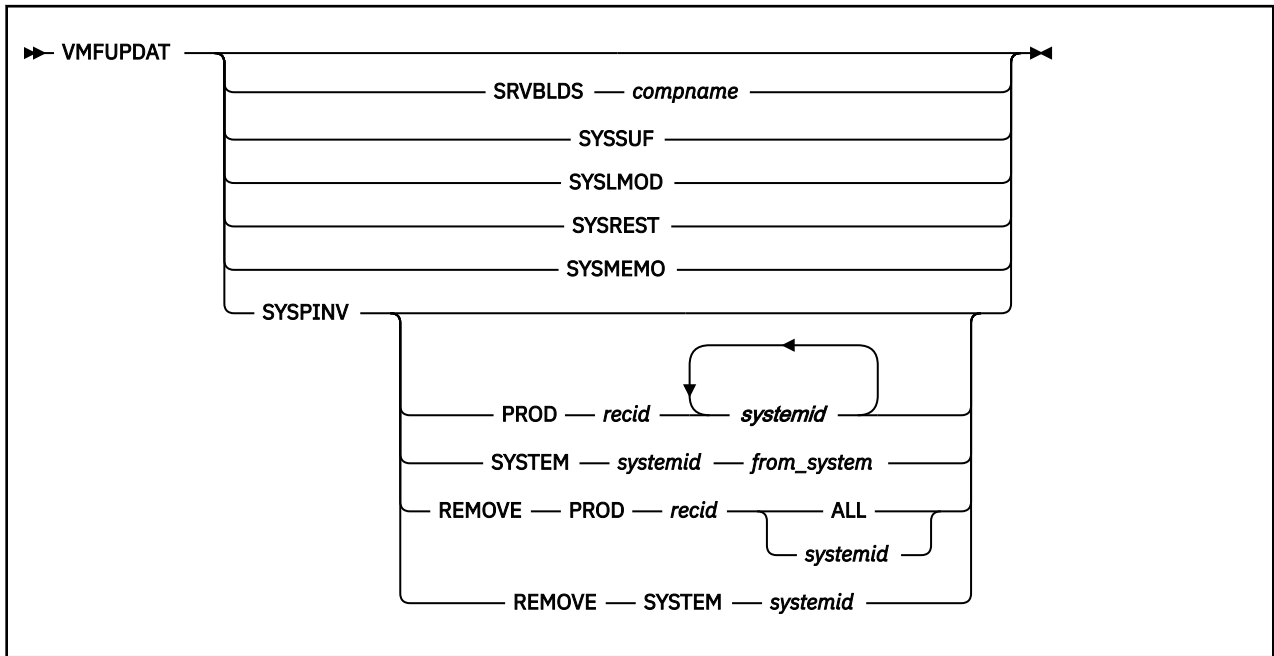
If you are unfamiliar with the z/VM HELP Facility, enter `help` to display the main HELP Menu. For information on the HELP command, enter:

```
help cms help
```

The VMFSUFIN EXEC issues the following return codes:

Return Code	Explanation
0	Command completed successfully.
24	Command failed because of a command line syntax error.
36	Command failed because a target disk or directory was not available.
100	Command failed because of an external error.

## VMFUPDAT EXEC



### Purpose

Use the VMFUPDAT EXEC to:

- change the manual status in the Service-Level Build Status table.
- change the INSTALL, BUILD, INCLUDE, INSPPF, BLDPPF and P2PPF tags in the System-Level Service Update table.
- change the local modification rework status in the System-Level Local Modification table.
- delete restart records in System-Level Restart table or SERVICE \$RESTART (from SERVICE EXEC) file.
- display memos.
- change the System-Level Product Inventory table.

### Operands

#### SRVBLDS

indicates the Service-Level Build Status table is to be updated.

#### *compname*

is the name of the component whose SRVBLDS table is to be updated.

#### SYSSUF

indicates the system-level service update facility table is to be updated.

#### SYSLMOD

indicates the System-Level Local Modification table is to be updated.

#### SYSREST

indicates the System-Level Restart table and SERVICE \$RESTART file are to be updated.

#### SYSMEMO

indicates the System-Level Memo Table entries should be displayed.

#### SYSPINV

indicates the System-Level Product Inventory table is to be updated.

**PROD**

indicates a product is to be added to one or more systems.

***recid***

is the 1- to 8-character alphanumeric identifier on the :RECID. tag in the PPF file for the product.

***systemid***

is the ID of the system to which the product is to be added.

**SYSTEM**

indicates that a system is to be cloned (new system ID information is added for each product that is associated with an existing system) OR changed (product information that pertains to an existing system ID is modified to reflect the new system name specified).

***systemid***

is the ID of the system being added (for system cloning), or that is replacing an existing system ID (for system change).

***from\_systemid***

is the ID of the system being cloned or changed. If the specified system ID exists in the system configuration file, system information is cloned. If the specified system ID does not exist in the system configuration file, system information is changed to the system name specified by the *systemid* operand.

**REMOVE PROD**

indicates a product is to be removed from one or more systems.

***recid***

is the 1-to-8 character alphanumeric identifier on the :RECID. tag in the PPF file for the product.

**ALL**

indicates a product is to be removed from all systems.

***systemid***

is the ID of the system from which the product is to be removed.

**REMOVE SYSTEM**

indicates a system is to be removed from all products.

***systemid***

is the ID of the system being removed.

**Usage Notes**

1. To update the system-level service update facility table (SYSSUF), from the VMSES/E System Inventory tables, use the PF6 VMFSUFTB key on the panel that is displayed after invoking VMFUPDAT SYSSUF.

**Examples**

- To update the Service-Level Build Status table enter:

```
VMFUPDAT SRVBLDS CMS
```

- To update the system-level service update facility table enter:

```
VMFUPDAT SYSSUF
```

- To update the System-Level Local Modification table enter:

```
VMFUPDAT SYSLMOD
```

- To update the System-Level Restart table enter:

```
VMFUPDAT SYSREST
```

- To add the system mymem2 to all the products that are on mymem1:

```
VMFUPDAT SYSPINV SYSTEM MYMEM2 MYMEM1
```

- To add the product 1VMVMC20 to more than one system:

```
VMFUPDAT SYSPINV PROD 1VMVMC20 MEMEM1 MYMEM2
```

- To remove the system mymem4 (that is, to remove the specified system from every product, or to remove all products from the specified system):

```
VMFUPDAT SYSPINV REMOVE SYSTEM MYMEM4
```

- To remove the product 1VMVMC20 from all systems:

```
VMFUPDAT SYSPINV REMOVE PROD 1VMVMC20 ALL
```

- To remove the product 1VMVMC20 from the mymem1 system:

```
VMFUPDAT SYSPINV REMOVE PROD 1VMVMC20 MYMEM1
```

## Panels

### VMFUPDAT Function Selection Panel

The VMFUPDAT Function Selection panel lets you select a specific VMFUPDAT function. It is automatically displayed if VMFUPDAT is entered without any operands. If you enter VMFUPDAT with any of its operands, you would go directly to the specific function panel and bypass this panel.

VMFUPDAT displays the following panels:

- SRVBLDS update panel
- SYSSUF update panel
- SYSLMOD update panel
- SYSREST update panel
- SYSMEMO update panel

**Note:** There is no panel interface for the SYSPINV function of VMFUPDAT.

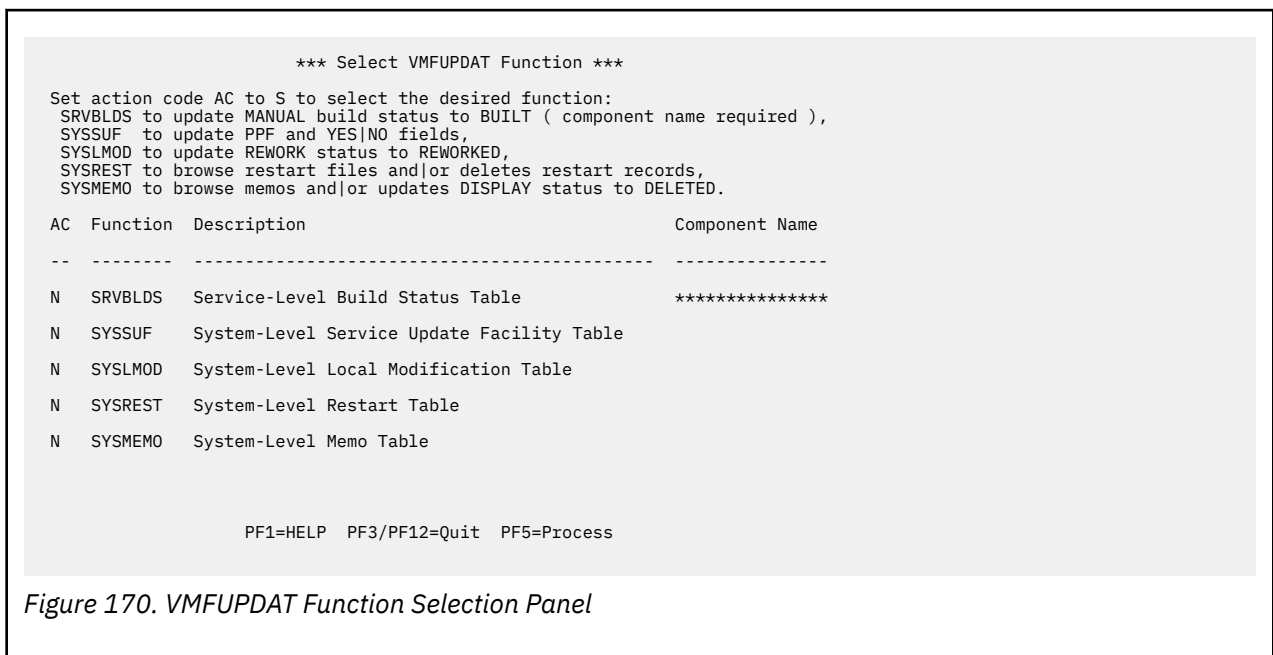


Table 25. Program Function (PF) Keys on the VMFUPDAT Function Update Panel

PF Key	Function	Explanation
PF1	Help	Displays a help panel that describes the content and meaning of the VMFUPDAT Update panel.
PF3/12	Quit	Exits from VMFUPDAT.
PF5	Process	Displays the selected function panel, or if no selection was made, exits the VMFUPDAT EXEC.
PF6	VMFSUFTB	Calls the VMFSUFTB command to create the SYSSUF table.

### SRVBLDS Update Panel

The SRVBLDS Update panel lets you update the status of parts in the special build list, UNKNOWN, in the Service-Level Build Status (SRVBLDS) table for the component name specified.

This panel displays the parts in a component's SRVBLDS table with a status of MANUAL. The records are displayed with an action code (AC) 'M', meaning the status of the parts in the special build list UNKNOWN, can be changed from MANUAL to BUILT by changing the action code (AC) to 'B'. When you press the Process Key (PF5), each part with an Action Code of 'B' is removed from the UNKNOWN record and the status of the record is changed to BUILT if all parts are removed.

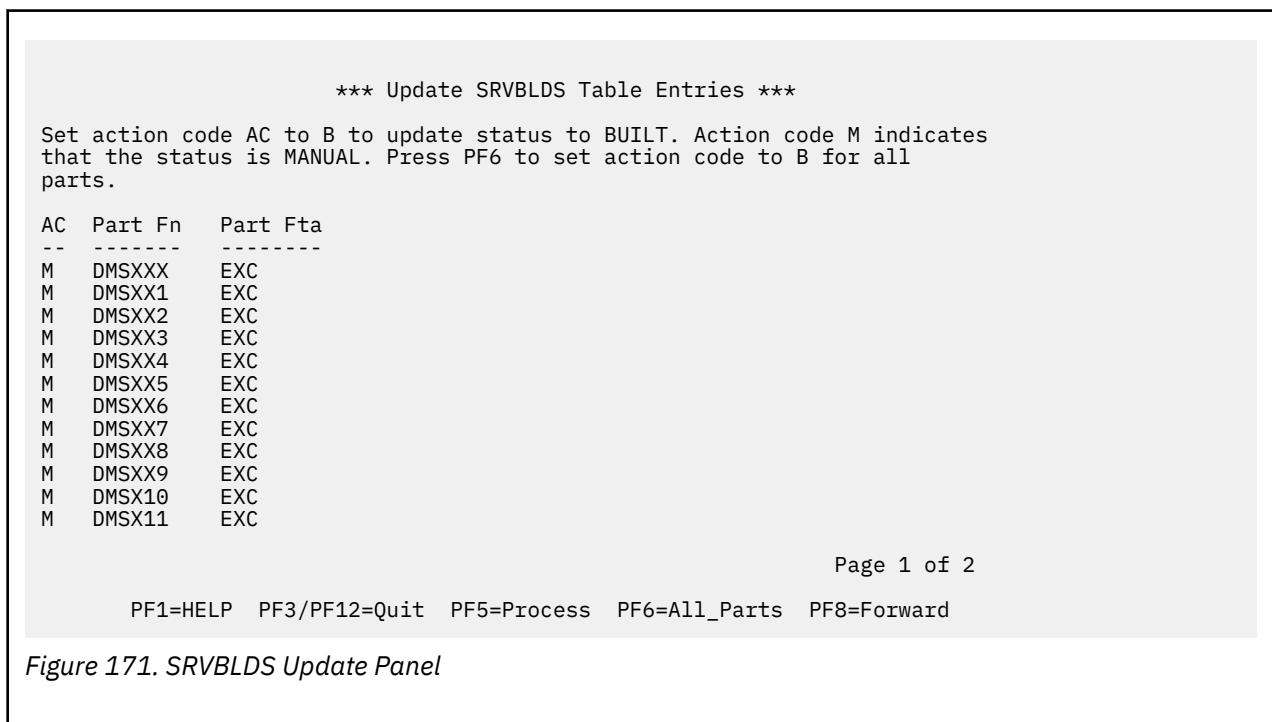


Table 26. Program Function (PF) Keys on the VMFUPDAT EXEC SRVBLDS Update Panel

PF Key	Function	Explanation
PF1	Help	Displays a help panel that describes the content and meaning of the SRVBLDS Update panel.
PF3/12	Quit	Exits from VMFUPDAT. No changes are made to records in the SRVBLDS table.
PF5	Process	Updates the appropriate records in the SRVBLDS table with changes made on the panel.
PF6	All_Parts	Sets the Action Code to 'B' for all parts.
PF7	Backward	Displays the previous page of the panel.
PF8	Forward	Displays the next page of the panel.

### SYSSUF Update Panel

The SYSSUF Update panel lets you update selected fields in the System-Level Service Update Facility (SYSSUF) table.

The column headings Compname, Prodid, Servlev, Prodlev, and Description are displayed for each record in the table. These headings can't be changed. Other fields that can be changed are:

**:INSTALL :BUILD :INCLUDE**

These fields can be changed to YES or NO.

**:INSPPF :BLDPPF :P2PPPF**

These fields can be changed to any valid PPF name or Component name combination. You can blank out the P2PPPF field if you want to remove the tag and data from the record in the SYSSUF table. Data is required for all the other fields.

A line is displayed at the bottom of each page that lets you change all the occurrences of a particular PPF name in the SYSSUF table. Enter the PPF name you want changed and the name you want it changed to in place of the '\*\*\*\*\*' fields.

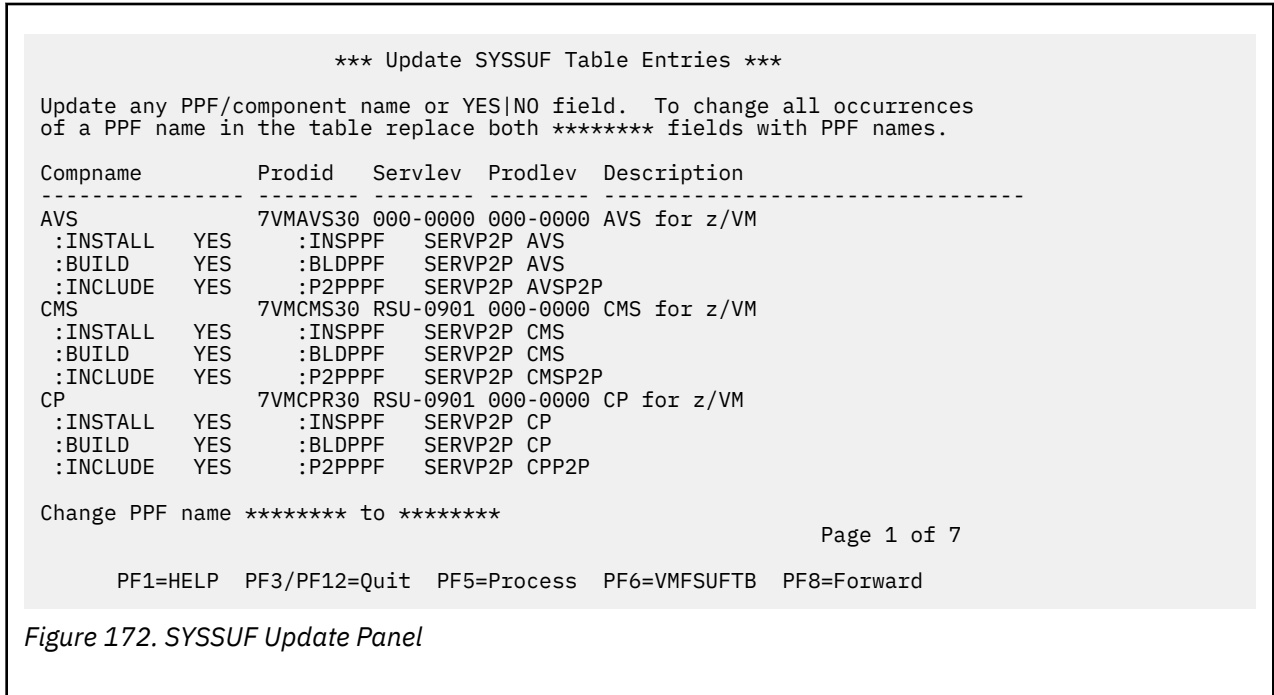


Figure 172. SYSSUF Update Panel

**Note:** The Servlev and Prodlev information displayed in Figure 172 on page 608 reflect the information for the system or SSI member where you are executing the command.

Table 27. Program Function (PF) Keys on the VMFUPDAT EXEC SYSSUF Update Panel

PF Key	Function	Explanation
PF1	Help	Displays a help panel that describes the content and meaning of the SYSSUF Update panel.
PF3/12	Quit	Exits from VMFUPDAT. No changes are made to records in the SYSSUF table.
PF5	Process	Updates the appropriate records in the SYSSUF table with changes made on the panel.
PF6	VMFSUFTB	Calls the VMFSUFTB command to update the SYSSUF table from the VMSES/E System Inventory tables on the 51D disk. The updates to the SYSSUF table are not final until the panel is exited with the process key (PF5). The VMFSUFTB key is only valid before any data is entered on the panel.
PF7	Backward	Displays the previous page of the panel.
PF8	Forward	Displays the next page of the panel.

**SYSLMOD Update Panel**

The SYSLMOD Update panel lets you update the status field of records in the System-Level Local Modification (SYSLMOD) table to REWORKED.

This panel displays records in the table with a status of REWORK. The records are displayed with an action code (AC) 'N', meaning that rework is NOT COMPLETE for the local modification. This Action Code

should be changed to 'C' if you have COMPLETED the rework for this local modification. When you press the Process Key (PF5), the status of each record with an Action Code of 'C' is updated to REWORKED.

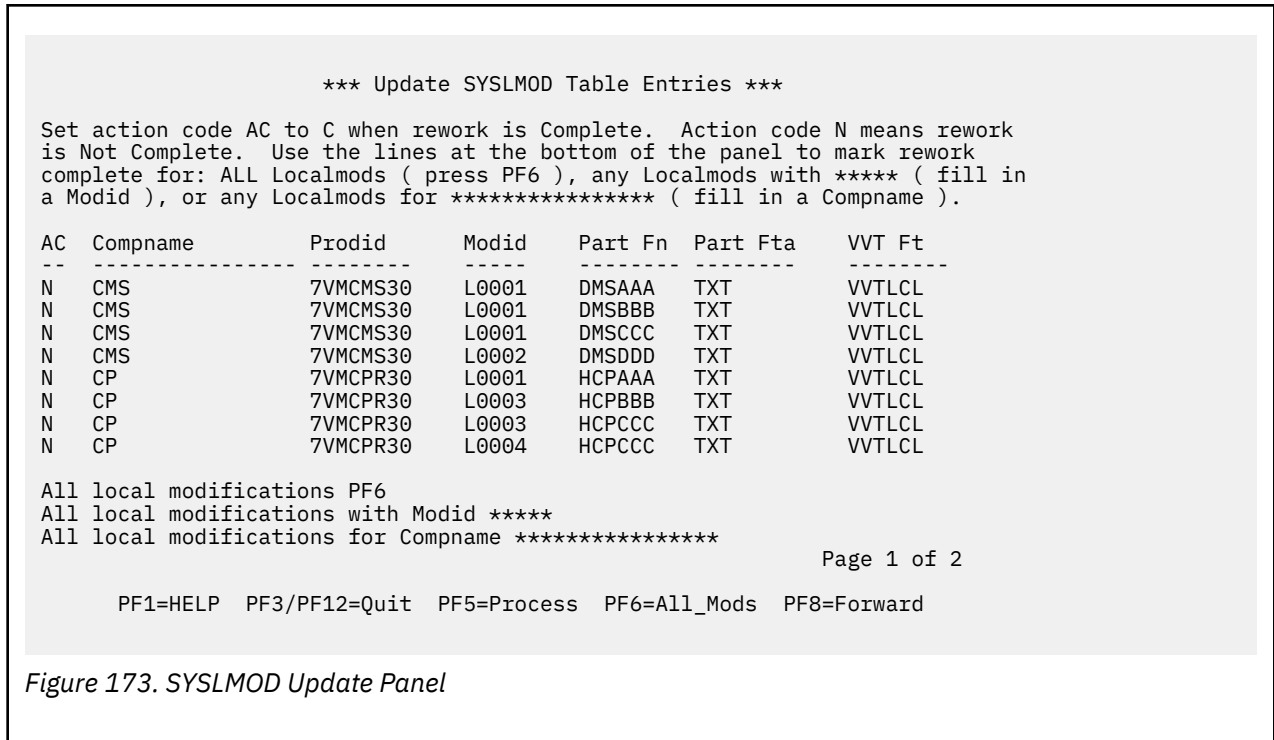


Figure 173. SYSLMOD Update Panel

Table 28. Program Function (PF) Keys on the VMFUPDAT EXEC SYSLMOD Update Panel

PF Key	Function	Explanation
PF1	Help	Displays a help panel that describes the content and meaning of the SYSLMOD Update panel.
PF3/12	Quit	Exits from VMFUPDAT. No changes are made to records in the SYSLMOD table.
PF5	Process	Updates the appropriate records in the SYSLMOD table with changes made on the panel.
PF6	All_Mods	Updates the Action Code to 'C' for all local modifications displayed on the panel.
PF7	Backward	Displays the previous page of the panel.
PF8	Forward	Displays the next page of the panel.

Three lines are displayed at the bottom of each page that let you mark rework complete for multiple local modifications:

**First Line (press PF6)**

Updates the Action Code to 'C' for ALL local modifications.

**Second Line (Enter a MODID, press enter or a PF key)**

Updates the Action Code to 'C' for any local modifications with the specified MODID.

**Third Line (Enter a COMPNAME, press enter or a PF key)**

Updates the Action Code to 'C' for any local modifications with the specified COMPNAME.

**SYSREST Update Panel**

The SYSREST Update panel lets you browse or delete records from the System-Level Restart table and the SERVICE \$RESTART file.

The SERVICE \$RESTART file contains the data needed to restart the SERVICE command. The VM SYSREST table contains records, each of which contains the data needed to restart a specific call to the VMFSUFIN command. For each record in the VM SYSREST table, the Package Name and Status is displayed with an Action Code (AC) of 'K'. You can change this Action Code to 'D' for DELETE, if you want the file or record deleted. The All-Recs Key (PF6) can be used to change all Action Codes to 'D'. When you press the Process

Key (PF5), the SERVICE \$RESTART file is erased if selected, and the selected VM SYSREST records are deleted. If all the records in the VM SYSREST table are deleted the table is erased.

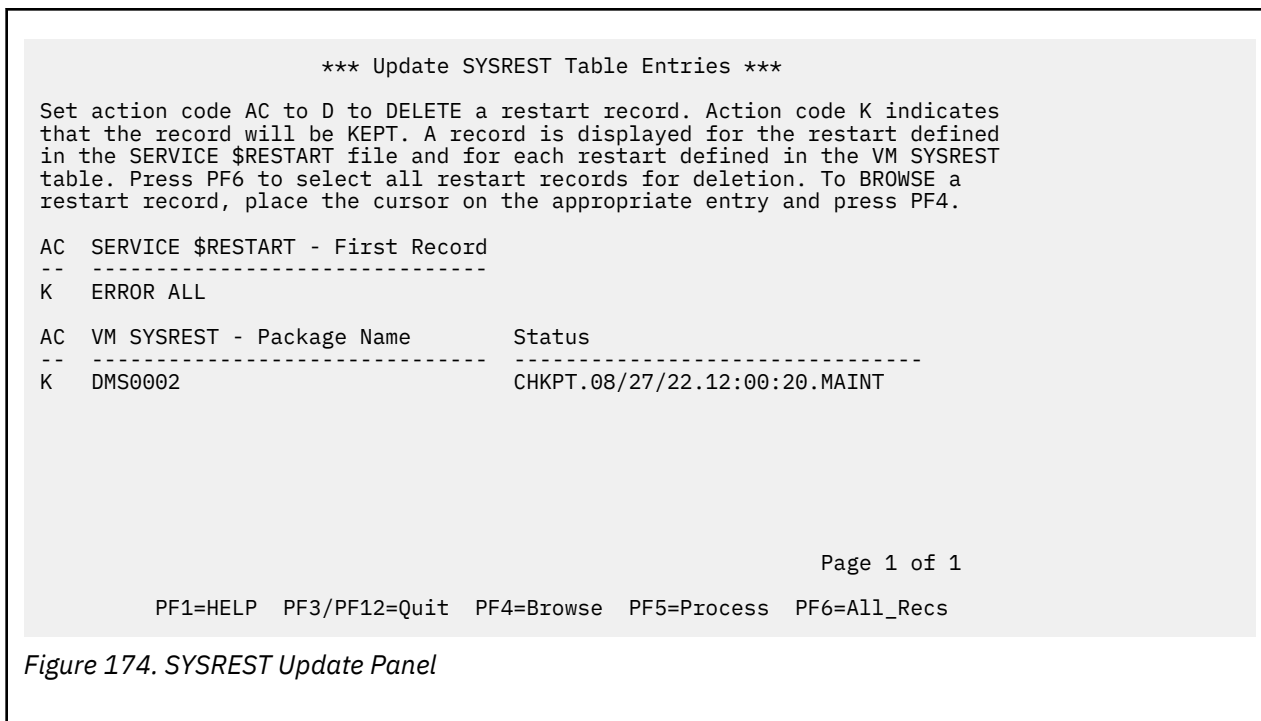


Table 29. Program Function (PF) Keys on the VMFUPDAT EXEC SYSREST Update Panel

PF Key	Function	Explanation
PF1	Help	Displays a help panel that describes the content and meaning of the SYSREST Update panel.
PF3/12	Quit	Exits from VMFUPDAT. No changes are made to records in the SYSREST table.
PF4	Browse	Displays the selected record using the BROWSE command.
PF5	Process	Updates the appropriate records in the SYSREST table with changes made on the panel.
PF6	All_Recs	Updates the Action Code to 'D' for all restart records.
PF7	Backward	Displays the previous page of the panel.
PF8	Forward	Displays the next page of the panel.

### SYSTEMMO Update Panel

The SYSTEMMO Update panel lets you browse available system memos and then delete the memos from the VM SYSTEMMO table if they are no longer needed. The actual memos are **not** deleted.

Several service deliverables come with MEMOs. These memos include:

- CORrective service Memos (COR MEMOs)
- PTF APAR MEMOs
- RSU MEMOs
- UMEMOs

During automated service processing, pointers to these memos are collected and placed in the VM SYSTEMMO table. The collected memos can be viewed in one place using the VMFUPDAT EXEC.

For each record in the VM SYSTEMMO table, an Action Code (AC), the Component, Memo Type, Memo File name (Memo Fn), Memo File Type (Memo Ft), and Memo Date/Time are displayed. You can browse a memo by placing the cursor on the desired entry and pressing PF4. An Action Code of 'K' indicates the status of the memo will be kept as DISPLAY. You can change this action code to 'D' for DELETE if you want



the memo to no longer be displayed. When you press PF5, the status of the memo is changed to DELETED in the SYSMEMO table.

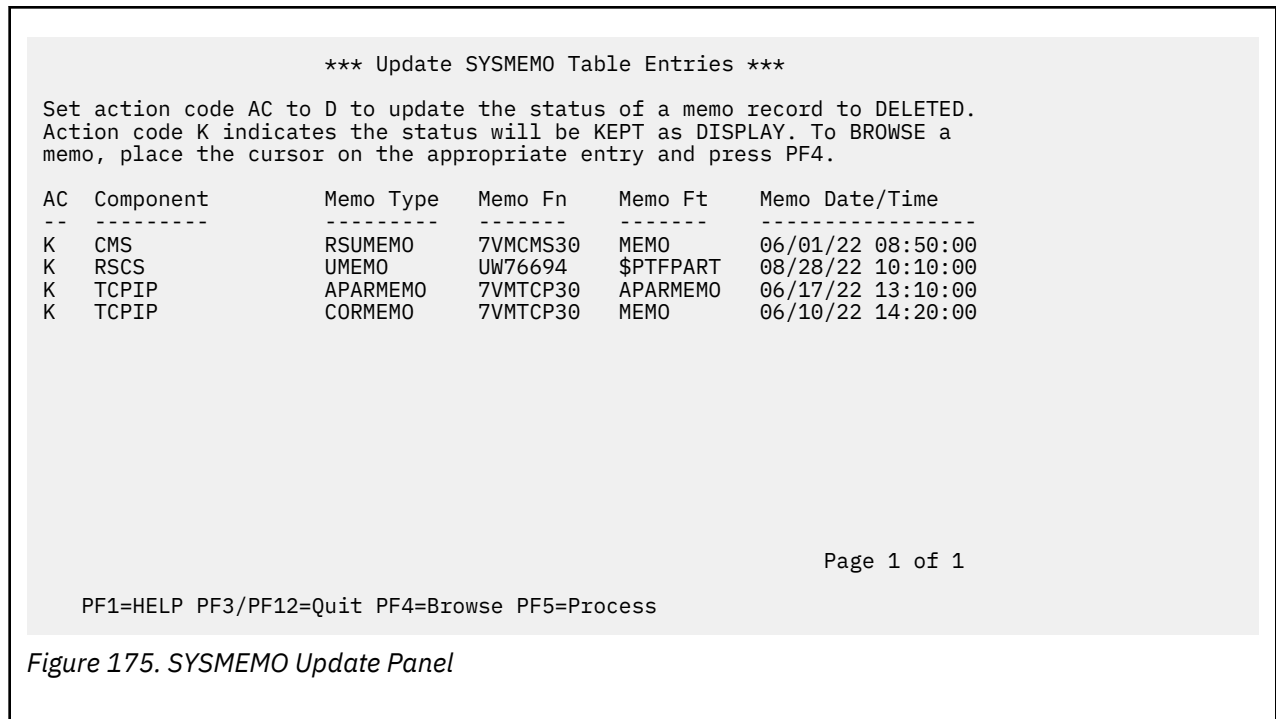


Table 30. Program Function (PF) Keys on the VMFUPDAT EXEC SYSMEMO Update Panel

PF Key	Function	Explanation
PF1	Help	Displays a help panel that describes the content and meaning of the SYSMEMO Update panel.
PF3/12	Quit	Exits from VMFUPDAT. No changes are made to records in the SYSMEMO table.
PF4	Browse	Displays the selected memo using the BROWSE command.
PF5	Process	Updates the appropriate records in the SYSMEMO table with changes made on the panel.
PF7	Backward	Displays the previous page of the panel.
PF8	Forward	Displays the next page of the panel.

## Input and Output Files

### Input Files

#### **ppfname PPF**

The usable form product parameter file.

#### **VMFHPSEL HPANEL**

The SELECT function HELP panel.

#### **VMFHPSUF HPANEL**

The Update SYSSUF HELP panel.

#### **VMFHPLMD HPANEL**

The Update SYSLMOD HELP panel.

#### **VMFHPRST HPANEL**

The Update SYSREST HELP panel.

#### **VMFHPLD HPANEL**

The Update SRVBLDS HELP panel.

#### **VMFHPMEM HPANEL**

The Update SYSMEMO HELP panel.

**VMFINS DEFAULTS**

The file containing the *sysid* default.

**Input/Output Files*****sysid* SYSSUF**

The system-level service update facility table.

***sysid* SYSLMOD**

The system-level local modification table.

***sysid* SYSREST**

The system-level restart table.

***sysid* SYSTEMEMO**

The system-level memo table.

**SERVICE \$RESTART**

The SERVICE restart file.

***bldid* SRVBLDS**

The service-level build status table.

***systemid* SYSPINV**

The system-level product table.

***recid* SRVPROD**

The service-level production status table.

**Temporary Files****VMFUPDAT \$\$\$UP\$\$\$**

The VMFUPDAT parameter file (used to pass parameters to the panel handler).

**\$\$\$VM\$\$\$ SYSSUF**

The SYSSUF temporary file (used for recovery).

**\$\$\$VM\$\$\$ SYSLMOD**

The SYSLMOD temporary file (used for recovery).

**\$\$\$VM\$\$\$ SYSREST**

The SYSREST temporary file (used for recovery).

**\$\$\$VM\$\$\$ SRVBLDS**

The SRVBLDS temporary file (used for recovery).

**\$\$\$VM\$\$\$ SYSTEMEMO**

The SYSTEMEMO temporary file (used for recovery)

**\$\$\$VM\$\$\$ SYSPINV**

The SYSPINV temporary file (used for recovery).

**Messages and Return Codes**

Appendix D, “[Module Identifiers for VMSES/E Messages](#),” on page 749 shows the format of VMSES/E messages and lists the identifiers used by each VMSES/E EXEC.

For a complete explanation of any VMSES/E message, use the HELP Facility to view the message explanation online or see the appropriate messages documentation. To display information about a specific message - VMF002E, for example - enter:

```
help msg vmf002e
```

If you are not familiar with the z/VM HELP Facility, enter `help` to display the main HELP Menu. For information about the HELP command, enter:

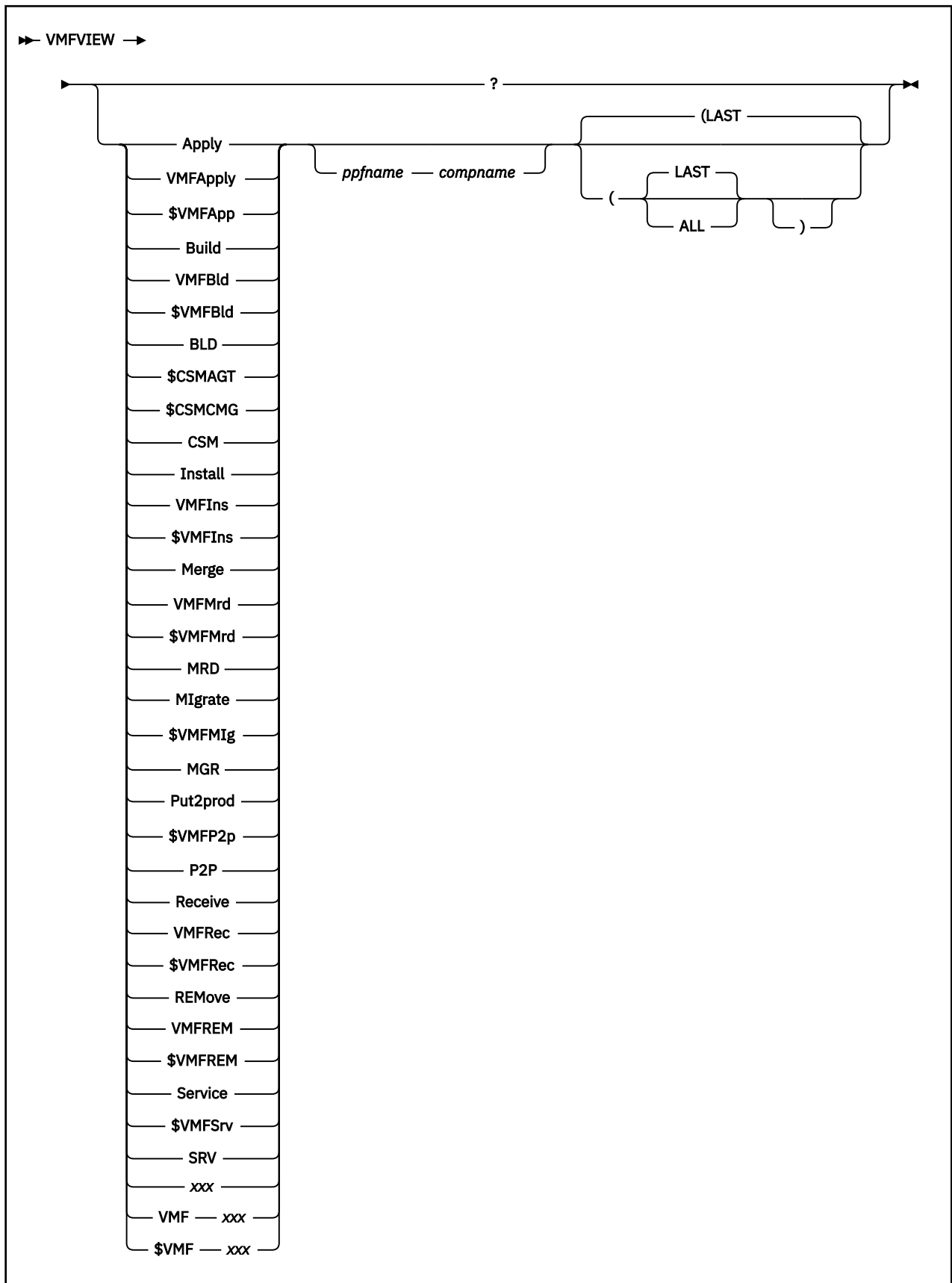
```
help cms help
```

Appendix D, “[Module Identifiers for VMSES/E Messages](#),” on page 749 shows the format of VMSES/E messages and lists the identifiers for each VMSES/E EXEC.

VMFUPDAT issues the following return codes:

<b>Return Code</b>	<b>Explanation</b>
0	Command completed successfully.
4	Command completed with one or more warning conditions.
24	Command failed because of a command line syntax error.
28	Command failed because a required file was not found.
36	Command failed because a target disk or directory was not available.
99	Command terminated by user.
100	Command failed because of an external error.

# VMFVIEW EXEC



## Purpose

VMFVIEW calls XEDIT to allow you to view the message logs. Using VMFVIEW, you can:

- Select which product or component's messages to view.
- Select which function's messages to view.
- View all the messages with the same message leaders.
- View all the messages with the same message number.
- View the help file for a specific message.
- Move backward and forward through the displayed message log.
- View only those PTFs that have been received or committed.

## Operands

**?**

displays a VMFVIEW help screen.

### Apply

**VMFApp**

**\$VMFApp**

views the apply message log.

### Build

**VMFBld**

**\$VMFBld**

**BLD**

views the build message log.

### \$CSMAGT

views the CSMAGENT message log.

### \$CSMCMG

**CSM**

views the SERVMGR message log.

### Install

**VMFIns**

**\$VMFIns**

views the install message log.

### Merge

**VMFMrd**

**\$VMFMrd**

**MRD**

views the merge message log.

### Migrate

**\$VMFMIg**

**MGR**

views the migrate message log.

### Put2prod

**\$VMFP2p**

**P2P**

views the put2prod message log.

### Receive

**VMFRec**

**\$VMFRec**

views the receive message log.

**REMove**  
**VMFREM**  
**\$VMFREM**

views the remove message log.

**Service**  
**\$VMFSrv**  
**SRV**

views the service message log.

**xxx**  
**VMFxxx**  
**\$VMFxxx**

views the user message log, \$VMFxxx \$MSGLOG.

***ppfname***

is the file name of the usable form product parameter file. The product parameter file must have a file type of PPF.

***compname***

is the name of the component as it is specified on the :COMPNAME tag in the product parameter file.

## Options

**LAST**

displays only messages from the most recent run in the message log. LAST is the default.

**ALL**

displays messages from all runs in the message log.

## Usage Notes

1. The *ppfname* and *compname* operands can be used for the install message log, except when the VMFINS command INFO option is used.
2. PF keys are defined in a tailorable file called VMFVIEW\$ PROFILE. To change the PF key assignments, edit this file and follow the instructions at the beginning of the file. (See [Table 31 on page 618](#) for the default PF key assignments.)
3. The initial set of messages displayed by VMFVIEW is defined in the file VMFVIEW\$ PROFILE. The default is the nonstatus category of messages.
4. The Help PF key (PF1 / PF13) performs one of two functions, depending on the position of the cursor:
  - a. When the cursor is on the command line, the VMFVIEW help screen is displayed.
  - b. When the cursor is on a line that contains a recognized message ID, the help screen for that VMSES/E message is displayed. If the requested message help is not found, or if a message ID cannot be identified, an error is reported.
5. The 'All' PF key (PF2) performs one of two functions, depending on the position of the cursor:
  - a. When the cursor is on the command line, the 'All' key displays all messages in the log, excluding any trace messages.
  - b. When the cursor is on a line that contains a recognized message ID, the 'All' key displays all of the messages in the log with the same message number. If no message ID is recognized, an error message is reported.
6. The 'All+Trace' PF key (PF14) performs one of two functions, depending on the position of the cursor:
  - a. When the cursor is on the command line, the 'All' key displays all messages in the log, including any trace messages.
  - b. When the cursor is on a line that contains a recognized message number, the 'All+Trace' key functions in the same manner as the 'All' (PF2) key.

**Note:** Results produced by using the 'All' (PF2) key and the 'All+Trace' (PF14) key are qualified by the LAST and ALL options of the VMFVIEW command.

7. To gather information about messages, you can issue commands, such as VMFSIM, FILELIST, and XEDIT, from the command line.
8. You may find it useful to put a subset of messages into a separate CMS file. You can use the XEDIT PUT command to do this.

For example, if you want to put all BD: (build) messages in a file to process separately, press the appropriate PF key to isolate this subset of messages. (The default is PF06 / PF18.) Then enter the following command on the XEDIT command line:

```
PUT * BUILD
```

This creates a CMS file with the name BUILD \$MSGLOG. The PUT command changes the current line on the XEDIT screen. You can enter TOP on the XEDIT command line to make the current line the top line of the file, or you can press another PF key to view a new subset of messages.

9. You may find it useful to add the invocation of VMFVIEW to your user exits.
10. If, while viewing messages from the LAST run, you wish to check messages in previous runs, use the 'ALL' XEDIT macro. This macro resets the 'SET SELECT' and 'SET DISPLAY' XEDIT subcommands used by VMFVIEW. To return to view only the LAST run, press one of the PF keys that is set to select a subset of messages.

## Examples

- To run VMFVIEW using the IBM-supplied defaults to view the last receive message log, enter:

```
VMFVIEW R
```

- To use VMFVIEW to view all apply message logs, enter:

```
VMFVIEW VMFA (ALL
```

- To use VMFVIEW to view the last build message log for a given product or component, enter:

```
VMFVIEW BLD ppfname compname
```

## Message Headers

The message headers that appear in the message logs are:

### BD

These are build messages that give information needed to rebuild the parts affected by the application of service.

### CK

These are items you *must check*. These messages contain information that may require user action.

### MS

These are mismatch messages that indicate conflicting control information that *must be investigated and corrected*.

### RO

These are requisite out of component messages. They indicate PTFs *must be applied to another component*.

### RQ

These are requisite messages that indicate there are PTFs (in the same component) that are missing and *must be applied*.

### ST

These are status messages. These messages give useful information but require no further action.

**SV**

These are severe messages. These indicate problems that resulted in the termination of a process. These problems *must be investigated and corrected*.

**TR**

These are tracing messages produced from having activated program-specific tracing. These messages provide supplementary information for diagnostic purposes.

**WN**

These are warning messages that *must be investigated*. They indicate situations that may or may not be a real problem. It is up to the user to decide after investigating the cause of the message.

**Default PF Key Assignments**

Table 31 on page 618 shows the default Program Function (PF) key assignments.

<i>Table 31. Default Program Function (PF) Key Assignments for the VMFVIEW EXEC</i>		
<b>PF Keys</b>	<b>Function</b>	<b>Explanation</b>
PF1 / PF13	Help	Display a HELP screen
PF2	All	Display ALL messages other than tracing messages
PF3 / PF15	Quit	Exit VMFVIEW
PF4 / PF16	Exception	Display CK:, MS:, RQ:, SV:, and WN: messages
PF5 / PF17	Status	Display ST: messages
PF6 / PF18	Build	Display BD: messages
PF7 / PF19	Backward	Display previous screen of messages
PF8 / PF20	Forward	Display following screen of messages
PF9 / PF21	OutCompReq	Display RO: messages
PF10 / PF22	Non-Stat	Display all messages except ST: messages
PF11 / PF23	Requisite	Display RQ: messages
PF12 / PF24	Severe	Display SV: messages
PF14	All+Trace	Display ALL messages, including tracing messages

**Input and Output Files**

**Input Files**

***ppfname* PPF**

The usable form product parameter file.

**VMFVIEW\$ PROFILE**

The VMFVIEW environment profile.

***fn* \$MSGLOG**

The message log.

**PPF Tags Used**

**:APPID**

The identifier of the product/component used during Apply.

**:BLDID**

The identifier of the product/component used during Build.



**:RECID**

The identifier of the product/component being received as it appears on the tape.

**Messages and Return Codes**

Appendix D, “Module Identifiers for VMSES/E Messages,” on page 749 shows the format of VMSES/E messages and lists the identifiers used by each VMSES/E exec.

For a complete explanation of any VMSES/E message, use the HELP Facility to view the message explanation online or see the appropriate messages documentation. To display information on a specific message, for example VMF002E, enter:

```
help msg vmf002e
```

If you are unfamiliar with the z/VM HELP Facility, enter help to display the main HELP Menu. For information on the HELP command, enter:

```
help cms help
```

The VMFVIEW EXEC issues the following return codes:

Return Code	Explanation
0	Command completed successfully.
4	Command completed with one or more warning conditions.
12	Command failed because of an internal error.
24	Command failed because of a command line syntax error.
28	Command failed because a required file was not found.
100	Command failed because of an external error.

**Recovery Information**

The VMFVIEW command can be restarted by reissuing the command.



## Chapter 21. Product Parameter File Syntax

This chapter describes the syntax of the records in the various types of product parameter files. For an overview of the various types of product parameter files, see [“The Source Product Parameter File”](#) on page 13.

### Structure of the Data in Product Parameter Files

All product parameter files (PPFs) consist of a series of records. There are two types of records: data records and comments.

#### Data Records

A data record begins with a tag. A tag begins with a colon and contains a keyword and a data value, which are separated by a period. The keyword identifies the function of the record. Depending on the function, the tag could be followed by zero or more values on the same line as the tag, or a list of zero or more statements on successive lines below the tag.

Unlike the data in the Software Inventory tables, all PPF tags must be in uppercase; but their values can be in mixed case. All PPF tags are required and must start in column one. A data record looks like this:

```
:COMPNAME.VMSES
```

:COMPNAME is the tag; VMSES is the value. The syntax checking function of the VMFPPF EXEC notifies you if there are missing tags.

#### Comments

Product parameter files can also contain comments. A comment begins with an asterisk (\*). Comments can appear on separate lines or on the same lines as tags and statements. In source and override product parameter files, comments that appear on separate lines come in three varieties, which are distinguished by the number of asterisks that begin the comment.

- Leading comments begin with a single asterisk. These comments are bound to the records that follow them. A blank must follow the asterisk.
- Trailing comments begin with two asterisks. These comments are bound to the records that precede them. A blank must follow the asterisks.
- Other comments include those that begin with more than two asterisks and those that do not have a blank between the asterisks and the text of the comment. These comments are not merged into the temporary or usable form PPF.

### File Structure of Product Parameter Files

There are four types of product parameter files; source, override, temporary, and usable form.

The source product parameter file is supplied by the product. It contains the recommended values for the various product parameters.

The override product parameter file may be supplied by the product or created by you. It contains overrides to the product-supplied parameters that are found in the source product parameter file.

Temporary product parameter files are created by the VMFOVER EXEC and erased from your system after the usable form product parameter file is created.

The usable form product parameter file is the final PPF file, and it is used to control VMSES/E exec processing.

## Product Parameter File Syntax

The structure of each of these types of PPFs is very similar. The syntax of the source PPF is described in its entirety in the following sections. When the syntax of the other types of PPFs is discussed, only the differences from the source PPF are described. [Figure 176 on page 622](#) shows the general relationship of the four types of product parameter files.

## Product Parameter File Processing

The VMFOVER EXEC applies override PPFs to a source PPF to create a temporary PPF. The VMFPPF EXEC calls the VMFOVER EXEC to dynamically build the temporary PPF. The VMFPPF EXEC uses this temporary PPF to create the usable form PPF. Throughout this book, the usable form product parameter file is referred to as simply the product parameter file or the PPF.

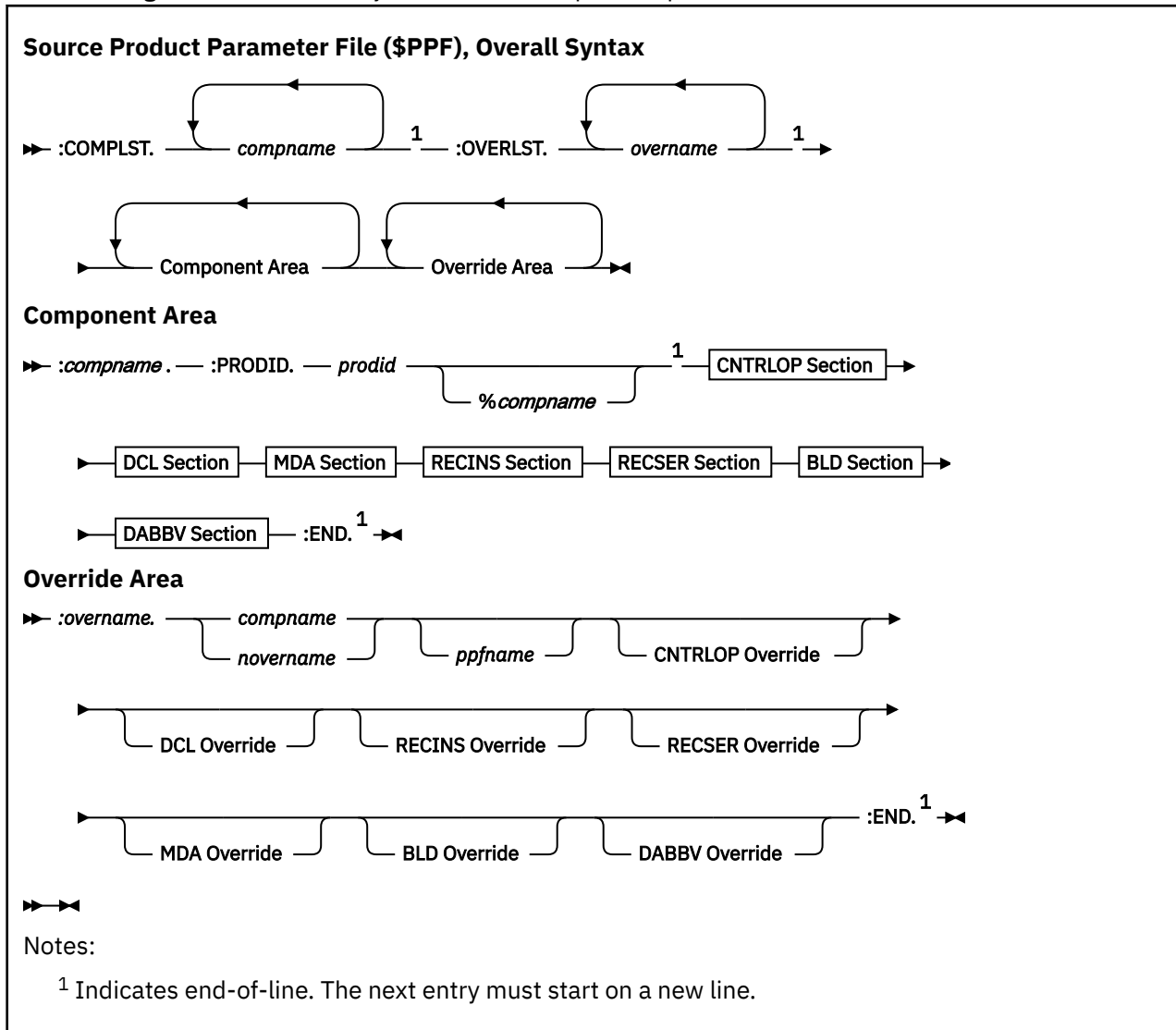


*Figure 176. Product Parameter File Relationship*

## Source Product Parameter File Syntax

The source product parameter file is supplied by the product. It contains the recommended, or default, values for the various parameters. It is made up of a header area, one or more component areas, and zero or more override areas.

The following shows the overall syntax for a source product parameter file.



### Header Area

The header area contains two tags; :COMPLST and :OVERLST. The :COMPLST tag is a list of component areas in the source PPF. The :OVERLST tag is a list of override areas in the source PPF. The header area must appear before any component or override areas, and the :COMPLST tag must precede the :OVERLST tag.

#### :COMPLST.

lists the component areas in the source PPF. There is generally only one component area per product. The parameters in a component area are usually modified with overrides rather than providing additional component areas.

#### *compname*

is the name of a component area. *compname* is a 1- to 16-character alphanumeric identifier.

The *compname* on the :COMPLST tag must match one of the :*compname* tags in the component area.

### **:OVERLST.**

lists the override areas in the source PPF. Each override area provides changes for a single component area, but multiple override areas can provide changes for the same component area.

#### ***overname***

is the name of an override area in the source PPF.

The *overname* on the :OVERLST tag should match one of the :*overname* tags in one of the override areas.

## Example

Figure 177 on page 624 shows an example of a header area in a source PPF.

```
:COMPLST.  VMSES  
:OVERLST.  VMSESUCENG VMSESINS VMSESPTFS
```

Figure 177. Sample Header Area of a Source PPF

## Component Area

Component areas are delimited by the *:compname* and :END tags and contain the :PRODID tag plus the following sections that contain the parameters for the product:

- CNTRLOP section (see “Control Options Section” on page 625)
- DCL section (see “Variable Declarations Section” on page 631)
- MDA section (see “Minidisk/Directory Assignments Section” on page 633)
- RECINS section (see “Receive Installation Tape Definition Section” on page 637)
- RECSER section (see “Receive Service Media Definition Section” on page 639)
- BLD section (see “Build Product Definition Section” on page 641)
- DABBV section (see “File Type Abbreviations Extensions Section” on page 644)

All the tags and sections in the component area of the source PPF are the same as in the component area of the usable form PPF. Remember that in the source PPF, variables defined in the variable declarations section have not yet been resolved.

The tags in the component area are:

### ***:compname.***

marks the beginning of a component area in a source PPF and is retained in the usable form PPF. *compname* is a 1-16 character uppercase name that identifies a product. The *:compname* tag is used by the VMFOVER EXEC to locate the component areas listed on the :COMPLST tag in source PPFs.

### **:PRODID.**

identifies the 7-25 character product/component/version-release-modification identifier for PTF validation and product requisites.

#### ***prodid***

is the 7-8 character alphanumeric identifier assigned to each product/version-release-modification level of the product (may also be referred to as *prodid* in other areas of this book). *prodid* also identifies the file name of the source PPF for the product.

#### ***%compname***

is the component name identifier, for example CMS, which is added to the *prodid* with a percent sign (%). *compname* is a 1-16 character alphanumeric identifier.

### **:END.**

marks the end of a component area.

## Example

Figure 178 on page 625 shows an example of the component area in a source PPF.

```
:VMSES.  
:PRODID. 1VMVMC23%MYCOMP  
:  
:END.
```

*Figure 178. Sample Component Area of a Source PPF*

## Control Options Section

The control options (CNTRLOP) section is delimited by the :CNTRLOP and :ECNTRLOP tags and identifies parameters used to control the operation of VMSES/E. Many control options can be overridden by corresponding options on the VMSES/E commands.

### Syntax

The following shows the syntax for the control options section of the product parameter file.





**:PRODESC.**

is the descriptive name for the product identified on the :PRODID tag.

**text**

is free-format text, which can include spaces. *text* ends with the start of the next tag.

**:VERSION.**

identifies the level of VMSES/E required by the product. If this tag is not specified, the version is assumed to be VMSES/E 1.1.1.

**version**

indicates the level of VMSES/E required by the product. The format is z/VM v.r.m.

**Note:** This field is reserved for use by IBM.

**:BCOMPNAME.**

identifies the base component.

**bcompname**

is the 1-16 character base component name assigned to the product, which is used together with *prodid* to stack the product on VMSES/E formatted tapes.

**:RECID.**

identifies the product.

**prodid**

is the 1-8 character alphanumeric product identifier assigned to the product, which is used together with *bcompname* to stack the product on VMSES/E formatted tapes. *prodid* is also used as the file name of the Software Inventory tables created or updated during receive processing.

**:APPID.**

specifies the file names of the select data file and version vector tables.

**appid**

is the primary apply identifier.

**sappid**

is a secondary apply identifier. *sappids* are required when there are different system versions and dependent products.

**Note:** The *appid* and *sappids* are the file names of the version vector tables used by the VMFSIM GETLVL and VMFSIM CHKLVL commands. VMFSIM GETLVL and VMFSIM CHKLVL search the tables in the order you specify until the information is located.

**:BLDID.**

identifies the file name of the Software Inventory tables for build processing.

**bldid**

is the 1-8 character alphanumeric identifier used as the file name of the Software Inventory tables created or updated during build processing. The default value is the same as *prodid*. Different *bldid* values can be used to create different systems.

**:LOG.**

identifies whether messages should be logged.

**YES**

indicates messages issued by the VMFINS, VMFREC, VMFAPPLY, VMFBLD and VMFMRDSK execs should be logged.

**NO**

indicates messages issued by the VMFINS, VMFREC, VMFAPPLY, VMFBLD and VMFMRDSK execs should not be logged.

**:RECVALL.**

indicates if missing serviceable parts will be received for committed PTFs.

**YES**

indicates the VMFREC EXEC will receive missing serviceable parts.

### **NO**

indicates the VMFREC EXEC will not receive missing serviceable parts of committed PTFs.

### **:SETUP.**

indicates whether VMFSETUP should be called by other VMSES/E commands to access the required minidisks or SFS directories for the product.

### **YES**

indicates VMFSETUP EXEC should be called by other VMSES/E commands to access the required minidisks or SFS directories for the product. If a user exit is specified, VMFSETUP is called after the user exit.

### **NO**

indicates VMFSETUP EXEC should not be called by other VMSES/E commands to access the required minidisks or SFS directories for the product.

### **PREEXit**

indicates VMFSETUP EXEC should be called by other VMSES/E commands to access the required minidisks or SFS directories for the product. If a user exit is specified, VMFSETUP is called before the user exit.

### **:SLVI.**

lists the system level and version indicators used to prefix and suffix the APAR number to identify the file type of update files for the product, as in X12345ZZ.

#### **x**

is the prefix.

#### **zz**

is the suffix.

### **:NLS.**

specifies the language identifier.

#### ***langid***

is the 1- through 5-character language identifier for the national language installed for the product. It is the second token in the VMFNLS LANGLIST file. *langid* supports products exploiting VMSES/E 1.1.

#### ***language code***

is a 3-character language code defined in *National Language Support Reference Manual Volume 2*, SE09-8002.

### **:CNTRL.**

identifies the file name of the control file used to service, assemble, or build this product.

#### ***cntrlfn***

is the 1- through 8-character file name of the control file. The file type must be CNTRL.

### **:ALTCNTRL.**

identifies the file name of the alternate control file used to service, assemble, or build this product.

#### ***cntrlfn***

is the 1- through 8-character file name of the alternate control file. The file type must be CNTRL.

### **:AXLIST.**

identifies the file name of the product-supplied apply and exclude lists shipped in service packages.

#### ***axname***

is the 1- through 8-character alphanumeric file name of the product-supplied apply and exclude lists.

### **:EXCLIST.**

identifies the file name of the user-supplied exclude list.

#### ***excname***

is the 1- through 8-character alphanumeric file name of the user-supplied exclude list. The file type of this file must be \$EXCLIST.

**:UPDTID.**

identifies the update level identifier used in the file types of AUX files and version vector tables created during VMFAPPLY EXEC processing. The AUX file must be listed in the control file specified on the :CNTRL tag.

***updateid***

is the 4- through 8-character update level identifier.

**:CKAUX.**

indicates if self-documenting information in replacement parts should be validated against AUX files during VMFBLD EXEC processing.

**Note:** The CKAUX function is not performed in VM/ESA 1.2.0 (and later). The tag remains for downward compatibility with VM/ESA 1.1.1.

**YES**

indicates self-documenting information in replacement parts should be validated against AUX files during VMFBLD EXEC processing.

**NO**

indicates if self-documenting information in replacement parts should not be validated against AUX files during VMFBLD EXEC processing.

**:CKSDI.**

indicates if self-documenting information in replacement parts should be validated against version vector tables during VMFAPPLY EXEC processing.

**YES**

indicates self-documenting information in replacement parts should be validated against version vector tables during VMFAPPLY EXEC processing.

**NO**

indicates self-documenting information in replacement parts should not be validated against version vector tables during VMFAPPLY EXEC processing.

**:CKVV.**

indicates if AUX files should be validated against the corresponding version vector tables during VMFBLD EXEC processing.

**YES**

indicates AUX files should be validated against the corresponding version vector tables during VMFBLD EXEC processing.

**NO**

indicates AUX files should not be validated against the corresponding version vector tables during VMFBLD EXEC processing.

**:CKGEN.**

indicates if AUX files should be validated against the corresponding version vector tables during VMFASM, VMFHASM, VMFHLASM, VMFNLS, VMFEXUPD, and GENCPBLS processing.

**YES**

indicates AUX files should be validated against the corresponding version vector tables.

**LOGMOD**

indicates AUX files should be validated against the corresponding version vector tables and local version vector tables should be updated using the information from the corresponding AUX files.

**NO**

indicates AUX files should not be validated against the corresponding version vector tables. The output file is named according to the VVT structure.

**NOVVT**

indicates AUX files should not be validated against the corresponding version vector tables. The output file is named according to the AUX/CNTRL file structure.

### **:RETAIN.**

identifies file modes that are restricted from use by VMSES/E. The VMFSETUP EXEC does not use any file mode listed here. File modes A thru D, S, and Y are automatically retained.

*fm*

is the file mode that is to be restricted from use by VMSES/E.

### **:USEREXIT.**

identifies the file name of a user exit to be called to setup at the beginning and cleanup at the end of each invocation of the VMFREC, VMFAPPLY, VMFBLD, VMFMRDSK, VMFASM, VMFHASM, VMFHLASM, VMFREPL, VMFNLS, VMFREM, VMFEXUPD, and GENCPBLS execs.

*exitname*

is the file name of the user exit.

### **PI**

You can give this exec any file name; the file type must be EXEC. Use this exec to do setup or cleanup tasks, such as accessing and linking minidisks. For example, you could use it to link and later detach minidisks required by the calling service function.

The calling function supplies three parameters to the user exec:

1. The name of the service function (in upper case)
2. SET-UP or CLEAN-UP, indicating initialization or termination
3. The return code which the calling service function is going to end with (on CLEAN-UP only)

The return code from the user exit is processed by the calling service function. On SET-UP, any return code other than 0 or 4 will cause the calling service function to stop. On CLEAN-UP, the return code will override the return code of the calling service function. If you do not wish to alter the return code of the calling service function, you must be sure to end the user exit with the return code that was passed from the calling service function.

### **PI end**

### **:PTFPFX.**

identifies the 2-character prefixes that are used to identify a PTF number for this product.

*pp*

is a 2-character prefix for this product.

### **:APARPFX.**

identifies the 2-character prefixes that are used to identify an APAR number for this product.

*aa*

is a 2-character prefix for this product.

### **:ECNTRLOP.**

marks the end of the control options section for the product.

## **Example**

Figure 179 on page 631 shows an example of a control options section.

```

:CNTRLOP.
:PRODESC. ACOMP for VM
:VERSION. z/VM v.r.m
:BCOMPNAME. MYCOMP
:RECID. 1VMVMC23
:APPID. 1VMVMC23 CMP$PSU$
:BLDID. 1VMVMC23
:LOG. YES
:RECVALL. NO

```

```

:SETUP. NO
:SLVI. Z/VF
:NLS. AMENG
:CNTRL. CMPVM
:AXLIST. CMPVM
:EXCLIST.
:UPDTID. AUXVM
:CKGEN. YES
:CKSDI. NO
:CKVV. NO
:RETAIN.
:USEREXIT.
:PTFFPX. UM
:APARPFX. VM
:ECNTRLOP.

```

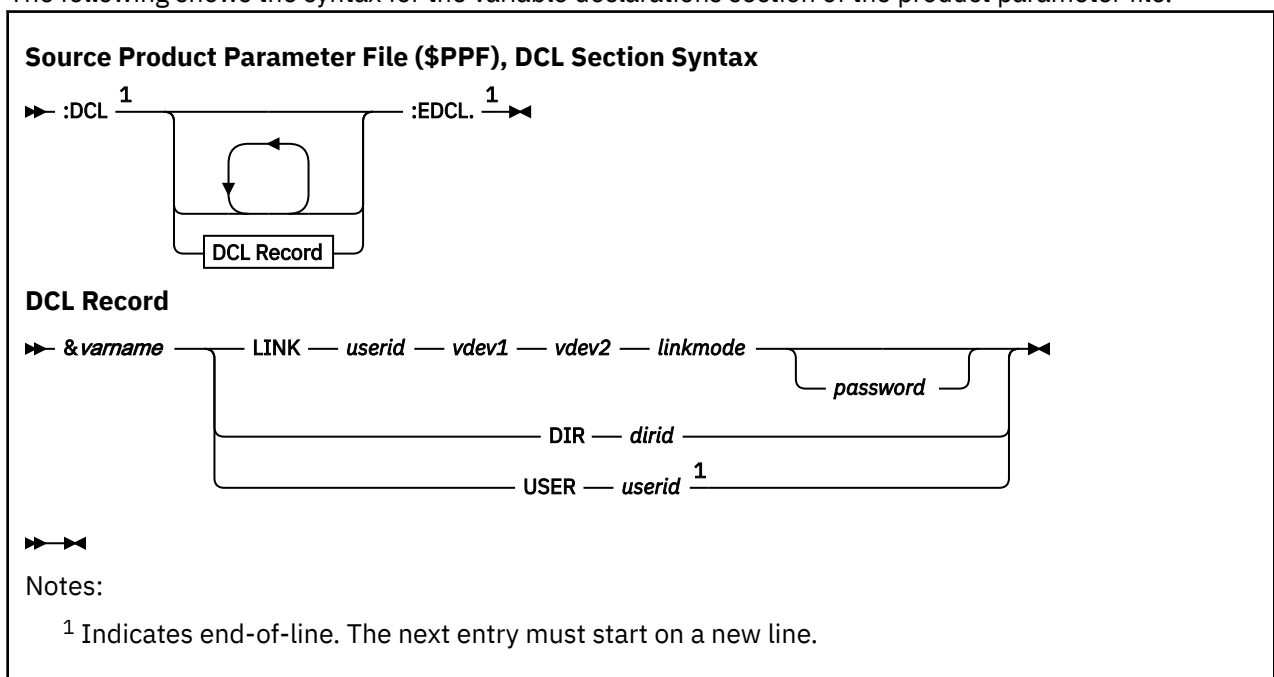
Figure 179. Sample :CNTRLOP Section of a PPF

## Variable Declarations Section

The variable declarations section identifies variables and the values assigned to them in the PPF. It is delimited by the :DCL and :EDCL tags. In source and override PPFs, the variables defined in this section are used in the minidisk/directory assignment section. In the usable form PPF, this section is used for resource allocation and minidisk linking. The VMFPPF EXEC substitutes the values of these variables into the minidisk/assignment section when it creates the usable form PPF. The variables can also be used as parameters on product processing exits in the receive installation tape definition and receive service media definition sections of the PPF.

## Syntax

The following shows the syntax for the variable declarations section of the product parameter file.



The tags in this section have the following syntax:

### **:DCL.**

marks the beginning of the variable declarations section for the product.

### **:EDCL.**

marks the end of the variable declarations section for the product.

## **DCL Record**

### **&varname**

is a variable name. The first character must be "&". This variable is used by the :MDA section. It is also used by the :USERDEF and :TARGET sections in the PRODPART file.

### **LINK**

identifies a minidisk target where product code will reside.

#### **userid**

is the user ID of the owner of the target minidisk.

#### **vdev1**

is the owner's address of the target minidisk.

#### **vdev2**

is the address at which to link the specified minidisk. It is the value that is substituted in the :MDA section for *&varname*.

#### **linkmode**

is the link mode, for example RR, W, M, and so on.

#### **password**

is the password for the access mode specified, if required.

**Note:** When defining a target minidisk for a multiconfiguration virtual machine, specify the user ID of the virtual machine as *userid*.

### **DIR**

identifies a fully-qualified SFS directory target where product code will reside. A fully qualified directory identifier is in the format *filepoolid:userid.qualifier* (*qualifier* can be multiple subdirectory identifiers, such as A.B.C).

#### **dirid**

is the value that is substituted in the :MDA section for *&varname*.

### **USER**

matches directly to a :USERDEF statement in a PRODPART file and (usually) identifies the name of a service virtual machine. The VMFINS resource manager needs this information to define virtual machine entries in the CP Directory.

#### **userid**

is the name of a service virtual machine.

## **Example**

Figure 180 on [page 633](#) shows an example variable declarations section.

```

:DCL.
&LMDZ DIR POOL1:MAINT $\nu$ rm.LOCALMOD MR
&SAMPZ DIR POOL1:MAINT $\nu$ rm.LOCALSMP MR
&DELTY LINK MAINT $\nu$ rm 5D6 5D6 MR
&DELTZ LINK MAINT $\nu$ rm 5D2 5D2 MR
&APPLX LINK MAINT $\nu$ rm 5A6 5A6 MR
&APPLY LINK MAINT $\nu$ rm 5A4 5A4 MR
&APPLZ LINK MAINT $\nu$ rm 5A2 5A2 MR
&BAS3Z LINK MAINT $\nu$ rm 5B4 5B4 MR
&BLD2Z LINK MAINT 193 193 MR
&BLD5Z LINK MAINT 19D 19D MR
&BLD6Z LINK MAINT $\nu$ rm 490 490 MR
&BLD7Z LINK MAINT $\nu$ rm 493 493 MR
&BLD8Z LINK MAINT $\nu$ rm 5E6 5E6 MR
&SRVU1 USER VMSERVU
:EDCL.

```

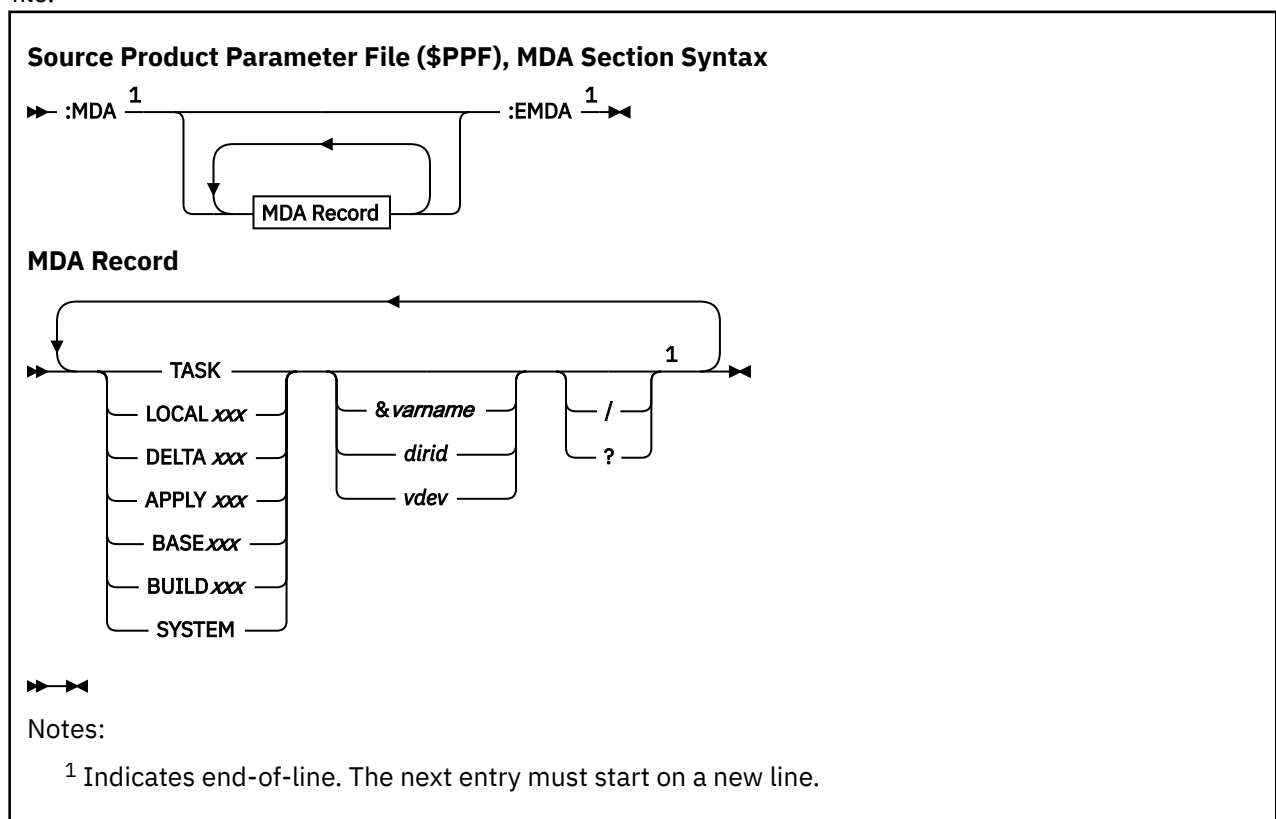
Figure 180. Sample :DCL Section of a PPF

## Minidisk/Directory Assignments Section

The minidisk/directory assignments section identifies the symbolic strings of minidisks or SFS directories that make up the service database of the product. It is delimited by the :MDA and :EMDA tags. In source and override PPFs, this section may contain variables defined in the variable declarations section. In the usable form PPF, however, all of these variables have been resolved by VMFPPF.

### Syntax

The following shows the syntax of the minidisk/directory assignments section of the product parameter file.



The tags in this section have the following syntax:

**:MDA.**

marks the beginning of the minidisk/directory assignments section for the product.

**:EMDA.**

marks the end of the minidisk/directory assignments section for the product.

### **MDA Record**

In the syntax shown above, *String* is a 1-8 character symbolic string name. The string name must begin in column 1. Duplicate string names are used for strings that require more than one line.

The valid string names are:

#### **TASK**

contains any minidisks or directories you want accessed before the service database defined for the product.

#### **LOCALxxx**

contains customized files such as local modifications and circumventive service.

#### **DELTAxxx**

contains PTFs. PTFs contain the raw materials required to build systems after applying service: update files, replacement parts, and PTF parts lists. The DELTA string also contains the service-level description table, the service-level requisite table, and the service-level receive status table.

#### **APPLYxxx**

contains the files that define maintenance levels: AUX files, version vector tables, the service-level apply status table, the select data file, and the service-level build status table.

#### **BASExxx**

contains the product raw materials: source files and base object files.

#### **BUILDxxx**

contains the usable system.

#### **SYSTEM**

contains any minidisks or directories you want accessed after the service database defined for the product.

**Note:** The *xxx* in the string names indicates you can define different strings within a string type. You can use this capability to give your strings more meaningful names. For more information, see [“The VMSES/E Database” on page 105](#).

#### **&varname**

is a variable that is defined in the variable declarations section on a LINK or DIR statement. (It would not be on a USER statement since this statement cannot be in the :MDA record.)

#### **dirid**

is the name of a fully-qualified SFS directory.

#### **vdev**

is the address of a minidisk.

**/**

causes VMFSETUP to access the string as read-only. For example:

```
DELTA 2D6 2D4/ 2D2/
```

indicates the 2D4 and 2D2 disks are to be accessed as read-only by accessing them as extensions of themselves (for example, mode/mode) by the VMFSETUP EXEC.

- If a minidisk or directory appears more than once, its first specification determines its status.
- If a minidisk or directory is already accessed read-write, it is released and reaccessed read-only.
- If a minidisk is empty (there are no CMS files on it), it cannot be accessed as read-only. When a read-only access is requested for an empty disk, a warning message is issued; and the disk is accessed as read/write. If the disk is linked read-only, an error message is issued and the disk is not accessed.

A minidisk address, directory name, or variable can appear in more than one string.

**Note:** The VMFSETUP EXEC accesses minidisks and directories by string name in the order they are specified above. Within each string, the minidisks and directories are accessed from left to right. The first occurrence of the disk or directory establishes its place in the access order.



?

causes VMFSETUP to not access the string. For example:

```
SYSTEM 1DF?
```

indicates the 1DF disk is not to be accessed, but will be linked if the LINK option is specified.

## Example

Figure 181 on page 635 shows an example minidisk/directory assignments section in a source PPF.

```
:MDA.
LOCALMOD    &LMODZ
LOCALSAM    &SAMPZ
APPLY       &APPLX &APPLY &APPLZ
DELTA       &DELTY &DELTZ
BUILD8      &BLD8Z

BUILD7      &BLD7Z
BUILD6      &BLD6Z
BUILD5      &BLD5Z
BUILD2      &BLD2Z
BASE3       &BAS3Z
SYSTEM
:EMDA.
```

Figure 181. Sample :MDA Section of a Source PPF

Figure 182 on page 635 shows an example minidisk/directory assignments section in a usable form PPF. Notice the variables have been resolved.

```
:MDA.
LOCALMOD    POOL1:MAINTurm.LOCALMOD
LOCALSAM    POOL1:MAINTurm.LOCALSMP
APPLY       5A6 5A4 5A2
DELTA       5D6 5D2
BUILD8      5E6

BUILD7      493
BUILD6      490
BUILD5      19D
BUILD2      193
BASE3       5B4
SYSTEM
:EMDA.
```

Figure 182. Sample :MDA Section of a Usable Form PPF

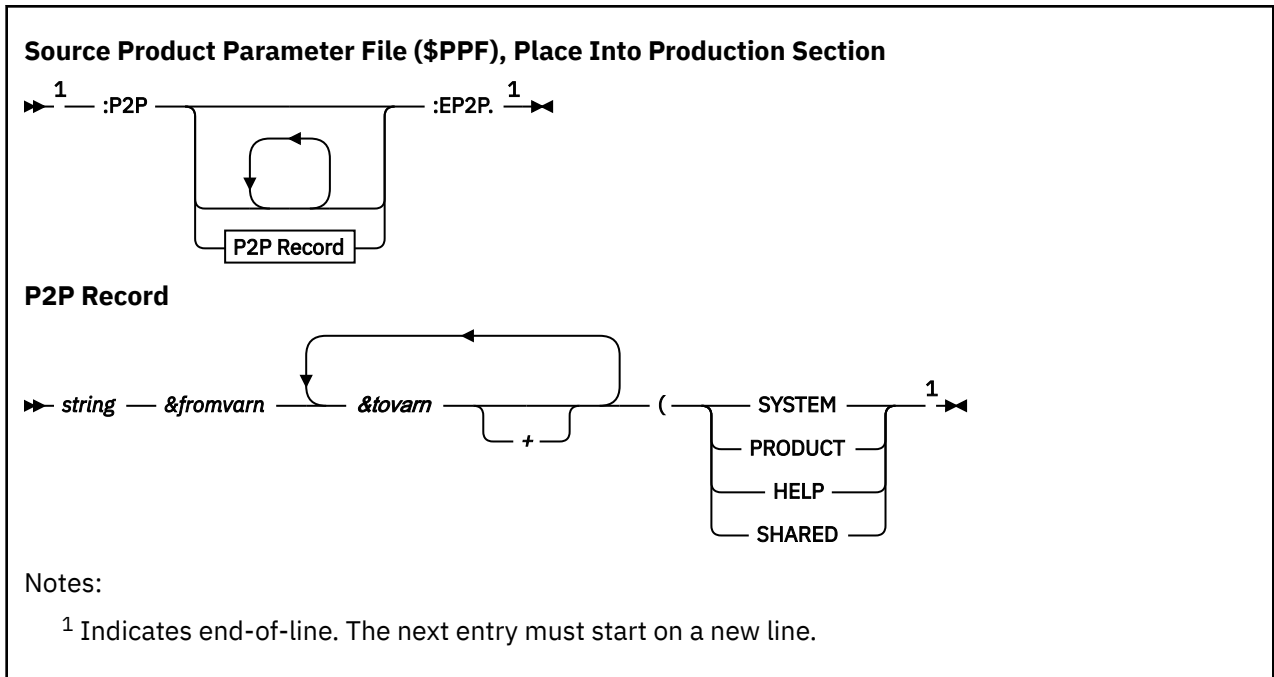
## Place Into Production Section

The place into production section identifies test build and production build minidisks or SFS directories to be used when copying serviced objects into production. Each set of minidisks are optionally flagged as a system disk, a product disk, or a help disk.

The place into production section is optional.

## Syntax

The following shows the syntax for the place into production section of the product parameter file.



The tags in this section have the following syntax:

- :P2P.**  
marks the beginning of the place into production section.
- :EP2P.**  
marks the end of the place into production section.

**P2P Record**

**string**  
is the 1–8 character symbolic MDA section string for a test build minidisk or SFS directory. See “MDA Record” on page 634 for a description of the allowed *string* values.

**&fromvarn**  
is the variable name from the DCL section that defines a test build minidisk or SFS directory.

**&tovarn**  
is the variable name from the DCL section that defines a production build minidisk or SFS directory.

**+**  
A "+" at the end of the variable name causes an upper case copy to occur.

**SYSTEM**  
identifies a CMS system disk, such as 190.

**PRODUCT**  
identifies a product code disk, such as 19E.

**HELP**  
identifies a product help disk, such as 19D.

**SHARED**  
identifies a disk that is shared by two or more systems.

**Example**

Figure 183 on page 637 shows an example variable declarations section.



part is in a separate CMS file within the tape file. Tape file names cannot appear in the :RECINS section more than once.

### ***phexec***

is the name of the VMFREC part handler used to process the parts in the tape file:

#### **VMFRCALL**

receives parts unconditionally.

#### **VMFRCAXL**

receives apply and exclude lists.

#### **VMFRCCOM**

receives parts conditionally.

#### **VMFRCPTF**

receives PTF parts lists conditionally.

#### **VMFRCUPP**

receives files unconditionally. The tape files are loaded to a target disk and changed to uppercase.

If the part handler name begins with a dash (-), the associated tape file is not received. For more information on the part handlers used by VMFREC, see [“A Closer Look at the VMFREC EXEC” on page 108](#).

## ***String***

In the syntax diagram, *String* is the 1-8 character upper-case symbolic string name to which the tape file is loaded. For more information on strings, see [“The VMSES/E Database” on page 105](#). The symbolic string must be defined in the minidisk/directory assignments section.

Valid string names are:

### **TASK**

contains any minidisks or directories you want accessed before the service database defined for the product.

### **LOCALxxx**

contains customized files such as local modifications and circumventive service.

### **DELTAxxx**

contains PTFs. PTFs contain the raw materials required to build systems after applying service: update files, replacement parts, and PTF parts lists. The DELTA string also contains the service-level description table, the service-level requisite table, and the service-level receive status table.

### **APPLYxxx**

contains the files that define maintenance levels: AUX files, version vector tables, the service-level apply status table, the select data file, and the service-level build status table.

### **BASExxx**

contains the product raw materials: source files and base object files.

### **BUILDxxx**

contains the usable system.

### **SYSTEM**

contains any minidisks or directories you want accessed after the service database defined for the product.

### **PPEXIT**

indicates a product processing exit to be run when it is encountered in the receive installation tape definition section.

#### ***ppexec***

is the file name of the product processing exit.

#### ***parms***

is a list of parameters passed to the product processing exit.

**Note:** This interface is intended for use by program products that make use of VMSES/E. IBM does not intend this interface for customer use.

## Example

Figure 184 on page 639 shows a sample receive installation tape definition section.

```

:RECINS.
AXLIST      VMFRCAXL DELTA      * Apply and Exclude Lists
PARTLST     VMFRCPTF DELTA     * $PTFPART Files
DELTA       VMFRCOM  DELTA     * Service
APPLY       VMFRCALL APPLY     * Service
TOOLS       VMFRCALL BUILD7    * Tools
SYSTEM      VMFRCALL BUILD6    * System Disk
NCHHELP     VMFRCUPP BUILD5    * Uppercase HELP Files
:ERECINS.
    
```

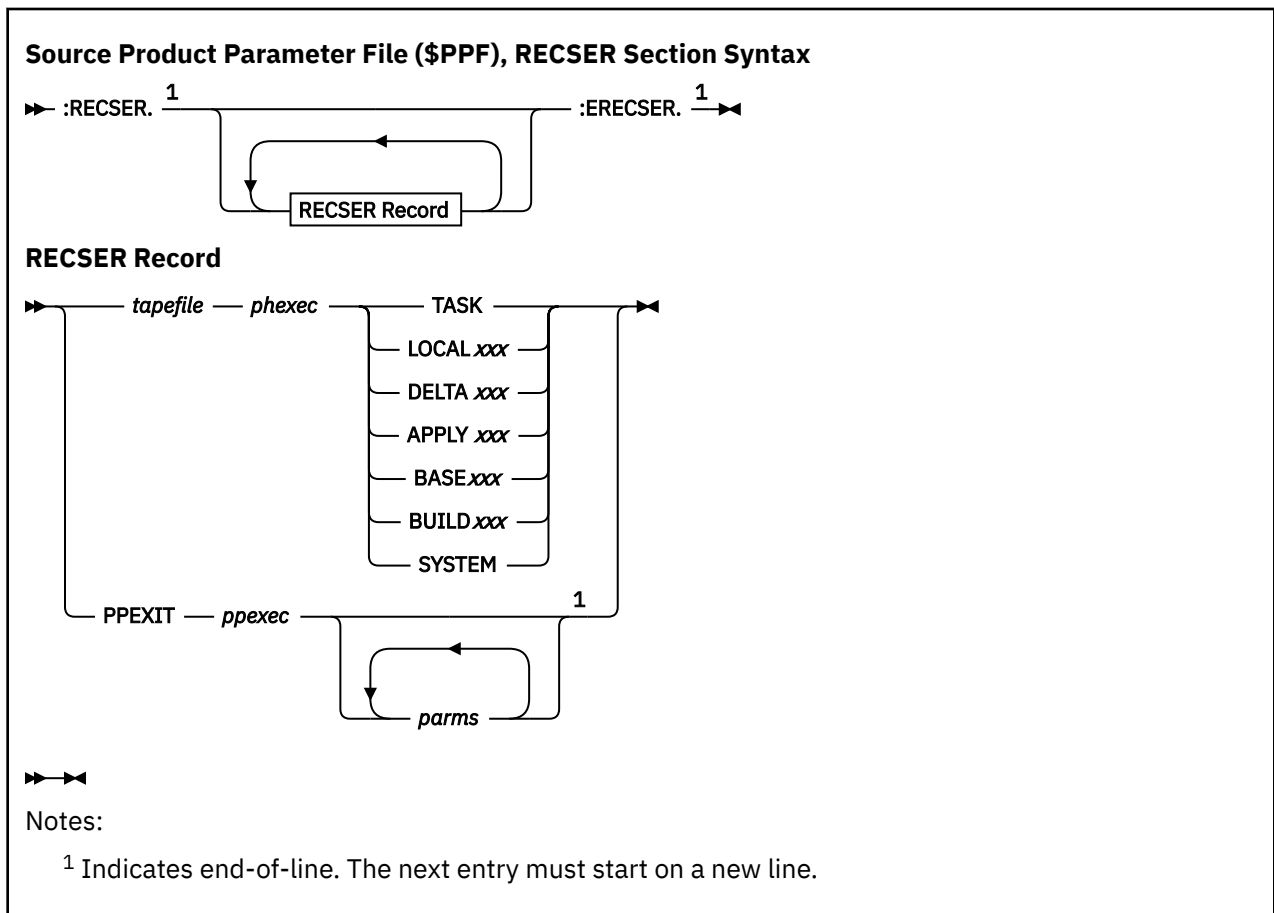
Figure 184. Sample :RECINS Section of a PPF

## Receive Service Media Definition Section

The receive service media definition section defines the layout of service tapes and envelopes for the product. It is delimited by the :RECSER and :ERECSER tags.

### Syntax

The following shows the syntax of the receive service media definition section of the product parameter file.



The tags in this section have the following syntax:

### **:RECSER.**

marks the beginning of the receive service media definition section for the product.

### **:ERECSER.**

marks the end of the receive service media definition section for the product.

### ***RECSER Record:***

#### ***tapefile***

is a 1-8 upper-case character symbolic tape file name. There is one *tapefile* for each tape file on the service media for the product. *tapefile* may start in any column. Each tape file contains all the product parts of the same or a related type. The attribute that binds the types of parts in a particular tape file is that they all require the same receive processing and are loaded to the same target. Each part is in a separate CMS file within the tape file. Tape file names cannot appear in the :RECSER section more than once.

#### ***phexec***

is the name of the VMFREC part handler used to process the parts in the tape file:

#### **VMFRCALL**

receives parts unconditionally.

#### **VMFRCAXL**

receives apply and exclude lists.

#### **VMFRCCOM**

receives parts conditionally.

#### **VMFRCPTF**

receives PTF parts lists conditionally.

#### **VMFRCUPP**

receives tape files unconditionally. The tape files are loaded to a target disk and changed to uppercase.

If the part handler name begins with a dash (-), the associated tape file is not received. For more information on the part handlers used by VMFREC, see [“A Closer Look at the VMFREC EXEC” on page 108](#).

### ***String***

In the syntax diagram, *String* is the upper-case symbolic string name to which the tape file is loaded. For more information on strings, see [“The VMSES/E Database” on page 105](#). The symbolic string must be defined in the minidisk/directory assignments section.

Valid string names are:

#### **TASK**

contains any minidisks or directories you want accessed before the service database defined for the product.

#### **LOCALxxx**

contains customized files such as local modifications and circumventive service.

#### **DELTAxxx**

contains PTFs. PTFs contain the raw materials required to build systems after applying service: update files, replacement parts, and PTF parts lists. The DELTA string also contains the service-level description table, the service-level requisite table, and the service-level receive status table.

#### **APPLYxxx**

contains the files that define maintenance levels: AUX files, version vector tables, the service-level apply status table, the select data file, and the service-level build status table.

#### **BASExxx**

contains the product raw materials: source files and base object files.

#### **BUILDxxx**

contains the usable system.

**SYSTEM**

contains any minidisks or directories you want accessed after the service database defined for the product.

**PPEXIT**

indicates a product processing exit to be run when it is encountered in the receive service media tape definition section.

***ppexec***

is the file name of the product processing exit.

***parms***

is a list of parameters passed to the product processing exit.

**Note:** This interface is intended for use by program products that make use of VMSES/E. IBM does not intend this interface for customer use.

**Example**

Figure 185 on page 641 shows an example receive service media definition section.

```
:RECSER.  
AXLIST      VMFRCAXL DELTA  
PARTLIST    VMFRCPTF DELTA  
DELTA       VMFRCCOM DELTA  
:ERECSESR.
```

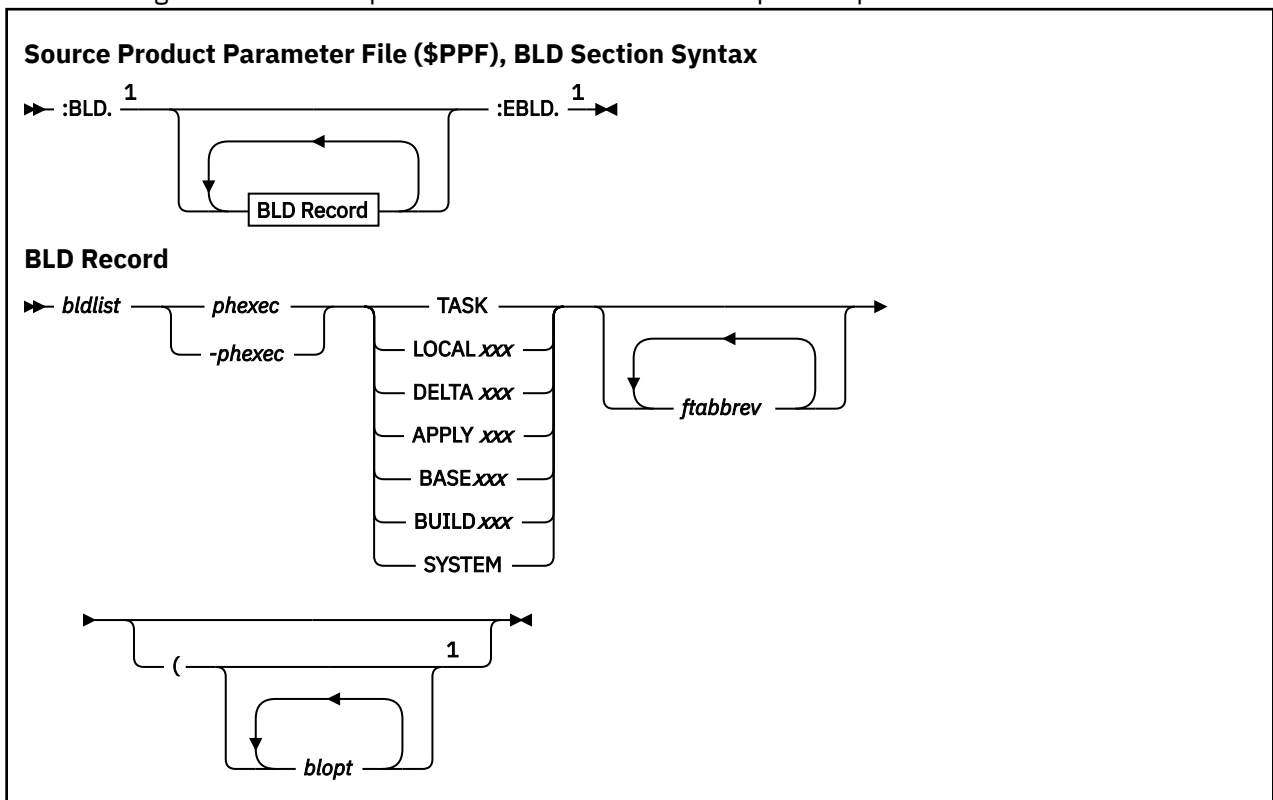
Figure 185. Sample :RECSER Section of a PPF

**Build Product Definition Section**

The build product definition section defines build processing for the product. It is delimited by the :BLD and :EBLD tags.

**Syntax**

The following shows the build product definition section of the product parameter file.





Notes:

<sup>1</sup> Indicates end-of-line. The next entry must start on a new line.

The tags in this section have the following syntax:

**:BLD.**

marks the beginning of the build product definition section for the product.

**:EBLD.**

marks the end of the build product definition section for the product.

### ***BLD Record***

***bldlist***

is a 1-8 upper-case character build list file name. There is one *bldlist* for each build list defined for the product. *bldlist* may start in any column. The attribute that binds objects in a particular build list is that they all require the same build processing and are built to the same target. Build list names cannot appear in the :BLD section more than once.

You should not remove build lists from product parameter files. If you do not want to ever build any object in a specific build list, create a product parameter file override and bypass it.

***phexec***

is the name of the VMFBLD part handler used to process the objects in the build list.

**VMFBDCLB**

builds callable services libraries.

**VMFBDCOM**

builds replacement objects (format 2 build lists).

**VMFBDCPY**

builds replacement text only (format 1 build lists).

**VMFBDDDR**

restores DDR image files to minidisks.

**VMFBDDL**

builds CMS/DOS Phase libraries.

**VMFBDCGEN**

builds generated objects, such as text decks.

**VMFBDLLB**

builds LOADLIBs.

**VMFBMMLB**

builds MACLIBs.

**VMFBDMOD**

builds MODULEs.

**VMFBNUC**

builds nuclei.

**VMFBDSBR**

identifies system objects (saved segments) that need to be built.

**VMFBDSEG**

builds segments.

**VMFBDTLB**

builds TXTLIBs.

For more information on the part handlers used by VMFBLD, see [“Creating Objects with VMFBLD” on page 325.](#)



**-phexec**

indicates you want to bypass the build list. If the part handler name begins with a dash (-), all objects in the associated build list are bypassed. See [Figure 186 on page 643](#) for an example.

**String**

In the syntax diagram, **String** is the upper-case symbolic string name to which the objects in the build list are built. For more information on strings, see [“The VMSES/E Database” on page 105](#). The symbolic string must be defined in the minidisk/directory assignments section.

Valid string names are:

**TASK**

contains any minidisks or directories you want accessed before the service database defined for the product.

**LOCALxxx**

contains customized files such as local modifications and circumventive service.

**DELTAxxx**

contains PTFs. PTFs contain the raw materials required to build systems after applying service: update files, replacement parts, and PTF parts lists. The DELTA string also contains the service-level description table, the service-level requisite table, and the service-level receive status table.

**APPLYxxx**

contains the files that define maintenance levels: AUX files, version vector tables, the service-level apply status table, the select data file, and the service-level build status table.

**BASExxx**

contains the product raw materials: source files and base object files.

**BUILDxxx**

contains the usable system.

**SYSTEM**

contains any minidisks or directories you want accessed after the service database defined for the product.

**ftabbrev**

are the file type abbreviations for the serviceable parts listed in the build list. The *ftabbrev* must be the 3-character PTF abbreviation or the real CMS file type for parts that are not serviced by replacement. This field is only valid for format 1 build lists.

**blopt**

are build list options which are passed to the part handler. For more information on build list options, see [“Build Lists” on page 141](#), [“VMFBLD EXEC” on page 305](#), or [“Creating Objects with VMFBLD” on page 325](#).

**Example**

[Figure 186 on page 643](#) shows an example build product definition section.

```
:BLD.VMF
BLHLP  VMFBDCOM  BUILD5VMF
BLSSES VMFBDCOM  BUILD8VMF
BLSYS  VMFBDCOM  BUILD6VMF
BLSRC  VMFBDCOM  BUILD7VMF
MLOAD  VMFBDMOD  BUILD8VMF
SLOAD  VMFBDMOD  BUILD6VMF
BLPPF  -VMFBDCOM BUILD7
:EBLD.
```

*Figure 186. Sample :BLD Section of a PPF*

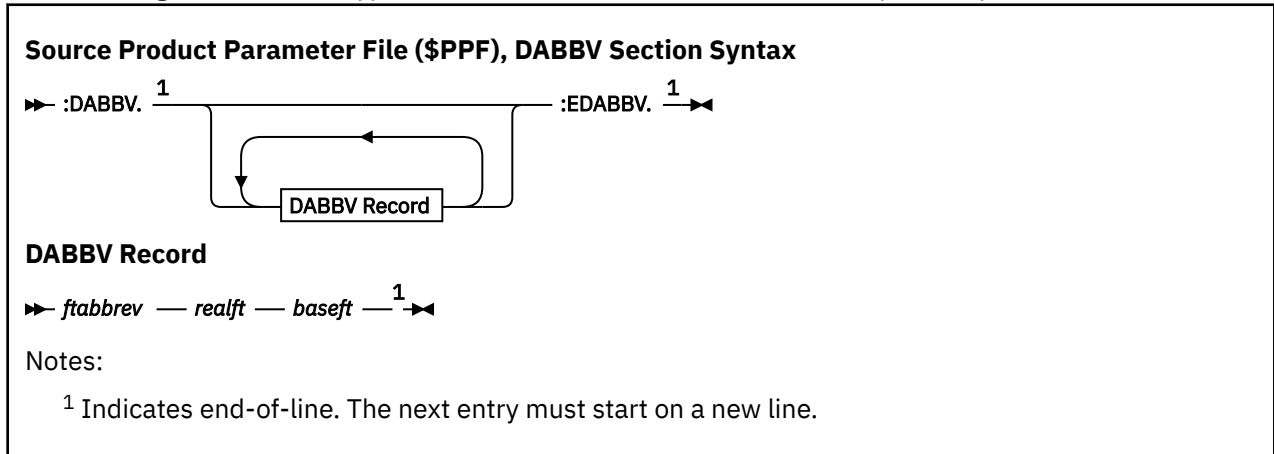
In [Figure 186 on page 643](#), the VMFBLPPF build list has been bypassed, so no objects will be built.

## File Type Abbreviations Extensions Section

The file type abbreviations extensions section defines file type abbreviations for the product, which override entries in the file type abbreviation table (VM SYSABRVT). It is delimited by the :DABBV and :EDABBV tags.

### Syntax

The following shows the file type abbreviations extensions section of the product parameter file.



The tags in this section have the following syntax:

**:DABBV.**

marks the beginning of the file type abbreviations extensions section for the product.

**:EDABBV.**

marks the end of the file type abbreviations extensions section for the product.

### **DABBV Record**

***ftabbrev***

is the 3-character abbreviation for the real CMS file type.

***realft***

is the real CMS file type, for example, TEXT or EXEC.

***baseft***

is the 1-8 character file type that corresponds to the base level of a serviceable part (for example, the level of the part that has never been serviced).

### Example

Figure 187 on page 644 shows an example file type abbreviations extensions section.

```
:DABBV.
NXT NEWTYPE NXT000000
AXT APPTYPE AXT000000
:EDABBV.
```

Figure 187. Sample :DABBV Section of a PPF

## Override Area

Override areas are delimited by the *:override* and :END tags and may contain the :PRODID tag and any of the following sections:

- “Control Options Section” on page 625
- “Variable Declarations Section” on page 631

- [“Minidisk/Directory Assignments Section” on page 633](#)
- [“Receive Installation Tape Definition Section” on page 637](#)
- [“Receive Service Media Definition Section” on page 639](#)
- [“Build Product Definition Section” on page 641](#)
- [“File Type Abbreviations Extensions Section” on page 644](#)

These sections contain parameters for the product.

An override area should only contain tags and sections for the component areas you want to change.

With the exception of the `:overname` tag, all of the tags and sections in the component area of the source PPF are the same as in the component area of the usable form PPF. The syntax of the tags that begin sections is extended to help you define changes that you want to make to the component area. Remember, in override areas, variables defined in the variable declarations section have not yet been resolved.

The override area has the following syntax (see [“Source Product Parameter File \(\\$PPF\), Overall Syntax” on page 623](#)):

**`:overname.`**

marks the beginning of an override area. `:overname` is a 1-8 character uppercase name or keyword that identifies a product. The `:overname` tag is used by the VMFOVER EXEC to locate the override areas listed on the `:OVERLST` tag.

**`compname`**

is a pointer to the component area that this override area changes.

**`novername`**

is a pointer to the next override area in the chain of override areas that eventually lead to a component area.

**`ppfname`**

is a pointer to the source or override PPF that contains `compname` or `novername`. If `ppfname` is not supplied, it defaults to the name of the source or override PPF that contains `overname`.

## Tag Extensions

VMSES/E provides extensions for the tags in the product parameter file. The syntax diagrams in the following sections show the extensions to the tags that begin each section of the product parameter file.

UPDATE indicates the statements in that section of the override area update the corresponding statements in the component or override area that is pointed to on the `:overname` tag. When UPDATE is used, the original data is commented out with an asterisk (\*) in the first column; and the new data is inserted on the following lines. If there is no original data that corresponds (first token matches) to the new data, the new data is added to the end of the section.

REPLACE indicates the statements in that section of the override area replace all the statements in the corresponding section in the component area or override area that is pointed to on the `:overname` tag. When REPLACE is used, the original data is completely removed (that is, the data is not commented out).

**Note:** Use REPLACE not UPDATE to override an empty section of the PPF.

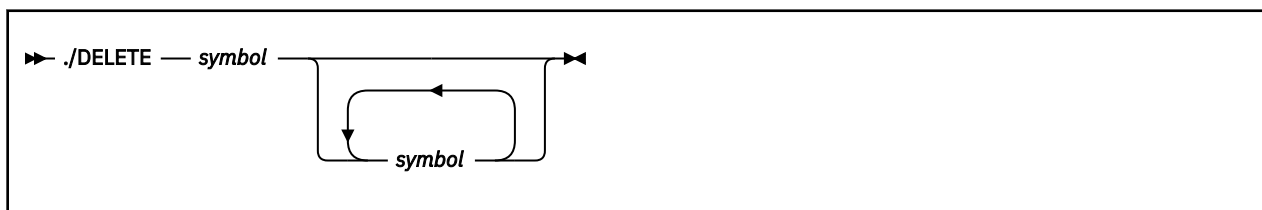
## Override Control Records

In addition to the update and replace capabilities, VMSES/E provides override control records to insert and delete data from updateable sections. The updateable sections are `:DCL`, `:MDA`, `:RECINS`, `:RECSER`, `:BLD` and `:DABBV`. Override control records should be used in conjunction with the UPDATE keyword.

In the following sections, additional notation is used to show the syntax of the override control records. For a description of this notation, see [“Syntax Notation” on page 142](#).

### Delete Override Control Record

The delete override control record has the following syntax:



#### **./DELETE**

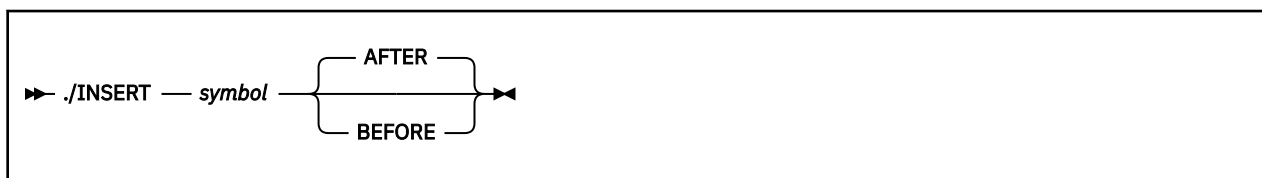
indicates records to be deleted from a section. Multiple `./DELETE` records are allowed in a section.

#### **symbol**

is the first token of a line to be deleted from the section. All occurrences of *symbol* will be deleted within the section. Leading and trailing comments are deleted along with the record.

### Insert Override Control Record

The insert override control record has the following syntax:



#### **./INSERT**

indicates the start of an insert block. All of the data in an insert block is inserted, even other control records and tags. No checking is done to ensure the validity of PPF symbols in an insert block.

#### **symbol**

is the token that indicates where to insert the block.

#### **BEFORE**

indicates the data in the insert block is to be inserted immediately preceding the first occurrence of *symbol* in the section, unless there are leading comments, in which case the insertion precedes the leading comments.

#### **AFTER**

indicates the data in the insert block is to be inserted immediately following the last occurrence of *symbol* in the section, unless there are trailing comments, in which case the insertion follows the trailing comments. AFTER is the default.

#### **./END**

indicates the end of an insert block. The `./END` record is required for each insert block.

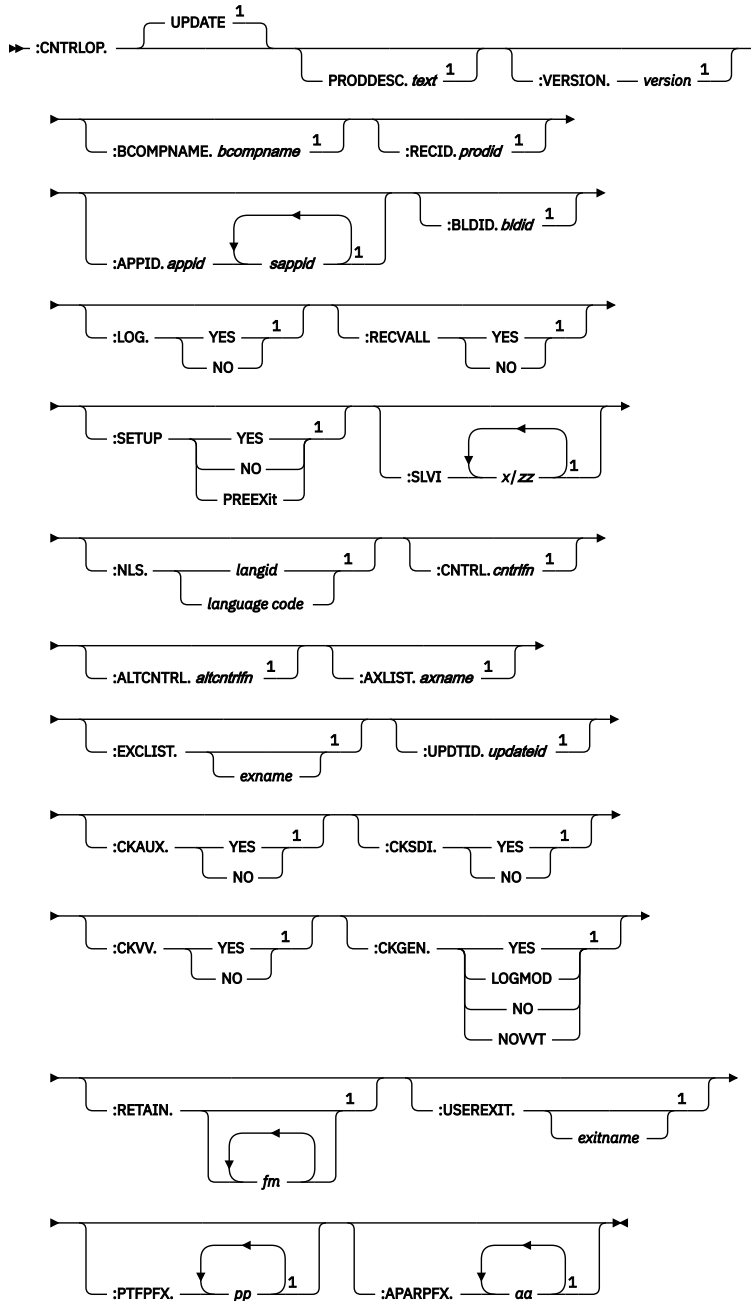
## Override Area Syntax

The following sections show the syntax for the override areas in the source product parameter file.

### **:CNTRLOP. UPDATE**

The extension to the `:CNTRLOP` tag introduces overrides for the control options section.

## Source Product Parameter File (\$PPF), CNTRLOP Section Override Syntax



## Notes:

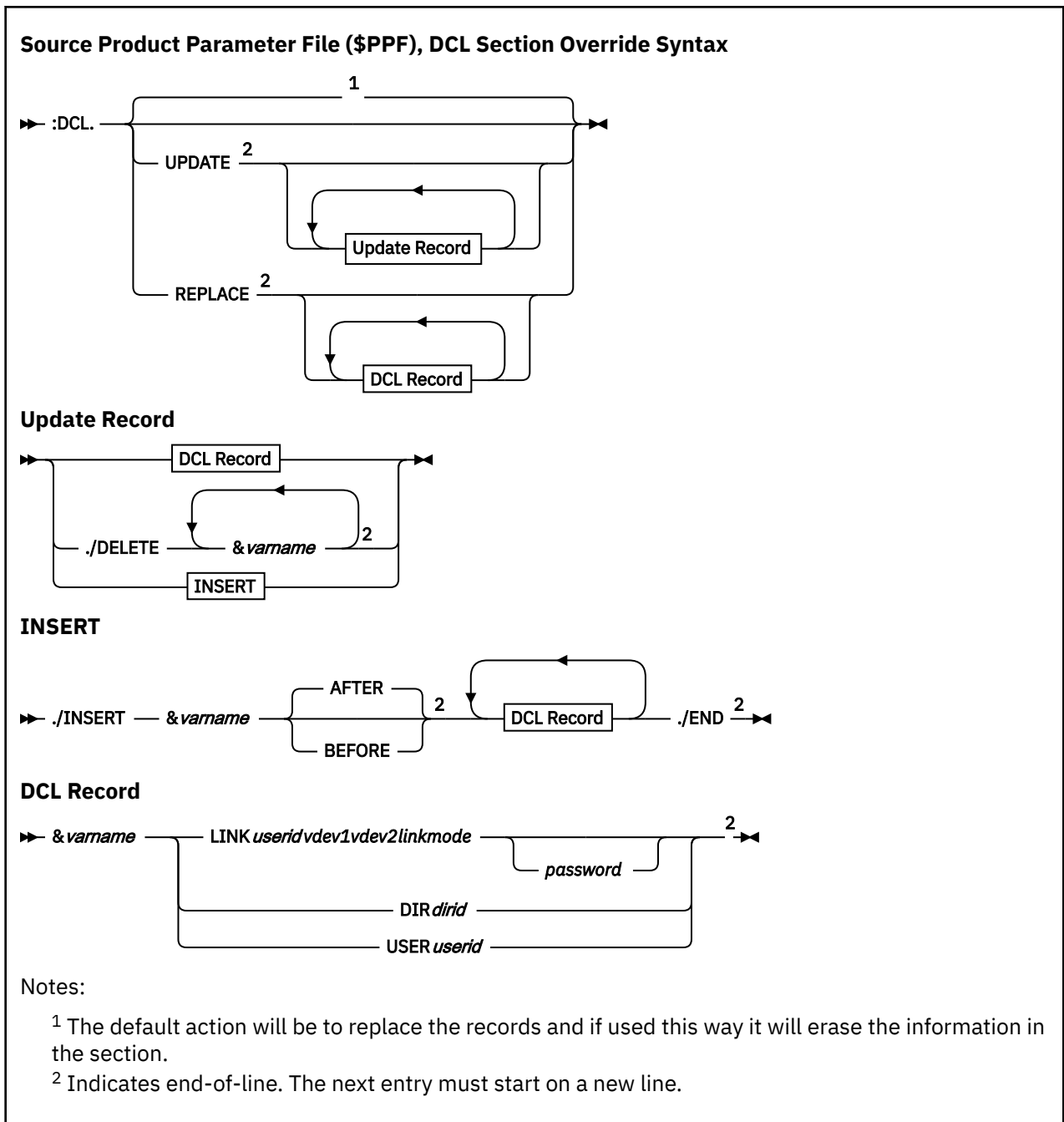
<sup>1</sup> Indicates end-of-line. The next entry must start on a new line.

**Note:** The REPLACE keyword is not valid for this section because all of the data in this section is tagged data.

For a complete description of the control options section, see “Control Options Section” on page 625.

**:DCL. UPDATE or REPLACE**

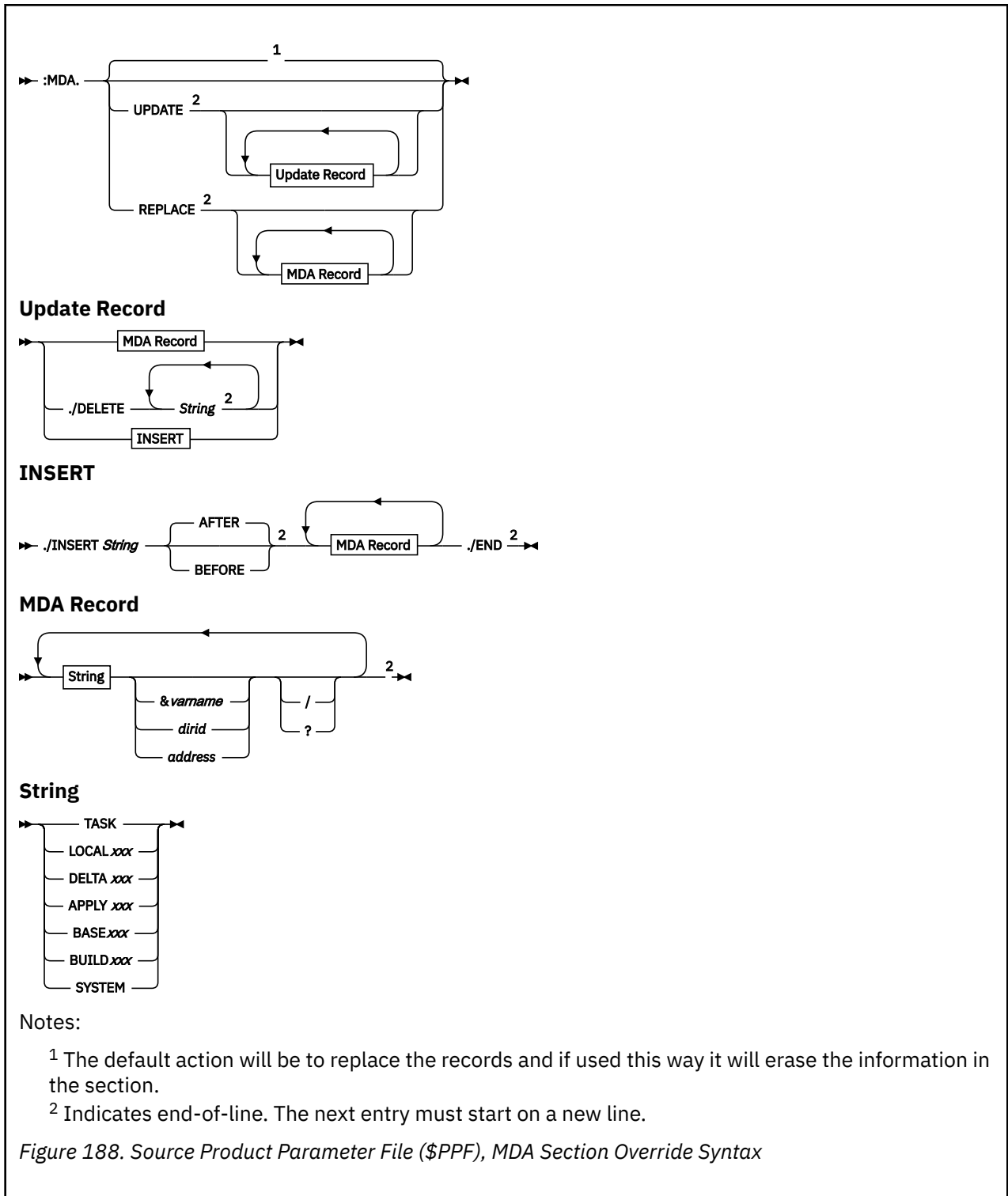
The extension to the :DCL tag introduces overrides for the variable declarations section.



For a complete description of the variable declarations section, see [“Variable Declarations Section”](#) on page 631.

**:MDA. UPDATE or REPLACE**

The extension to the :MDA tag introduces overrides for the minidisk/directory assignments section.

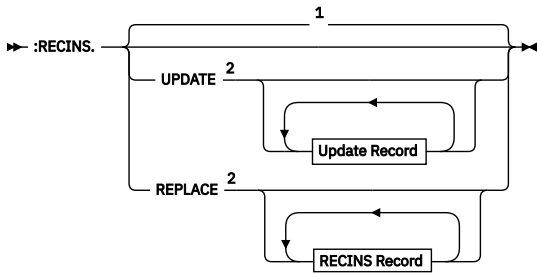


For a complete description of the minidisk/directory assignments section, see [“Minidisk/Directory Assignments Section”](#) on page 633.

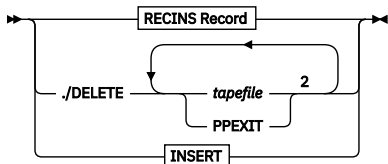
**:RECINS. UPDATE or REPLACE**

The extension to the :RECINS tag introduces overrides for the receive installation tape definition section.

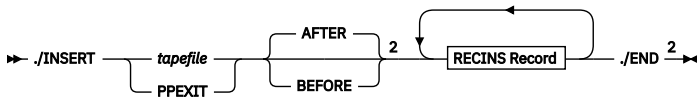
**Source Product Parameter File (\$PPF), RECINS Section Override**



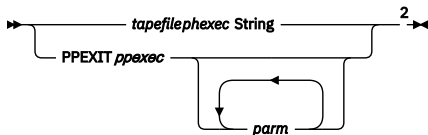
**Update Record**



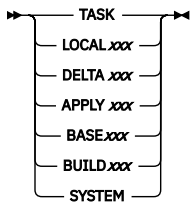
**INSERT**



**RECINS Record**



**String**



**Notes:**

- <sup>1</sup> The default action will be to replace the records and if used this way it will erase the information in the section.
- <sup>2</sup> Indicates end-of-line. The next entry must start on a new line.

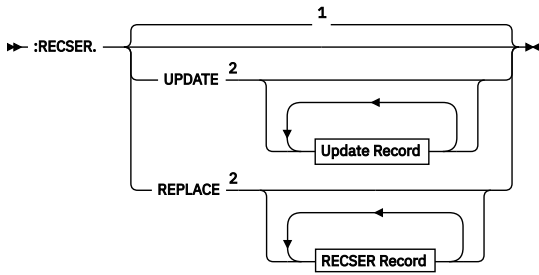
For a complete description of the receive installation tape definition section, see [“Receive Installation Tape Definition Section”](#) on page 637.

**:RECSER. UPDATE or REPLACE**

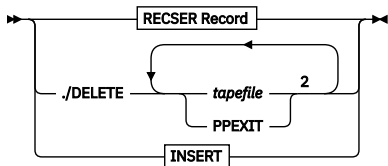
The extension to the :RECSER tag introduces overrides for the receive service media definition section.



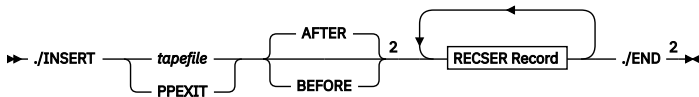
**Source Product Parameter File (\$PPF), RECSER Section Override**



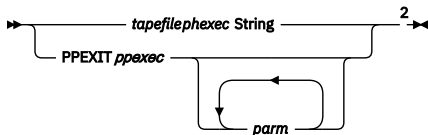
**Update Record**



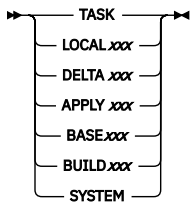
**INSERT**



**RECSER Record**



**String**



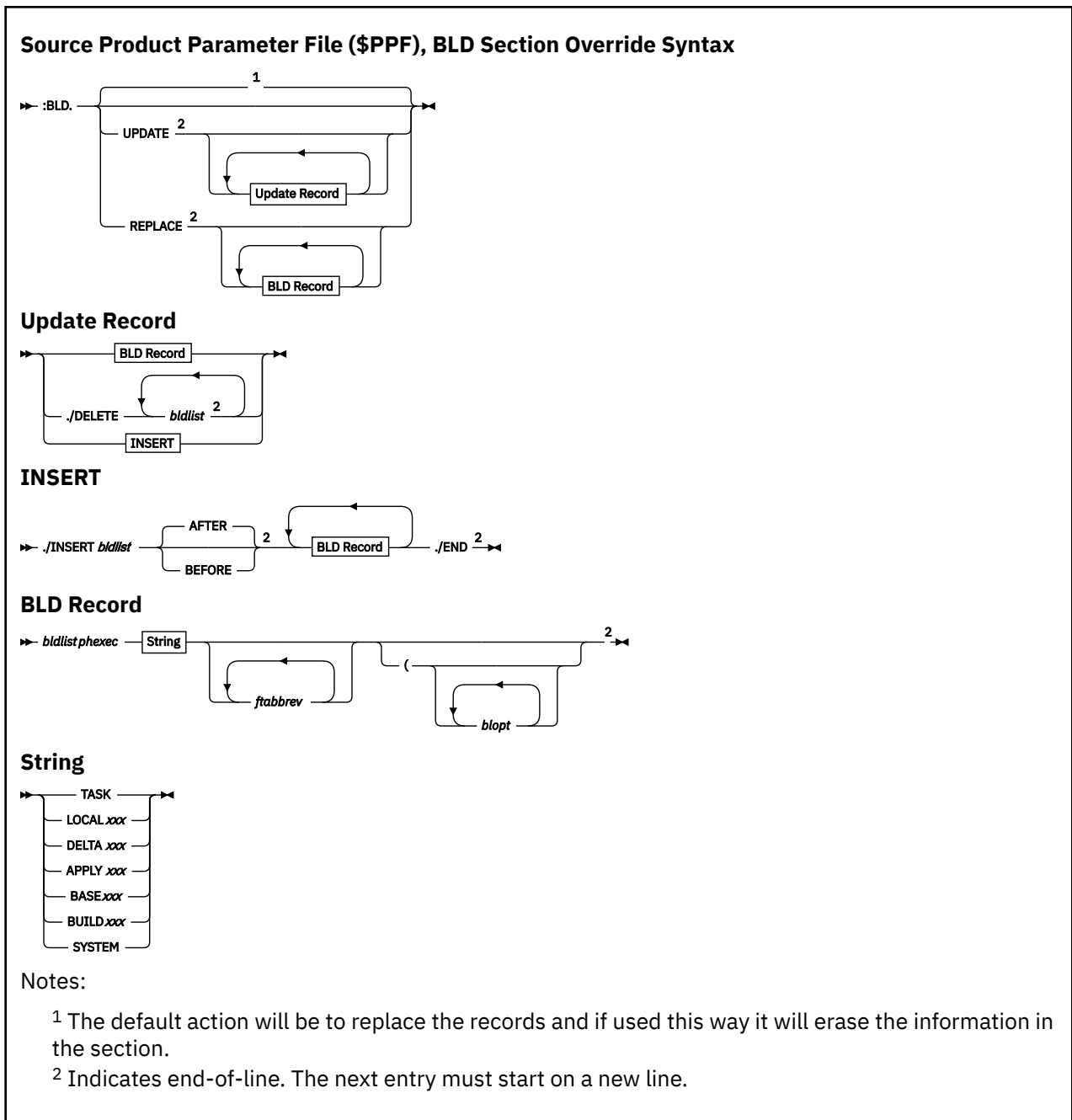
**Notes:**

- <sup>1</sup> The default action will be to replace the records and if used this way it will erase the information in the section.
- <sup>2</sup> Indicates end-of-line. The next entry must start on a new line.

For a complete description of the receive service media definition section, see [“Receive Service Media Definition Section”](#) on page 639.

**:BLD. UPDATE or REPLACE**

The extension to the :BLD tag introduces overrides for the build product definition section.



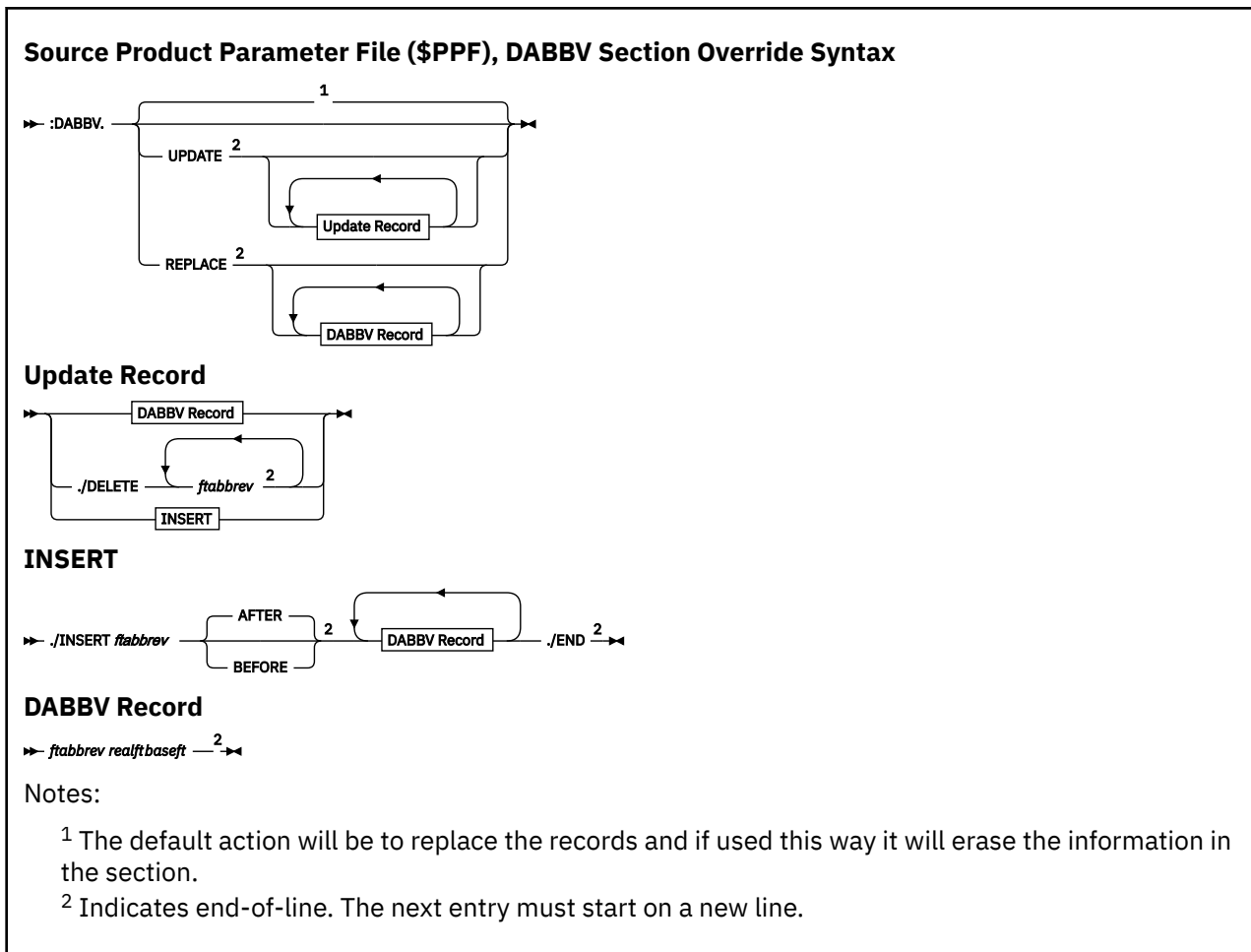
**Note:**

You should not remove build lists from product parameter files when you create overrides. Instead, you should create an override and bypass them. For more information, see [“Build Product Definition Section”](#) on page 641.

For a complete description of the build product definition section, see [“Build Product Definition Section”](#) on page 641.

**:DABBV. UPDATE or REPLACE**

The extension to the :DABBV tag introduces overrides for the file type abbreviations extensions section.



For a complete description of the file type abbreviations section, see [“File Type Abbreviations Extensions Section”](#) on page 644.

## Override Product Parameter File Syntax

Override product parameter files may be supplied by the product or created locally. Override PPFs contain a header area and one or more override areas.

### Header Area

The header area contains only the :OVERLST tag, which is a list of override areas in the override PPF. The header area must appear before any override areas. The syntax of the :OVERLST tag is the same as in source PPFs.

The header area of an override PPF shown in [Figure 189](#) on page 653.

```
:OVERLST. VMSESUCENG VMSESINS VMSESPTFS
```

*Figure 189. Sample Header Area of an Override PPF*

### Override Area

Override areas in override PPFs have the same syntax as override areas in source PPFs. For more information, see [“Override Area”](#) on page 644.

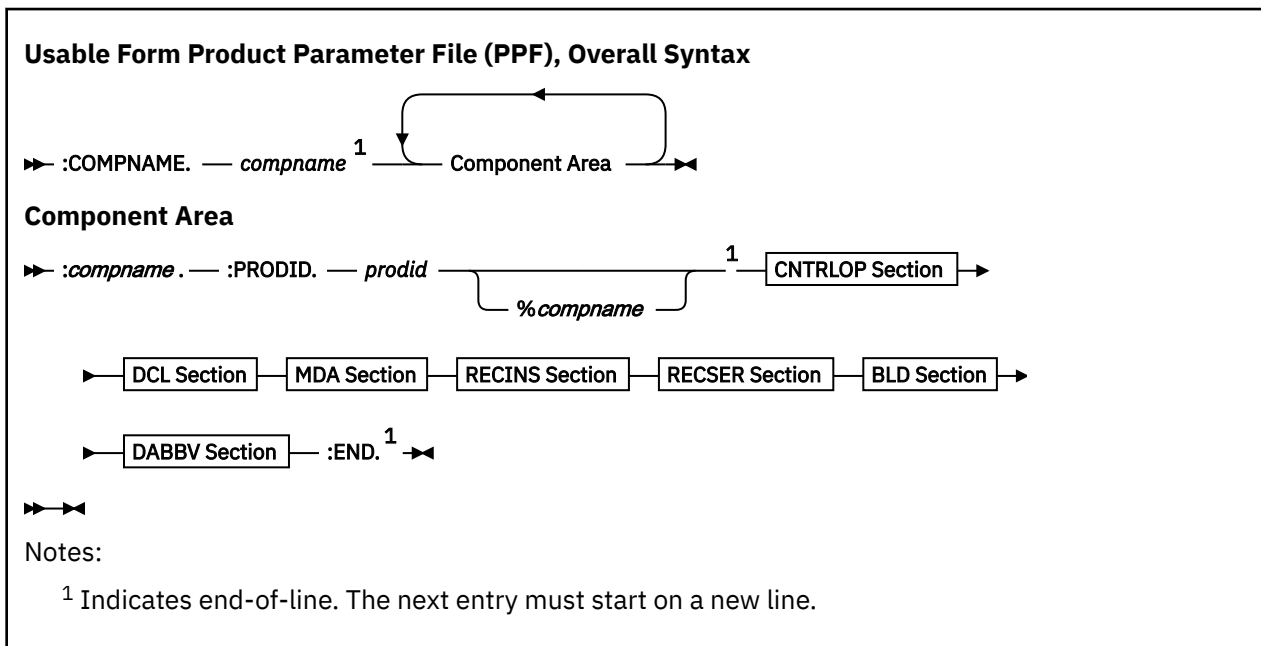
## Temporary Product Parameter File Syntax

Temporary product parameter files are the output of the VMFOVER EXEC. They have the same syntax as source product parameter files, except there can only be one component area.

For more information on the syntax of source PPFs, see [“Source Product Parameter File Syntax”](#) on page 623.

## Usable Form Product Parameter File Syntax

The usable form product parameter file is the final PPF entity that is actually used to control VMSES/E exec processing. It is made up of one or more component areas.



## Component Area

Component areas are delimited by the :COMPNAME and :END tags and contain the :*compname* and :PRODID tags plus the following sections which contain the parameters for the product:

- [“Control Options Section”](#) on page 625
- [“Variable Declarations Section”](#) on page 631
- [“Minidisk/Directory Assignments Section”](#) on page 633
- [“Receive Installation Tape Definition Section”](#) on page 637
- [“Receive Service Media Definition Section”](#) on page 639
- [“Build Product Definition Section”](#) on page 641
- [“File Type Abbreviations Extensions Section”](#) on page 644

The component area has the following syntax:

### **:COMPNAME.**

marks the beginning of a component area in a usable form PPF. The :COMPNAME tag is inserted by the VMFOVER EXEC when it creates a temporary PPF. A product may have multiple component areas if there are different sets of parameters to process it. When this is done, *compname* can be used to provide a meaningful name to each set of parameters for the product.

### ***compname***

1-16 character uppercase name that identifies a product. The value of *compname* is taken from the :*compname* tag in the source PPF.

## Component Area

The tags in the component area are:

### **:compname.**

marks the beginning of a component area in a source PPF and is retained in the usable form PPF. It is a 1-16 character uppercase name that identifies a product. The *:compname* tag is used by the VMFOVER EXEC to locate the component areas listed on the :COMPLST tag in source PPFs.

### **:PRODID.**

identifies the 7-25 character product/component/version-release-modification identifier for PTF validation and product requisites.

### **prodid**

is the 7-8 character alphanumeric identifier assigned to each product/version-release-modification level of the product (may also be referred to as *prodid* in other areas of this book). *prodid* also identifies the file name of the source PPF for the product.

### **%compname**

is the component name identifier, for example CMS, which is added to the *prodid* with a percent sign (%). *compname* is a 1- to 16-character alphanumeric identifier.

### **:END.**

marks the end of a component area.

For information on the:

- CNTRLOP section, see the [“Control Options Section”](#) on page 625
- DCL section, see the [“Variable Declarations Section”](#) on page 631
- RECINS section, see the [“Receive Installation Tape Definition Section”](#) on page 637
- RECSER section, see the [“Receive Service Media Definition Section”](#) on page 639
- MDA section, see the [“Minidisk/Directory Assignments Section”](#) on page 633
- BLD section, see the [“Build Product Definition Section”](#) on page 641
- DABBV section, see the [“File Type Abbreviations Extensions Section”](#) on page 644

## Example

Figure 190 on page 655 shows an example of a component area.

```
:COMPNAME. MYCOMP
:MYCOMP.
:PRODID. 1VMVMC23%MYCOMP
:
:END.
```

Figure 190. Sample Component Area of a Usable Form PPF

## Examples of Overrides

The following examples show the use of override PPFs to change a source PPF. The first three examples show single-level overrides, in which the override area points directly to a component area. The fourth example is a multi-level example in which an override area points to another override area, which then points to a component area. The last example is an override file for system (multiple product) use.

## Inserting a Record

This override inserts a record in the :MDA section of a source PPF.

### Override PPF (INSERT \$PPF)

```
:OVERLST. MYCOMP
:MYCOMP. MYCOMP 1VMVMC23
```

## Product Parameter File Syntax

```
:MDA. UPDATE  
./INSERT LOCALSAM BEFORE  
LOCALMOD 3C4  
./END  
:END.
```

### Source PPF (1VMVMC23 \$PPF)

```
:  
:MYCOMP.  
:  
:MDA.  
LOCALSAM 3C2  
DELTA 3D6 3D4 3D2  
:
```

### Usable Form PPF (INSERT PPF)

```
:  
:COMPNAME.MYCOMP  
:MYCOMP.  
:  
:MDA.  
LOCALMOD 3C4  
LOCALSAM 3C2  
DELTA 3D6 3D4 3D2  
:
```

## Deleting Records

This override deletes two records from the :MDA section of a source PPF.

### Override PPF (DELETE \$PPF)

```
:OVERLST. MYCOMP  
:MYCOMP. MYCOMP 1VMVMC23  
:MDA. UPDATE  
./DELETE LOCALMOD LOCALSAM  
:END.
```

### Source PPF (1VMVMC23 \$PPF)

```
:  
:MYCOMP.  
:  
:MDA.  
LOCALMOD 3C4  
LOCALSAM 3C2  
DELTA 3D6 3D4 3D2  
:
```

### Usable Form PPF (DELETE PPF)

```
:  
:COMPNAME.MYCOMP  
:MYCOMP.  
:  
:MDA.  
DELTA 3D6 3D4 3D2  
:
```

## Updating a Record - Single-Level Override

This override updates a record in the :RECSER section of the product parameter file.

## Override PPF (UPDATE \$PPF)

```
:OVERLST. MYCOMP
:MYCOMP. MYCOMP 1VMVMC23
:RECSER. UPDATE
PARTLST VMFRCCOM DELTA1
:END.
```

## Source PPF (1VMVMC23 \$PPF)

```
:
:MYCOMP.
:
:RECSER.
AXLIST VMFRCAXL DELTA
PARTLST VMFRCCOM DELTA
TEXT VMFRCTXT DELTA
:
```

## Usable Form PPF (UPDATE PPF)

```
:
:COMPNAME.MYCOMP
:MYCOMP.
:
:RECSER.
AXLIST VMFRCAXL DELTA
*PARTLST VMFRCCOM DELTA
PARTLST VMFRCCOM DELTA1
TEXT VMFRCTXT DELTA
:
```

## Updating a Record - Multi-Level Override

The first override in this example updates the :CNTRL tag and points to the second override. The second override is an alias override which only changes the *compname*. Alias override PPFs are sometimes provided by the product to provide easy-to-remember synonyms for *ppfname* and *compname*.

## Control File Override PPF (MULTI \$PPF)

```
:OVERLST. MYCOMP
:MYCOMP. MYCOMP ESA
:CNTRL. DMSLCL
:END.
```

## Alias Override PPF (ESA \$PPF)

```
:OVERLST. MYCOMP
:MYCOMP. MYCOMP 1VMVMC23
:END.
```

## Source PPF (1VMVMC23 \$PPF)

```
:
:MYCOMP.
:
:CNTRL. DMSVM
:
```

## Usable Form PPF (MULTI PPF)

```
:
:COMPNAME.MYCOMP
:MYCOMP.
:
```

## Product Parameter File Syntax

```
*:CNTRL. DMSVM  
:CNTRL. DMSLCL  
:
```



## Chapter 22. Software Inventory Syntax

This chapter describes the syntax of the files and tables in the Software Inventory. For an overview of the Software Inventory, see [Chapter 15, “Introduction to the Software Inventory,”](#) on page 163.

### Structure of the Data in the Software Inventory Tables

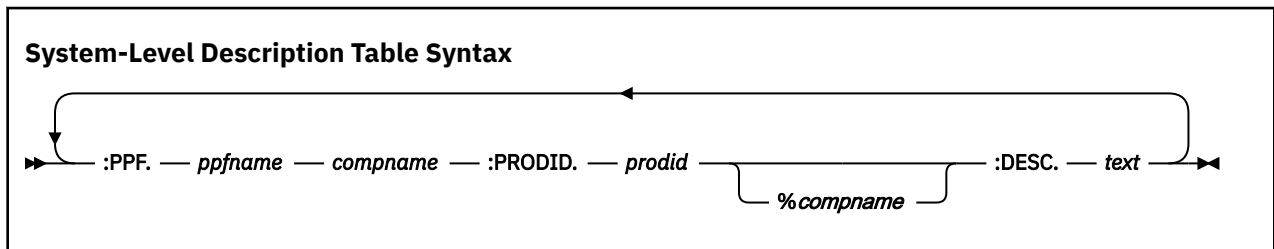
Tables in the Software Inventory are made up of logical records. A logical record consists of all the fields between one key field and the next. A key field identifies the major grouping of information contained in each logical record. The data in each key field is unique throughout a table, duplicate key field values are not allowed. A field consists of everything between one tag and the next, even if it spans multiple lines. The name of the field is the same as the tag in the field.

A tag is an alphanumeric string that starts with a colon (:) and ends with a period (.). The tag identifies the nature of the data following the tag. [Figure 191 on page 659](#) illustrates the data structure.



Figure 191. Software Inventory Data Structure

For example, in the system-level description table shown here, there are three fields, PPF, PRODID, and DESC. PPF is the key field, consisting of the tag :PPF and its data *ppfname compname*. The PRODID field consists of the tag :PRODID and its data *prodid%compname*. The percent sign (%) is a delimiter. The DESC field consists of the tag :DESC and its data *text*. Each logical record in the table begins with the key field PPF.



### Delimiters

In the syntax diagrams in the following sections, the percent sign (%) and the period (.) are entered as delimiters to separate entries. Do not enter blank spaces before or after the delimiter.

### The System-Level Software Inventory

The system-level Software Inventory files are:

- [“The Product Parts \(PRODPART\) File”](#) on page 660
- [“The Saved Segment Data \(SEGDATA\) File”](#) on page 677
- [“The Migration Parts Table \(prodid MIGPvrn\)”](#) on page 682
- [“The System-Level Description Table \(VM SYSDESCCT\)”](#) on page 684
- [“The System-Level Memo Table \(VM SYSMEMO\)”](#) on page 685
- [“The System-Level Requisite Table \(VM SYSREQT\)”](#) on page 686
- [“The System-Level Receive Status Table \(VM SYSRECS\)”](#) on page 688

- [“The System-Level Apply Status Table \(VM SYSAPPS\)” on page 690](#)
- [“The System-Level Build Status Table \(VM SYSBLDS\)” on page 692](#)
- [“The System-Level Service Update Facility Table \(VM SYSSUF\)” on page 693](#)
- [“The System-Level Product Inventory Table \(VM SYSPINV\)” on page 696](#)
- [“The System-Level Restart Table \(VM SYSREST\)” on page 696](#)
- [“The System-Level Local Modification Table \(VM SYSLMOD\)” on page 699](#)
- [“The System-Level Base APAR Table \(VM SYSAPARS\)” on page 701](#)
- [“The File Type Abbreviation Table \(VM SYSABRVT\)” on page 702](#)
- [“The Parts Catalog \(VMSES PARTCAT\)” on page 703](#)

All files in the system-level Software Inventory, with the exception of the parts catalog and the file type abbreviation table, reside on an inventory disk.

## The Software Inventory Defaults

The Software Inventory disk, by default, is the MAINT $\nu$ rm 51D disk, and it is accessed as your D-disk. It may also be a Shared File System directory. The Production Inventory disk is the PMAINT 41D disk

All files in the system-level Software Inventory, except the PRODPART file, the SEGDATA file, the file type abbreviation table, and the parts catalog, use the system name as their file name. The default system name is VM.

The descriptions in this chapter assume you are using the defaults for the Software Inventory disk or directory and the system name.

## Changing the Software Inventory Defaults

To change the default Software Inventory disk, you must specify the SIDISK option using the VMSES/E commands on which it is supported. To change the file mode of the Software Inventory disk, you must first add the new file mode to the :RETAIN tag in the product parameter file so it is not released by VMFSETUP. You can then access it at the new file mode and use the SIMODE option.

VMSES/E commands that support the SIDISK option automatically access the Software Inventory disk. All other VMSES/E commands assume it is already accessed.

The default system name is VM. To change the system name, specify the SYSTEM option using the VMSES/E commands on which it is supported. You might use different system names when you are supporting multiple systems.

## The Product Parts (PRODPART) File

The product parts (PRODPART) file, *prodid* PRODPART, contains information used for product installation and is supplied by the product owner or product packager on the product installation media. It resides on the Software Inventory minidisk or directory. The information in the PRODPART file is used to update entries in the system-level Software Inventory each time a product is loaded onto your system.

The PRODPART file has the following major sections:

- A header section to identify the product being installed
- A loadable unit section to define installable subsets of the product and identify requisite relationships
- A parts section to define the tailorable parts contained in the product
- A product parameters section to define product unique installation parameters
- A saved segment definitions section to define default information for building saved segments

The following shows the syntax of the complete PRODPART file.



```
:RECID.1VMVMC23
:PRODESC.MYCOMP of z/VM
:PONUM.5664123
:PROCTYPE.VMSES
:SERVLEV.910-9101
```

## Loadable Unit Section

The loadable unit section of the PRODPART file defines installable subsets of the product being installed and identifies their requisite relationships.

The following terms are used in the description of the entries in the loadable unit section.

- **Product** designates a single product or *piece* of a product that is installed and serviced independently.

For example, CMS is a component of z/VM. It may be installed independently and is serviced independently of the rest of the z/VM product. PVM is another example of a product. It is a separately installed product that is serviced independently of any other product.

**Note:** The term "product" refers to both products and components of products, unless there is a specific need to differentiate between the two.

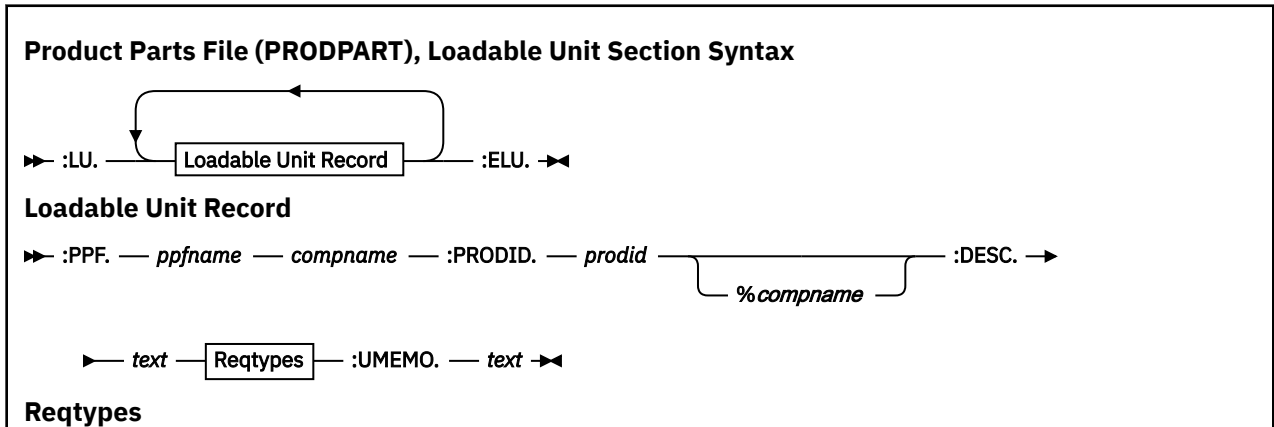
- **Loadable unit** designates a portion of a product that may be installed independently of the rest of the product, but is serviced as part of the product.

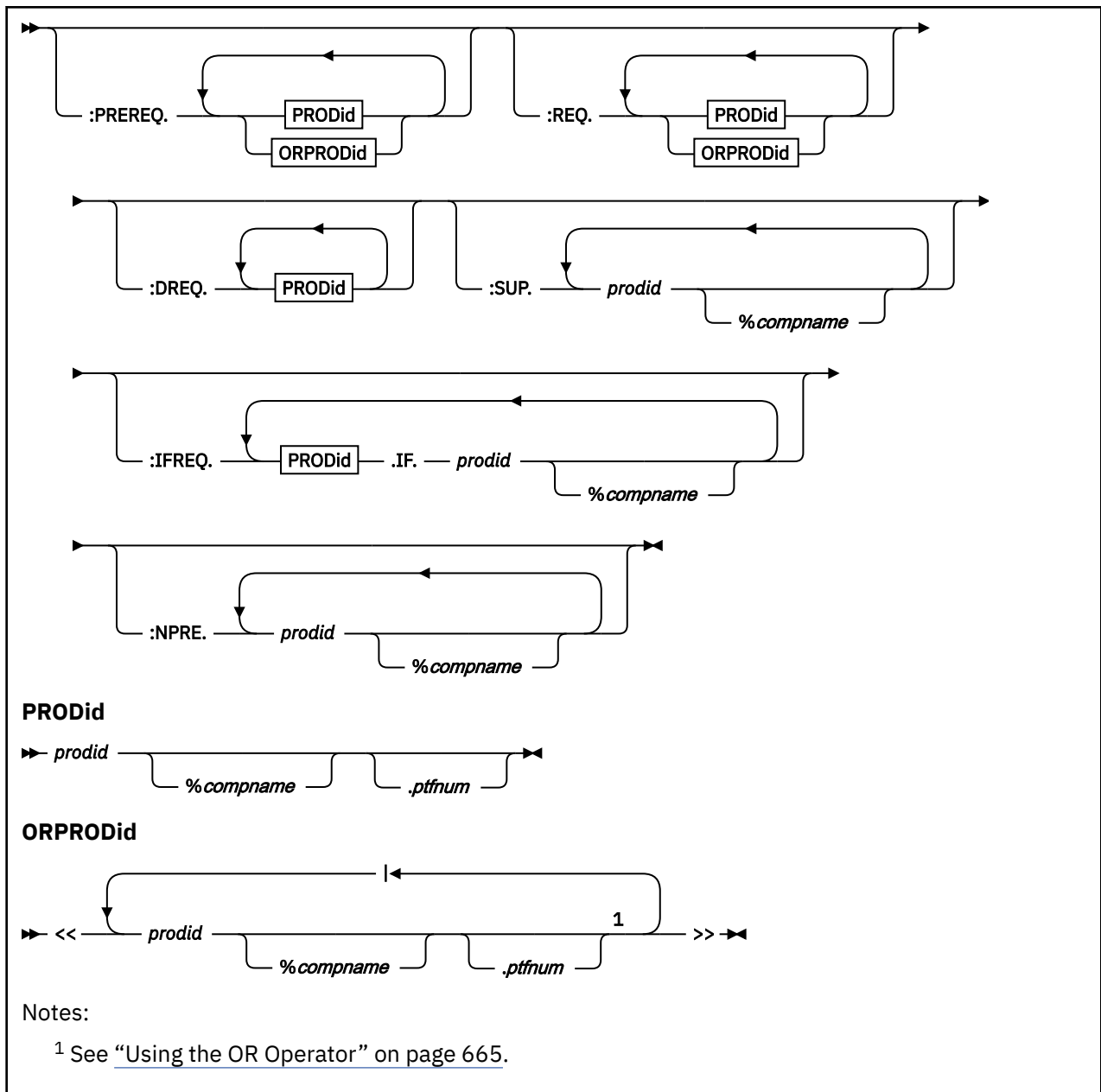
For example, the NetView® DASD Conservation Option (DCO) is a loadable unit of the NetView product. This means you load only a portion of the full NetView product from tape to the system when you install the NetView DCO. However, when you apply service to the installed version of NetView (NetView DCO), there is only one service stream; and the application of the service is to the **entire** product not just the loadable unit.

A loadable unit is defined in the system-level Software Inventory tables as the product's *prodid* and a *compname* (component name) that describes the loadable unit name separated by a percent sign. For example, the *prodid* for NetView is 5664204. The loadable unit for the full product is 5664204%NETVIEW. The loadable unit for NetView DCO is 5664204%DCO.

## Syntax

Product Parts File (PRODPART), Loadable Unit Section Syntax shows the syntax of the loadable unit section of the PRODPART file.





The tags in the loadable unit section are defined as follows:

**:LU.**

indicates the start of the section describing the loadable units that are contained on the installation media for the product.

**Loadable Unit Record**

The loadable unit record describes the loadable unit.

**Note:** Every product must have at least one loadable unit defined. If the product consists of only one loadable unit, the :PPF tag contains the *prodid* and *compname* from the :RECID and :BCOMPNAME tags in the PPF. The :DESC tag contains the same description as the :PRODDDESC tag in the header section.

There can be multiple loadable unit records, and they can contain the following tags and data:

**:PPF.**

identifies the product parameter file for a particular loadable unit.

***ppfname***

is the 1-8 character file name of the product parameter file associated with the loadable unit.

### ***compname***

is the 1-16 character alphanumeric identifier of the component contained within the product parameter file associated with a loadable unit.

### **:PRODID.**

specifies the identifier that is used for PTF validation and product requisites.

### ***prodid***

is the 7-8 character alphanumeric identifier assigned to the product by IBM. *prodid* is the file name of the PRODPART file that is shipped with the product.

### **%*compname***

is a 1-16 character component identifier preceded by a percent sign (%). For example, the identifier for CMS is %CMS.

### **:DESC.**

indicates a description for the loadable unit.

### ***text***

is the description of the loadable unit.

## ***Reqtypes***

The loadable unit section also contains tags that identify the requisites for the loadable unit.

The tags defining loadable unit requisites are required only if they are applicable to the loadable unit. For example, if the loadable unit does not contain any prerequisites, the :PREREQ tag would not be identified in this section.

As you can see after "[Reqtypes](#)" in [Product Parts File \(PRODPART\), Loadable Unit Section Syntax](#), the following types of requisites are specified in the loadable units section:

### **:PREREQ.**

identifies other products or loadable units that must be installed before this loadable unit can be installed correctly.

### **:REQ.**

identifies other products or loadable units that must be installed before this loadable unit can operate correctly.

### **:DREQ.**

identifies other products or loadable units that must be installed before this loadable unit can be installed correctly. Unlike requisites defined on the :PREREQ tag, these requisites are no longer satisfied when the requisite product or loadable unit is superseded. This occurs when a product requires a specific level of another product and newer levels of the product will not meet the requirements.

### **:SUP.**

identifies other products or loadable units replaced by this product, such as a new version or release of a product. This also implies the superseded product never needs to be installed once this loadable unit is installed.

### **:IFREQ.**

identifies conditional requisites. Specifically, this identifies particular products or loadable units that must be installed before this product or loadable unit can operate **if** the specified product or loadable unit is installed.

### **:NPRE.**

identifies other products or loadable units that cannot exist on a system at the same time as this product.

The variables specified with these tags are:

### ***prodid***

is the 7-8 character alphanumeric identifier assigned to the product by IBM. *prodid* is the file name of the PRODPART file that is shipped with the product.

**%compname**

is the component name identifier preceded by a percent sign (%), for example %CMS. *compname* is a 1-16 character alphanumeric identifier.

**ptfnum**

is a specific PTF number (for example UV12345). A PTF number indicates the requisite is a specific service level of the product, and the PTF must be installed.

**Using the OR Operator**

If a product's requisites can be satisfied by any number of products, the "OR" operator can be used in the requisite definition. The format of the "OR" statement is shown in [Figure 192 on page 665](#).

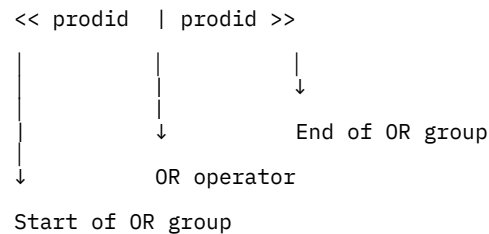


Figure 192. Requisite OR Format Syntax

**Note:** There must be a blank space after << and before >>. [Figure 193 on page 665](#) shows two prerequisites for a loadable unit: product PRODABC and any one of products PRODX, PRODY, or PRODZ.

```
:PREREQ. PRODABC << PRODX | PRODY | PRODZ >>
```

Figure 193. :PREREQ Tag Using OR Format

**:UMEMO.**

identifies the start of user memo text for the loadable unit.

**text**

is the user memo text for the loadable unit.

**:ELU.**

identifies the end of the section containing the loadable units.

**Example**

The following shows an example of the loadable unit section.

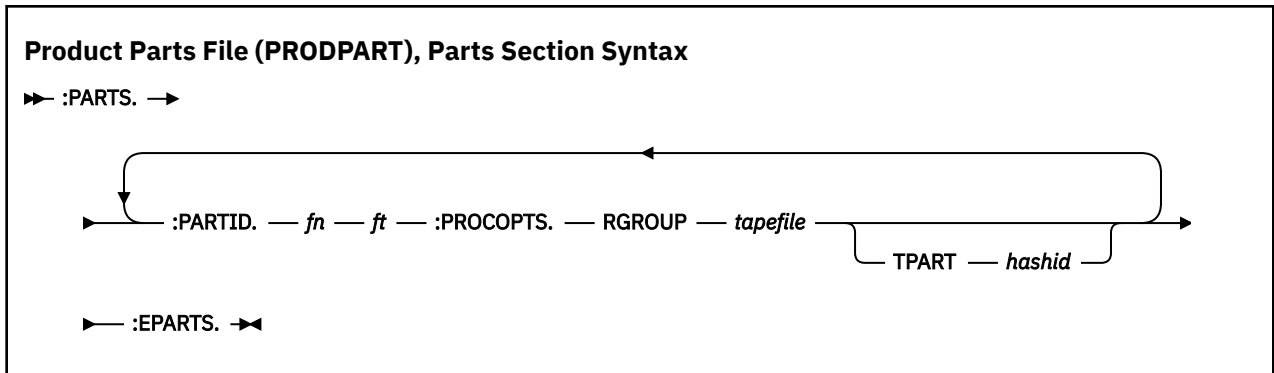
```
:LU.
:PPF.1VMVMC23 MYCOMP
:DESC.MYCOMP for z/VM
:UMEMO.
Special instructions go here
:PPF.1VMVMC23 MYCOMPSRC
:PROPID.1VMVMC23%MYCOMP
:DESC.Optional Source Code for MYCOMP
:PREREQ.1VMVMP11
:SUP.1VMVMC12
:ELU.
```

**Parts Section**

The parts section of the PRODPART file defines the tailorable parts contained in the product.

**Syntax**

[Product Parts File \(PRODPART\), Parts Section Syntax](#) shows the syntax of the parts section.



The tags in the parts section are defined as follows:

**:PARTS.**

identifies the start of a parts definition block.

**:PARTID.**

identifies the tailorable part.

***fn***

is the 1-8 character file name of the part.

***ft***

is the 1-8 character file type of the part.

**:PROCOPTS.**

identifies the processing options that are used when the part is installed.

**RGROUP**

indicates in which tape file the parts were shipped. The parts are identified on the :PARTID tag.

***tapefile***

is the alphanumeric identifier assigned in the :RECINS. section of the PPF to the tape file containing the parts identified on the :PARTID tag. This data is required to allow previous levels of the product to be deleted when you migrate to a new level.

**TPART**

assigns the unique identifier for the tailorable file.

***hashid***

is a hash code, a unique identifier assigned to a tailored part of the product being installed, that represents the file in its unmodified form. The *hashid* is used during product migration to determine if a tailored file has been modified by the user.

**:EPARTS.**

identifies the end of a parts definition block.

**Example**

The following shows an example of the parts section.

```
:PARTS.
:PARTID.HCPRDEVS SAMPEXEC
:PROCOPTS.RGROUP SAMPLES TPART 12A67DD8900001
:PARTID.DMKSYS 3375
:PROCOPTS.RGROUP SAMPLES TPART 12AAA238900001
:EPARTS.
```

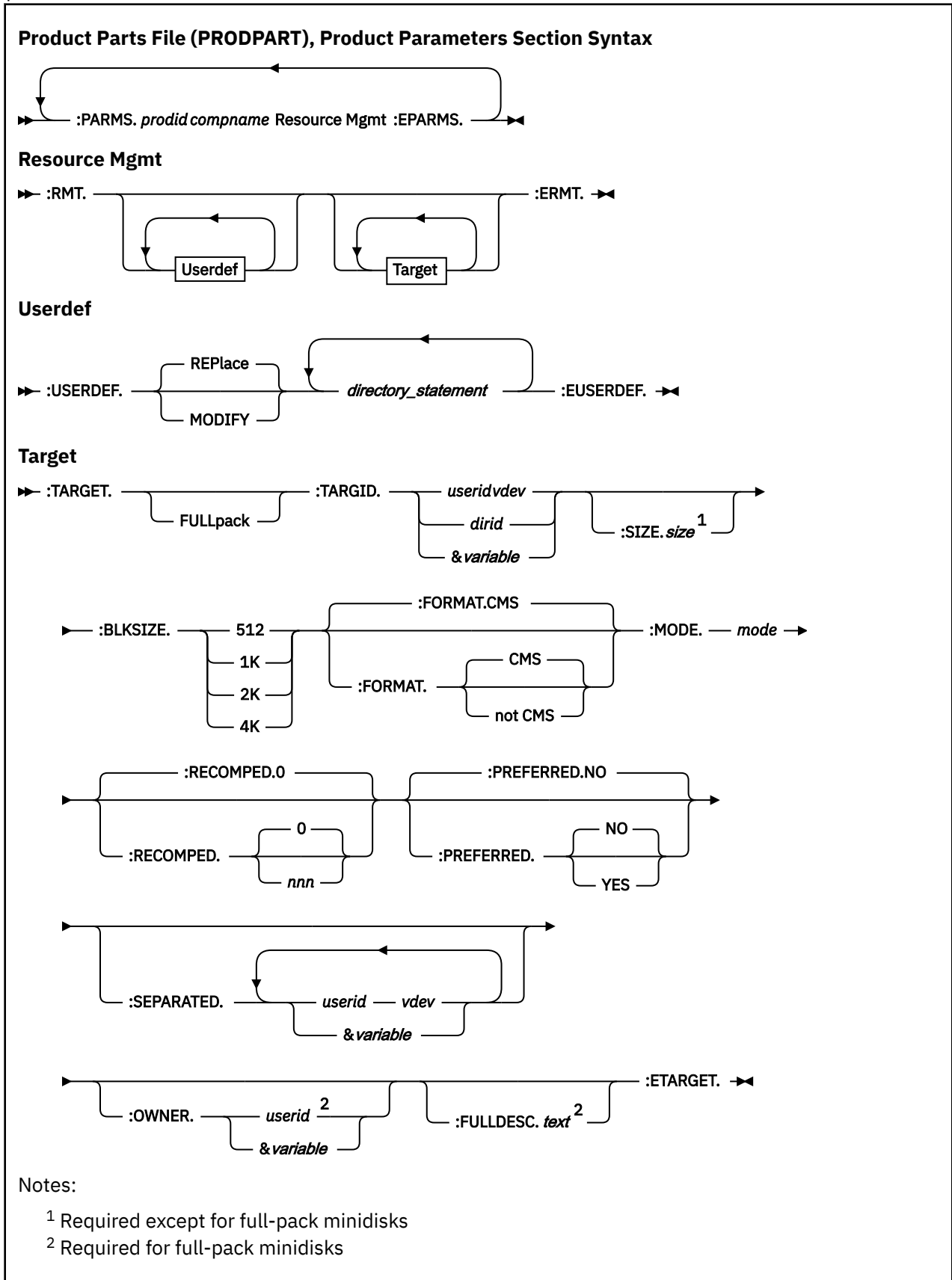
**Product Parameters Section**

The product parameters section of the PRODPART file defines product-unique installation parameters. It contains a resource management subsection, which is further subdivided into user definition subsections and target subsections.



## Syntax

Product Parts File (PRODPART), Product Parameters Section Syntax shows the syntax of the product parameters section.



The tags in the product parameters section are defined as follows:

### **:PARMS.**

identifies the start of the product parameters section.

The product parameters section in a PRODPART file is optional; its existence is dependent on the product's installation requirements.

#### ***prodid***

is the 7-8 character alphanumeric identifier assigned to the product by IBM. *prodid* is the file name of the PRODPART file that is shipped with the product.

#### ***compname***

is the name of the component, for example CMS. *compname* is a 1-16 character alphanumeric identifier.

**Note:** Each :PARMS section applies to one or more loadable units. If the *prodid* and *compname* are not specified on the :PARMS tag, the section applies to all loadable units. If they are specified, they must match the *prodid* and *compname* from a :PRODID tag in the loadable units section.

### **:RMT.**

identifies the start of the resource management subsection.

### **:USERDEF.**

identifies the start of the user definition subsection. This section defines a user ID to be added to or modified in the CP directory before your product can be installed or used.

#### **REPLACE**

replaces the entry if the user ID is already defined in the CP directory. REPLACE is the default.

#### **MODIFY**

adds the specified lines to the existing entry.

#### ***directory\_statement***

defines the user ID using directory statements, excluding MDISK statements. These statements sometimes appear exactly as they would in the directory. In other cases, there can be variable substitution. This can be a one value substitution or a two value substitution. A two value variable substitution is, for example,

```
LINK &BLD3Z 190 RR
```

The &BLD3Z is the user ID and the address.

### **:EUSERDEF.**

identifies the end of the user definition subsection.

### **:TARGET.**

identifies the start of a target subsection. This section defines the minidisk or SFS directory requirements for the product.

#### **FULLPACK**

indicates a full-pack minidisk is required. A blank indicates either a smaller minidisk or an SFS directory.

### **:TARGID.**

identifies a target minidisk or SFS directory.

#### ***userid***

is a user ID.

#### ***vdev***

is a minidisk address.

#### ***dirid***

is a fully-qualified SFS directory name.

#### ***&variable***

is a variable defined for a minidisk or directory in the :DCL section of the product parameter file. Variable names begin with an ampersand (&).

**Note:** When defining a target minidisk for a subconfiguration entry of a multiconfiguration virtual machine definition, the subconfiguration ID must be hardcoded as the user ID in the *userid vdev* pair. A variable cannot be used in the target minidisk definition.

**:SIZE.**

defines the size (in blocks) of the minidisk or directory. This tag is not required for a full-pack minidisk.

***size***

is the number of blocks used by the minidisk or directory based on the :BLKSIZE defined.

**:BLKSIZE.**

defines the size of the blocks for the minidisk or directory.

**512, 1K, 2K, or 4K**

is the block size. Shared File System directories always use 4K blocks.

**:FORMAT.**

is the format of the minidisk.

**CMS**

indicates the minidisk is in CMS format. The default is CMS.

**not CMS**

indicates the minidisk is not in CMS format.

**:MODE.**

indicates the read/write mode of the minidisk or directory.

Although this tag is required, its value is ignored for directories.

***mode***

is the read/write mode, for example, RR, RW, MR, and so forth.

**:RECOMPED.**

indicates the size of the non-CMS formatted area on the minidisk.

**0**

is the default.

***nnn***

is the number of blocks that are not to be formatted.

**:PREFERRED.**

indicates whether the minidisk is preferred (placed as near to the center of the pack as possible).

**NO**

indicates the minidisk is not preferred. NO is the default.

**YES**

indicates the minidisk should be placed as near to the center of the pack as possible.

**:SEPARATED.**

lists minidisks that must be placed on a different pack from this minidisk.

***userid***

is the user ID of the owner of the minidisk.

***vdev***

is the address of the minidisk.

***&variable***

is a variable defined for a minidisk in the :DCL section of the product parameter file. Variable names begin with an ampersand (&).

**:OWNER.**

indicates the owner of a full-pack minidisk. This tag is required for full-pack minidisks.

***userid***

is the user ID of the minidisk owner.

### **&variable**

is a variable defined for a minidisk or directory in the :DCL section of the product parameter file. Variable names begin with an ampersand (&).

### **:FULLDESC.**

indicates the reason for using a full-pack minidisk. This tag is required for full-pack minidisks.

### **text**

is the reason for using a full-pack minidisk.

### **:ETARGET.**

identifies the end of the target subsection.

### **:EPARMS.**

identifies the end of the product parameters section.

## **Example**

the following shows an example of the product parameters section.

```
:PARMS.1VMVMC23 MYCOMP
:RMT.
:USERDEF.
USER &CMS1 NOLOG 24M 32M G
  AUTOLOG AUTOLOG1 OP1 MAINT
  ACCOUNT ACT4 CMSTST
  MACH XC
  IPL 190
  CONSOLE 009 3215
  SPOOL 00C 2540 READER A
  SPOOL 00D 2540 PUNCH A
  SPOOL 09E 1403 A
  LINK &BLD3Z 190 RR
  LINK &PRODZ 19E RR
  LINK &BLD5Z 19D RR
:EUSERDEF.
:TARGET.
  :TARGID.&CMSB0
  :SIZE.300
  :BLKSIZE.4K
  :FORMAT.CMS
  :MODE.MR
  :PREFERRED.YES
:ETARGET.
:TARGET.
  :TARGID.&SRVU0
  :SIZE.450
  :BLKSIZE.4K
  :FORMAT.CMS
  :MODE.MR
  :PREFERRED.YES
:ETARGET.
:ERMT.
:EPARMS.
```

## **Saved Segment Definitions Section**

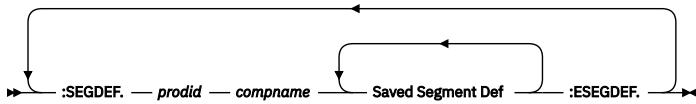
The saved segment definitions section of the PRODPART file defines default information for building saved segments. It contains one or more saved segment definition blocks, each of which contains one or more object subsections, each of which defines a saved segment.

**Note:** The saved segment definitions section does not define saved systems.

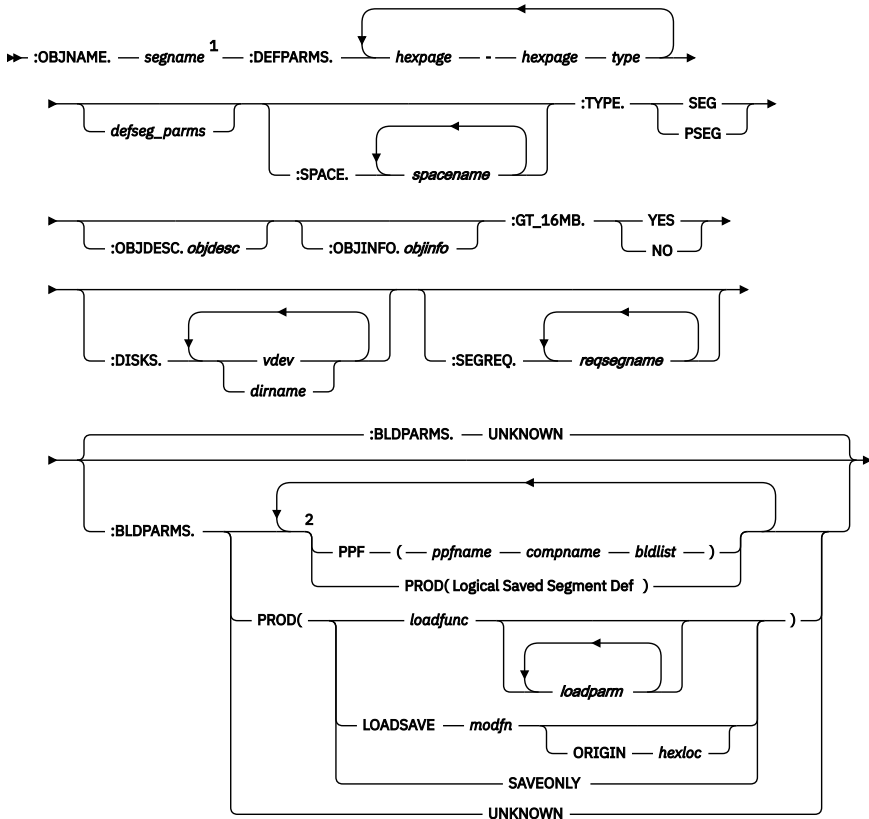
## **Syntax**

[Product Parts File \(PRODPART\), Saved Segment Definitions](#) shows the syntax of the saved segment definitions section.

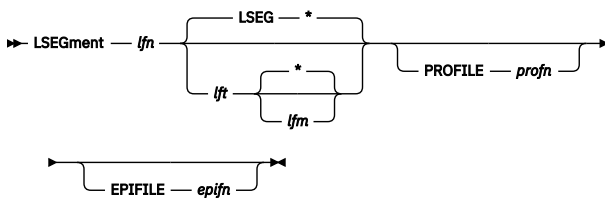
**Product Parts File (PRODPART), Saved Segment Definitions**



**Saved Segment Def**



**Logical Saved Segment Def**



**Notes:**

- <sup>1</sup> The remaining tags in the saved segment definition can be specified in any order.
- <sup>2</sup> This loop is valid only for a physical saved segment that contains logical saved segments. 'PSEG' must be specified on the :TYPE tag.

The tags and data in the saved segment definitions section are defined as follows:

**:SEGDEF.**

indicates the start of a saved segment definition block.

**prodid**

is the 7-8 character alphanumeric identifier assigned to the product by IBM.

**compname**

is the 1-16 character component name.

**Note:** The values of *prodid* and *compname* match the values on the :PRODID tag in the loadable unit section of the PRODPART file.

### **:OBJNAME.**

indicates the start of a saved segment definition.

#### ***segname***

is the name of the saved segment.

### **:DEFPARMS.**

indicates the storage range and other information that the VMFBLD EXEC specifies on the DEFSEG command to define the saved segment to CP.

#### ***hexpage***

is the hexadecimal address of a page in storage.

#### ***type***

is a page descriptor code that indicates the type of virtual machine access that is permitted to the page range.

#### ***defseg\_parms***

are optional DEFSEG command operands. The following operands are permitted: LOADNSHR, RSTD, and SECURE.

**Note:** Do not specify the SPACE operand on this tag. Segment spaces are identified on the :SPACE tag.

For information about the syntax and usage of the DEFSEG command, see [z/VM: CP Commands and Utilities Reference](#).

### **:SPACE.**

lists the segment spaces in which the saved segment is a member.

**Note:** The VMFBLD EXEC uses the first name listed on this tag as the primary segment space when issuing the DEFSEG command for the member.

#### ***spacename***

is the name of a segment space.

### **:TYPE.**

indicates whether the saved segment contains CMS logical saved segments.

#### **SEG**

indicates the saved segment does not contain logical saved segments. One set of build parameters is allowed on the :BLDPARMS tag.

#### **PSEG**

indicates the saved segment is a physical saved segment that contains logical saved segments. Multiple sets of build parameters are allowed on the :BLDPARMS tag, each set defining one or more logical saved segments.

### **:OBJDESC.**

provides a description of the saved segment.

#### ***objdesc***

is free-format text that describes the saved segment.

### **:OBJINFO.**

provides special installation information.

#### ***objinfo***

is free-format text that describes any special requirements for installing the saved segment. This field is informational; its content does not affect the generation of the saved segment.

### **:GT\_16MB.**

indicates whether the saved segment can be loaded above the 16MB line.

#### **YES**

the saved segment can be loaded above 16MB.

**Note:** This does not mean that the range specified on the :DEFPARMS tag actually defines the saved segment above 16MB, only that it can.

**NO**

the saved segment cannot be loaded above 16MB. The range specified on the :DEFPARMS tag must be less than 16MB.

**:DISKS.**

lists the minidisks and SFS directories to be accessed by the VMFBLD EXEC when building the saved segment.

**Note:** If 'PPF' is specified on the :BLDPARMS tag, the minidisks and directories specified on this tag are accessed before the minidisks and directories defined in the product parameter file. If 'UNKNOWN' is specified on the :BLDPARMS tag, the minidisks and directories specified on this tag are not accessed.

***vdev***

is a minidisk virtual device number.

***dirname***

is a fully qualified SFS directory name.

**:SEGREQ.**

lists the saved segments that must be built before this saved segment is built.

**Note:** All requisite saved segments must be defined in the same system saved segment build list and SEGDATA file as this saved segment.

***reqsegname***

is the name of a requisite saved segment.

**:BLDPARMS.**

identifies the parameters that the VMFBLD EXEC uses to build the saved segment.

If 'PSEG' is specified on the :TYPE tag, the VMFBLD EXEC creates a physical segment definition file (*segname* PSEG) that contains the logical segment records, then issues the SEGGEN command to build the physical and logical saved segments. Multiple sets of build parameters (for VMSES/E and non-VMSES/E products) can be specified on this tag to identify the logical saved segments to include in the physical saved segment.

**UNKNOWN**

indicates a build function for this saved segment cannot be issued by the VMFBLD EXEC. This forces VMFBLD to issue only the DEFSEG command to define the saved segment to CP. After VMFBLD has completed, the user must issue the function that actually loads and saves the saved segment. This is the default.

**PPF**

indicates the build parameters are defined in the specified product parameter file and build list. Depending on the build list option specified (ACCESS, LINK, or NOACCESS), VMFSETUP might be called to link and access the disks specified in the product parameter file for the product when the segment is built.

***ppfname***

is the name of the product parameter file.

***compname***

is the name of the component section in the product parameter file.

***bldlist***

is the name of the product saved segment build list.

**PROD**

indicates the build parameters are not specified in a product parameter file. The VMFBLD EXEC uses the parameters specified on this tag to build the saved segment (after issuing the DEFSEG command to define the saved segment to CP).

### **LSEgment**

identifies a logical segment definition file, which defines a CMS logical saved segment to be included in the physical saved segment.

**Note:** 'PSEG' must be specified on the :TYPE tag.

### ***lfn***

is the file name of the logical segment definition file.

### ***lft***

is the file type of the logical segment definition file. If the file type is not specified here, the SEGGEN command uses a default of LSEG.

**Note:** Do not use PROFILE or EPIFILE as the file type of the logical segment definition file.

### ***lfn***

is the file mode of the logical segment definition file. If the file mode is not specified here, the SEGGEN command uses a default of \*.

### **PROFILE**

indicates a routine is to be run before information is loaded into the logical saved segment. For information about using a profile when building a logical saved segment, see [z/VM: CP Planning and Administration](#).

### ***profn***

is the file name of the profile routine.

### **EPIFILE**

indicates a routine is to be run after information is loaded into the logical saved segment. For information about using an epifile when building a logical saved segment, see [z/VM: CP Planning and Administration](#).

### ***epifn***

is the file name of the epifile routine.

### ***loadfunc***

is the name of the routine the VMFBLD EXEC calls to load and save the saved segment.

Return codes from *loadfunc* are processed as follows:

**0**

Segment built without any errors

**1-4**

Segment built, one or more warnings issued

**all others**

Segment build failed

### ***loadparm***

is a parameter to be passed to the *loadfunc* routine.

The following built-in variables are also available to indicate data to be passed to the *loadfunc* routine. When the saved segment is built, the VMFBLD EXEC resolves the variables as indicated:

### **&RANGE**

Gets the hexadecimal page ranges, page descriptor codes, and optional DEFSEG operands from the :DEFPARMS tag

### **&SPACE**

Gets the primary segment space name from the :SPACE tag

### **&ORIGIN**

Gets the starting load address from the :DEFPARMS tag

### **&SEGNAME**

Gets the name from the :OBJNAME tag



**LOADSAVE**

indicates the VMFBLD EXEC calls the built-in LOADSAVE function, which issues the LOADMOD command to load a specified relocatable module. LOADSAVE then issues the SAVESEG command to save the module as a saved segment.

See the usage notes on the LOADMOD command in *z/VM: CMS Commands and Utilities Reference*.

**modfn**

is the file name of the relocatable module. The file type must be MODULE.

**SAVEONLY**

indicates the VMFBLD EXEC calls the built-in SAVEONLY function, which issues only the SAVESEG command.

**ORIGIN**

specifies a load address for the module.

**hexloc**

is the hexadecimal storage address where the module is to be loaded.

**Note:** It is recommended that the ORIGIN keyword be specified and that the built-in variable &ORIGIN be used to indicate the storage location. When the saved segment is built, the VMFBLD EXEC resolves the &ORIGIN variable to get the load address from the :DEFPARMS tag. If the ORIGIN keyword is not specified, CMS selects any available storage location. If an actual address is specified, it must be within the range defined on the :DEFPARMS tag.

**:ESEGDEF.**

identifies the end of a saved segment definition block.

**Example**

The following shows an example of the saved segment definitions section.

```
*****
:SEGDEF.1VMVMC23 MYCOMP
*****
:OBJNAME. HELPINST
:DEFPARMS. C00-CFF SR
:TYPE. PSEG
:OBJDESC. CMSINST AND HELP LSEGS
:GT_16MB. NO
:BLDPARMS. PPF(ESA 1VMVMC23 DMSSBINS) PPF(ESA 1VMVMC23 DMSSBHLP):OBJNAME. CMSPIPES
:DEFPARMS. 1900-19FF SR
:TYPE. PSEG
:OBJDESC. CMS PIPES SEGMENT
:GT_16MB. YES
:BLDPARMS. PPF(ESA 1VMVMC23 DMSSBPIP)
:
:ESEGDEF.
```

**Example PRODPART File**

The following shows an example of a complete PRODPART file.

```
*=====
* Header Section
*=====
:RECID.1VMVMC23
:PONUM.5654030
:PRODDDESC.MYCOMP component for z/VM
:SERVLEV.000-0000
:PROCTYPE.VMSES*=====
* Loadable Units Section - equivalent of different components
*=====
:LU.
:PPF.1VMVMC23 MYCOMP
:PRODDID.1VMVMC23%MYCOMP
:DESC.MYCOMP component for z/VM
:PREREQ.1VMVMP11
:DREQ.1VMVME10
```

## Software Inventory Syntax

```
:SUP.
:ELU.*=====
* Parts Section - lists all files for this Product/Component
*=====
:PARTS.
*-----
:PARTID.SERVER PROFILE
:PROCOPTS.TPART 00000000000000

:PARTID.DMSNGP SAMPLE
:PROCOPTS.TPART 00000000000000

:PARTID.VMSYS POOLDEF
:PROCOPTS.TPART 00000000000000:PARTID.VMSYSR POOLDEF
:PROCOPTS.TPART 00000000000000

:PARTID.VMSYSU POOLDEF
:PROCOPTS.TPART 00000000000000

:PARTID.VMSERVS DMSPARMS
:PROCOPTS.TPART 00000000000000

:
*-----
:EPARTS.*=====
```

```
* Segdef Section - default segment definitions for CMS
*=====
*****
:SEGDEF.1VMVMC23 MYCOMP
*****
:OBJNAME. HELPINST
:DEFPARMS. C00-CFF SR
:TYPE. PSEG
:OBJDESC. CMSINST AND HELP LSEGS
:GT_16MB. NO
:BLDPARMS. PPF(ESA MYCOMP DMSSBINS) PPF(ESA MYCOMP DMSSBHLP):
:ESEGDEF.*****
```

```
*=====
* Parms Section - resource allocation information section
*=====
*****
:PARMS.1VMVMC23 MYCOMP
*****
:RMT.
:USERDEF.
USER &CMS1 NOLOG 24M 32M G
AUTOLOG AUTOLOG1 OP1 MAINT
ACCOUNT ACT4 CMSTST
MACH XC
IPL 190
CONSOLE 009 3215
SPOOL 00C 2540 READER A
SPOOL 00D 2540 PUNCH A
SPOOL 00E 1403 A
LINK &BLD3Z 190 RR
LINK &PRODZ 19E RR
LINK &BLD5Z 19D RR
:EUSERDEF.*
:USERDEF.
USER &CMSBU NOLOG 24M 24M G
ACCOUNT 3 SYSTEM
MACH ESA
OPTION ACCT
IPL 190 PARM AUTOCR
CONSOLE 009 3215
SPOOL 00C 2540 READER *
SPOOL 00D 2540 PUNCH A
SPOOL 00E 1403 A
LINK &BLD3Z 190 RR
:EUSERDEF.
*
::TARGET.
:TARGID.&CMSB0
:SIZE.300
:BLKSIZE.4K
:FORMAT.CMS
:MODE.MR
```

```

: PREFERRED . YES
: ETARGET .
: TARGET .
: TARGID . &SRVU0
: SIZE . 450
: BLKSIZE . 4K
: FORMAT . CMS
: MODE . MR
: PREFERRED . YES
: ETARGET .
: TARGET .
: TARGID . &SRVU1
: SIZE . 1500
: BLKSIZE . 4K
: FORMAT . CMS
: MODE . R
: PREFERRED . YES
: ETARGET .:

: ERMT .

: EPARMS .

```

## The Saved Segment Data (SEGDATA) File

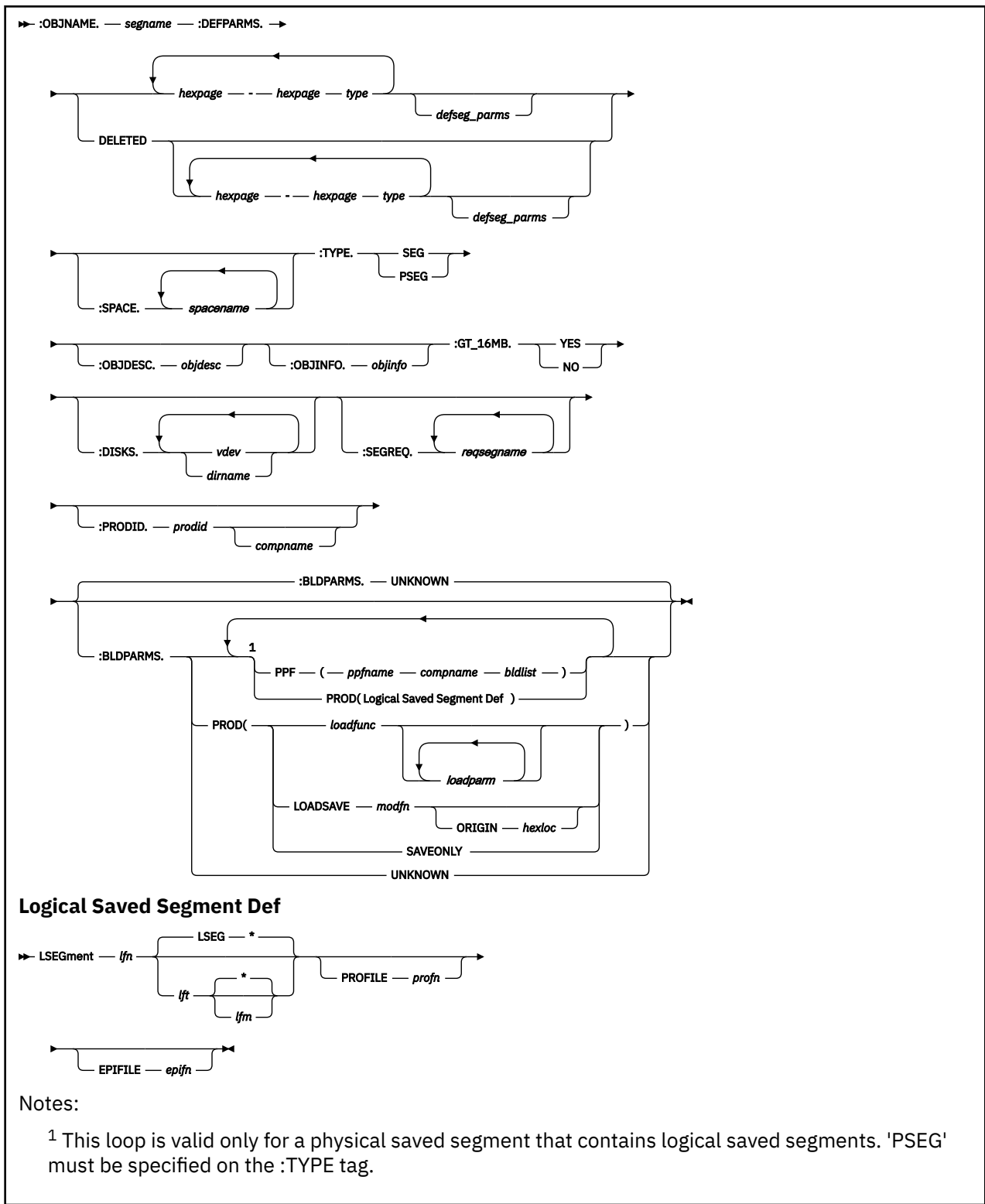
The saved segment data (SEGDATA) file, *bldlist* SEGDATA, contains customized saved segment definitions for building the set of saved segments identified in the system saved segment build list that has the same file name. The SEGDATA file resides on the same disk as the system saved segment build list. It is updated by the user through the VMFSGMAP saved segment mapping tool and is processed by VMFBLD.

**Note:** The SEGDATA file does not define saved systems.

### Syntax

SEGDATA File Syntax shows the syntax of the SEGDATA file.





The tags and data in the SEGDATA file are defined as follows:

**:OBJNAME.**

indicates the start of a saved segment definition.

***segname***

is the name of the saved segment.

**:DEFPARMS.**

indicates the storage range and other information that the VMFBLD EXEC specifies on the DEFSEG command to define the saved segment to CP.

***hexpage***

is the hexadecimal address of a page in storage.

***type***

is a page descriptor code that indicates the type of virtual machine access that is permitted to the page range.

***defseg\_parms***

are optional DEFSEG command operands. The following operands are permitted: LOADNSHR, RSTD, and SECURE.

**Note:** The SPACE operand is not specified on this tag. Segment spaces are identified on the :SPACE tag.

**DELETED**

indicates the saved segment is to be deleted. No DEFSEG command is issued.

**Note:** If any DEFSEG data is specified in this field, it is saved in the SEGDATA file following the DELETED keyword.

**:SPACE.**

lists the segment spaces in which the saved segment is a member.

**Note:** The VMFBLD EXEC uses the first name listed on this tag as the primary segment space when issuing the DEFSEG command for the member.

***spacename***

is the name of a segment space.

**:TYPE.**

indicates whether the saved segment contains CMS logical saved segments.

**SEG**

indicates the saved segment does not contain logical saved segments. One set of build parameters is allowed on the :BLDPARMS tag.

**PSEG**

indicates the saved segment is a physical saved segment that contains logical saved segments. Multiple sets of build parameters are allowed on the :BLDPARMS tag, each set defining one or more logical saved segments.

**:OBJDESC.**

provides a description of the saved segment.

***objdesc***

is free-format text that describes the saved segment.

**:OBJINFO.**

provides special installation information.

***objinfo***

is free-format text that describes any special requirements for installing the saved segment. This field is informational; its content does not affect the generation of the saved segment.

**:GT\_16MB.**

indicates whether the saved segment can be loaded above the 16MB line.

**YES**

the saved segment can be loaded above 16MB.

**Note:** This does not mean that the range specified on the :DEFPARMS tag actually defines the saved segment above 16MB, only that it can.

### **NO**

the saved segment cannot be loaded above 16MB. The range specified on the :DEFPARMS tag must be less than 16MB.

### **:DISKS.**

lists the minidisks and SFS directories to be accessed by the VMFBLD EXEC when building the saved segment.

**Note:** If 'PPF' is specified on the :BLDPARMS tag, the minidisks and directories specified on this tag are accessed before the minidisks and directories defined in the product parameter file. If 'UNKNOWN' is specified on the :BLDPARMS tag, the minidisks and directories specified on this tag are not accessed.

### ***vdev***

is a minidisk virtual device number.

### ***dirname***

is a fully qualified SFS directory name.

### **:SEGREQ.**

lists the saved segments that must be built before this saved segment is built.

**Note:** All requisite saved segments must be defined in the same system saved segment build list and SEGDATA file as this saved segment.

### ***reqsegname***

is the name of a requisite saved segment.

### **:PRODID.**

identifies the PRODPART file that contains the default definition for the saved segment.

### ***prodid***

is the 7-8 character alphanumeric identifier assigned to the product.

### ***compname***

is the 1-16 character component name. The component name is necessary only if the PRODPART file contains saved segment definitions for more than one component.

### **:BLDPARMS.**

identifies the parameters the VMFBLD EXEC uses to build the saved segment.

If 'PSEG' is specified on the :TYPE tag, the VMFBLD EXEC creates a physical segment definition file (*segname* PSEG) that contains the logical segment records, then issues the SEGGEN command to build the physical and logical saved segments. Multiple sets of build parameters (for VMSES/E and non-VMSES/E products) can be specified on this tag to identify the logical saved segments to include in the physical saved segment.

### **UNKNOWN**

indicates a build function for this saved segment cannot be issued by the VMFBLD EXEC. This forces VMFBLD to issue only the DEFSEG command to define the saved segment to CP. After VMFBLD has completed, the user must issue the function that actually loads and saves the saved segment. This is the default.

### **PPF**

indicates the build parameters are defined in the specified product parameter file and build list. Depending on the build list option specified (ACCESS, LINK, or NOACCESS), VMFSETUP might be called to link and access the disks specified in the product parameter file for the product when the segment is built.

### ***ppfname***

is the name of the product parameter file.

### ***compname***

is the name of the component section in the product parameter file.

### ***bldlist***

is the name of the product saved segment build list.

**PROD**

indicates the build parameters are not specified in a product parameter file. The VMFBLD EXEC uses the parameters specified on this tag to build the saved segment (after issuing the DEFSEG command to define the saved segment to CP).

**LSEGment**

identifies a logical segment definition file, which defines a CMS logical saved segment to include in the physical saved segment.

**Note:** 'PSEG' must be specified on the :TYPE tag.

***lfn***

is the file name of the logical segment definition file.

***lft***

is the file type of the logical segment definition file. If the file type is not specified here, the SEGEN command uses a default of LSEG.

**Note:** PROFILE and EPIFILE are not valid as the file type of the logical segment definition file.

***lfn***

is the file mode of the logical segment definition file. If the file mode is not specified here, the SEGEN command uses a default of \*.

**PROFILE**

indicates a routine is to be run before information is loaded into the logical saved segment.

***profn***

is the file name of the profile routine.

**EPIFILE**

indicates a routine is to be run after information is loaded into the logical saved segment.

***epifn***

is the file name of the epifile routine.

***loadfunc***

is the name of the routine the VMFBLD EXEC calls to load and save the saved segment.

***loadparm***

is a parameter to be passed to the *loadfunc* routine.

The following built-in variables are also available to indicate data to be passed to the *loadfunc* routine. When the saved segment is built, the VMFBLD EXEC resolves the variables as indicated:

**&RANGE**

Gets the hexadecimal page ranges, page descriptor codes, and optional DEFSEG operands from the :DEFPARMS tag

**&SPACE**

Gets the primary segment space name from the :SPACE tag

**&ORIGIN**

Gets the starting load address from the :DEFPARMS tag

**&SEGNAME**

Gets the name from the :OBJNAME tag

**LOADSAVE**

indicates the VMFBLD EXEC calls the built-in LOADSAVE function, which issues the LOADMOD command to load a specified relocatable module. LOADSAVE then issues the SAVESEG command to save the module as a saved segment.

***modfn***

is the file name of the relocatable module. The file type must be MODULE.

**SAVEONLY**

indicates the VMFBLD EXEC calls the built-in SAVEONLY function, which issues only the SAVESEG command.

**ORIGIN**

specifies a load address for the module.

**hexloc**

is the hexadecimal storage address where the module is to be loaded.

**Note:** If the built-in variable &ORIGIN is specified, the VMFBLD EXEC resolves the variable to get the load address from the :DEFPARMS tag. If an actual load address is specified, it must be within the range defined on the :DEFPARMS tag.

**Example**

The following shows an example of the SEGDATA file.

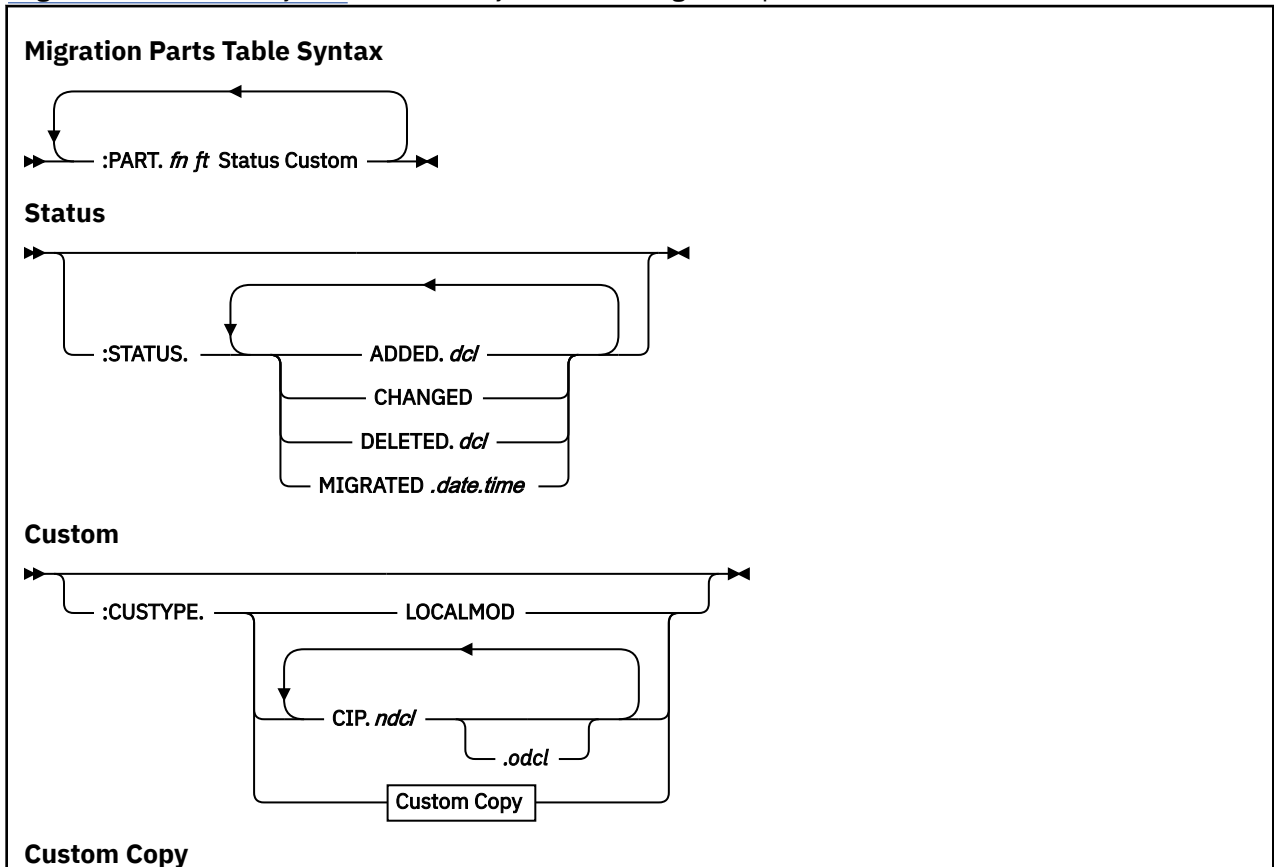
```
:OBJNAME. HELPINST
:DEFPARMS. C00-CFF SR
:TYPE. PSEG
:OBJDESC. CMSINST AND HELP LSEGS
:GT_16MB. NO
:PRODID. 1VMVMC23
:BLDPARMS.PPF(ESA MYCOMP DMSSBINS) PPF(ESA MYCOMP DMSSBHLF)
```

**The Migration Parts Table (*prodid MIGPvrm*)**

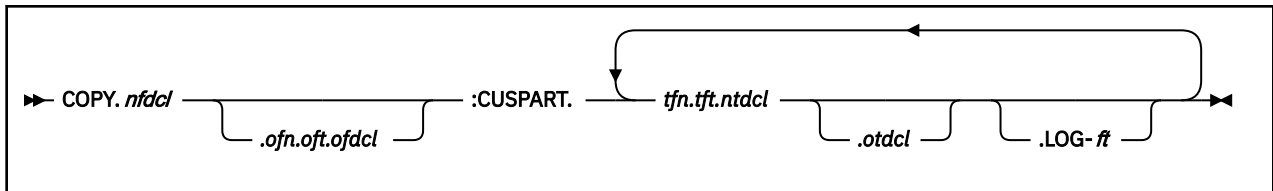
The Migration Parts table contains a list of parts that have been changed or can be customized.

The file name of this table is the PRODID of a component, product or feature. The file type of this table indicates the version, release and modification of a z/VM release from which you can migrate. The Migration Parts table resides on the Software Inventory disk.

Migration Parts Table Syntax shows the syntax of the migration parts table.





**:PART.**

identifies a part

***fn***

is the 1- to 8-character file name of the part.

***ft***

is the 1- to 8-character file type of the part.

**:STATUS.**

is the status of the part

**ADDED**

indicates that the part has been added to a disk

***dcl***

is the variable name of the disk in the component's DCL section of its PPF, (for example, `&BUILD2`), to which the part was added.

**CHANGED**

indicates that the contents of the part has changed. (New part is considered changed.)

**DELETED**

indicates that the part has been deleted from a disk.

***dcl***

is the variable name of the disk in the component's DCL section of its PPF, (for example, `&BUILD2`), to which the part was deleted.

**MIGRATED**

indicates that a customized part has been migrated.

***date***

indicates the month, day and year the part was migrated. The format for date is *mm/dd/yy*.

***time***

indicates the hour, minute and second the part was migrated. The format for time is *hh:mm:ss*.

**:CUSTYPE.**

indicates how the part can be customized

**LOCALMOD**

indicates that this part is customized by local modification

**CIP**

indicates that the part is changed-in-place. This means the part is customized by being modified on the disk where it resides.

***ndcl***

is the variable name of the disk in the component's DCL section of its PPF, (for example, `&BUILD2`), where the part resides in the to release.

***odcl***

is the variable name of the disk in the component's DCL section of its PPF, (for example, `&BUILD2`), where the part resides in the from release.

**COPY**

indicates that the part is customized by being copied and then modified.

***nfdcl***

is the variable name of the disk in the component's DCL section of its PPF, (for example, `&BUILD2`) where the unmodified part resides on the to release.

**ofn**

is the 1- to 8-character file name of the unmodified part on the from release.

**oft**

is the 1- to 8-character file type of the unmodified part on the from release.

**ofdcl**

is the variable name of the disk in the component's DCL section of its PPF, (for example, &BUILD2) where the unmodified part resides on the from release.

**:CUSPART.**

identifies the target characteristics of a COPY part

**tfn**

is the 1- to 8-character file name of the modified part. It can be ? to indicate that file name of the modified part is user-determined.

**tft**

is the 1- to 8-character file type of the modified part. It can be ? to indicate that the file type of the modified part is user-determined.

**ntdcl**

is the variable name of the disk in the component's DCL section of its PPF, (for example, &BUILD2) where the modified part resides on the to release.

**otdcl**

is the variable name of the disk in the component's DCL section of its PPF, (for example, &BUILD2) where the modified part resides on the from release.

**LOG-ft**

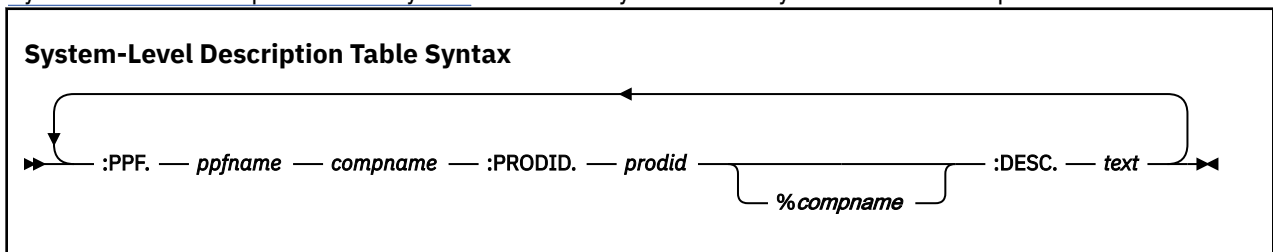
is 1- to 8- character file type to which the part is to be copied. LOG indicates that a product exit log has been updated for this part.

## The System-Level Description Table (VM SYSDESCT)

The system-level description table contains the descriptions of the products that have been received on the system. The system-level description table resides on the Software Inventory disk, and it is updated during receive processing for installation media. Information in the PRODPART files is used to update this table.

### Syntax

System-Level Description Table Syntax shows the syntax of the system-level description table.

**:PPF.**

indicates a product parameter file (PPF) was used to load the product.

:PPF is the key field for the system-level description table.

**ppfname**

is the file name of the PPF that was used to load the product.

**compname**

is the name of the component in the PPF that was used to load the product. *compname* is a 1-16 character alphanumeric identifier.

**:PRODID.**

identifies the product that was loaded.

**prodid**

is the 7-8 character alphanumeric identifier from the :RECID tag (for example, 1VMVMC23).

**%compname**

is the component name preceded by a percent sign (%), for example, %CMS. *compname* is a 1-16 character alphanumeric identifier.

**:DESC.**

identifies the product description.

**text**

is the description of the product.

**Example**

The following shows an example of the system-level description table.

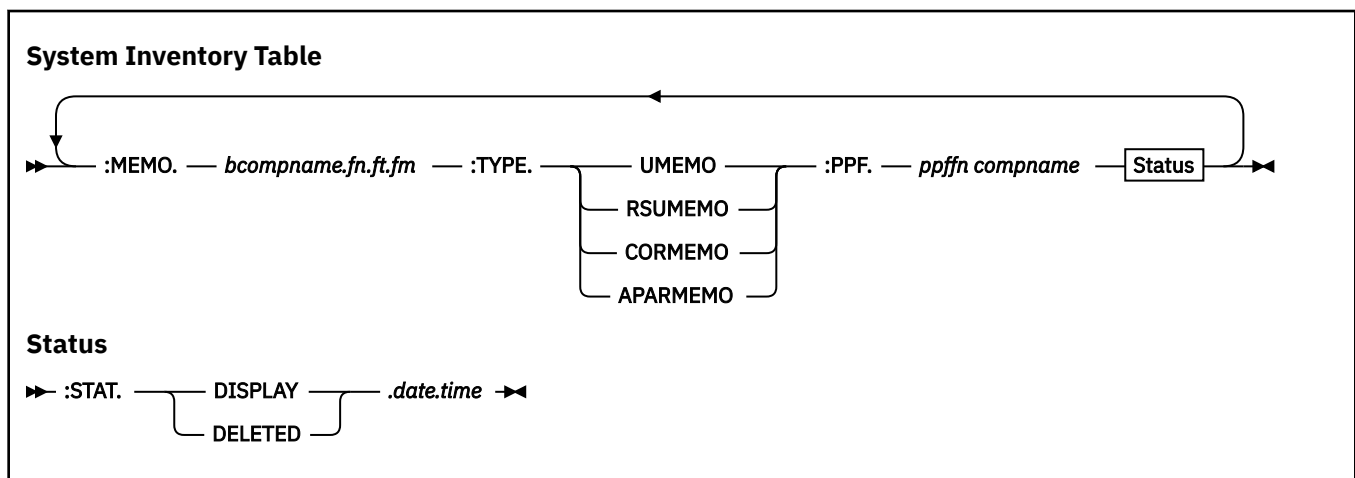
```
:PPF. ESAINS MYCOMP :PRODID. 1VMVMC23 :DESC. MYCOMP Component of z/VM
:PPF. 5684120A WLFS :PRODID. 5684120A :DESC. WORK STATION LAN
```

**The System-Level Memo Table (VM SYMEMO)**

The system-level memo table contains a list of memos collected during automated service processing. The memos included are:

- PTF APAR MEMO
- RSU MEMO
- UMEMO
- COR MEMO

This table gives you the ability to display all the memos in one place (using the VMFUPDAT EXEC).

**Syntax****:MEMO.**

identifies there is a Memo

**bcompname**

is the 7-8 character alphanumeric identifier defined on the bcompname tag in the PPF file

**fn ft fm**

is the fileid of the file containing the memo

**:TYPE.**

is the type of MEMO specified

### UMEMO

indicates UMEMO data is in the PTF \$PTFPART file

### RSUMEMO

indicates a Product Installation Memo or latest Recommended Service Upgrade (RSU) Memo, if the product is RSU supported

### CORMEMO

indicates a CORrective Service Memo

### APARMEMO

indicates an APAR MEMO, which was created from the APAR section of the CORrective Service memo

### :PPF.

identifies the product parameter file. This field is used for UMEMO display only.

#### *ppffn*

is the file name of the PPF file associated with the memo

#### *compname*

is the component name. *compname* is a 1-16 character alphanumeric identifier.

### :STAT.

is the status of the MEMO specified

### DISPLAY

indicates the record will be displayed by VMFUPDAT SYSMEMO

### DELETED

indicates the record will not be displayed by VMFUPDAT SYSMEMO

### date

indicates the month, day, and year the memo was processed. The format for date is *mm/dd/yy*.

### time

identifies the hour, minute, and second the memo was processed. The format for time is *hh:mm:ss*.

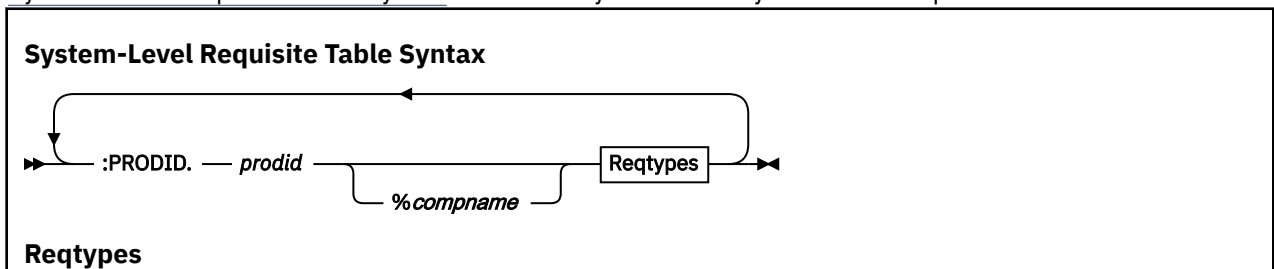
## The System-Level Requisite Table (VM SYSREQT)

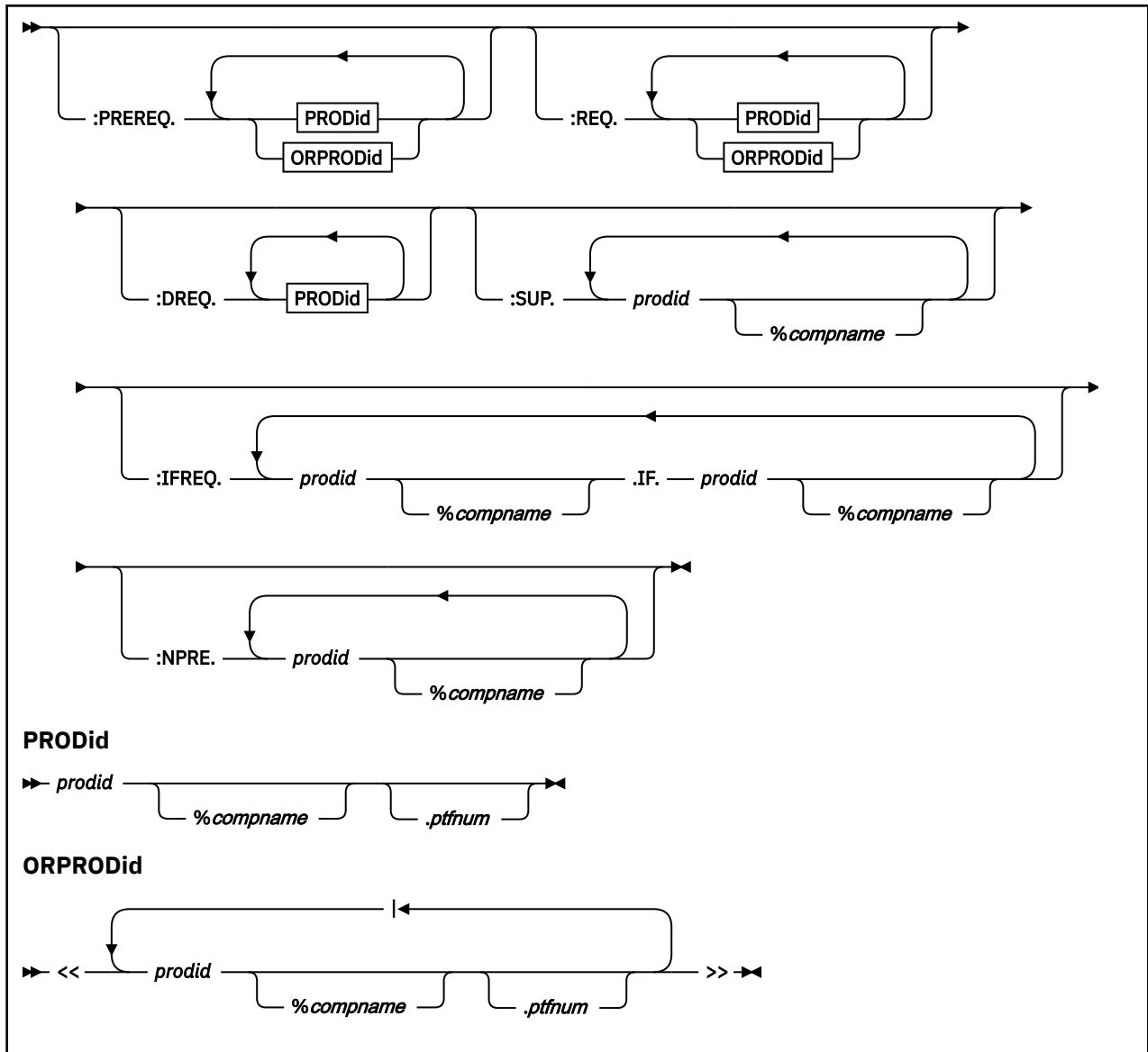
The system-level requisite table contains the relationships between products that will be installed or have been installed on the system.

The system-level requisite table resides on the Software Inventory disk, and it is updated during receive processing for installation media. This table is updated using information in the PRODPART files.

## Syntax

System-Level Requisite Table Syntax shows the syntax of the system-level requisite table.



**:PRODID.**

identifies the product.

:PRODID is the key field for the system-level requisite table.

***prodid***

is the 7-8 character alphanumeric identifier assigned to the product by IBM (for example, 1VMVMC23). *prodid* is the file name of the PRODPART file that is shipped with the product.

***%compname***

is the component name preceded by a percent sign (%), for example %CMS. *compname* is a 1-16 character alphanumeric identifier.

***ptfnum***

is a specific PTF number, for example UV12345.

**Reqtypes**

The system-level requisite table shows the following types of requisites:

**:PREREQ.**

identifies other products or loadable units that must be installed before this loadable unit can be installed correctly.

### **:REQ.**

identifies other products or loadable units that must be installed before this loadable unit can operate correctly.

### **:DREQ.**

identifies other products or loadable units that must be installed before this loadable unit can be installed correctly. Unlike requisites defined on the :PREREQ tag, these requisites are no longer satisfied when the requisite product or loadable unit is superseded. This occurs when a product requires a specific level of another product and newer levels of the product will not meet the requirements.

### **:SUP.**

identifies other products or loadable units replaced by this product, such as a new version or release of a product. This also implies the superseded product never needs to be installed once this loadable unit is installed.

### **:IFREQ.**

identifies conditional requisites. Specifically, this identifies particular loadable units that must be installed with this product or loadable unit **if and only if** the specified product/loadable unit or product/PTF is installed.

### **:NPRES.**

identifies other products or loadable units that cannot exist on a system at the same time as this product.

The variables specified with these tags are:

#### ***prodid***

is the 7-8 character alphanumeric identifier assigned to the product by IBM (for example 1VMVMC23). *prodid* is the file name of the PRODPART file that is shipped with the product.

#### ***%compname***

is the component name preceded by a percent sign (%), for example %CMS. *compname* is a 1-16 character alphanumeric identifier.

#### ***ptfnum***

is a specific PTF number (for example UV12345). Specifying a PTF number indicates the PTF must also be installed.

## Example

Figure 194 on page 688 shows an example of the system-level requisite table.

```
:PRODID.57000ABC%PRODABC :PREREQ.57000B40 << 57000Y40 | 57000W40 >>
      :COREQ. 57000DEF%PRODDEF
      :SUP. 57000X40
      :IFREQ. 57000W40.IF.57000T40
      :NPRES. 57000Z40
```

Figure 194. System-Level Requisite Table Example

## The System-Level Receive Status Table (VM SYSRECS)

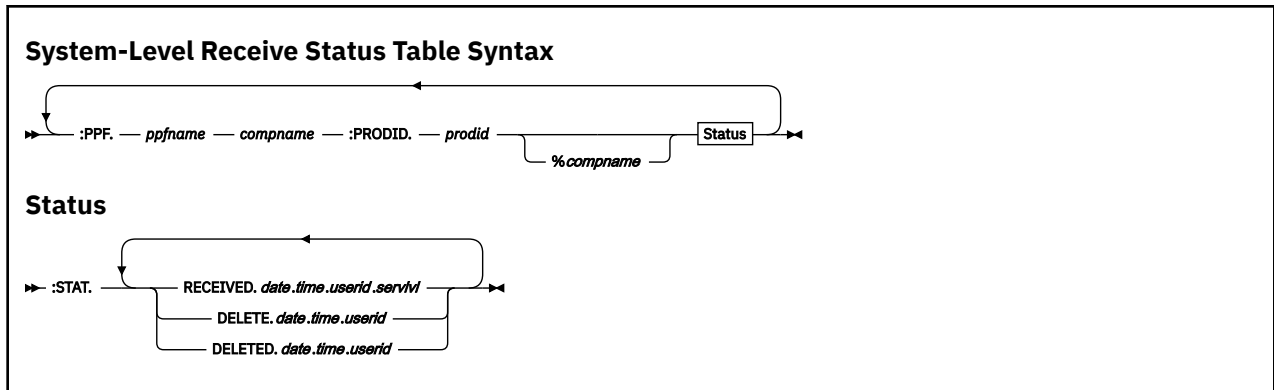
The system-level receive status table contains a list of all products that have been received on the system.

The system-level receive status table resides on the Software Inventory disk, and it is updated during receive processing for installation media.

You can access the information in this table to determine which product parameter file is used for a specific product and to determine which products have been received on the system.

## Syntax

[System-Level Receive Status Table Syntax](#) shows the syntax of the system-level receive status table.

**:PPF.**

indicates a product parameter file (PPF) was used to load the product.

:PPF is the key field for the system-level receive status table.

***ppfname***

is the file name of the PPF that was used to load the product.

***compname***

is the name of the component in the PPF that was used to load the product.

**:PRODID.**

identifies the product.

***prodid***

is the 7-8 character alphanumeric identifier assigned to the product by IBM (for example, 1VMVMC23). *prodid* is the file name of the PRODPART file that is shipped with the product.

***%compname***

is the component name preceded by a percent sign (%), for example %CMS). *compname* is a 1-16 character alphanumeric identifier.

**:STAT.**

indicates the status of the product on the system.

**RECEIVED**

means the product has been received.

**DELETE**

means you have asked to delete this product and the delete is pending.

**DELETED**

means the product has been deleted.

***date***

identifies the month, day, and year the product was processed. The format for date is *mm/dd/yy*.

***time***

identifies the hour, minute, and second the product was processed. The format for time is *hh:mm:ss*.

***userid***

is the userid that was used when the product was processed.

***servlvl***

identifies the service level of the product when it was processed.

**Example**

Figure 195 on page 690 shows an example of the system-level receive status table.

```
:PPF.VMESA MYCOMP :PRODID.1VMVMC23%MYCOMP :STAT.RECEIVED.06/01/89.10:10:10.MAINT.9101-911
:PPF.VMESA MYCOMP2 :PRODID.1VMVMD23%MYCOMP2 :STAT.DELETED.01/02/90.11:11:11.JONES
:PPF.5684120A WLFS :PRODID.5684120A%WLFS :STAT.RECEIVED.01/02/90.11:11:11.JONES
:PPF.5684142B LANRES :PRODID.5684142B%LANRES :STAT.RECEIVED.01/02/90.11:11:11.JONES
```

Figure 195. System-Level Receive Status Table Example

## The System-Level Apply Status Table (VM SYSAPPS)

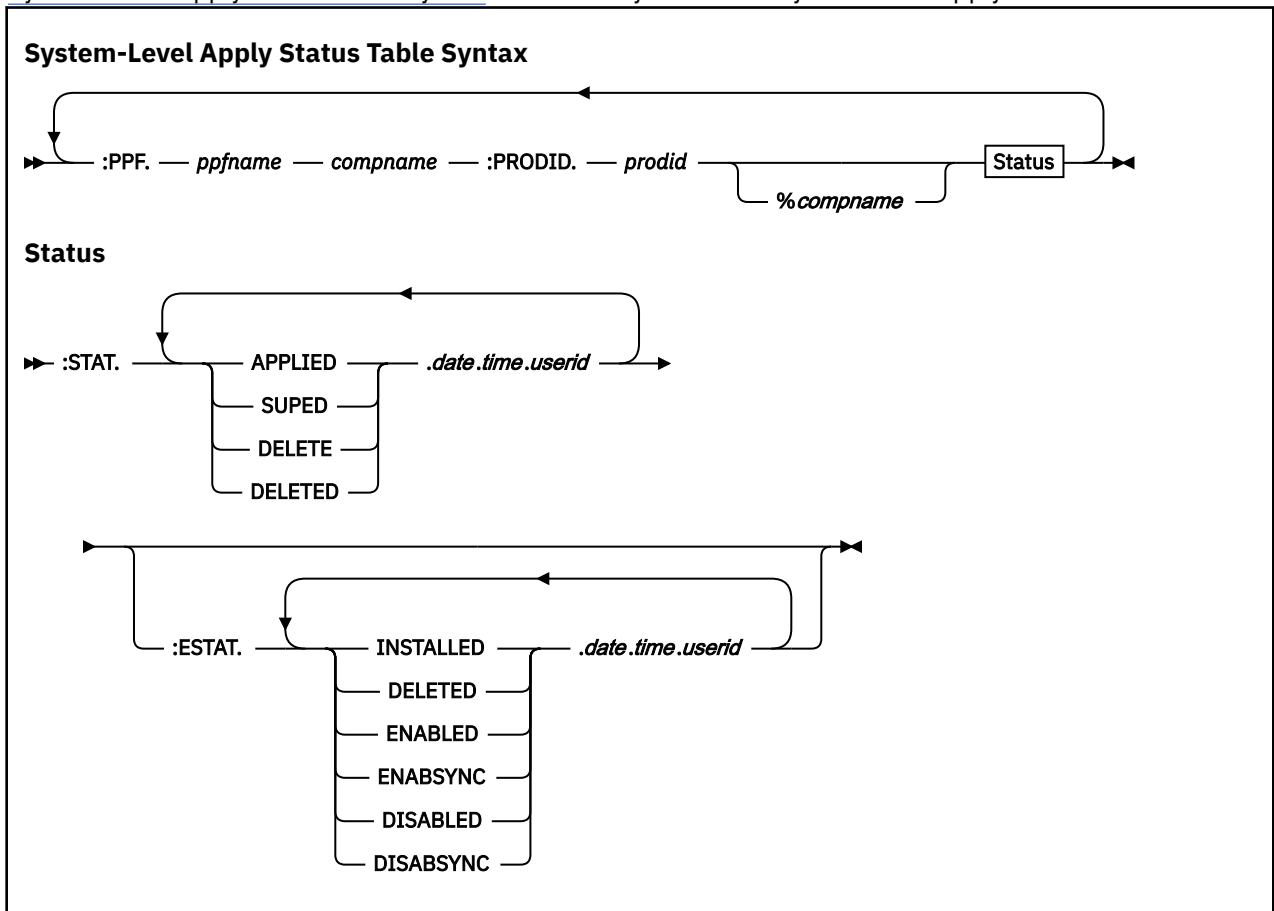
The system-level apply status table contains a list of all products that have been applied on the system. A product is considered to be applied when it has passed requisite checking.

The system-level apply status table resides on the Software Inventory disk, and it is updated during receive processing for installation media.

You can access the information in this table to determine which product parameter file is used for a specific product, which products have been applied on the system, and which products defined to the system are enabled or disabled.

### Syntax

System-Level Apply Status Table Syntax shows the syntax of the system-level apply status table.



**:PPF.**

indicates the product parameter file (PPF) that was used to load the product.

:PPF is the key field for the system-level apply status table.

***ppfname***

is the file name of the PPF that was used to load the product.



***compname***

is the name of the component in the PPF that was used to load the product. *compname* is a 1-16 character alphanumeric identifier.

**:PRODIG.**

identifies the product.

***prodid***

is the 7-8 character alphanumeric identifier assigned to the product by IBM (for example, 1VMVMC23). *prodid* is the file name of the PRODPART file that is shipped with the product.

***%compname***

is the component name preceded by a percent sign (%), for example %CMS. *compname* is a 1-16 character alphanumeric identifier.

**:STAT.**

is the status of the product on the system.

**APPLIED**

means requisite checking as been successfully completed.

**SUPED**

means the product has been superseded by another product.

**DELETE**

means you have asked to delete the product and the delete is pending.

**DELETED**

means the product has been deleted.

***date***

identifies the month, day, and year the product was processed. The format for date is *mm/dd/yy*.

***time***

identifies the hour, minute, and second the product was processed. The format for time is *hh:mm:ss*.

***userid***

identifies the user ID that was used when the product was processed.

**:ESTAT.**

is the enablement status of the product on the system.

**INSTALLED**

means the product is installed on the system.

**DELETED**

means the product has been deleted.

**ENABLED**

means the product is enabled to the system.

**ENABSYNC**

means the product is enabled to the system (same as ENABLED).

**DISABLED**

means the product is disabled to the system.

**DISABSYNC**

means the product is disabled to the system (same as DISABLED).

**Example**

Figure 196 on page 692 shows an example of the system-level apply status table.

```

:PPF.VMESA MYCOMP :PRODID.1VMVMC23%MYCOMP :STAT.APPLIED.03/31/21.11:11:11.MAINT
:PPF.VMESA MYCOMP2 :PRODID.1VMVMD23%MYCOMP2 :STAT.APPLIED.03/31/21.10:11:11.MAINT
:PPF.5684120A WLFS :PRODID.5684120A%WLFS :STAT.APPLIED.03/31/21.12:10:10.TMG
:PPF.5684142B LANRES :PRODID.5684142B%LANRES :STAT.APPLIED.03/31/22.12:10:10.TMG
:ESTAT.ENABLED.03/31/22.12:10:10.TMG INSTALLED.03/31/22.12:10:10.TMG
:PPF.5684100E PVM :PRODID.5684100E%PVM :STAT.APPLIED.10/31/22.08:09:00.P684100E
:ESTAT.DISABSYNC.11/01/22.10:15:00.P684100E DISABLED.10/31/22.08:09:00.P684100E
INSTALLED.10/31/22.08:09:00.P684100E
    
```

Figure 196. System-Level Apply Status Table Example

## The System-Level Build Status Table (VM SYSBLDS)

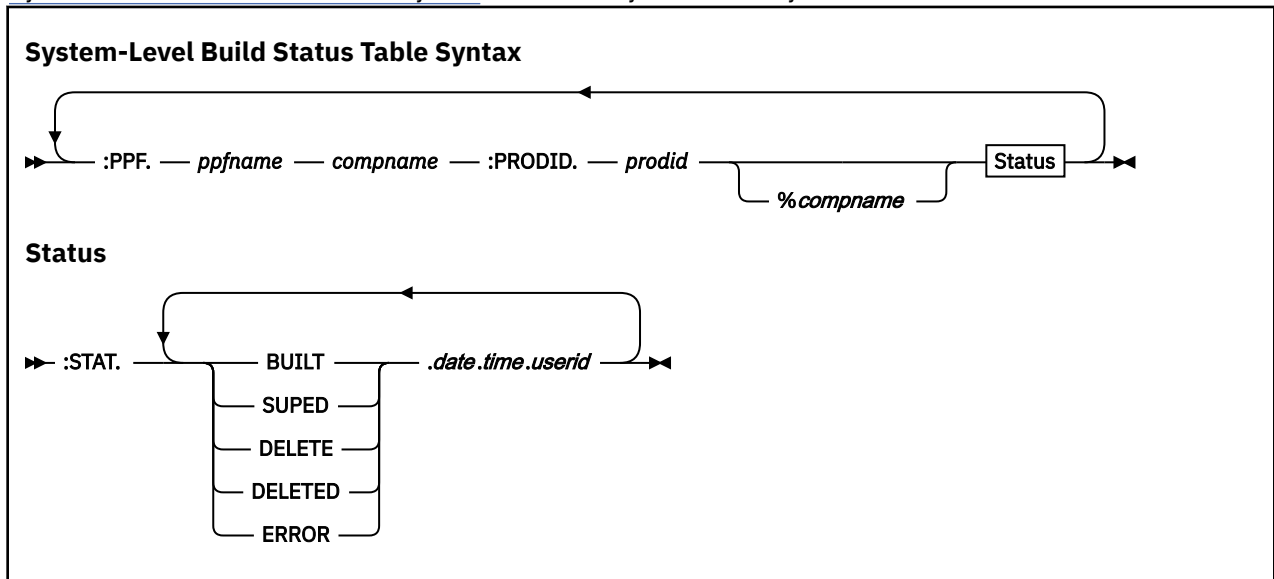
The system-level build status table contains a list of all products that have been built on the system.

The system-level build status table resides on the Software Inventory disk, and it is updated during build processing for installation media.

You can access the information in this table to determine which products have been completely installed on the system.

### Syntax

System-Level Build Status Table Syntax shows the syntax of the system-level build status table.



**:PPF.**

identifies the product parameter file (PPF) that was used to build the product.

:PPF is the key field for the system-level build status table.

**ppfname**

is the file name of the PPF that was used to build the product.

**compname**

is the name of the component in the PPF that was used to build the product. *compname* is a 1-16 character alphanumeric identifier.

**:PRODID.**

identifies the product.

**prodid**

is the 7-8 character alphanumeric identifier assigned to the product by IBM (for example, 1VMVMC23). *prodid* is the file name of the PRODPART file that is shipped with the product.

**%compname**

is the component name preceded by a percent sign (%), for example %CMS. *compname* is a 1-16 character alphanumeric identifier.

**:STAT.**

is the status of the product on the system.

**BUILT**

means the product has been built.

**SUPED**

means the product has been superseded by another product.

**DELETE**

means you have asked to delete the product and the delete is pending.

**DELETED**

means the product has been deleted.

**ERROR**

means an error occurred while the product was being processed.

**date**

identifies the month, day, and year the product was processed. The format for date is *mm/dd/yy*.

**time**

identifies the hour, minute, and second the product was processed. The format for time is *hh:mm:ss*.

**userid**

identifies the user ID that was used when the product was processed.

**Example**

The following [Figure 197 on page 693](#) shows an example of the system-level build status table.

```
:PPF.ESAINS MYCOMP :PRODID.1VMVMC23%MYCOMP :STAT.BUILT.03/31/22.10:10:10.MAINT
:PPF.ESATEST MYCOMP2 :PRODID.1VMVMD23%MYCOMP2 :STAT.SUPED.04/30/22.11:11:11.MAINT
:PPF.ESATEST MYCOMP :PRODID.1VMVMC23%MYCOMP :STAT.DELETED.04/30/22.12:12:12.MAINT
:PPF.5684120A WLFS :PRODID.5684120A%WLFS :STAT.BUILT.04/02/22.01:15:15.JONES
:PPF.5684142B LANRES :PRODID.5684142B%LANRES :STAT.BUILT.04/02/22.02:15:15.JONES
```

Figure 197. System-Level Build Status Table Example

**The System-Level Service Update Facility Table (VM SYSSUF)**

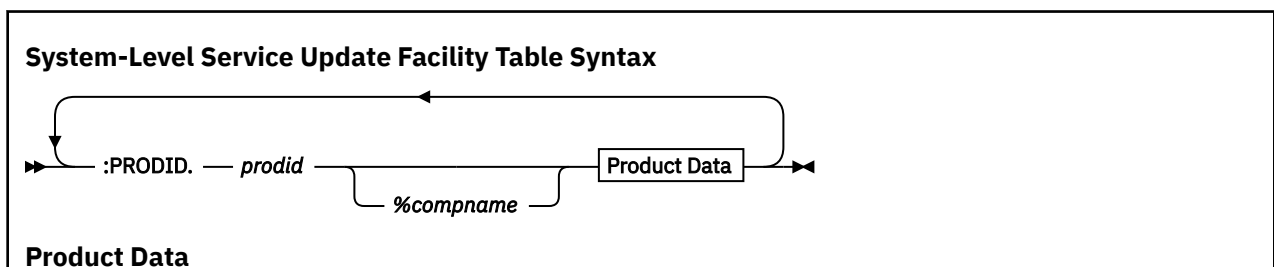
The system-level service update facility table contains a list of all products that are installed on the system and related data needed by the automated service commands. The LOCALMOD, PUT2PROD, and SERVICE EXECs use this table.

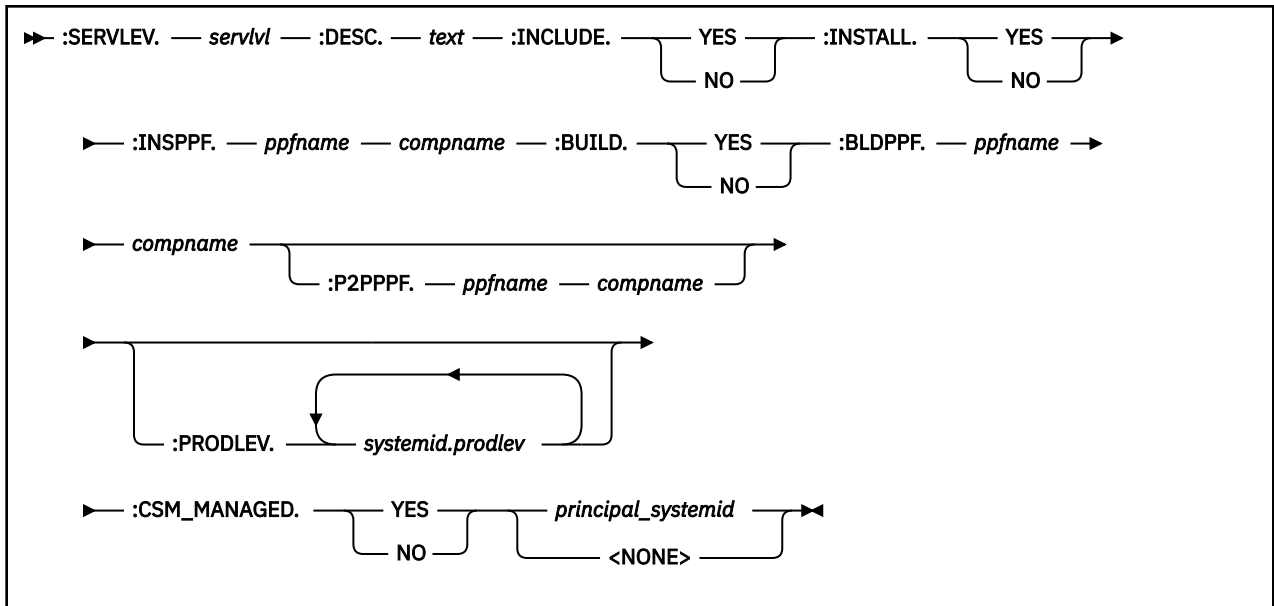
The system-level service update facility table resides on the software inventory disk and is updated by VMFSUFTB EXEC.

You can access the information in the system-level service update facility table to determine the parameters used by the automated service commands. You can then tailor the table using the VMFSIM MODIFY or VMFUPDAT SYSSUF command.

**Syntax**

[System-Level Service Update Facility Table Syntax](#) shows the syntax of the system-level Service Update Facility table.





**:PRODID.**

identifies the product.

**prodid**

is the 7- through 8-character alphanumeric identifier assigned to the product by IBM (for example, 1VMVMC23).

**%compname**

is the component name preceded by a percent sign (%) (for example, %CMS). *compname* is a 1- through 16-character alphanumeric identifier.

**:SERVLEV.**

identifies the RSU, ESO, or SDO test service level. On a system managed by z/VM CSM, the value is the z/VM CSM service level.

**servlvl**

is the RSU, ESO, SDO, or z/VM CSM test service level of the product.

**:DESC.**

identifies the product description.

**text**

is the description of the product.

**:INCLUDE.**

obsolete.

**YES**

default shipped value.

**NO**

obsolete.

**:INSTALL.**

indicates whether or not to install service (RSU or COR) for this product.

**YES**

installs the RSU or COR (this is the initial value set by VMFSUFTB).

**NO**

does not install the RSU or COR.

**:INSPPF.**

identifies the PPF used to install the RSU.

**ppfname**

is the file name of the PPF used to install the RSU.

**compname**

is the name of the component in the PPF used to install the RSU.

**:BUILD.**

indicates whether the automated service commands should run VMFBLD for the product, *prodid*, after installing the RSU.

**YES**

runs VMFBLD (this is the initial value set by VMFSUFTB).

**NO**

does not run VMFBLD.

**:BLDPPF.**

identifies the PPF used to run VMFBLD.

**ppfname**

is the file name of the PPF used to run VMFBLD.

**compname**

is the name of the component in the PPF used to run VMFBLD.

**:P2PPPF.**

identifies the PPF used by the PUT2PROD EXEC.

**ppfname**

is the file name of the PPF used by the PUT2PROD EXEC.

**compname**

is the name of the component in the PPF used by the PUT2PROD EXEC.

**:PRODLEV.**

identifies the RSU, ESO, or SDO production service level. On a system managed by z/VM CSM, the value is the z/VM CSM production level.

**systemid.prodlev**

*systemid* is the ID of the system and *prodlev* is the RSU, ESO, SDO, or z/VM CSM production service level of the product.

The VMFUPDAT SYSSUF panel displays only one RSU, ESO, SDO, or z/VM CSM production service level: the level associated with the ID of the system on which the VMFUPDAT command was executed.

**:CSM\_MANAGED.**

indicates the management state for the product, with respect to z/VM Centralized Service Management (z/VM CSM).

**YES**

service for the product is actively controlled using z/VM CSM support, and is managed by the named z/VM CSM principal system (*principal\_systemid*).

**NO**

service for the product is not being controlled using z/VM CSM. If a principal system is cited, management of the subject system by z/VM CSM has been suspended. If the principal system is cited as <NONE>, the subject system is not being managed by z/VM CSM.

## Example

Figure 198 on page 695 shows an example of the system-level Service Update Facility table.

```
:PROVID.1VMVMC23%MYCOMP :SERVLEV.RSU-0901 :DESC.MYCOMP z/VM
:INCLUDE.YES :INSTALL.YES :INSPPF.SERV2P MYCOMP :BUILD.YES :BLDPPF.SERV2P MYCOMP
:P2PPPF.SERV2P MYCOMP2P :PRODLEV.SYSID1.RSU-0901
```

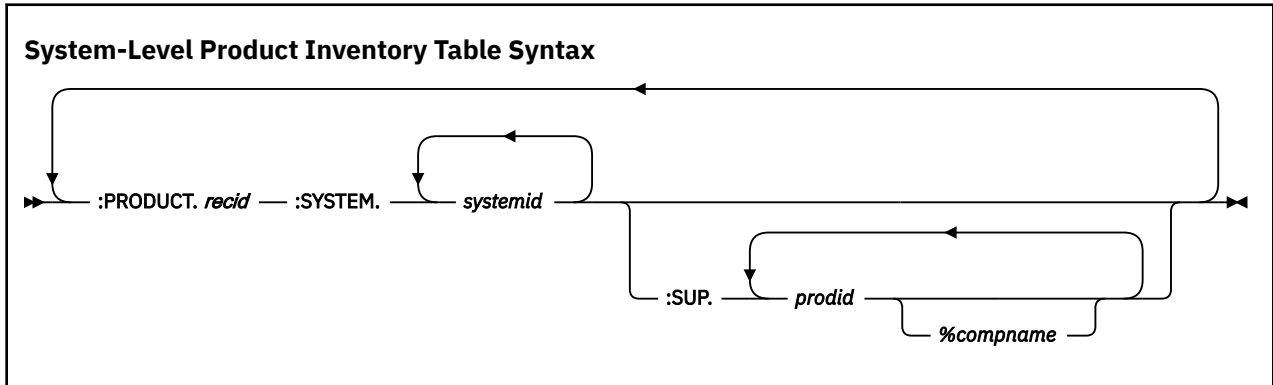
Figure 198. System-Level Service Update Facility Table Example

## The System-Level Product Inventory Table (VM SYSPINV)

The system-level Product Inventory table (VM SYSPINV) specifies which products are installed on which systems or members. The system-level Product Inventory table resides on the production inventory disk (by default, PMAINT 41D).

### Syntax

[System-Level Product Inventory Table Syntax](#) shows the syntax of the system-level Product Inventory table.



#### **:PRODUCT.**

identifies the product.

#### **recid**

is the 1–8 character alphanumeric product identifier.

#### **:SYSTEM.**

identifies the system.

#### **systemid**

is the ID of a system.

#### **:SUP**

identifies the products replaced (superseded) by this product. That is, the value of this tag is a list of the PRODIDs which the product in the `:PRODUCT` tag supersedes.

#### **prodid**

is the 7-8 character alphanumeric identifier assigned to the product by IBM (for example, 1VMVMC23).

#### **%compname**

is the component name preceded by a percent sign (%) (for example, %CMS). *compname* is a 1-16 character alphanumeric identifier.

### Example

Figure 199 on page 696 shows an example of the system-level Product Inventory table.

```
:PRODUCT.1VMVMC23 :SYSTEM.MEMSYS1 MEMSYS2 :SUP.1VMVMC22
:PRODUCT.5748890A :SYSTEM.MEMSYS2
```

Figure 199. System-Level Product Inventory Table Example

## The System-Level Restart Table (VM SYSREST)

The system-level restart table contains records used to restart VMFSUFIN EXEC.

The system-level restart table resides on the software inventory disk and is updated by the VMFSUFIN EXEC.

You can access the information in this table to determine which invocations of VMFSUFIN are incomplete and are awaiting restart.

The VM SYSREST table contains these restart records:

- Checkpoint Restart Record

A checkpoint restart record is created when a VMFSUFIN service install step fails. This record, which is identified by a status of CHKPT in the :STAT tag, allows VMFSUFIN to restart at the failing step. The record is used when VMFSUFIN is called with the RESTART option for one of the *prodid*s listed in the record. The service install process is restarted using only data from the checkpoint restart record.

- Key Restart Record

A key restart record is created when VMFSUFIN is called with the KEY option and a record with a matching key cannot be found in the SYSREST table. This record, which is identified by a status of KEY in the :STAT tag, allows VMFSUFIN to install service from data passed by two separate calls. The record is used when VMFSUFIN is called with a matching key. The service install process is restarted using data from both the VMFSUFIN command and the key restart record.

- Initial Restart Record

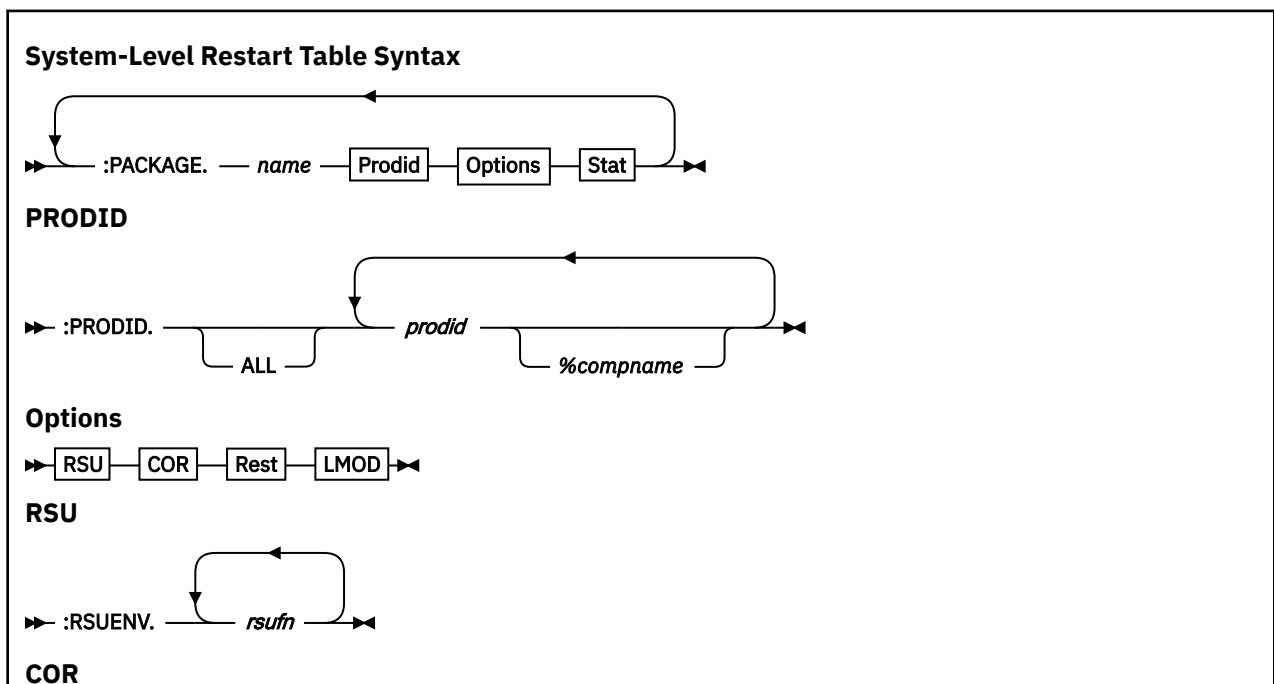
An initial restart record is created when VMFSUFIN begins processing and remains until it is converted into a checkpoint restart record. This record, which is identified by a status of INIT in the :STAT tag, allows VMFSUFIN to restart from the beginning. The record is used when VMFSUFIN is called with the RESTART option for one of the *prodid*s listed in the record and a checkpoint restart record cannot be found. The service install process is restarted using only data from the initial restart record.

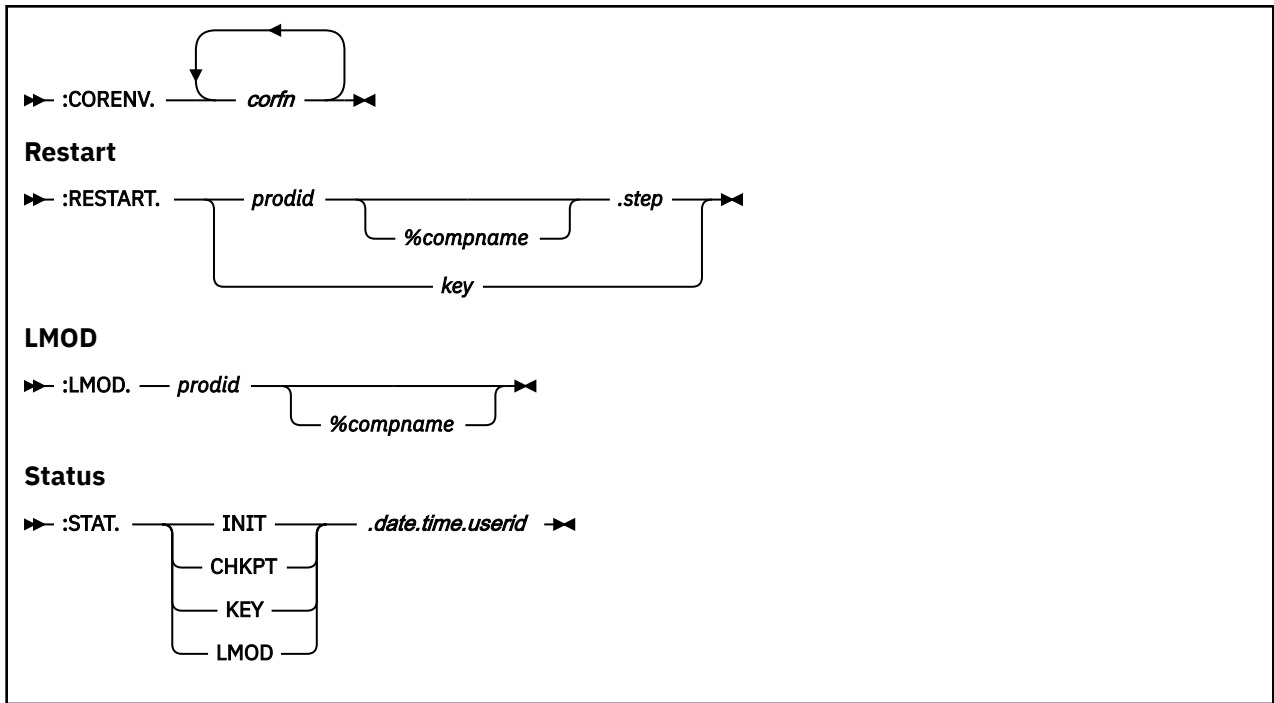
- Local Modification Restart Record

A local modification restart record is created when VMFSUFIN does not complete build processing because local modifications requiring rework were encountered. This record, which is identified by a status of LMOD in the :STAT tag, allows VMFSUFIN to complete build processing after local modifications are reworked. The record is used when VMFSUFIN is called with the RESTART option for one of the *prodid*s listed in the record. The service install process is restarted using only data from the local modification restart record.

## Syntax

System-Level Restart Table Syntax shows the syntax of the system-level restart table.





**: PACKAGE .**

identifies the service package.

**name**

is the file name of the RSU envelope or the COR envelope (whichever is received first), the tape name in the form *dfn.dft.prod*, or if the BUILD option was specified a name in the form *BUILD.prod*.

**dfn**

is the file name of the tape descriptor file.

**dft**

is the file type of the tape descriptor file.

**prod**

is the product identifier (*prodid*) of the first product on the tape.

**: PROPID .**

identifies the product.

**ALL**

indicates the ALL operand was specified on the VMFSUFIN command. The actual products to be processed follow the ALL keyword.

**prodid**

is the 7-8 character alphanumeric identifier assigned to the product by IBM (for example, 1VMVMC23).

**%compname**

is the component name preceded by a percent sign (%) (for example, %CMS). *compname* is a 1-16 character alphanumeric identifier.

**: RSUENV .**

identifies the RSU.

**rsufn**

is the file name of the RSU envelope.

**: CORENV .**

identifies the COR bucket.



**corfn**

is the file name of the COR bucket envelope.

**:RESTART .**

identifies the restart data.

**prodid**

is the identifier of the product to restart. *prodid* is the 7-8 character alphanumeric identifier assigned to the product by IBM.

**%compname**

is the component name of the product to restart.

**step**

is the service step at which to restart. The valid steps are: SETUP, MERGE, PSU, INSTALL, APPLYRSU, RECEIVE, APPLYCOR, BUILD, CLEANUP.

**key**

is the character string used to match this key restart record with a call to VMFSUFIN with a KEY option.

**:LMOD .**

identifies the local modification restart product.

**prodid**

is the identifier of the first product for which a local modification was found. *prodid* is the 7-8 character alphanumeric identifier assigned to the product by IBM.

**%compname**

is the component name of the product.

**:STAT .**

is the status of this restart record.

**INIT**

indicates this is an initial restart record.

**CHKPT**

indicates this is a checkpoint restart record.

**KEY**

indicates this is a key restart record.

**LMOD**

indicates this is a local modification restart record.

**date**

identifies the month, day, and year this record was created. The format for date is mm/dd/yy.

**time**

identifies the hour, minute, and second this record was created. The format for time is hh:mm:ss.

**userid**

identifies the user ID used when this record was created.

## Example

Figure 200 on page 699 shows an example of the system-level restart table.

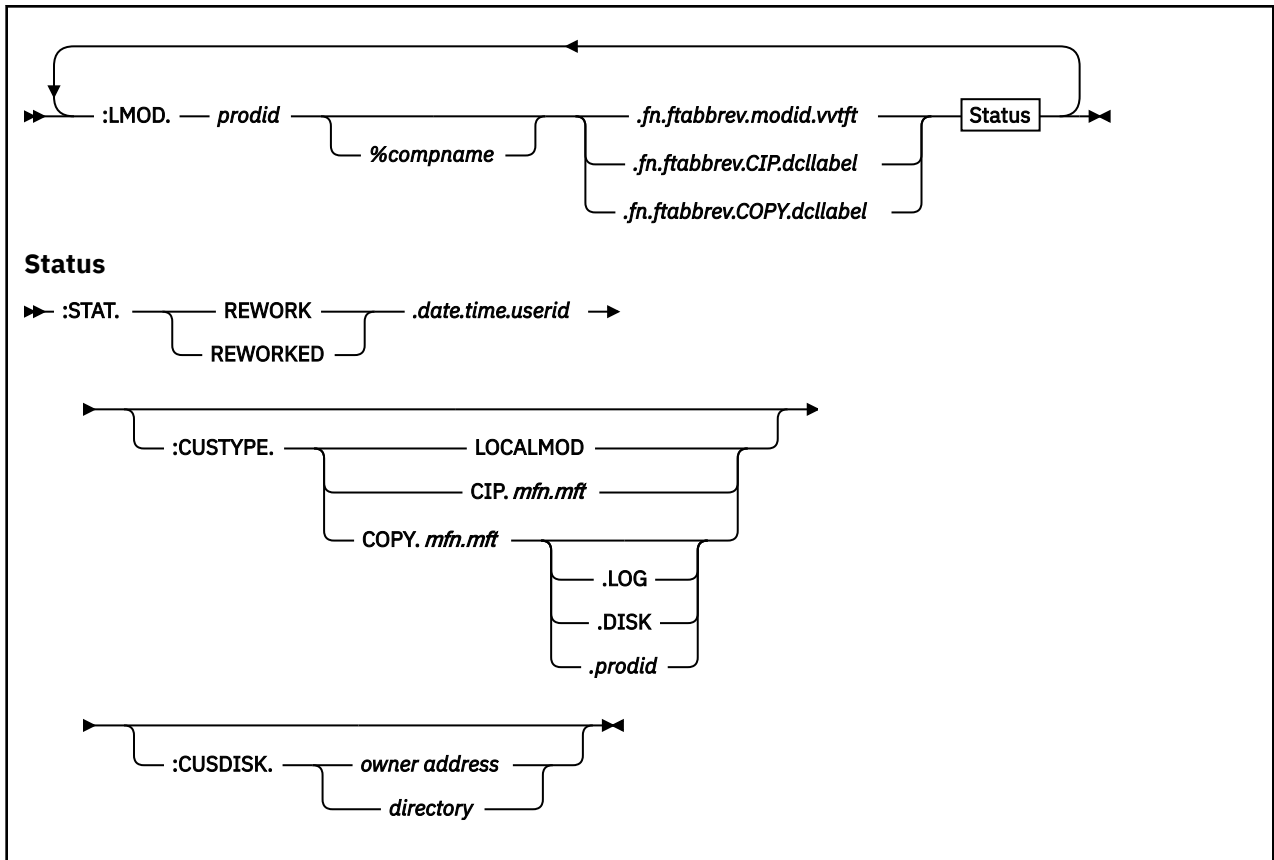
```
:PACKAGE.BUILD.7VMCMS30%CMS :PRODID.7VMCMS30%CMS :STAT.LMOD.10/03/22.12:50:48.MAINT730
:RESTART.7VMCMS30%CMS.BUILD :LMOD.7VMCMS30%CMS
```

Figure 200. System-Level Restart Table Example

## The System-Level Local Modification Table (VM SYSLMOD)

The System-Level Local Modification table contains local modifications that were previously applied to parts that were serviced using the VMFSUFIN EXEC or SERVICE EXEC and require work.

The System-Level Local Modification table resides on the Software Inventory disk. It is updated by the VMFSUFIN EXEC and the VMFUPDAT EXEC.



## :LMOD.

identifies the local modification.

### **prodid**

is the 7- or 8-character alphanumeric identifier assigned to the product by IBM (for example, 1VMVMC23).

### **%compname**

is the component name preceded by a percent sign (%) (for example, %CMS). *compname* is a 1-through 16-character alphanumeric identifier.

### **fn**

is the 1- through 8-character file name of the part.

### **ftabbrev**

is the file type abbreviation for the part. *ftabbrev* must be the 3-character file type abbreviation for the part or the real CMS file type of an update-only part.

### **modid**

is the local modification number.

### **vtft**

is the file type of the local version vector table where the modification is identified.

### **CIP**

indicates that the part is changed in place.

### **COPY**

indicates that the part is copied, then modified.

### **dcllabel**

is the variable name of the disk in the component's DCL section of its PPF, where the part resides.

## :STAT.

is the status of the local modification.

**REWORK**

indicates the local modification needs to be reworked.

**REWORKED**

indicates the local modification has been reworked.

***date***

identifies the month, day, and year this record was created or last updated. The format for date is mm/dd/yy.

***time***

identifies the hour, minute, and second this record was created or last updated. The format for time is hh:mm:ss.

***userid***

identifies the User ID that was used when this record was created or last updated.

**:CUSTYPE.**

indicates how the part is customized.

**LOCALMOD**

indicates the part is customized by local modification.

**CIP**

indicates that the part is changed in place.

**COPY**

indicates that the part is copied, then modified.

***mfn mft***

is the file name and file type of the migrated part.

**LOG**

indicates that there is information in the product migration exit log for the migrated part.

**DISK**

indicates that the part was migrated by disk migration.

***prodid***

indicates that the part is migrated by disk migration when product *prodid* is migrated.

**:CUSDISK.**

indicates where the part resides.

***owner address***

the user ID and address of the minidisk.

***directory***

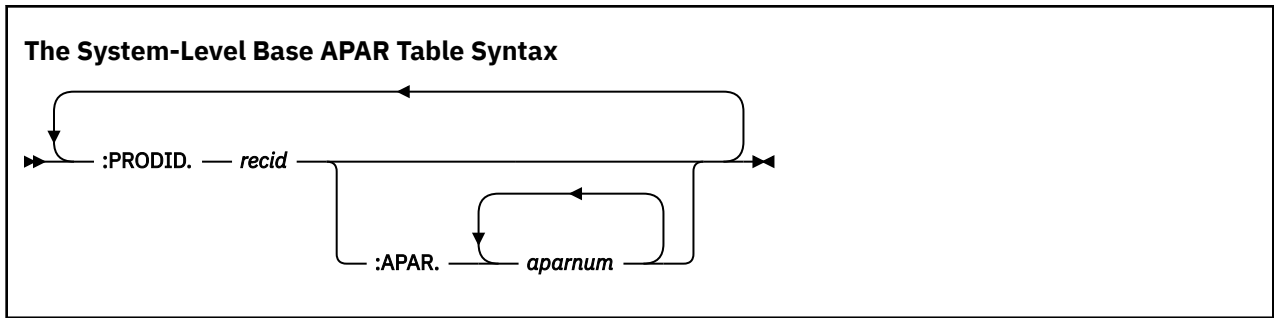
the SFS directory name.

## The System-Level Base APAR Table (VM SYSAPARS)

The system-level Base APAR table (VM SYSAPARS) contains a list of all APARs included in the base of all supported z/VM products and components. The system-level base APAR table resides on the production inventory disk (by default, PMAINT 41D).

### Syntax

The [System-Level Base APAR Table Syntax](#) shows the syntax of the system-level Base APAR table.



**:PRODID.**

identifies the product.

**recid**

is the 1–8 character alphanumeric product identifier.

**:APAR.**

identifies the APAR.

**aparnum**

is the 1–8 character alphanumeric APAR number.

**Example**

Figure 201 on page 702 shows an example of the system-level base APAR table.

```
:PRODID.1VMVMC23
:APAR.VM23456 VM23457 VM23458 VM23459 VM23460

:PRODID.1VMVMC11
:APAR.VM12345 VM12346

:PRODID.1VMVMC10
:APAR.

:PRODID.1VMVMC09
:APAR.
```

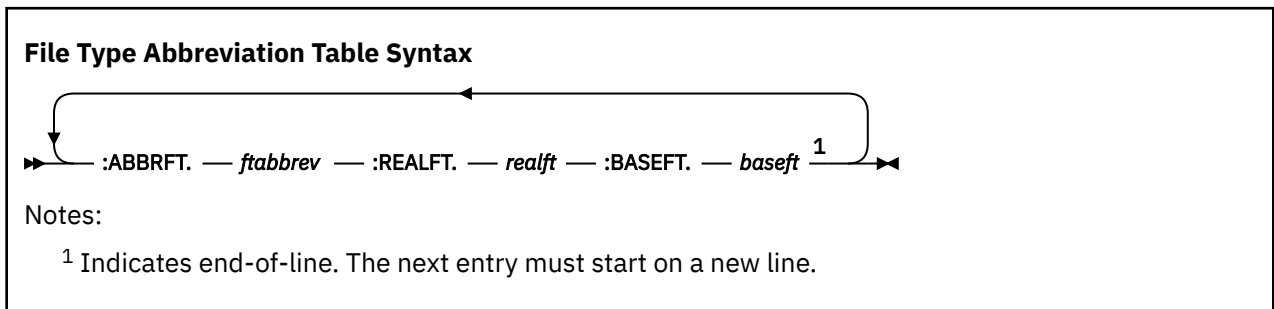
Figure 201. System-level Base APAR Table Example

**The File Type Abbreviation Table (VM SYSABRVT)**

The file type abbreviation table contains a map of the 3-character abbreviations for file types to their corresponding real CMS file types and their base file types. This translation is required by VMSES/E when processing PTF-numbered parts. The file type abbreviation table is shipped as part of the VMSES/E component.

**Syntax**

File Type Abbreviation Table Syntax shows the syntax of the file type abbreviation table.



**:ABBRFT.**

indicates a file type abbreviation.

:ABBRFT is the key field for the file type abbreviation table.

***ftabbrev***

is the 3-character file type abbreviation.

**:REALFT.**

indicates a real file type.

***realft***

is the 1-8 character CMS file type corresponding to the abbreviated file type.

**:BASEFT.**

indicates a base file type.

***baseft***

is the 8-character identifier assigned to identify the file type of a backup for base level parts. A base level part is a part of the system that has never been serviced. This identifier allows the renaming of parts that have never been serviced. These parts can be used as backups to restore the base level of the part if a PTF that replaces the base level of the part needs to be removed.

## Example

Figure 202 on page 703 shows an example of the file type abbreviation table.

:ABBRFT.CPY	:REALFT.COPY	:BASEFT.CPY000000
:ABBRFT.EXC	:REALFT.EXEC	:BASEFT.EXC000000
:ABBRFT.HCM	:REALFT.HELPCMS	:BASEFT.HCM000000
:ABBRFT.MAC	:REALFT.MACRO	:BASEFT.MAC000000
:ABBRFT.TXT	:REALFT.TEXT	:BASEFT.TXT000000
:ABBRFT.XED	:REALFT.XEDIT	:BASEFT.XED000000
:ABBRFT.\$EX	:REALFT.\$EXEC	:BASEFT.\$EX000000
:ABBRFT.\$XE	:REALFT.\$XEDIT	:BASEFT.\$XE000000

Figure 202. File Type Abbreviation Table Example

## The Parts Catalog (VMSES PARTCAT)

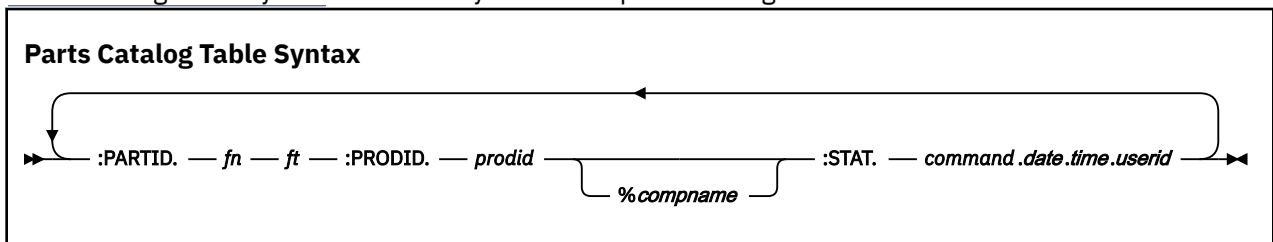
The parts catalog identifies all files on a minidisk or Shared File System directory that were stored there by VMSES/E. A parts catalog exists on every minidisk or SFS directory on which VMSES/E operates.

The parts catalog also identifies:

- To which product the file belongs
- The VMSES/E command that put the file on the minidisk or directory
- The VMSES/E command that modified the file on the minidisk or directory

## Syntax

Parts Catalog Table Syntax shows the syntax of the parts catalog.



**:PARTID.**

identifies a part that has been created or modified on the minidisk or directory.

:PARTID is the key field for the parts catalog.

***fn***

is the file name of a part.

***ft***

is the file type of a part.

**:PRODID.**

identifies the product to which the part belongs.

***prodid***

is the 7-8 character alphanumeric identifier assigned to the product by IBM (for example, 1VMVMC23). *prodid* is the file name of the PRODPART file that is shipped with the product.

***%compname***

is the component name preceded by a percent sign (%), for example %CMS. *compname* is a 1-16 character alphanumeric identifier.

**:STAT.**

indicates the last operation that was performed on the part.

***command***

is the VMSES/E command that put the part on the minidisk or SFS directory or the VMSES/E command that last modified it on that minidisk or directory.

***date***

identifies the month, day, and year the part was updated. The date is in the format *mm/dd/yy*.

***time***

identifies the hour, minute, and second the part was updated. The time is in the format *hh:mm:ss*.

***userid***

identifies the user ID that was used when the part was updated.

## Example

Figure 203 on page 704 shows an example of the parts catalog.

```
:PARTID.DMSABC COPY :PRODID.1VMVMC23 :STAT.VMFREC.11/11/22.:10:10:10.MAINT
:PARTID.DMSXXX TEXT :PRODID.1VMVMC23 :STAT.VMFREC.11/11/22.10:10:10.MAINT
:PARTID.RECEIVE EXEC :PRODID.1VMVMC23 :STAT.VMFREC.11/11/22.10:10:10.MAINT
```

Figure 203. Parts Catalog Table Example

## The Service-Level Software Inventory

The following files are in the service-level Software Inventory:

- “The PTF Parts (\$PTFPART) File” on page 705
- “The Service-Level Description Table (recid SRVDESCT)” on page 712
- “The Service-Level Requisite Table (recid SRVREQT)” on page 713
- “The Service-Level Receive Status Table (recid SRVRECS)” on page 715
- “The Service-Level Apply Status Table (appid SRVAPPS)” on page 716
- “The Service-Level Build Status Table (bldid SRVBLDS)” on page 717
- “The Service-Level Production Status Table (prodid SRVPROD)” on page 720
- “The Version Vector Table (appid VVTlvld)” on page 721

*prodid*, *appid*, and *bldid* are the values assigned to the :RECID, :APPID, and :BLDID tags in the product's product parameter file. *vlev* identifies the maintenance level and is taken from characters 4-n of the value on the :UPDTID tag in the product's product parameter file.

All files in the service-level Software Inventory (including the \$PTFPART files) reside on each product's APPLY and DELTA strings. (Remember, VMSES/E maintains service-level Software Inventories only for products that use VMSES/E for service).

## The PTF Parts (\$PTFPART) File

The PTF Parts file (\$PTFPART), *ptfnum* \$PTFPART, contains information used for product service and is supplied by the product service packaging group on the product service media. It resides on the product's DELTA string. The information in the \$PTFPART file is used to update entries in the service-level Software Inventory each time product service is received onto your system.

The \$PTFPART file has three major sections (shown in [“PTF Parts File \(\\$PTFPART\), Overall Syntax”](#) on page 705):

- A header section that identifies the service being received
- A requisite section that defines requisite PTF relationships
- A parts definition section that defines the parts being serviced

“PTF Parts File (\$PTFPART), Overall Syntax” on page 705 shows the syntax of the complete \$PTFPART file.

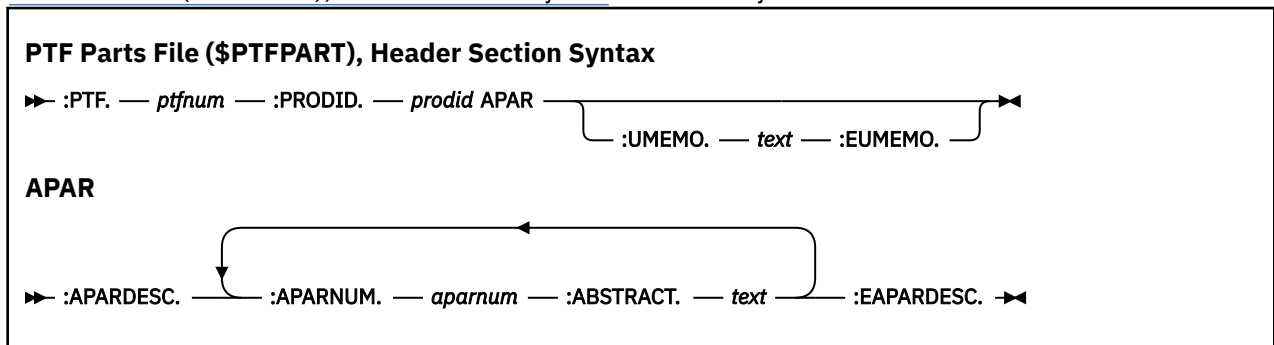


### Header Section

The header section of the \$PTFPART file identifies the service being received.

#### Syntax

PTF Parts File (\$PTFPART), Header Section Syntax shows the syntax of the header section.



The tags in the header section are defined as follows:

#### **:PTF.**

identifies the program temporary fix (PTF).

#### ***ptfnum***

is the 7-character PTF number.

#### **:PRODID.**

identifies the product the PTF belongs to.

#### ***prodid***

is the 7-8 character alphanumeric identifier assigned to the product by IBM. This number is used as the file name of the base product parameter file and PRODPART file for the product. It also identifies the product on the installation and service media.

#### **:APARDESC.**

identifies the start of an authorized program analysis report (APAR) definition block.

#### **:APARNUM.**

identifies an APAR associated with the PTF.

***aparnum***

is the 7-character APAR number.

**:ABSTRACT.**

describes the APAR. For source-maintained products, this record is used as the comment field in the AUX file created by VMFAPPLY.

***text***

is the description of the APAR. *text* may span multiple lines and is ended by the next :APARNUM tag or by the :EAPARDESC tag.

**:EAPARDESC.**

identifies the end of an APAR block.

**:UMEMO.**

identifies the start of the user memo block for this PTF.

***text***

documents the PTF. This free-form text is terminated by the :EUMEMO tag.

**:EUMEMO.**

identifies the end of the user memo block for this PTF.

### ***Example***

Figure 204 on page 706 shows an example of the header section.

```
:PTF.UV12345
:PROID.1VMVMC23
:APARDESC.
:APARNUM.VM23456
:ABSTRACT.Support for VMSES/E
:EAPARDESC.
:UMEMO.
Special instructions go here
:EUMEMO.
```

Figure 204. \$PTFPART File Header Section Example

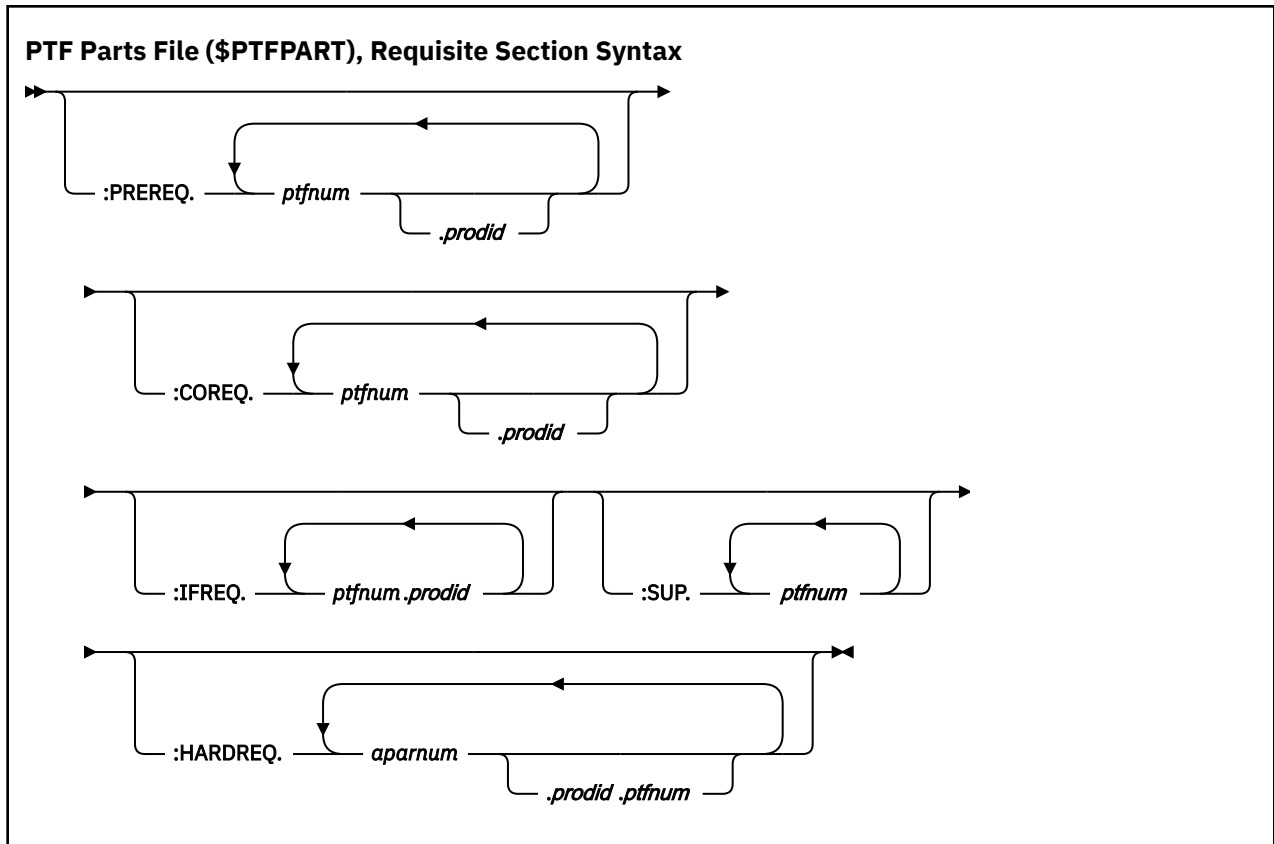
## **Requisite Section**

The requisite section of the \$PTFPART file defines requisite PTF relationships.

### ***Syntax***

PTF Parts File ([\\$PTFPART](#)), [Requisite Section Syntax](#) shows the syntax of the requisite section.





### Requisite Specifications

Data on these tags can span multiple lines and is ended by the occurrence of a tag starting in column 1. The tags in the requisite section are defined as follows:

#### **:PREREQ.**

identifies other PTFs that must be applied before this PTF is applied.

#### **:COREQ.**

identifies other PTFs that must be applied at the same time this PTF is applied. No specific order is required for applying corequisite PTFs.

#### **:IFREQ.**

identifies PTFs for another product that must be applied to that product if the product is also installed on your system. No specific order is required for applying if-requisite PTFs.

#### **:SUP.**

identifies PTFs that have been completely replaced by this PTF.

#### **:HARDREQ.**

identifies logical requisites and line intersection of source updates that are required for this PTF. If this PTF supersedes another PTF all hard requisites that were identified in the superseded PTFs should be included in this PTF.

**Note:** Hard requisites (HARDREQs) are a subset of the prerequisite change only. All corequisites and if-requisites are by definition hard requisites.

The variables specified with these tags are:

#### ***ptfnum***

is the 7-character PTF number.

#### ***prodid***

is the 7-8 character identifier assigned to the product being serviced.

*ptfnum.prod* represents a requisite relationship that is outside of the product being serviced.

**aparnum**

is the 7-character APAR number.

*aparnum.prodId.ptfnum* represents a functional requisite relationship that is outside of the product being serviced. The PTF number identified is the first PTF shipped for the product that contained the required APAR.

**Example**

Figure 205 on page 708 shows an example of the requisite section.

```
:PREREQ.UV56789 UV00001.1VMVMC23 UV99999
:COREQ.UV00002 UV00003.1VMVMC23
:SUP.UV00008
:IFREQ.UV00010.1VMVMP11
:HARDREQ.VM22222 VM33333.1VMVMP11.UV00004 VM44444.1VMVMP11.UV00001
```

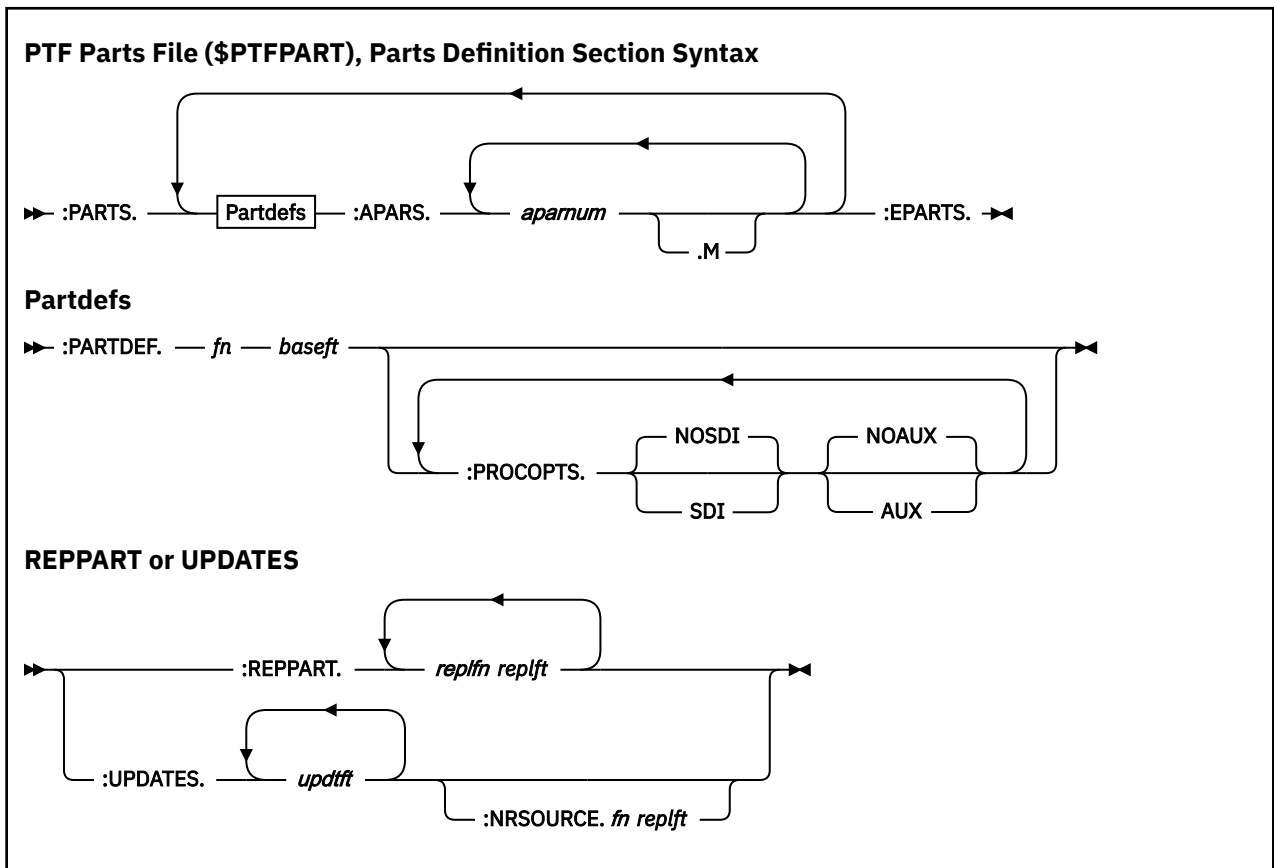
Figure 205. \$PTFPART File Requisite Section Example

**Parts Definition Section**

The parts definition section of the \$PTFPART file defines the parts being serviced.

**Syntax**

PTF Parts File (\$PTFPART), Parts Definition Section Syntax shows the syntax of the parts definition section.



The tags in the parts definition section are defined as follows:

**:PARTS.**

identifies the start of a part definition block.

**:PARTDEF.**

identifies the base part that is being serviced and the start of the parts definition section.

***fn***

is the 1-8 character file name of the base part being serviced.

***baseft***

is the 1-8 character file type of the base part being serviced.

If the part is maintained by source updates, the base part is the file name and file type of the source file.

If the part being serviced is a TEXT deck serviced by replacement only, the base part file name is the same as the file name of the base source file used to generate the TEXT deck. The base part file type is TEXT.

If the part is serviced by replacement only, the base file name is the same as the file name of the replacement part being shipped with the PTF. The base file type is the translation of the three-character abbreviation used in the file type of replacement part being shipped.

**:PROCOPTS.**

identifies the options that are to be used by VMSES/E when the base part is processed.

The options specified on this tag are dependent on the characteristics of the base part or replacement part being serviced. All options in effect, whether specified or by default, are used to process all replacement parts following the :PROCOPTS tag until another :PROCOPTS tag is specified to redefine the options.

**NOSDI**

indicates the parts identified on the :REPPART tag do not contain self-documenting information. NOSDI is the default.

**SDI**

indicates parts identified on the :REPPART tag contain self-documenting information. This information is used to verify the APARs listed for the part are included in the replacement parts specified on the :REPPART tag.

This option is only valid for a part specified on the :REPPART tag.

**NOAUX**

indicates an AUX file should not be built during apply processing for the base part on the :PARTDEF tag. NOAUX is the default.

**AUX**

indicates an AUX file should be built during apply processing for the base part on the :PARTDEF tag.

**:REPPART.**

identifies a list of replacement files that were shipped with this PTF for the base file identified on the :PARTDEF tag. Each replacement part must contain all APARs identified on the :APARS tag for this part.

***replfn***

is the 1-8 character file name of a replacement part.

***replft***

is the 8-character file type of a replacement part.

**:UPDATES.**

identifies a list of APAR source update files that were shipped with this PTF for the base file identified on the :PARTDEF tag.

***updtft***

is the 8-character file type of a source update file. An update file identified on this tag has a file type in this format: *raaaaacm*.

***r***

is a release identifier.

***aaaaaa***

is characters 3-7 of the APAR number.

**cm**

is 2-character component identifier.

For example, the file type of an update file could be R12345DS.

**:NRSOURCE.**

identifies a new or refreshed source file that is shipped with this PTF for the base file identified on the :PARTDEF tag. If the new or refreshed source has APARs merged into the file, those APARs are flagged with an M on the :APARS tag.

**fn**

is the 1-8 character file name of a new or refreshed source file.

**replft**

is the 8-character file type of a replacement part.

**:APARS.**

identifies all APARs that have been shipped in PTFs, including this PTF, for the base part. The APARs are listed in the same order that they shipped in previous PTFs. The left-most APAR number is the current APAR being fixed; the right-most APAR number was the first APAR fixed on the base part. If the *aparnum* is followed by ".M", the update associated with the APAR has been merged into the source code.

It is recommended that even though this tag is optional you should code this tag for the product. Keep in mind that if you have a product that has already created PTFs without using this tag then you will need to continue to not use this tag until a new version, release, or modification level of the product becomes available. When you have the new level, start using the tag when creating PTFs.

**aparnum**

is the 7-character APAR number.

**M**

indicates an APAR that was merged into a base, updateable, source file. No blanks are allowed in the construction of the variable.

**Note:** This tag is not required if the product being serviced does not ship source update files with its PTFs. If source updates are shipped for any part in the product, the :APARS tag must be specified in the :PARTDEF section for every part.

**:EPARTS.**

identifies the end of a parts definition block.

**Example**

[Figure 206 on page 711](#) shows an example of the parts definition section.

```

:PARTS.

:PARTDEF.RECEIVE $EXEC
:PROCOPTS.AUX
:REPPART.RECEIVE EXC12345
:UPDATES.P23456VF
:APARS.VM23456 VM34567

:PARTDEF.DMSABC ASSEMBLE
:PROCOPTS. AUX SDI
:REPPART.DMSABC TXT12345 DMSABC TXG12345
:PROCOPTS. NOAUX NOSDI
:REPPART.DMSABC MOD12345
:UPDATES.P23456VF
:NRSOURCE.DMSABC ASM12345
:APARS.VM23456 VM34567.M

:PARTDEF.CMSCALL MACRO
:PROCOPTS. AUX
:UPDATES.P23456VF
:APARS.VM23456 VM34567

:PARTDEF.DMSXYZ TEXT
:PROCOPTS. AUX SDI
:REPPART.DMSXYZ TXT12345
:APARS.VM23456 VM34567

:EPARTS.

```

Figure 206. \$PTFPART File Parts Section Example

## Example \$PTFPART File

Figure 207 on page 712 shows an example of the \$PTFPART file.

```

:PTF.UV12345
:PROPID.1VMVMC23
:APARDESC.
:APARNUM.VM23456
:ABSTRACT.Support for VMSES/E
:EAPARDESC.
:UMEMO.
Special instructions go here
:EUMEMO.
:PREREQ.UV56789 UV00001.1VMVMC23 UV99999
:COREQ.UV00002 UV00003.1VMVMC23
:SUP.UV00008
:IFREQ.UV00010.1VMVMP11
:HARDREQ.VM22222 VM33333.1VMVMP11.UV00004 VM44444.1VMVMP11.UV00001

:PARTS.
:PARTDEF.RECEIVE $EXEC
:PROCOPTS.AUX
:REPPART.RECEIVE EXC12345
:UPDATES.P23456VF
:APARS.VM23456 VM34567

:PARTDEF.DMSABC ASSEMBLE
:PROCOPTS. AUX SDI
:REPPART.DMSABC TXT12345 DMSABC TXG12345
:PROCOPTS. NOAUX NOSDI
:REPPART.DMSABC MOD12345
:UPDATES.P23456VF
:NRSOURCE.DMSABC ASM12345
:APARS.VM23456 VM34567.M

:PARTDEF.CMSCALL MACRO
:PROCOPTS. AUX
:UPDATES.P23456VF
:APARS.VM23456 VM34567

:PARTDEF.DMSXYZ TEXT
:PROCOPTS. AUX SDI
:REPPART.DMSXYZ TXT12345
:APARS.VM23456 VM34567

:EPARTS.

```

Figure 207. Example \$PTFPART File

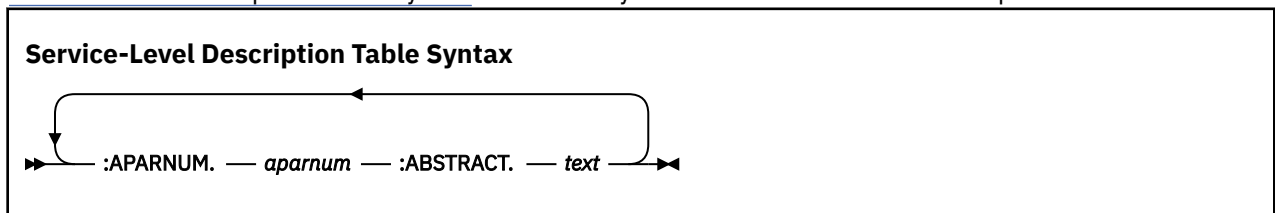
## The Service-Level Description Table (recid SRVDESCT)

The service-level description table contains the abstract information for an APAR that has been received on the system.

The service-level description table resides on the product's DELTA disk string and is updated by the VMFREC EXEC during receive processing for service tapes. Information in the \$PTFPART files is used to update this table. The VMFAPPLY EXEC uses this information to add comments to the AUX file it builds for a serviced part.

### Syntax

Service-Level Description Table Syntax shows the syntax of the service-level description table.



#### **:APARNUM.**

identifies the APAR number.

:APARNUM is the key field for the service-level description table.

***aparnum***

is the 7-character APAR number.

**:ABSTRACT.**

identifies the beginning of the abstract. A description of the APAR is contained in the abstract.

***text***

a brief description of the APAR.

**Example**

Figure 208 on page 713 shows an example of the service-level description table.

```
:APARNUM.VM23456 :ABSTRACT.Fix problem with CMS IPL
:APARNUM.VM22222 :ABSTRACT.DMSABC branches to location FFFFFFFF
```

Figure 208. Service-Level Description Table Example

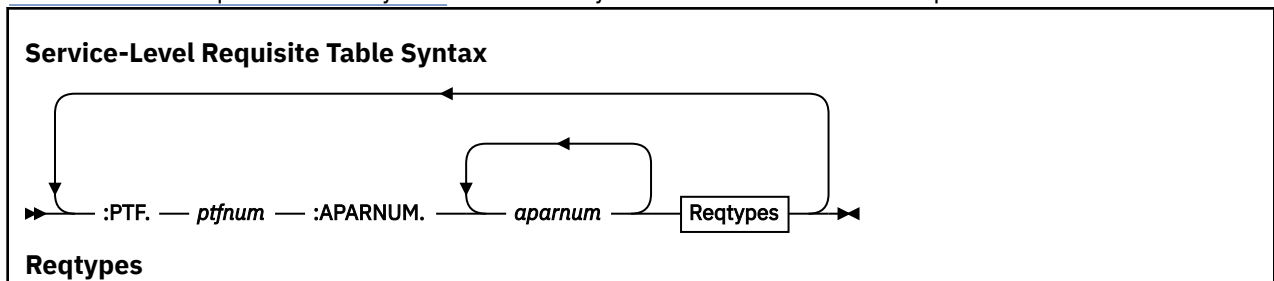
**The Service-Level Requisite Table (recid SRVREQT)**

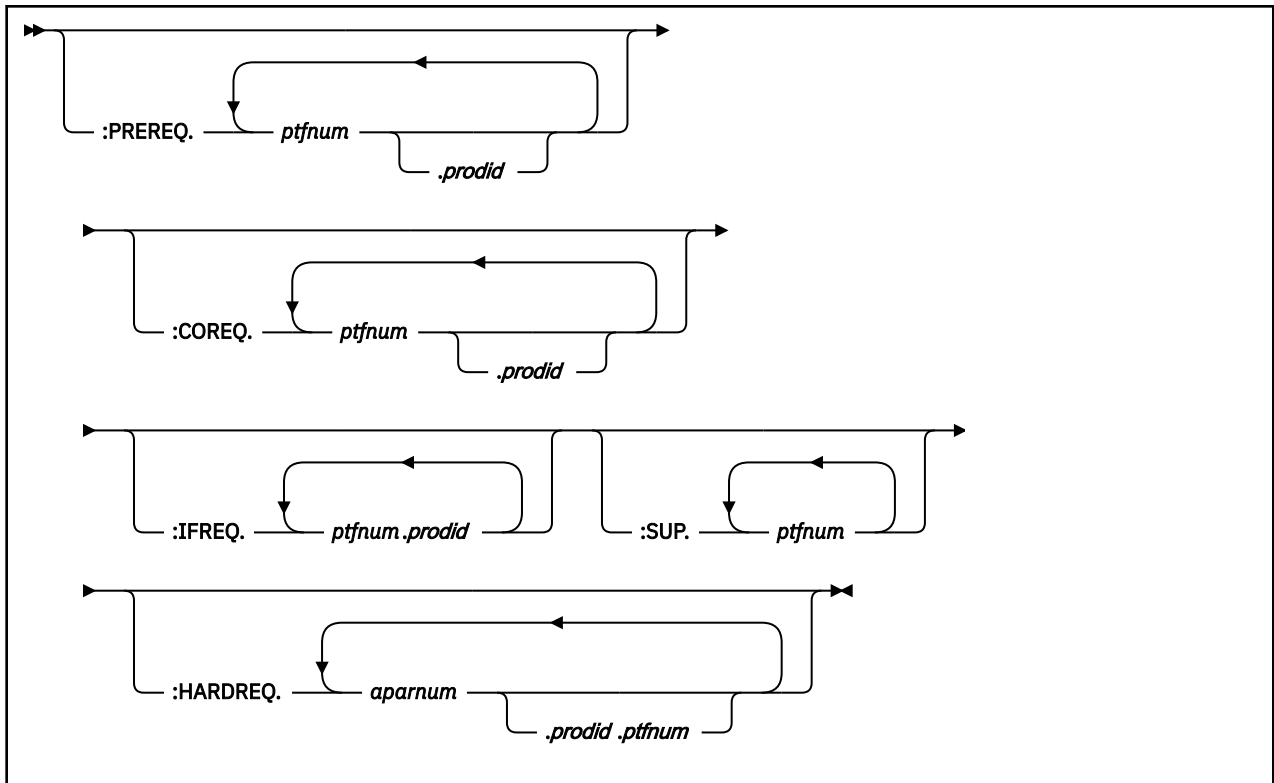
The service-level requisite table contains the relationships between PTFs that have been received on a system and a mapping of PTFs to APARs.

The service-level requisite table resides on the product's DELTA disk string and is updated by the VMFREC EXEC during receive processing for service media. Information in the \$PTFPART files is used to update this table.

**Syntax**

Service-Level Requisite Table Syntax shows the syntax of the service-level requisite table.





**:PTF.**

identifies the PTF number.

:PTF is the key field for the service-level requisite table.

***ptfnum***

is the 7-character PTF number.

**:APARNUM.**

identifies the APAR number.

***aparnum***

is the 7-character APAR number.

**Reqtypes**

The requisite tags are defined as follows:

**:PREREQ.**

identifies other PTFs that must be applied before this PTF is applied.

**:COREQ.**

identifies other PTFs that must be applied at the same time this PTF is applied. No specific order is required for applying corequisite PTFs.

**:IFREQ.**

identifies PTFs for another product that must be applied to that product if the product is also installed on your system. No specific order is required for applying if-requisite PTFs.

**:SUP.**

identifies PTFs that have been completely replaced by this PTF.

**:HARDREQ.**

identifies logical requisites and line intersection of source updates that are required for this PTF. If this PTF supersedes another PTF all hard requisites that were identified in the superseded PTFs should be included in this PTF.

**Note:** Hard requisites (HARDREQs) are a subset of the prerequisite change only. All corequisites and if-requisites are by definition hard requisites.



The variables specified on these tags are:

***ptfnum***

is the 7-character PTF number.

***prodid***

is the 7-8 character identifier assigned to the product being serviced.

*ptfnum.prodid* represents a requisite relationship that is outside the product being serviced.

***aparnum***

is the 7-character APAR number.

*aparnum.prodid.ptfnum* represents a functional requisite relationship that is outside the component being serviced. The PTF number identified is the first PTF shipped for the product that contained the required APAR.

## Example

Figure 209 on page 715 shows an example of the service-level requisite table.

```
:PTF.UV12345 :APARNUM.VM12345
:PREREQ. UV23456
:COREQ. UV45678 UV56789
:SUP. UV77777 UV88888
:IFREQ. UV66666.1VMVMC23
:HARDREQ.VM00001
```

Figure 209. Service-Level Requisite Table Example

## The Service-Level Receive Status Table (recid SRVRECS)

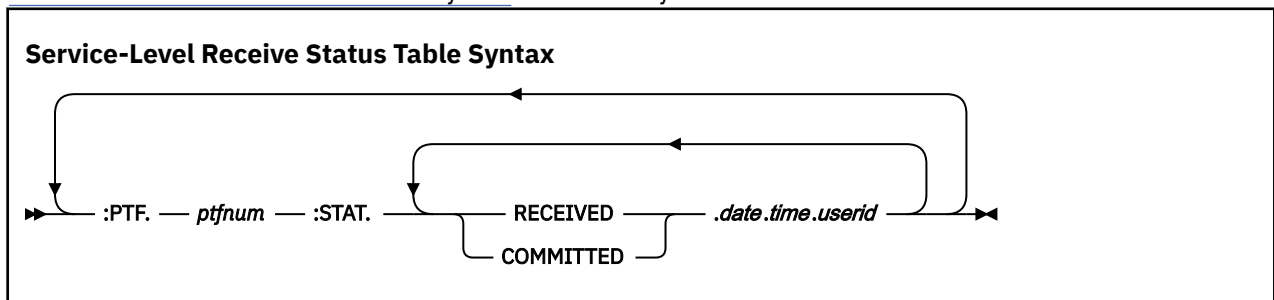
The service-level receive status table contains a list of all PTFs that have been received for the product.

The service-level receive status table resides on the product's DELTA disk string and is updated by the VMFREC EXEC as PTFs are processed during receive processing for service media.

You can access the information in this table to determine which PTFs have been received for the product.

## Syntax

Service-Level Receive Status Table Syntax shows the syntax of the service-level receive status table.



**:PTF.**

identifies the PTF.

:PTF is the key field for the service-level receive status table.

***ptfnum***

is the 7-character PTF number.

**:STAT.**

provides the status of the PTF.

**RECEIVED**

indicates the PTF has been received.

**COMMITTED**

indicates the PTF has been committed.

**date**

identifies the month, day, and year the PTF was processed. The format of date is *mm/dd/yy*.

**time**

identifies the hour, minute, and second the PTF was processed. The format of time is *hh:mm:ss*.

**userid**

identifies the user ID that was used when the PTF was processed.

**Example**

Figure 210 on page 716 shows an example of the service-level receive status table.

```
:PTF.UV12345 :STAT.RECEIVED.03/03/21.11:11:11.SMITH
:PTF.UV23456 :STAT.RECEIVED.02/03/21.12:12:12.JONES
:PTF.UV01234 :STAT.COMMITTED.04/05/22.22:22:12.JONES
                RECEIVED.01/10/21.06:06:06.MIKED
```

Figure 210. Service-Level Receive Status Table Example

**The Service-Level Apply Status Table (appid SRVAPPS)**

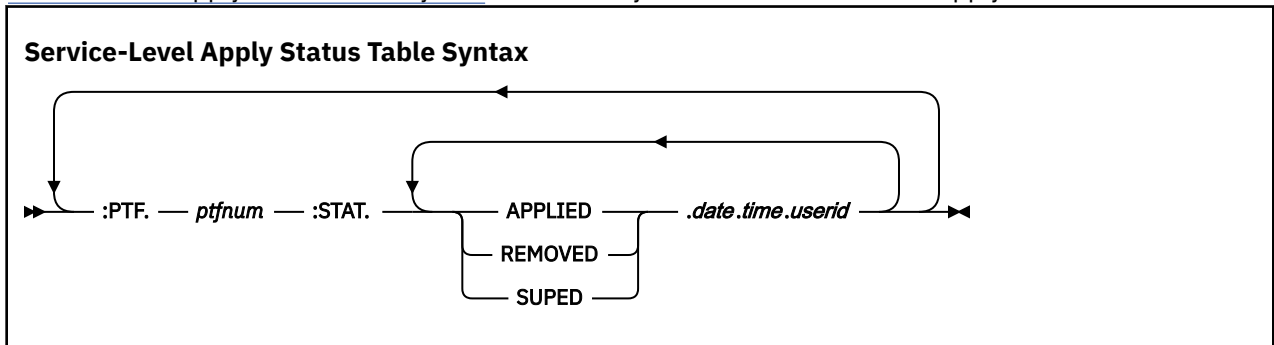
The service-level apply status table contains a list of all PTFs that have been applied to the product.

The service-level apply status table resides on the product's APPLY disk string and is updated by the VMFAPPLY EXEC during apply processing for service media.

You can access the information in this table to determine which PTFs have been applied or superseded for the product.

**Syntax**

Service-Level Apply Status Table Syntax shows the syntax of the service-level apply status table.



**:PTF.**

identifies the PTF.

:PTF is the key field for the service-level apply status table.

**ptfnum**

is the 7-character PTF number.

**:STAT.**

indicates the status of the PTF for the product.

**APPLIED**

indicates the PTF has been applied to the product.

**REMOVED**

indicates the PTF has been removed from the product.

**SUPED**

indicates the PTF has been superseded.

**date**

identifies the month, day, and year the PTF was processed. The format of date is *mm/dd/yy*.

**time**

identifies the hour, minute, and second the PTF was processed. The format of time is *hh:mm:ss*.

**userid**

identifies the user ID that was used when the PTF was processed.

**Example**

Figure 211 on page 717 shows an example of the service-level apply status table.

```
:PTF.UV12345 :STAT.APPLIED.03/03/22.22:22:22.JONES
:PTF.UV23456 :STAT.APPLIED.02/03/22.11:11:11.SMITH
:PTF.UV01234 :STAT.SUPED.11/10/22.12:12:12.SMITH
              APPLIED.06/06/21.02:02:02.JONES
```

Figure 211. Service-Level Apply Status Table Example

**The Service-Level Build Status Table (bldid SRVBLDS)**

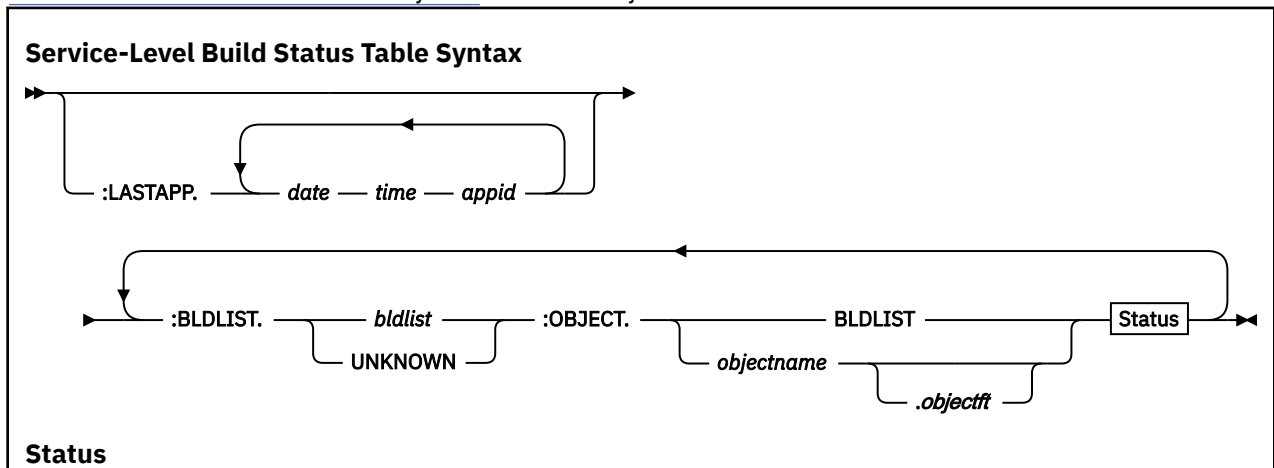
The service-level build status table contains a list of all objects that have been or need to be built for the product.

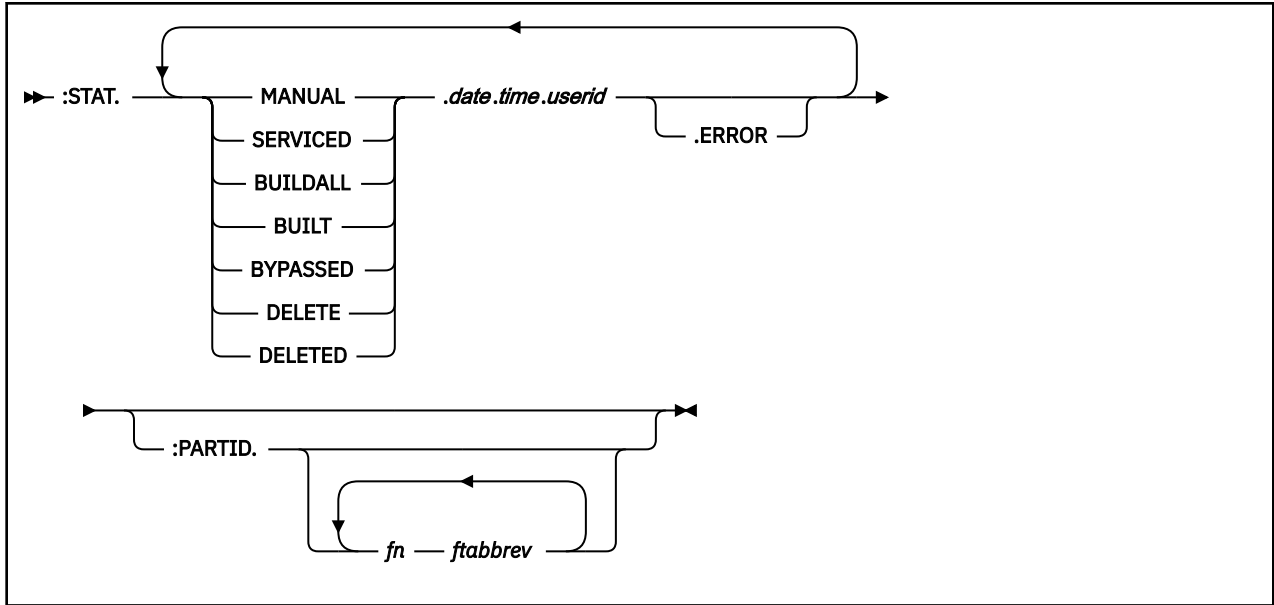
The service-level build status table resides on the product's APPLY disk and is updated by the VMFBLD EXEC as objects are generated during build processing for service media.

You can access the information in this table to determine which objects need to be generated and which objects have already been built.

**Syntax**

Service-Level Build Status Table Syntax shows the syntax of the service-level build status table.





**:LASTAPP.**

identifies the last processed block of information in the select data file.

**date**

identifies the month, day, and year of the last apply that updated the table. The format of date is *mm/dd/yy*.

**time**

identifies the hour, minute, and second of the last apply that updated the table. The format of time is *hh:mm:ss*.

**appid**

is the file name of the select data file (*appid \$SELECT*).

**Note:** The *appid* file name is the same value specified on the :APPID tag in the product parameter file.

**:BLDLIST.**

identifies the build list that contained the object identified on the :OBJECT tag.

**bldlist**

is the file name of the build list.

**UNKNOWN**

is the name of a special build list. This build list is used as a place holder for any serviceable parts not included in any defined objects.

**Note:** :BLDLIST and :OBJECT must be used together as the key field for the service-level build status table.

**:OBJECT.**

identifies an object in the specified build list.

**BLDLIST**

identifies a special object and is used to provide the overall status of a build list. For example, each member of a MACLIB is represented as a separate object; and the BLDLIST object gives you the status of the entire MACLIB.

**objname**

is the name of the object. For format 2 build lists, the *objectft* must also be specified to fully identify an object name.

**objectft**

is the file type of the object. It is only specified for format 2 build lists.

**:STAT.**

indicates the build status of the object.

**MANUAL**

indicates the object requires manual processing. This status is only assigned to the special build list named UNKNOWN.

**SERVICED**

indicates the object has been serviced but not built.

**BUILDALL**

indicates the user requested this object be built with the ALL option on the VMFBLD command and the object still needs to be built.

**BUILT**

indicates the object has been built.

**BYPASSED**

indicates that the building of an object has been bypassed because the object is superseded by the same object on a higher release level.

**DELETE**

indicates the object has been removed from the build list and the corresponding object must be deleted.

**DELETED**

indicates the object has been deleted.

***date***

identifies the month, day, and year the object was processed. The format of date is *mm/dd/yy*.

***time***

identifies the hour, minute, and second the object was processed. The format of time is *hh:mm:ss*.

***userid***

identifies the user ID that was used when the object was processed.

**.ERROR**

indicates an error was detected by the build part handler when the object was processed. .ERROR remains in effect until the object is successfully processed.

**:PARTID.**

identifies one or more parts contained in an object.

This tag is only used for wildcard objects and the special build list named UNKNOWN.

***fn***

is the file name of a part contained in an object.

***ftabbrev***

is the file type abbreviation for a part contained in an object. The *ftabbrev* must be the 3-character PTF abbreviation or the real CMS file type for parts not serviced by replacement.

**Attention**

Because entries in the select data file, service-level build status table, and build lists are used to delete objects, you should use care when modifying these files.

**Example**

Figure 212 on page 720 shows an example of the service-level build status table.

```

:LASTAPP.08/05/22 12:01:21 1VMVMC23 08/02/22 10:04:92 1VMVMP11
:BLDLIST.UNKNOWN :OBJECT.BDLIST :STAT.MANUAL.08/05/22.12:01:21.MAINT :PARTID.HCPXYZ TXT
:BLDLIST.HCPEXC :OBJECT.ABC.EXEC :STAT.BUILT.08/05/22.10:24:13.MAINT
:BLDLIST.HCPEXC :OBJECT.BDLIST :STAT.BUILT.08/05/22.10:24:13.MAINT

:BLDLIST.HCPMODS :OBJECT.HCPABC.MODULE :STAT.BUILT.08/05/22.10:35:21.MAINT
:BLDLIST.HCPMODS :OBJECT.HCPDEF.MODULE :STAT.BUILDALL.08/05/22.11:16:10.MAINT.ERROR
:BLDLIST.HCPMODS :OBJECT.BDLIST :STAT.BUILDALL.08/05/22.11:16:10.MAINT.ERROR
:BLDLIST.CPLOAD :OBJECT.BDLIST :STAT.BUILT.08/05/22.10:11:40.BILL
    
```

Figure 212. Service-Level Build Status Table Example

## The Service-Level Production Status Table (prodid SRVPROD)

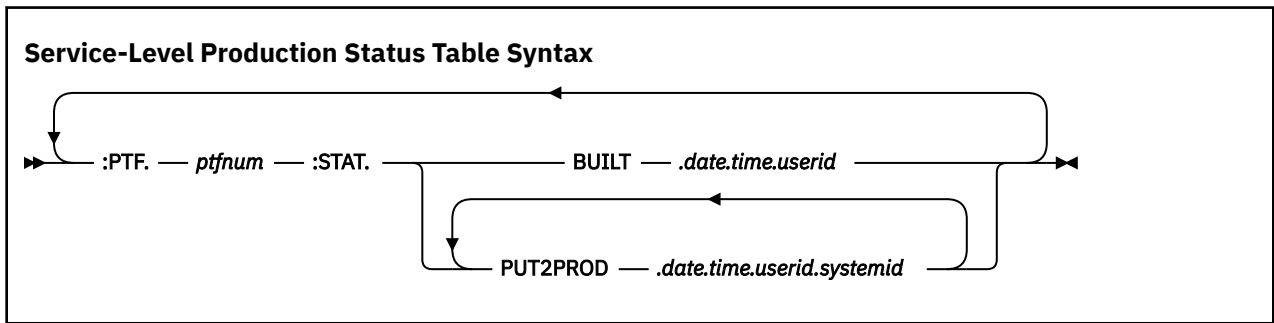
The service-level production status table contains a list of all PTFs that have been built into the product.

The service-level production status table resides on the Production Inventory disk (by default, PMAINT 41D) and is updated by the SERVICE EXEC and the PUT2PROD EXEC.

You can access the information in this table to determine which PTFs have been built or put into production.

### Syntax

Service-Level Production Status Table Syntax shows the syntax of the service-level production status table.



**:PTF.**

identifies the PTF.

:PTF is the key field for the service-level production status table.

***ptfnum***

is the 7-character PTF number.

**:STAT.**

indicates the status of the PTF for the product.

**BUILT**

indicates the PTF has been built into the product.

**PUT2PROD**

indicates the PTF has been put into production for the product.

***date***

identifies the month, day, and year the PTF was processed. The format of date is *mm/dd/yy*.

***time***

identifies the hour, minute, and second the PTF was processed. The format of time is *hh:mm:ss*.

***userid***

identifies the user ID that was used when the PTF was processed.

***systemid***

identifies the ID of the system on which the PTF was processed.

## Example

Figure 213 on page 721 shows an example of the service-level production status table.

```
:PTF.UV12345 :STAT.BUILT.03/03/22.22:22:22.MAINT
:PTF.UV23456 :STAT.BUILT.02/03/22.11:11:11.MAINT
:PTF.UV01234 :STAT.PUT2PROD.11/10/22.12:12:12.MAINT.SYS01
                BUILT.06/06/21.02:02:02.MAINT
```

Figure 213. Service-Level Apply Status Table Example

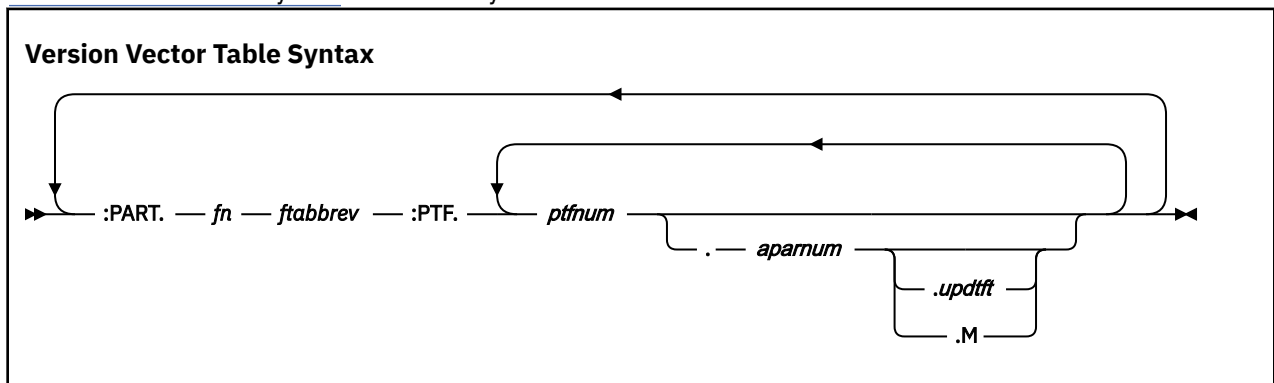
## The Version Vector Table (appid VVTlvld)

The version vector table contains a history of all PTFs that have been applied to a product at a specific maintenance level.

The version vector table resides on the product's APPLY disk and is updated by the VMFAPPLY EXEC as PTFs are applied. The VMFBLD EXEC uses this table to determine the current level of a part being processed.

## Syntax

Version Vector Table Syntax shows the syntax of the version vector table.



### **:PART.**

identifies the serviced part.

`:PART` is the key field for the version vector table.

### **fn**

is the 1-8 character file name of the serviced part.

### **ftabbrev**

is the 3-character file type abbreviation for the part. The *ftabbrev* must be the 3-character PTF abbreviation or the real CMS file type for parts that are not serviced by replacement.

### **:PTF.**

identifies the PTFs associated with the serviced part.

### **ptfnum**

is the 7-character PTF number. The numeric portion of the PTF number is concatenated to the *ftabbrev* to form the file type of the serviceable part associated with the PTF level of this part.

### **aparnum**

is the 7-character APAR number associated with the PTF.

### **updtft**

is the 8-character file type of the source update file that contains the changes for the APAR identified.

### **M**

indicates the PTF and APAR have been merged into a new or refreshed source file for the part.

## Example

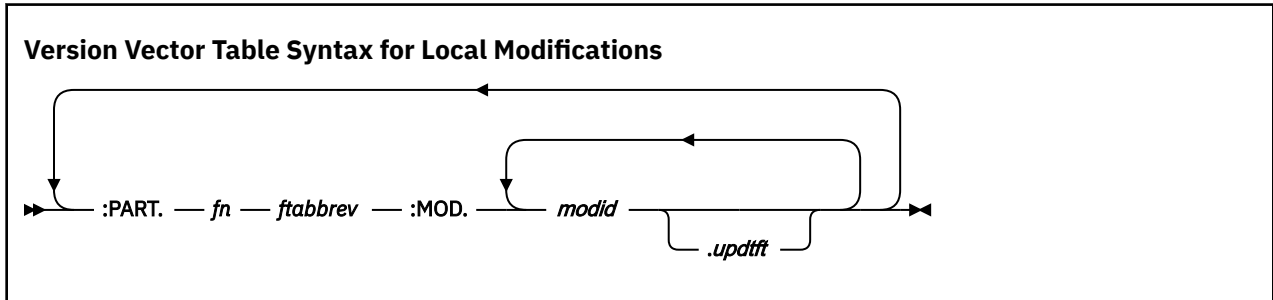
Figure 214 on page 722 shows an example of the version vector table.

```
:PART.DMSABC TXT :PTF. UV12345.VM00001 UV23456.VM00002
:PART.RECEIVE EXC :PTF. UV12345.VM00001 UV34567.VM00003
:PART.FILELIST EXC :PTF. UV12345.VM00001.P00001DS UV34567.VM00003.M
:PART.FSOPEN MACRO :PTF. UV12345.VM00001.P00001DS
```

Figure 214. Version Vector Table Example

## Version Vector Table Entries for Local Modifications

“Version Vector Table Syntax for Local Modifications” on page 722 shows the syntax version vector table entries for local modifications.



### **:PART.**

identifies the serviced part.

### **fn**

is the 1-8 character file name of the serviced part.

### **ftabbrev**

is the 3-character file type abbreviation for the part. The *ftabbrev* must be the 3-character PTF abbreviation or the real CMS file type for parts that are not serviced by replacement.

### **:MOD.**

identifies the local modifications associated with the serviced part.

### **modid**

is a 7-character local tracking number assigned to the modification. The first two characters indicate a local tracking number follows. It is recommended the first two characters be LC. Characters 3-7 are a 5-character identifier for your local modification, which you can create according to your own tracking scheme. It is recommended the first character be an L. Characters 3-7 of *lclmodid* are concatenated to the *ftabbrev* to form the file type of the serviceable part associated with this modification level of the part. For example, LCL1234 is a local modification tracking number. In this example, L1234 would be concatenated to the *ftabbrev* to form the file type of the serviceable part.

We recommend you start the local tracking number with LCL to ensure it does not interfere with service delivered by IBM. If you use characters other than LCL, make sure they are unique for your product.

### **updtft**

is the 8-character file type of the source update file that contains the changes for the local modification identified.

Figure 215 on page 722 shows an example of a version vector table entry for a local modification.

```
:PART.RECEIVE EXC :MOD. LCL1010.UPDT1010
```

Figure 215. Version Vector Table Local Modification Entry Example



## Appendix A. Related Commands and EXECs

This topic is a general reference for commands and EXECs related to VMSES/E that can be used during product installation.

The tools described in this section are:

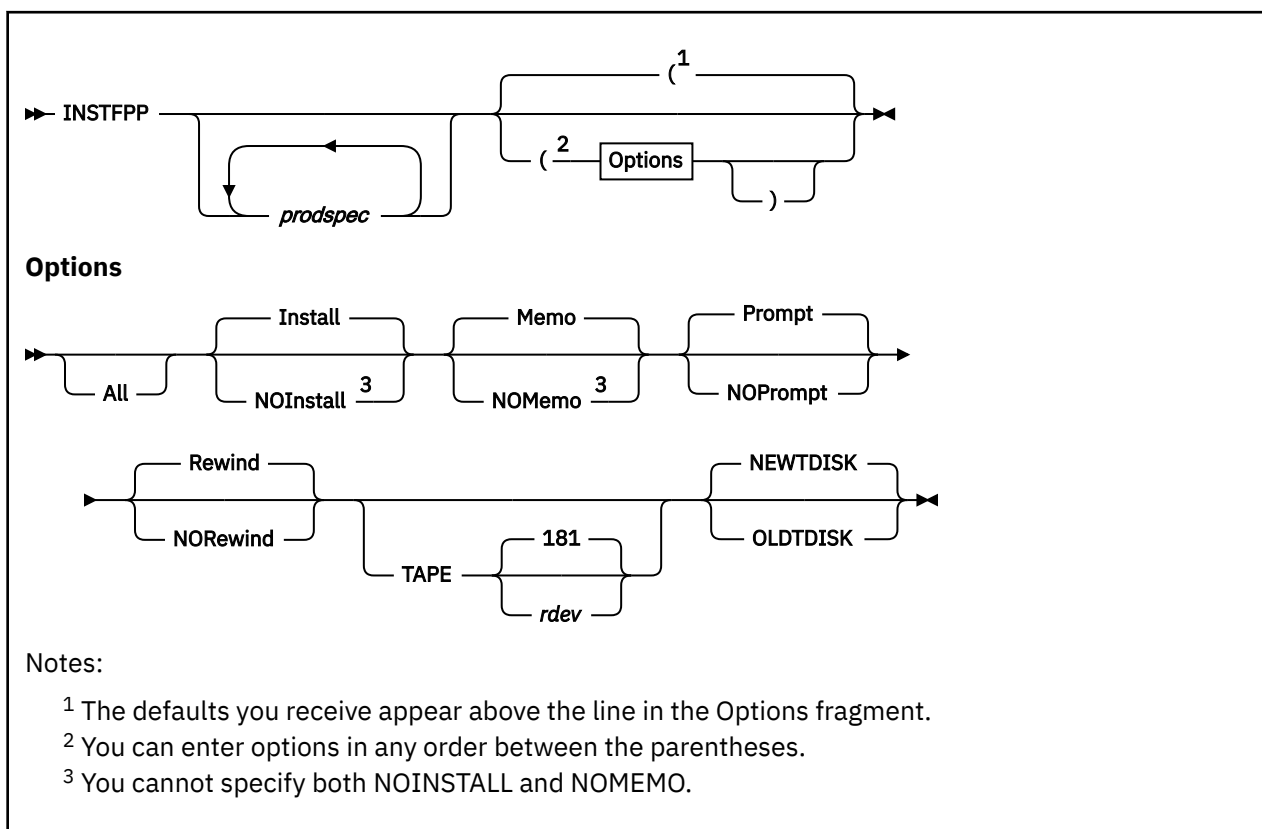
- INSTFPP EXEC
- The Patch Facility
- SNTINFO EXEC

Other tools used in system generation are described in:

- [z/VM: Installation Guide](#)
- [z/VM: Service Guide](#)
- [z/VM: CP Messages and Codes](#)
- [z/VM: CMS and REXX/VM Messages and Codes](#)
- [z/VM: Other Components Messages and Codes](#)
- [z/VM: CP Planning and Administration](#)
- [z/VM: CMS Commands and Utilities Reference](#)

Table 16 on page 229 lists the tools and indicates in which book they are described.

### INSTFPP EXEC



#### Purpose

Use the INSTFPP EXEC to install optional feature products from optional feature product tapes. INSTFPP gets input information from the \$DASD\$ CONSTS file and updates the PROD LEVEL file. You can run INSTFPP in panel mode, or you can specify products on the command line.

## Operands

### *prodspec*

is a product specification code, consisting of the product number and the feature identification code as listed on the SDO Product tapes. You can enter multiple product specification codes to indicate the products you want processed from the optional feature product tape.

To create the product specification code, attach the feature identification code to the end of the product number. If no feature identification code exists for a product, specify just the product number. Enter the codes without imbedded hyphens or other punctuation.

INSTFPP scans the files on the stacked tape and processes the selected optional feature products.

## Options

### **All**

processes all the products on the tape. ALL is the default if you do not enter product specification codes. You cannot use this option if you enter product identification codes.

### **Install**

installs the selected products. INSTALL is the default.

### **NOInstall**

processes the selected products and does not install them. If you have specified NOMEMO, you cannot specify NOINSTALL.

### **Memo**

prints the product *Memo-to-Users* from the tape for each selected product. MEMO is the default.

### **NOMemo**

processes the selected products without printing a *Memo-to-Users* for each product. If you have specified NOINSTALL, you cannot specify NOMEMO.

### **Prompt**

displays prompts that ask if you want to process the specified optional feature products. PROMPT is the default.

### **NOPrompt**

eliminates the prompts that ask if you want to install the specified optional feature products.

### **Rewind**

rewinds the tape before and after product installation. REWIND is the default.

### **NORewind**

does not rewind the tape before and after product installation. The tape must be positioned at the start of the first tape file for a product. Only products located after that initial tape position can be installed. This option is not available with INSTFPP panels.

### **TAPE**

specifies the real address of the tape drive on which the optional feature product tape is mounted. Use this option only if the tape drive is not currently attached as 181.

### **181**

is the default. INSTFPP attaches the device as 181.

### *rdev*

is the real address of the tape drive on which the optional feature product tape is mounted.

### **NEWTDISK**

Allows INSTFPP to define the size of the temporary disk space to be used for staging during product installation. NEWTDISK is the default.

**OLDTDISK**

Allows INSTFPP to use user-defined temporary disks for staging during product installation. Use this option for products requiring a large amount of disk space for the product installation and a specific search order to be able to run correctly. This option is not available when you use the INSTFPP panels. See the Usage Notes for more information.

**Using the INSTFPP Panels**

To run INSTFPP in panel mode:

1. Enter the INSTFPP command with no parameters from a terminal that displays at least 20 lines. If you enter any parameters, the INSTFPP panels are not displayed.
2. When the Installation Options panel (VMFINSP01) appears on your screen (see [Figure 216 on page 725](#)), enter the real tape drive address. Change the defaults, if necessary, and press Enter.

```

VMFINSP01          INSTFPP INSTALLATION OPTIONS
-----
If appropriate, change any defaults and then press the ENTER key.

Real tape address (will be attached as 181):
Process all products on the tape (Y/N)?          ----Y
Be prompted before each product is processed (Y/N)?      Y

Alternatives:
  (1) Install the product(s) and print the memo(s)
  (2) Only print the product memo(s)
  (3) Only install the product(s)
Enter (1, 2, or 3):      1

-----
PF1=Help    2=          3=Quit    4=          5=          6=
PF7=        8=          9=        10=         11=         12=Cursor
====>

```

*Figure 216. INSTFPP Installation Options Panel*

3. If you changed the default to *not* process all products on the tape, the Product Selection Panel (VMFINSP02) is displayed (see [Figure 217 on page 726](#)). All products included on the tape are listed. Type an **X** next to each product you want to process. Press PF5 to process the selected products, or press PF3 to quit.

```

VMFINSP02          INSTFPP PRODUCT SELECTION PANEL          Line 14 of 112
-----
Type an X next to the product(s) you want to install.
When you have finished, press the PF5 key to begin
the installation process.

- 5668801      Graphical Data Display Manager/Interactive Map Definition
- 5684007 PC   Graphical Data Display Manager Base PCLK Feature
- 5748XXB     Display Management System for CMS
- 5688004 DS   Structured Query Language/Data System Full Product
- 5688004 US   Structured Query Language/Data System User Facility Subset
- 5688004     Structured Query Language/Data System
- 5688004 NL   Structured Query Language/Data System
- 5688004     Structured Query Language/Data System
- 5668962     Assembler H
- 5668909     PL/I Optimizing Compiler, Library and PLITEST Facility
- 5668958     VS COBOL II
- 5664318     Virtual Machine/Interactive Productivity Facility
- 5664318 KA  VM/Interactive Productivity Facility
-----
PF1=Help      2=          3=Quit          4=Return      5=Execute    6=
PF7=Backward  8=Forward  9=Sort(prodid) 10=Sort(desc) 11=Sort(X) 12=Cursor

====>

```

*Figure 217. A Sample INSTFPP Product Selection Panel*

### The PROD LEVEL File

INSTFPP updates a file, named PROD LEVEL, with the results of each optional feature product installation. Figure 218 on page 726 shows examples of PROD LEVEL file entries.

```

-----
5664318 Virtual Machine / Interactive Productivity Facility (VM/IPF)
VER n REL n MOD n VM PUT nnnn. SERVICE LEVEL nnn
Time and date of entry: &hms; dd mm yy
*** Product installed and verified successfully
-----
5664282 - INTERACTIVE SYSTEM PRODUCTIVITY FACILITY 2.2.1 for VM/SP 6
VER n REL n MOD n PUT LEVEL N/A SERVICE LEVEL n
Time and date of entry: &hms; dd mm yy
*** Product installed and verified successfully

```

*Figure 218. Sample PROD LEVEL File Entries*

1. It is recommended, but not required, that you be logged on to the MAINT<sub>vr</sub>m user ID. The user ID must have all privilege classes.
2. Your virtual storage size must be 16MB unless otherwise indicated by the product *Memo-to-Users*.
3. Access 5E5 as mode B.
4. By default, the PROD LEVEL file and all product memos are placed on the MAINT<sub>vr</sub>m 319 minidisk. If your user ID does not have write access to the MAINT<sub>vr</sub>m 319 minidisk, these files will be placed on your 191 minidisk.
5. User IDs for the optional feature products being installed must have valid nonrestricted passwords in the CP directory. You cannot install a product whose password is set to NOLOG.
6. You must have available at least 30 contiguous cylinders of 3390 temporary disk space (or the equivalent).
7. You should have a hardcopy of your CP directory available. Many product installation execs link to the user minidisks in write mode using default passwords. If the link attempt fails, you might be asked to enter the write password or multiple access password of the minidisk.
8. Appropriate spooling control options must be in effect to direct the virtual printer spool files INSTFPP produces with the PRINT command to the desired real printer. You might have to enter the CP SPOOL and TAG commands.

If your printer handles only uppercase characters, use the FOLD option of the CP LOADBUF command. If your printer does not accept the LOADBUF command, use the PRINT command with the UPCASE option. In addition, if your printer cannot print special characters contained in a product *Memo-to-Users*, review the *Memo-to-Users* on the terminal.

For more information about these commands, see [z/VM: CP Commands and Utilities Reference](#) and [z/VM: CMS Commands and Utilities Reference](#).

9. INSTFPP requires access to the \$DASD\$ CONSTS file.
10. Attach the tape drive containing the feature product tape as your 181, or use the TAPE option to specify the real address of the tape and let INSTFPP attach the drive. If a device is already defined as 181, INSTFPP issues a warning message and redefines the device before attaching the tape as 181.
11. You should review the *Memo-to-Users* for each product you plan to install. To print all the product memos and save them, enter:

```
instfpp (noinstall memo
```

Each product *Memo-to-Users* is named *Iprodid MEMOfc*. *fc* is the 1-2 character feature code.

12. If you enter the INSTFPP command with arguments, you can specify up to 130 characters on the CMS command line.
13. If you spooled the console STOP before invoking INSTFPP, a console log is spooled to your reader.
14. INSTFPP cannot properly restore minidisks accessed as read-only extensions with a subset defined. INSTFPP reaccesses minidisks as read-only extensions with no subset specification.
15. INSTFPP leaves the tape drive containing the optional feature product tape attached as virtual address 181.
16. If any products do not install correctly:
  - a. Try to solve the problem by using the console log, the product *Memo-to-Users*, the PROD LEVEL file, and other product-specific documentation.
  - b. Ready the tape.
  - c. Enter the INSTFPP command.
  - d. Reinstall the products that did not install correctly. Also reinstall any products that have these products as prerequisites.
  - e. If product installation is abnormally terminated (by an irrecoverable error or you enter HX) so that INSTFPP is unable to restore the invocation environment, log off and log on again to make sure the environment is properly restored before you reenter the INSTFPP command.
17. To use the OLDTDISK option:
  - a. The product should require more than 4500 4K blocks for the installation.
  - b. You should define two temporary disks (001 and 002) that are each slightly larger than the largest disk required.
  - c. You should access 002 as the A-disk, 001 as the C-disk, and 191 as the D-disk.
  - d. You should issue the INSTFPP command with the OLDTDISK option. The OLDTDISK option cannot be used with the INSTFPP panels.

### Messages and Return Codes

For a complete explanation of any VMSES/E message, use the HELP Facility to view the message explanation online or see the appropriate messages documentation. To display information on a specific message, for example DMS002E, enter:

```
help msg dms002e
```

## Patch Facility

If you are unfamiliar with the z/VM HELP Facility, enter `help` to display the main HELP Menu. For information on the HELP command, enter:

```
help cms help
```

### Responses

For each entry that INSTFPP makes in the PROD LEVEL file, there is an update response associated with it. The possible update responses and their explanations are:

**\*\*\* Product installed and verified successfully**

The product installed correctly and was verified successfully.

**\*\*\* Product files loaded; see the Memo to Users to complete installation**

The product files have been loaded successfully. See the product Memo to Users for information about how to complete the installation of the product and how to verify that it installed correctly. In some cases, the product Memo to Users refers to other documentation.

**\*\*\* Product installed; manual verification required**

The product installed, but it was not verified automatically. See the product Memo to Users for information about how to make sure the product installed correctly. In some cases, the product Memo to Users refers to other documentation.

**\*\*\* Product installed; verification failed**

The product installed, but the automatic verification failed. Try to install the product again after correcting any problems; if it still does not verify correctly, contact your support personnel.

**\*\*\* Product Installation EXEC failed; RC = rc**

The product installation exec failed, and the return code passed back by this exec to INSTFPP is *rc*. See the product Memo to Users or product installation exec prolog to see what this return code means. If you cannot fix the problem, contact your support personnel.

## The Patch Facility

---

You can use the patch facility to make changes to TEXT files in the CP module, CP nucleus, or CMS nucleus.

The patch facility lets you maintain (patch) object code when neither source nor object deck replacements are available. These patches are a temporary solution to object code problems until you receive a replacement file from IBM.

The procedure for maintaining object code is similar to that for maintaining source code. However, with object code, rather than changing source and reassembling the source file, you will apply changes directly to the TEXT files as the nucleus is generated.

**Note:** Do not log patches in the version vector table (VVT). VMSES/E does not support the logging of patches in the VVT.

## Controlling Patches

You control patches with three kinds of files:

- Control files
- AUXiliary files
- Update files

**Note:** As with source updates to base ASSEMBLE files, patches to TEXT files do not cause any changes to the original TEXT files because temporary TEXT files are created. The temporary TEXT files are used to generate the nucleus, then erased.

The following section describes the files and elements you need to understand to make object code patches.

## Control File

The control file contains a MACLIB statement and a list of file types, one for each level of AUX file. It also contains the text deck qualifier for each level. A keyword, TX\$, must follow the last AUX file name on a line to flag the AUX file or update as containing text patches.

**Note:** The patch facility creates a temporary control file with a file name of \$\$\$TUP\$\$ CNTRL. The use of this name for any control file is restricted to the patch facility.

## AUX File

The AUX file contains a list of the file types of update files. The update files contain source updates to source or updates to TEXT files.

**Note:** Any entry in the AUX file which contains the name of a patch update file must contain the TX\$ control word between columns 8 and 13.

## Load Map

The load map function contains information from both the map of CSECT external symbol resolution and service level information. The map also includes local patches and corequisite and prerequisite information. The date and time listed in the load map for each patched text file are the original date and time, not the date and time when the temporary deck was created.

## TEXT File

The TEXT file contains the data elements that have been assembled by the customer for source maintenance or provided (already assembled) by IBM for object maintenance. You use it to create executable modules. These files contain APAR corequisite and prerequisite information.

**Note:** The patch facility creates temporary text files with a file type of TXT\$TUP\$.

## Example of a Patch Update File

The following are the functionally equivalent HCPLDR control statements contained in the update file, which will appear as comments to the update program:

```
./ * * CO-REQ: nnnnnnnn
./ * * PREREQ: nnnnnnnn
./ * * IFREQ: nnnnnnnn
./ * NAME CSECTname
./ * *
./ * VER disp data
./ * REP disp data
./ * ICS name size
```

These statements are equivalent to the loader control statements in content. \$VMFPAT\$ EXEC reformats the statements to make them acceptable to HCPLDR module. \$VMFPAT\$ is called by VMFBLD part handlers, VMFBDNUC and VMFBMOD.

**Note:** Preallocated patch areas are not required because the loader can expand any CSECT in the executable module as it is created from the text files. Addressability to the expanded area is available in every CSECT although space for a patch area need not be. This results in reduced size of executable modules because patch areas do not consume space until they are used.

The following is an explanation of each statement:

**Note:** Each statement below must be preceded by “./”.

### \* NAME CSECTname

This statement identifies which CSECT in a text file is to be patched. The *CSECTname* must match the SD name from the ESD for the module. If you omit this statement, the CSECT with the same name as the file name is patched.

### \* \* comment

This is a comment to the loader EXEC.

### \*\* CO-REQ: nnnnnnnn

This is an APAR requisite comment. Use it to indicate dependencies on APARs or other patches. The following format is suggested:

```
./ * * CO-REQ: VM23418
./ * * PREREQ: VM23418
./ * * IFREQ: VM23444
```

When there are no dependencies, enter NONE after the keywords CO-REQ:, PREREQ:, and IFREQ:.

### \* VER *disp data*

This statement verifies the patch is applied at the correct point in the executable module. You need at least one verify statement for each patch. As much data as is required to assure uniqueness should be verified. For example, you may include a verify of the date in the copyright constant of the module prologue.

The *disp* is the displacement, in hex, of the location to be patched in the CSECT.

The *data* is the existing old data in hex to be replaced. This may be up to eleven 4-digit fields separated by commas. A comma may not follow the last halfword.

### \* REP *disp data*

This statement contains the data to be replaced in the CSECT.

The *disp* contains the displacement in hex of the location to be patched in the CSECT.

The *data* is the new data in hex that will replace the existing data. This may be up to eleven 4-digit fields separated by commas. A comma may not follow the last halfword.

### \* ICS *name size*

This statement expands the CSECT in the executable module (nucleus).

The *name* is the same name used in the NAME statement. If a NAME statement is not present, *name* must match the file name of the patch file in which it is included.

The *size* is the total size required for the CSECT.

The following are the functions used in the object service process:

- VMFBLD EXEC

The VMFBLD part handlers, VMFBDNUC and VMFBDMOD, run the \$VMFPAT\$ EXEC.

- \$VMFPAT\$ EXEC

This EXEC exploits the Verify and Replace capabilities of the loader module.

It gathers TEXT files into an executable nucleus or module based upon control information.

**Note:** This function does not change previous HCPLDR command syntax except to include a NOPATCH option.

This EXEC provides the same level of control for local service, when source is not available, as is currently provided for source file updates. This means the nucleus or module generation process automatically locates and tries to reapply a local patch each time a replacement update file arrives from IBM.

This results in one of two possibilities:

1. The local fix is applied and you receive replacement update files. You must then evaluate whether:

- The local fix is obsolete.

If the replacement update file contains the APAR fix that replaces the local fix, you can remove the local fix by removing the object update from the AUX file or the control file.

- The local fix is still needed.

If the replacement update file does not contain an APAR fix that replaces the local fix you will leave the local fix in place.



2. The local fix is not applied and you receive replacement service files. You must then evaluate whether:

- The local fix is obsolete.

If the replacement update file contains an APAR fix that replaces the local fix, you just accept the replacement deck.

- The replacement update file does not contain an APAR fix equivalent to the local fix.

If the replacement update file does not contain an APAR fix that replaces the local fix, you may:

- Choose the APAR fix as more important, and just accept the replacement update file.
- Fix both problems by accepting the replacement update file and reworking the local fix.

## Compatibility with HCPLDR

A temporary composite TEXT file is created by the \$VMFPAT\$ EXEC; the composite TEXT file is a copy of the IBM-supplied TEXT file with the patch update file merged into it. The temporary text deck will have the same date and time as the original. The NAME statement is not included in a composite TEXT file. The NAME statement is used only to identify the correct CSECT. When this statement is not present, the patch update file changes the CSECT with the same name as the TEXT file.

A composite TEXT file may include:

- The patch AUX file entry, placed following existing comments but before the first ESD statement.
- An APAR requisite comment, placed after the AUX file comment, but before the first ESD statement.
- An optional ICS statement, placed after an APAR requisite statement, but before the first ESD statement.
- Required VER and REP statements, placed after any ICS statement, but before the first RLD statement or before the END statement if there is no RLD statement in the TEXT deck.
- Patch update file comments, included as they are encountered.

For each NAME statement in a patch update file and for each patch update file without a NAME statement, a VERIFY statement must be present or the patch is not accepted by the \$VMFPAT\$ EXEC. The composite TEXT file will have the same file name and its file type will be TXT\$TUP\$.

The \$VMFPAT\$ EXEC goes through the load list one module at a time, using the CNTRL file to determine the file type. For each temporary composite TEXT file, the EXEC provides a file name with the file type TXT\$TUP\$. In addition, the EXEC creates a \$\$\$TUP\$\$ CNTRL file that contains the file type qualifier (\$TUP\$ that produces a file type of TXT\$TUP\$) of the composite TEXT file created by the loader EXEC.

The \$VMFPAT\$ EXEC calls the HCPLDR MODULE by passing it a temporary control file named \$\$\$TUP\$\$ along with other parameters. All temporary files such as TXT\$TUP\$ or \$\$\$TUP\$\$ have a file mode of 3; these files are erased automatically as soon as they are used.

## Usage Notes

The patch facility is provided to give you:

- The same control you have with source updates and IBM update file updates.
- The same tracking capability so that no previously applied patch will be lost or ignored when IBM replacement service is applied.

The following guidelines are recommendations to follow:

1. Keep each fix to a TEXT file in a separate update file.

This also applies to source update fixes; each source update fix should be in a separate update file.

Each fix should have an alphanumeric number that is the file type of the update file.

2. Keep all local fix descriptions for the same TEXT file in the same AUX file, unless a fix applies to a different control file level.

## Patch Facility

Local fixes for the same TEXT file should not be distributed over AUX files (different control file levels) arbitrarily. Local service should be easily distinguished from IBM service and should always be applied last. Local service can be distributed over separate control files for the purpose of maintaining different service levels with a single structure of AUX and update files. Each level can be built from a different control file containing only the desired level identifiers.

3. Never place local patches in AUX files from IBM. In other words, keep your local service separate from IBM service. Local service should be easily distinguished from IBM service and should always be applied last.
4. Patches to TEXT files should be applied only when no source file is available. Mixing source updates and TEXT file patches for the same module will lead to confusion and is not recommended.

### Example of Local Service to TEXT Files

Table 32 on page 732 shows an example of a control structure containing patches at more than one level. This is an appropriate use of the patch facility.

<i>Table 32. Control Structure Containing Patches at Multiple Levels</i>		
File Name	File Type	Contents
PRODSYS	CNTRL	L3 AUXLCL3 TX\$ P1 AUXP1 TEXT AUXVM
CMSSYS	CNTRL	L2 AUXLCL2 TX\$ L3 AUXLCL3 TX\$ P2 AUXP2 P1 AUXP1 TEXT AUXVM
TESTSYS	CNTRL	L1 AUXLCL1 TX\$ L2 AUXLCL2 TX\$ L3 AUXLCL3 TX\$ P3 AUXP3 P2 AUXP2 P1 AUXP1 TEXT AUXVM
HCPXYZ	AUXLCL1	PATCH3 TX\$ APAR3
HCPXYZ	PATCH3	./ * VER 24 4780,C204 ./ * REP 24 4700
HCPXYZ	AUXLCL2	PATCH2 TX\$ APAR2
HCPXYZ	PATCH2	./ * VER 254 47F0,6062 ./ * REP 254 4700 ./ * VER 260 5810,7042,5010 ./ * REP 260 58F0,7042,50F0
HCPXYZ	AUXLCL3	PATCH1 TX\$ APAR1
HCPXYZ	PATCH1	./ * VER 254 4740,6062 ./ * REP 254 47F0

*Table 32. Control Structure Containing Patches at Multiple Levels (continued)*

File Name	File Type	Contents
HCPXYZ	TEXT	ESD TXT RLD END

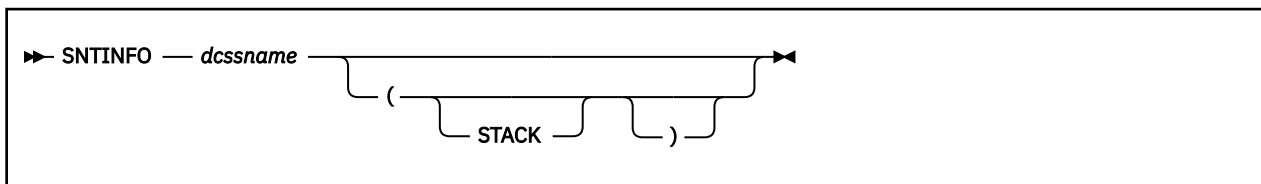
### Example of Local Service to ASSEMBLE Files

Table 33 on page 733 shows an example of a control structure containing a temporary patch over a local source update. Although this works, it is not recommended. Whenever source code is available, you should use source updates instead of patches.

*Table 33. Control Structure Containing Temporary Patch Over Local Source Update*

File Name	File Type	Contents
XASYS	CNTRL	L1 AUXLCL1 TX\$ P1 AUXP1 TEXT AUXVM
HCPXYZ	AUXLCL1	TEMP02 TX\$ PROB1 LCFIX5 TX\$ APAR2
HCPXYZ	TEMP02	./ * VER 24 4780,C204 ./ * REP 24 4700
HCPXYZ	LCFIX5	./ ADD 12340000 XYZLOOP TM FLAG,X'01' CHECK COMPLETE B0 DONE EXIT LOOP
HCPXYZ	TXTL1	ESD TXT RLD END

## SNTINFO EXEC



### Purpose

Use the SNTINFO EXEC to get discontinuous saved segment (DCSS) information directly from CP.

### Operands

#### *dcssname*

is the name of the discontinuous saved segment you want information about.

## Options

### STACK

puts the saved segment information on the stack (LIFO) and does not display it on the terminal. No error messages are issued when the STACK option is specified.

## Usage Notes

1. You can run SNTINFO from the CMS command line or call it from a REXX or EXEC2 exec.
2. If you do not specify the STACK option, the following line (8 tokens) is displayed on the terminal:

```
START(HEX): start  END(HEX): end  SIZE(HEX): size  CC: cc
```

3. If you specify the STACK option, the following line (4 tokens) is pushed LIFO onto the stack:

```
start end size cc
```

### ***start***

is the saved segment start load address in hexadecimal.

### ***end***

is the saved segment end load address in hexadecimal.

### ***size***

is the saved segment size in hexadecimal.

### ***cc***

is the saved segment condition code from the CP DIAGNOSE command.

## Messages and Return Codes

For a complete explanation of any VMSES/E message, use the HELP Facility to view the message explanation online or see the appropriate messages documentation. To display information on a specific message, for example DMS002E, enter:

```
help msg dms002e
```

If you are unfamiliar with the z/VM HELP Facility, enter help to display the main HELP Menu. For information on the HELP command, enter:

```
help cms help
```

## Appendix B. Input/Output Files

Table 34 on page 735 summarizes the files used for input and output by the VMSES/E EXECs. Files are arranged in alphabetical order by file type, with variables following special characters.

For explanations of how these files are used, see the discussions of the individual EXECs in Chapter 20, “VMSES/E Exec and Command Format Summaries,” on page 229.

<i>Table 34. Input/Output Files</i>			
<b>File ID</b>	<b>Used as input by:</b>	<b>Provided as output by:</b>	<b>Used as a temporary file by:</b>
<i>Bponum</i>	VMFINS BUILD		
<i>Vponum</i>	VMFINS BUILD		
<i>axlist APxnnnn\$</i>			VMFREC
HCPLDL ASSEMBLE	GENCPBLS		
<i>\$fn</i> ASSEMBLE			VMFASM, VMFHASM, VMFHLASM, VMFNLS
<i>fn</i> ASSEMBLE	VMFASM, VMFHASM, VMFHLASM, VMFNLS		
<i>partname</i> ASSEMBLE			VMFBLD
<i>fn</i> AUX\$\$\$\$			VMFASM, VMFHASM, VMFHLASM, VMFNLS, VMFSIM CHKLVL
HCPLDL AUX <i>lvlid</i>	GENCPBLS	GENCPBLS	
HCPMDLAT AUX <i>lvlid</i>	GENCPBLS		
<i>fn</i> AUX <i>lvlid</i>	VMFASM, VMFEXUPD, VMFHASM, VMFHLASM, VMFNLS		
<i>partid</i> AUX <i>lvlid</i>	VMFSIM CHKLVL	VMFAPPLY, VMFREM	
<i>\$VMFSIM</i> CNTRL			VMFASM, VMFHASM, VMFHLASM, VMFNLS, VMFSIM CHKLVL
<i>cntrlfn</i> CNTRL	GENCPBLS, LOCALMOD, VMFAPPLY, VMFASM, VMFBLD, VMFEXUPD, VMFHASM, VMFHLASM, VMFNLS, VMFPSU, VMFQOBJ, VMFREPL, VMFSIM CHKLVL, VMFSIM GETLVL, VMFREM		
<i>cntrlfn</i> CNTRLEXT	VMFQOBJ, VMFSIM CHKLVL, VMFSIM GETLVL		

## Input and Output Files

<i>Table 34. Input/Output Files (continued)</i>			
<b>File ID</b>	<b>Used as input by:</b>	<b>Provided as output by:</b>	<b>Used as a temporary file by:</b>
VMFINS CONSOLE		VMFINS BUILD, VMFINS DELETE, VMFINS INSTALL, VMFINS MIGRATE	
\$DASD\$ CONSTS	VMFCNVT, VMFMRDSK		
\$MLB\$ COPY			VMFBLD
\$CLB\$ CSLCNTRL			VMFBLD
\$CLB\$ CSLLIB			VMFBLD
<i>libname</i> CSLLIB		VMFBLD	
\$CLB\$ CSLSEG			VMFBLD
<i>libname</i> CSLSEG		VMFBLD	
VMFINS DEFAULTS	VMFENRPT, VMFINS BUILD, VMFINS DELETE, VMFINS DISABLE, VMFINS ENABLE, VMFINS INSTALL, VMFINS MIGRATE, VMFUPDAT		
SERVICE DISKMAP		VMFREC	
<i>fn</i> DLCS	VMFNLS		
<i>fname</i> DLKEDERR		VMFBLD	
\$DLB\$ DLKEDIT			VMFBLD
<i>libname</i> DLKEDIT		VMFBLD	
\$DLBB\$ DOSLIB			VMFBLD
\$DLB\$ DOSLIB			VMFBLD
<i>libname</i> DOSLIB		VMFBLD	
<i>fname</i> DOSLNK			VMFBLD
<i>ppfname</i> ERASE		VMFINS DELETE, VMFINS INSTALL, VMFINS MIGRATE	
<i>cploadblist</i> <i>cploadftype</i>		GENCPBLS	
<i>bldlist</i> EXCnnnnn	VMFBLD, VMFSGMAP, VMFREM	VMFSGMAP	
<i>prodid</i> EXEC		VMFINS DELETE, VMFINS DISABLE, VMFINS ENABLE, VMFINS INSTALL, VMFINS MIGRATE	
\$SRCLST\$ EXEC			VMFMRDSK
\$TRGLST\$ EXEC			VMFMRDSK
\$\$\$TLL\$\$ EXEC			VMFBLD

<i>Table 34. Input/Output Files (continued)</i>			
<b>File ID</b>	<b>Used as input by:</b>	<b>Provided as output by:</b>	<b>Used as a temporary file by:</b>
<i>axlist</i> EXxxxxx\$			VMFREC
VMFHPBLD HPANEL	VMFUPDAT		
VMFHPLMD HPANEL	VMFUPDAT		
VMFHPMEM HPANEL	VMFUPDAT		
VMFHPRST HPANEL	VMFUPDAT		
VMFHPSSEL HPANEL	VMFUPDAT		
VMFHPSUF HPANEL	VMFUPDAT		
VMFNLS LANGLIST	VMFBLD, VMFINS BUILD, VMFINS INSTALL, VMFNLS, VMFQOBJ		
VMPFXALL BITMAP	VMFBTMAP	VMFBTMAP	
VMPFX-aa BITMAP		VMFBTMAP	
\$CLB\$ LIBMAP			VMFBLD
<i>libname</i> LIBMAP		VMFBLD	
<i>\$fn</i> LISTING		VMFNLS	
<i>fn</i> LISTING			VMFNLS
<i>objname</i> LISTING		VMFBLD	
\$LLB\$ LKEDMAP			VMFBLD
\$LLB\$ LOADLIB			VMFBLD
\$MLB\$ MACLIB			VMFBLD
\$TMP\$ MACLIB			GENCPBLS
HCPMDLAT MACRO	GENCPBLS		
<i>\$objname</i> MAP			VMFBLD
<i>bldlist</i> MAP		VMFBLD	
<i>objname</i> MAP		VMFBLD	
<i>prodid</i> MIGPvrn	System upgrade utilities	System upgrade utilities	
<i>\$objname</i> MODULE			VMFBLD
<i>fn</i> OLDDATA		VMFSIM CHKLVL, VMFSIM COMPTBL, VMFSIM GETLVL, VMFSIM QUERY, VMFSIM SRVDEP, VMFSIM SRVREQ, VMFSIM SYSDEP, VMFSIM SYSREQ	
SUF OUT		VMFSUFTB	

## Input and Output Files

<i>Table 34. Input/Output Files (continued)</i>			
<b>File ID</b>	<b>Used as input by:</b>	<b>Provided as output by:</b>	<b>Used as a temporary file by:</b>
VMSES PARTCAT	VMFINS DELETE, VMFINS MIGRATE	VMFAPPLY, VMFBLD, VMFCOPY, VMFERASE, VMFINS BUILD, VMFINS DELETE, VMFINS MIGRATE, VMFMRDSK, VMFREC, VMFREM	
<i>prodid</i> PLANINFO		VMFINS DELETE, VMFINS INSTALL, VMFINS MIGRATE	
<i>ppfname</i> PPF	GENCPBLS, LOCALMOD, SERVICE, VMFAPPLY, VMFASM, VMFBTMAP, VMFBLD, VMFEXUPD, VMFHASM, VMFHLASM, VMFINS BUILD, VMFINS DELETE, VMFINS DISABLE, VMFINS ENABLE, VMFINS MIGRATE, VMFMRDSK, VMFNLS, VMFPSU, VMFQMDA, VMFQOBJ, VMFREC, VMFREPL, VMFSETUP, VMFSGMAP, VMFSIM CHKLVL, VMFSIM GETLVL, VMFVIEW, VMFREM, VMFUPDAT	VMFPPF	
<i>ppfname1</i> PPF	VMFINS INSTALL		
<i>ppfname2</i> PPF		VMFINS INSTALL	
VMFINS PRODLIST	VMFINS INSTALL, VMFINS MIGRATE	VMFINS INSTALL, VMFINS MIGRATE	
<i>prodid</i> PRODPART	VMFINS DELETE, VMFINS INSTALL, VMFINS MIGRATE, VMFPSU, VMFSGMAP, VMFSIM INIT		
<i>prodid</i> PRODSYS		VMFINS DELETE, VMFINS DISABLE, VMFINS ENABLE, VMFINS INSTALL, VMFINS MIGRATE	
VMFVIEW\$ PROFILE	VMFVIEW		
VMSESE PROFILE	VMFBLD		
<i>appid</i> PSUPLAN		VMFPSU	
VMFENRPT REPORT		VMFENRPT	
<i>fn</i> REPOS	VMFNLS		



<i>Table 34. Input/Output Files (continued)</i>			
<b>File ID</b>	<b>Used as input by:</b>	<b>Provided as output by:</b>	<b>Used as a temporary file by:</b>
<i>bldlist</i> SEGDATA	VMFBLD, VMFSGMAP	VMFSGMAP	
\$CLB\$ SEGMAP			VMFBLD
<i>fn</i> SEGMAP		VMFSGMAP	
<i>libname</i> SEGMAP		VMFBLD	
VMPUT SERVICE		VMFREC	
<i>fn</i> SERVLINK	SERVICE, VMFINS, VMFREC, VMFSUFIN		
<i>fn</i> SIMDATA	VMFSIM CHKLVL, VMFSIM LOGMOD, VMFSIM MODIFY, VMFSIM QUERY, VMFSIM SRVDEP, VMFSIM SRVREQ, VMFSIM SYSDEP, VMFSIM SYSREQ	VMFSIM CHKLVL, VMFSIM COMPTBL, VMFSIM GETLVL, VMFSIM QUERY, VMFSIM SRVDEP, VMFSIM SRVREQ, VMFSIM SYSDEP, VMFSIM SYSREQ	
<i>appid</i> SRVAPPS	SERVICE, VMFAPPLY, VMFBLD, VMFBTMAP, VMFINS BUILD, VMFMRDSK, VMFSIM SRVDEP, VMFSIM SRVREQ, VMFREM	VMFAPPLY, VMFREM	
<i>bldid</i> SRVBLDS	VMFBLD, VMFINS BUILD, VMFQOBJ, VMFUPDAT	VMFBLD, VMFINS BUILD, VMFUPDAT	
\$\$\$VM\$\$\$ SRVBLDS			VMFUPDAT
<i>recid</i> SRVDESCT	VMFREM	VMFBTMAP, VMFREC, VMFSIM INIT, VMFREM	
<i>prodid</i> SRVPROD	SERVICE	SERVICE, PUT2PROD	
<i>recid</i> SRVRECS	SERVICE, VMFBTMAP, VMFREC, VMFREM	VMFREC, VMFSIM INIT, VMFREM	
<i>recid</i> SRVREQT	SERVICE, VMFAPPLY, VMFSIM SRVDEP, VMFSIM SRVREQ, VMFREM	VMFREC, VMFSIM INIT, VMFREM	
CSM SVCSTAT	SERVMGR	SERVMGR	
CSM SYSSTAT	SERVMGR	SERVMGR	
VM SYSABRVT	VMFBLD, VMFINS BUILD, VMFREC, VMFSIM CHKLVL, VMFSIM GETLVL		
VM SYSAPARS	VMFBLD		
<i>appid</i> SYSAPPS	VMFSIM SYSDEP, VMFSIM SYSREQ		

## Input and Output Files

<i>Table 34. Input/Output Files (continued)</i>			
<b>File ID</b>	<b>Used as input by:</b>	<b>Provided as output by:</b>	<b>Used as a temporary file by:</b>
<i>sysid</i> SYSAPPS	VMFBTMAP, VMFENRPT, VMFINS BUILD, VMFINS MIGRATE, VMFSUFTB, SERVICE	VMFINS BUILD, VMFINS DELETE, VMFINS DISABLE, VMFINS ENABLE, VMFINS INSTALL, VMFINS MIGRATE	
<i>sysid</i> SYSBLDS		VMFINS BUILD, VMFINS DELETE	
<i>sysid</i> SYSDESCT	VMFENRPT, VMFINS BUILD, VMFINS DELETE, VMFINS DISABLE, VMFINS ENABLE, VMFINS MIGRATE, VMFSUFTB	VMFINS INSTALL, VMFINS MIGRATE, VMFSIM INIT	
<i>sysid</i> SYSLMOD	SERVICE, VMFUPDAT	SERVICE, VMFUPDAT	
<i>sysid</i> SYSMEMO	SERVICE, VMFUPDAT	SERVICE, VMFUPDAT	
VM SYSLMOD		LOCALMOD	
\$\$\$VM\$\$\$ SYSLMOD			VMFUPDAT
\$\$\$VM\$\$\$ SYSMEMO			VMFUPDAT
VM SYSPINV	PUT2PROD, SERVICE, VMFBLD		
<i>sysid</i> SYSRECS	VMFINS BUILD, VMFINS DELETE, VMFINS MIGRATE, VMFSUFTB	VMFINS DELETE, VMFINS INSTALL, VMFINS MIGRATE	
<i>sysid</i> SYSREQT	VMFINS BUILD, VMFINS DELETE, VMFINS MIGRATE, VMFSIM SYSDEP, VMFSIM SYSREQ	VMFINS INSTALL, VMFINS MIGRATE, VMFSIM INIT	
<i>sysid</i> SYSREST	SERVICE, VMFSUFIN, VMFUPDAT	SERVICE, VMFSUFIN, VMFUPDAT	
\$\$\$VM\$\$\$ SYSREST			VMFUPDAT
<i>sysid</i> SYSSUF	LOCALMOD, PUT2PROD, SERVICE, VMFBTMAP, VMFSUFIN, VMFSUFTB, VMFUPDAT	SERVICE, PUT2PROD, VMFSUFTB, VMFUPDAT	
\$\$\$VM\$\$\$ SYSSUF			VMFUPDAT
\$LLB\$ TEXT			VMFBLD
\$TLB\$ TEXT			VMFBLD
\$fn TEXT			VMFASM, VMFHASM, VMFHLASM, VMFNLS
<i>filename</i> TEXT		VMFBLD	
<i>fn</i> TEXT		VMFASM, VMFHASM, VMFHLASM, VMFNLS	

<i>Table 34. Input/Output Files (continued)</i>			
<b>File ID</b>	<b>Used as input by:</b>	<b>Provided as output by:</b>	<b>Used as a temporary file by:</b>
<i>libname</i> TEXT		VMFBLD	
<i>objname</i> TIMESTMP			VMFBLD
\$CLB\$ TXTLIB			VMFBLD
\$TLB\$ TXTLIB			VMFBLD
<i>libname</i> TXTLIB		VMFBLD	
HCPLDL TXT <i>modid</i>			GENCPBLS
<i>fn</i> TXT <i>nnnnn</i>		VMFASM, VMFHASM, VMFHLASM, VMFNLS	
<i>fn</i> UPDATES			VMFASM, VMFEXUPD, VMFHASM, VMFHLASM, VMFNLS
<i>objname</i> UPDATES			VMFBLD
<i>fn</i> UPDLOG		VMFEXUPD, VMFNLS	
<i>appid</i> VVT\$PSU\$	VMFPSU		
<i>appid</i> VVT <i>lvld</i>	GENCPBLS, LOCALMOD, VMFAPPLY, VMFASM, VMFBLD, VMFEXUPD, VMFHASM, VMFHLASM, VMFINS BUILD, VMFPSU, VMFQOBJ, VMFREPL, VMFSIM CHKLVL, VMFSIM GETLVL, VMFNLS, VMFREM	GENCPBLS, VMFEXUPD, VMFSIM CHKLVL, VMFNLS, VMFREPL, VMFAPPLY, VMFREM	VMFPSU
<i>fn</i> VVT <i>lvld</i>	VMFSIM LOGMOD	VMFSIM LOGMOD	
RETRY \$APPLIST		VMFAPPLY	
<i>fn</i> \$APPLIST	VMFAPPLY	VMFSIM COMPTBL, VMFSIM SRVDEP, VMFSIM SRVREQ, VMFSIM SYSDEP, VMFSIM SYSREQ	
<i>appid</i> \$APRCVRY	VMFAPPLY, VMFBLD, VMFINS BUILD, VMFMRDSK, VMFREM	VMFAPPLY	
<i>appid</i> \$ASTATS	VMFREM	VMFREM	
<i>recid</i> \$BRREQT	VMFREM	VMFREM	
<i>fn</i> \$CONS	PUT2PROD, SERVICE	PUT2PROD, SERVICE	
<i>fn</i> \$CONSOLD	PUT2PROD, SERVICE	PUT2PROD, SERVICE	
<i>prodid</i> \$CORymdd	VMFREC		
<i>recid</i> \$DSTATS	VMFREM	VMFREM	
<i>fn</i> \$DLCS			VMFNLS

## Input and Output Files

Table 34. Input/Output Files (continued)			
File ID	Used as input by:	Provided as output by:	Used as a temporary file by:
<i>fn</i> \$EXCLIST	VMFAPPLY, VMFPSU		
\$CRDSK\$ \$FILES\$		VMFMRDSK	
\$TLB\$ \$HISTORY			VMFBLD
<i>objname</i> \$HISTORY			VMFBLD
<i>prodid</i> \$INSnnnn	VMFREC		
SETUP \$LINKS	VMFINS BUILD, VMFINS DELETE, VMFINS MIGRATE, VMFSETUP	VMFSETUP	
<i>objname</i> \$MAP		VMFBLD	
<i>appid</i> \$MISSING		VMFAPPLY	
<i>fn</i> \$MSGLOG	VMFVIEW		
\$CSMAGT \$MSGLOG		CSMAGENT	
\$CSMCMG \$MSGLOG		SERVMGR	
\$CSMCMG \$MSGvrm		SERVMGR	
\$VMFlogid \$MSGLOG		VMFOVER, VMFPSU	
\$VMFAPP \$MSGLOG		VMFAPPLY	
\$VMFBLD \$MSGLOG		VMFBLD	
\$VMFINS \$MSGLOG		VMFINS BUILD, VMFINS DELETE, VMFINS DISABLE, VMFINS ENABLE, VMFINS INSTALL, VMFINS MIGRATE, VMFREC	
\$VMFLMD \$MSGLOG	LOCALMOD	LOCALMOD	
\$VMFP2P \$MSGLOG		PUT2PROD	
\$VMFREC \$MSGLOG		VMFREC	
\$VMFRM \$MSGLOG		VMFREM	
\$VMFSRV \$MSGLOG		SERVICE	
VMFRMT \$NEWCP\$			VMFINS DELETE, VMFINS INSTALL, VMFINS MIGRATE
<i>bdl</i> list \$NUCEXEC		VMFBLD	
VMSES \$PARTCAT	VMFMRDSK		
\$PPFT\$ \$PPF			VMFOVER

<i>Table 34. Input/Output Files (continued)</i>			
<b>File ID</b>	<b>Used as input by:</b>	<b>Provided as output by:</b>	<b>Used as a temporary file by:</b>
<i>ppfname</i> \$PPF	PUT2PROD, SERVICE, VMFBTMAP, VMFINS BUILD, VMFINS DELETE, VMFINS DISABLE, VMFINS ENABLE, VMFINS MIGRATE, VMFOVER, VMFPPF		
<i>ppfname1</i> \$PPF	VMFINS INSTALL		
<i>ppfname2</i> \$PPF		VMFINS INSTALL	
<i>prodid</i> \$PPF	VMFINS BUILD		
<i>ppfname</i> \$PPFTEMP		VMFOVER	VMFINS DELETE, VMFPPF
SERVICE \$PRODS	PUT2PROD, SERVICE	PUT2PROD, SERVICE, VMFBLD	
<i>systemid</i> \$PRODS	PUT2PROD	PUT2PROD, SERVICE	
<i>fn</i> \$PTFPART	VMFSIM INIT, VMFAPPLY, VMFREM		
<i>prodid</i> \$PUTnnnn	VMFREC		
LOCALMOD \$RESTART	LOCALMOD	LOCALMOD	
SERVICE \$RESTART	SERVICE, VMFUPDAT	SERVICE, VMFUPDAT	
<i>appid</i> \$RMRCVRY	VMFAPPLY, VMFBLD, VMFREC, VMFMRDSK, VMFREM, VMFINS BUILD	VMFREM	
VMSBR \$SELECT	VMFBLD	VMFBLD	
<i>appid</i> \$SELECT	VMFAPPLY, VMFBLD, VMFINS BUILD, VMFREM	GENCPBLS, VMFEXUPD, VMFSGMAP, VMFNLS, VMFREPL, VMFREM	
<i>xxx</i> \$PSU\$ \$SELECT		VMFPSU	
<i>appid</i> \$SNAPPS	VMFREM	VMFREM	
<i>recid</i> \$SNDESCT	VMFREM	VMFREM	
<i>recid</i> \$SNRECS	VMFREM	VMFREM	
<i>recid</i> \$SNREQT	VMFREM	VMFREM	
<i>recid</i> \$SRDESCT	VMFREM	VMFREM	
<i>recid</i> \$SRRECS	VMFREM	VMFREM	
<i>recid</i> \$SRREQT	VMFREM	VMFREM	
<i>appid</i> \$SRVAPPS	VMFAPPLY, VMFREM	VMFAPPLY, VMFREM	
<i>appid</i> \$STATS	VMFAPPLY	VMFAPPLY	
VMFRMT \$TMPCP\$			VMFINS DELETE, VMFINS INSTALL, VMFINS MIGRATE

## Input and Output Files

<i>Table 34. Input/Output Files (continued)</i>			
<b>File ID</b>	<b>Used as input by:</b>	<b>Provided as output by:</b>	<b>Used as a temporary file by:</b>
CUSSRC \$VMFREST			VMFINS MIGRATE
IBMSRC \$VMFREST			VMFINS MIGRATE
<i>appid \$VNIvlid</i>	VMFREM	VMFREM	
<i>appid \$VVlvid</i>	VMFAPPLY, VMFREM	VMFAPPLY, VMFREM	
<i>appid \$\$SELECT</i>	VMFAPPLY, VMFREM	VMFAPPLY, VMFREM	
VMFUPDAT \$\$\$UP\$\$\$			VMFUPDAT
<i>\$fn \$ft</i>		VMFEXUPD	VMFEXUPD
<i>fn \$ft</i>	VMFEXUPD		
<i>fn \$langid</i>			VMFNLS
<i>prodid \$\$EXEC\$\$</i>		VMFINS DELETE, VMFINS DISABLE, VMFINS ENABLE, VMFINS INSTALL, VMFINS MIGRATE	
<i>apptablefn apptableft</i>	VMFSIM SRVDEP, VMFSIM SRVREQ, VMFSIM SYSDEP, VMFSIM SYSREQ		
<i>objname cntrlfn</i>		VMFBLD	
<i>fn ctlfile</i>		VMFNLS	
<i>fn ft fm</i>	VMFCOPY, VMFERASE		
<i>fn2 ft2 fm2</i>		VMFCOPY	
<i>fn listft</i>	VMFSIM INIT		
<i>listfn listft</i>	VMFBLD		
VMFINFO <i>mmdhhtt</i>		VMFINFO	
INS <i>nnnn</i>	VMFREC		
PUT <i>nnnn</i>	VMFREC		
<i>partname objtype</i>			VMFBLD
<i>fn out_ft</i>		VMFEXUPD, VMFREPL	
<i>reqtablefn reqtableft</i>	VMFSIM SRVDEP, VMFSIM SRVREQ, VMFSIM SYSDEP, VMFSIM SYSREQ		
<i>table1fn table1ft</i>	VMFSIM COMPTBL		
<i>table2fn table2ft</i>	VMFSIM COMPTBL		
<i>tablefn tableft</i>	VMFSIM MODIFY, VMFSIM QUERY	VMFSIM MODIFY	
HCPLDL <i>updtft</i>		GENCPBLS	

<i>Table 34. Input/Output Files (continued)</i>			
<b>File ID</b>	<b>Used as input by:</b>	<b>Provided as output by:</b>	<b>Used as a temporary file by:</b>
<i>fn updtft</i>	VMFASM, VMFEXUPD, VMFHASM, VMFHLASM, VMFNLS		
<i>fn xxxnnnnn</i>		VMFASM, VMFEXUPD, VMFHASM, VMFHLASM, VMFNLS, VMFREPL	
<i>COR ymdd</i>	VMFREC		
<i>fn {EXEC EXCnnnnn}</i>	VMFQOBJ		
<i>fn {OBJDATA ft}</i>	VMFQOBJ	VMFQOBJ	





---

## Appendix C. VMSES/E Sample Files

This topic contains information about sample files that are supplied for use with VMSES/E.

### **LEVELCHK SAMPEXEC**

This sample can be used to perform a comparison evaluation of a given z/VM CSM service level, and the analogous information that is created for a z/VM CSM managed system (or an intended such system), or, a pairing of two of the same such entities.

The LEVELCHK SAMPEXEC part, which can be found on the MAINTCSM 5E6 disk, can be used to create an executable LEVELCHK EXEC. For information about using the LEVELCHK sample, see the comments in the LEVELCHK SAMPEXEC file.



## Appendix D. Module Identifiers for VMSES/E Messages

Messages issued by VMSES/E are in the following format:

`VMFmmmnnnnx`

where:

- VMF is the 3-character component identifier for VMSES/E.
- *mmm* is the 3-character identifier for the VMSES/E module that originated the message. [Table 35 on page 749](#) lists these identifiers in alphabetical order.
- *nnnn* is the 4-character message number.
- *x* is the 1-character severity code.

For a complete explanation of any VMSES/E message, use the HELP Facility to view the message explanation online or see the appropriate message documentation.

*Table 35. Message IDs*

Message ID ( <i>mmm</i> )	Originating Module	Message Format
APP	VMFAPPLY EXEC	VMFAPP <i>nnnnx</i>
ASM	VMFASM EXEC	VMFASM <i>nnnnx</i>
BDC	VMFBDCOM EXEC	VMFBDC <i>nnnnx</i>
BDP	VMFBDCPY EXEC	VMFBDP <i>nnnnx</i>
BDM	VMFBDMOD EXEC	VMFBDM <i>nnnnx</i>
BDN	VMFBNUC EXEC	VMFBDN <i>nnnnx</i>
BDS	VMFBDSSEG EXEC	VMFBDS <i>nnnnx</i>
BFS	VMFBDBFS EXEC	VMFBFS <i>nnnnx</i>
BLD	VMFBLD EXEC	VMFBLD <i>nnnnx</i>
BMP	VMFBTMAP EXEC	VMFBMP <i>nnnnx</i>
BRW	VMFBRWSE XEDIT	VMFBRW <i>nnnnx</i>
CAG	CSMAGENT EXEC	VMFCAG <i>nnnnx</i>
CHK	\$VMFCHK\$ XEDIT	VMFCHK <i>nnnnx</i>
CLB	VMFBDCLB EXEC	VMFCLB <i>nnnnx</i>
CMG	SERVMGR EXEC	VMFCMG <i>nnnnx</i>
CNV	VMFCNVT EXEC	VMFCNV <i>nnnnx</i>
COP	VMFCOPY EXEC	VMFCOP <i>nnnnx</i>
CPB	CSMPKGBD EXEC	VMFCPB <i>nnnnx</i>
CPL	CSMPKGLD EXEC	VMFCPL <i>nnnnx</i>
CPR	CSMP2PRD EXEC	VMFCPR <i>nnnnx</i>
CSQ	CSMQUERY EXEC	VMFCSQ <i>nnnnx</i>

Table 35. Message IDs (continued)

Message ID (mmm)	Originating Module	Message Format
CSV	CSMSRVMT EXEC	VMFCSVnnnnx
CSY	CSMSYSMT EXEC	VMFCSYnnnnx
CTL	\$CSMUTL\$ EXEC	VMFCTLnnnnx
CTR	CSMTRNSP EXEC	VMFCTRnnnnx
CWS	VMFCWSRC EXEC	VMFCWSnnnnx
DDR	VMFBDDDR EXEC	VMFDDRnnnnx
DEF	VMFDELF EXEC	VMFDEFnnnnx
DEL	VMFDEL EXEC	VMFDELnnnnx
DEP	VMFDEP EXEC	VMFDEPnnnnx
DLB	VMFBDDLb EXEC	VMFDLBnnnnx
E2E	VMFE2E MODULE	VMFE2Ennnnx
ENR	VMFENRPT EXEC	VMFENRnnnnx
ERA	VMFERASE EXEC	VMFERAnnnnx
EXU	VMFEXUPD EXEC	VMFEXUnnnnx
GCB	GENCPBLS EXEC	VMFGCBnnnnx
GEN	VMFBDGEN EXEC	VMFGENnnnnx
GPA	VMFGPARM EXEC	VMFGPAnnnnx
HAH	VMFHASHM MODULE	VMFHAHnnnnx
HAM	VMFHASM EXEC	VMFHAMnnnnx
HAS	VMFHASH EXEC	VMFHASnnnnx
INF	VMFINFO EXEC	VMFINFnnnnx
INM	VMFINMI EXEC	VMFINMnnnnx
INS	VMFINS EXEC	VMFINSnnnnx
INT	VMFINST EXEC	VMFINTnnnnx
LDP	VMFLDPRD EXEC	VMFLDPnnnnx
LDR	\$\$LDR\$\$ XEDIT	VMFLDRnnnnx
LDS	VMFLDS MODULE	VMFLDSnnnnx
LLB	VMFBDLLB EXEC	VMFLLBnnnnx
LMD	LOCALMOD EXEC	VMFLMDnnnnx
MEM	VMFMEMO EXEC	VMFMEMnnnnx
MHR	VMFMHR EXEC	VMFMHRnnnnx
MIG	VMFMIG EXEC	VMFMIGnnnnx
MKO	VMFMKOVr XEDIT	VMFMKOnnnnx
MLB	VMFBDMLB EXEC	VMFMLBnnnnx

Table 35. Message IDs (continued)

Message ID (mmm)	Originating Module	Message Format
MLO	VMFMLOAD EXEC	VMFMLOnnnnx
MRD	VMFMRDSK EXEC	VMFMRDnnnnx
MSG	VMFMSG EXEC	VMFMSGnnnnx
NLS	VMFNLS EXEC	VMFNLSnnnnx
OVE	VMFOVER EXEC	VMFOVEnnnnx
P2P	PUT2PROD EXEC	VMFP2Pnnnnx
PAT	\$VMFPAT\$ EXEC	VMFPATnnnnx
PLA	VMFPLAN EXEC	VMFPLAnnnnx
PMD	VMFBDPMD EXEC	VMFPMDnnnnx
PPF	VMFPPF EXEC	VMFPPFnnnnx
PRD	PRODU TL EXEC	VMFPRDnnnnx
PSU	VMFPSU EXEC	VMFPSUnnnnx
QMD	VMFQMDA EXEC	VMFQMDnnnnx
QOB	VMFQOBJ EXEC	VMFQOBnnnnx
RCA	VMFRCALL EXEC	VMFRCAnnnnx
RCX	VMFR CAXL EXEC	VMFR CXnnnnx
RCC	VMFRCCOM EXEC	VMFRCCnnnnx
RCP	VMFR CPTF EXEC	VMFR CPnnnnx
RDT	VMFRDTBL MODULE	VMFRDTnnnnx
REC	VMFREC EXEC	VMFRE Cnnnnx
REM	VMFREM EXEC	VMFRE Mnnnnx
REO	VMFRECON EXEC	VMFRE Onnnnx
REP	VMFREPL EXEC	VMFRE Pnnnnx
REQ	VMFREQC EXEC	VMFRE Qnnnnx
RES	VMFREST EXEC	VMFRE Snnnnx
RFL	VMFRFL EXEC	VMFRFLnnnnx
RMC	VMFRMTC EXEC	VMFRMCnnnnx
RMP	VMFRMTP EXEC	VMFRMPnnnnx
RMT	VMFRMT EXEC	VMFRMTnnnnx
RWL	VMFRWLIB EXEC	VMFRWLnnnnx
SAV	VMFSAVE EXEC	VMFSAVnnnnx
SBR	VMFBDSBR EXEC	VMFSBRnnnnx
SDM	\$VMFMSG\$ XEDIT	VMFSDMnnnnx
SET	VMFSETUP EXEC	VMFSETnnnnx

Table 35. Message IDs (continued)

Message ID (mmm)	Originating Module	Message Format
SGP	\$VMFSEG\$ EXEC	VMFSGPnnnnx
SGM	VMFSGMAP EXEC	VMFSGMnnnnx
SIM	VMFSIM EXEC	VMFSIMnnnnx
SIP	VMFSIMPC EXEC	VMFSIPnnnnx
SPC	VMFSPTCC XEDIT	VMFSPCnnnnx
SPL	VMFSPLIT XEDIT	VMFSPLnnnnx
SPP	VMFSPLTP XEDIT	VMFSPPnnnnx
SPT	VMFSPLTC XEDIT	VMFSPTnnnnx
SRV	SERVICE EXEC	VMFSRVnnnnx
SUI	VMFSUFIN EXEC	VMFSUInnnnx
SUT	VMFSUFTB EXEC	VMFSUTnnnnx
TBD	VMFTBDEF EXEC	VMFTBDnnnnx
TLB	VMFBDTLB EXEC	VMFTLBnnnnx
UPD	VMFUPDAT EXEC	VMFUPDnnnnx
UTL	\$VMFUTL\$ EXEC	VMFUTLnnnnx
VIE	VMFVIEW EXEC	VMFVIEnnnnx
VW1	\$VMFVW1\$ XEDIT	VMFVW1nnnnx
VW2	\$VMFVW2\$ XEDIT	VMFVW2nnnnx

## Appendix E. Tape Formats Supported by VMSES/E

The VMSES/E VMFINS EXEC can process product tapes in any of these formats:

- VMSES/E product tape format
- z/VM System Delivery Offering (SDO) format
- VMSES/E service tape format

Tapes in any of these formats may have one or more physical volumes.

### VMSES/E Product Media Format

This section describes the format of media containing products in VMSES/E format.

Each volume of a product tape in VMSES/E format contains the following files, in the following order:

1. VMFREC required tape file 1. This file contains:
  - a. A multi-volume directory, INS *nnnn*
  - b. (On the first volume only) Special installation tools, if any, for the product
  - c. (On the first volume only) \$PPF and PPF files for all components on the tape
  - d. (On the first volume only) PRODPART files for all components on the tape
  - e. *prodid* or *compname* MEMO file
2. VMFREC required tape file 2. This file contains:
  - a. *Ovrnmns* files for each component on this tape volume. *Ovrnmns* is the file type associated with a product identifier. The variables mean:
    - o** is constant.
    - v** is the version number of the *prodid*.
    - r** is the release number of the *prodid*.
    - m** is the modification level of the *prodid*.
    - nn** is the number of tape files in the product (delimited by tape marks). The number of tape files (*nn*) includes one of the tape header files, all of the product header files, and all of the product tape files.
    - s** is a Boolean flag that indicates the *prodid* is supported by VMSES/E with a product parameter file.
  - b. \$PPF files for each component on this tape volume
  - c. PRODPART files for each component on this tape volume
  - d. *prodid* or *compname* MEMO file
3. VMFREC required tape file for the first component. This file contains:
  - a. *Ovrnmns* files for the first component
  - b. A product content directory for the first component
4. *prodid* or *compname* MEMO file
5. The first code file for the first component :

And as many more files as necessary. Each new component begins with a VMFREC required tape file and a MEMO file. If a component is split between two or more volumes, there is a VMFREC required tape file and a MEMO file for the component on each volume.

## z/VM System Delivery Offering Format

---

Product tapes in z/VM System Delivery Offering format are created by combining product tapes for several products. Depending on the way the tapes are combined, these tapes are also called merged product tapes or stacked product tapes.

The first volume of a stacked product tape in z/VM SDO format contains the following files, in the following order:

1. A combined "first tape file" for all products on the tape. This file contains:
  - a. The installation exec for each product
  - b. \$PPF files for each product (if they exist)
  - c. PRODPART files for each product (if they exist)
  - d. Post-processing execs for each product (if they exist)
  - e. A "zero file" for each product, containing the number of tape files for the product

The first tape file serves as a table of contents for the tape.

2. *prodid* or *compname* MEMOs for each product on the tape
3. The zero file, post-processing execs, and either the \$PPF and PRODPART files or an installation exec for the first product
4. The first code file for the first product :

And as many more files as necessary. Each new product begins with a tape file containing the zero file, post-processing execs, and either the \$PPF and PRODPART files or an installation exec.

Subsequent volumes of a stacked product tape do not contain a first tape file or the collected MEMO files. They simply continue the first volume.

A merged product tape in z/VM SDO format looks like the second volume of a stacked product tape. It contains:

1. The zero file, post-processing execs, and either the \$PPF and PRODPART files or an installation exec for the first product
2. The first code file for the first product :

And as many more files as necessary. Each new product begins with a tape file containing the zero file, post-processing execs, and either the \$PPF and PRODPART files or an installation exec.

## VMSES/E Service Tape Formats

---

The RSU (recommended service upgrade) and the corrective service tape use similar formats:

- The first tape file on the service tape contains these files:
  - Tape descriptor file, which is described in [“The Tape Descriptor File” on page 125](#).
  - Service tape document, which is described in [“The Tape Document” on page 125](#).
- **Note:** A file called \$LEVEL MAP may exist in tape file 1. This file exists for compatibility with previous versions of the service tools.
- The second tape file on the service tape contains files related to each product. This includes the program level file, described in [“The Program Level File” on page 127](#), and the Memo-to-Users, described in [“The Memo-to-Users” on page 127](#).
- The first tape file for each product contains a program level file and the product contents directory (described in [“The Product Contents Directory” on page 127](#)). If the product or component does not



support using a product parameter file, the first tape file for the product or component contains a program level file and a product service exec.

- The rest of the tape files for each product on the service tape contain the service for each product.



---

## Appendix F. Servicing Non-VMSES/E SNA Products

This topic contains information about how to service the Systems Network Architecture (SNA) products that are not in VMSES/E format. These products are object maintained and serviced by replacing TEXT files.

**Do not use this topic to service any products supported by VMSES/E.** Although many terms used in this topic are similar or even identical to terms used in the VM/ESA service procedures, the procedures are not compatible. For example, a product parameter file discussed in this topic does not follow the same format as the VM/ESA product parameter file discussed in the rest of this book. The procedures described in this topic must stand apart from service procedures described in *z/VM: Service Guide*.

The service programs used in this topic are:

### **VMFMERGE**

applies PTFs from the DELTA disk to the MERGE disk. For more information about this program, see “VMFMERGE EXEC” on page 774.

### **VMFREMOV**

removes PTFs applied by the VMFMERGE EXEC procedure. For more information about this program, see “VMFREMOV EXEC” on page 777.

### **VMFZAP**

applies ZAPs and maintains a record of them in the ZAP log. For more information about this program, see “VMFZAP EXEC” on page 779.

Before you try to do any service processing using these execs, you must consider the following to determine the amount of virtual storage you need to define for your virtual machine.

- The number of minidisks accessed and the number of files on each minidisk accessed.
- The size of files (merge log, reqby log, ZAP log) needing to be updated, and other files used during processing.
- The number of execs loaded into virtual storage. You can find this out by using the CMS command EXECMAP. See *z/VM: Service Guide* for more information about this command.
- The number of nucleus extensions loaded into virtual storage. You can find this out by using the CMS command NUCXMAP. See *z/VM: Service Guide* for more information about this command.

A description of the types of disks, the files, logs, and lists used during service processing follows. In addition, there are generic descriptions of how to do different types of object code service.

---

## Types of Disks

VMFMERGE, VMFREMOV, and VMFZAP use several types of disks during processing. Although we refer to each of these as a single disk, there may be multiple occurrences of each type. The execs access these disks using mode letters E-N. When processing ends, the execs restore your disk hierarchy to its original condition.

The five types of disks are:

<b>Disk</b>	<b>Contents</b>
<b>BASE</b>	Original product files as shipped on the product installation media.
<b>DELTA</b>	Changed portions of the product. These files have different names from the corresponding files that are found on the BASE disk. There is also a service control file (containing all the information needed to install a program temporary fix (PTF)) for each change on this disk. There is an exclude list, at least one apply list, and possibly a remove list.

Disk	Contents
<b>MERGE</b>	Changes that have been applied. These files have been copied from the DELTA disk and renamed to match the corresponding file on the BASE disk. There is also a log containing changes (merge log) which have been merged into the product and a log containing the requisite relationships (reqby log) of the merged or superseded changes.
<b>ZAP</b>	TEXT files that have been copied from the BASE or MERGE disk and ZAPped. There is a log containing ZAPs (ZAP log) that have been applied to the product.
<b>RUN</b>	Actual working version of the product. The files on this disk are created by the product build exec from the files on the BASE, MERGE, and ZAP disks. There is also a log containing service (service log) of PTFs and ZAPs that have been applied to the product.

## Service Control File

The service control file (SCF) describes a Program Temporary Fix (PTF). There is an SCF for each PTF. SCFs are built by the change team and shipped with the PTFs in the delta file of the Program Update Tape (PUT). The file name of the SCF is the PTF number and the file type must be SCF.

The service control file contains all the information needed to install a PTF. The following is an example of a service control file.

```
:ptf.UV00006
:prodid.5664175
:prereq.UV00005 UV00007
:coreq.UV00056.5748RC2
:sup.Z00002 Z00003
:changes.
:element.DSIMNT TEXT
:replace.TXTP0006
:element.DSIXXX TEXT
:replace.TXTP0004
:element.DSIYYY TEXT
:replace.TXTP0001
:echanges.
:apartext.PP00004 - Split DSIXXX for new base register
:apartext.PP00009 - Update loadlist to add DSINEW
```

Figure 219. Example of a Service Control File (UV00006 SCF)

Where:

**:ptf.**

is the PTF number. This number is the same as the file name of the SCF.

**:prodid.**

is the seven character product identifier. (You may also specify a one character suffix for the release or level of the product. This suffix is determined by the service group).

**:prereq.**

is the SCF file name of all PTFs that must be merged before you can merge the PTF specified on the *:ptf.* tag. During processing, if the prerequisite cannot be found, then processing ends.

If a prerequisite change is not for this product, the associated *prodid* must be specified as part of the PTF name on *:prereq.* tag. For example, *ppppppp.prodid*, where *ppppppp* is the PTF number within another product. A message is displayed by VMFMERGE telling you there is a prerequisite PTF for another product, specifically the product associated with the *prodid* listed on the *:prereq.* tag.

You can omit PRODID if the products are the same as for this PTF.

**:coreq.**

is the SCF file name of all PTFs that need to be merged together with this PTF.

If a corequisite change is not for this product, the same rules apply as with *:prereq* for specifying *prodid*. A message is displayed by VMFMERGE and VMFREMOV telling you there is a corequisite PTF

for another product, specifically the product associated with the *prodid* listed on the *:coreq.* tag. As shown in the example above, the corequisite UV00056 is for product 5748RC2.

The maximum number of characters (including blanks) you can specify on a *:coreq.* tag is 256.

**:sup.**

are the PTFs or ZAPs that are no longer needed as a result of this PTF. Specifying this tag prevents superseded PTFs or ZAPs from being reapplied.

**:changes.**

indicates the beginning of the list of elements changed. The change list is a table with an entry for each element affected. Each element begins with an *:element* tag and ends with the next *:element* tag.

**:element.**

specifies the CMS file name and file type of an element defined or replaced by this PTF. A separate tag is coded for every element changed.

**:replace.**

specifies the CMS file type of an object replacement file for the associated element.

**:echanges.**

indicates the ending of the list of elements changed.

**:apartext.**

is an APAR number followed by a description of the problem reported by the APAR. The service group specifies this information.

## Product Parameter File

---

The product parameter file contains records that identify the various product minidisk addresses used for installation and service. A parameter file containing the default minidisk addresses is shipped with the product on the product installation media. (You can change or add to the default addresses.) The VMFMERGE, VMFREMOV, VMFZAP, and product execs read this file for the information needed to access minidisks. Before you use these execs, you must access the minidisk containing the product parameter file.

The file name of the file is the product identifier, and the file type must be VMFPARM.

Each record in the file contains a keyword indicating the type of disk and one or more virtual addresses. The format of the records is a keyword followed by one or more values. VMFMERGE, VMFREMOV, and VMFZAP recognize the following keywords.

**BASE**

the virtual address(es) of the minidisk(s) containing the base product code.

**DELTA**

the virtual address(es) of the DELTA Disk(s).

**MERGE**

the virtual address(es) of the MERGE Disk(s).

**ZAP**

the virtual address(es) of the ZAP Disk(s).

**RUN**

the disk for executable code.

**Note:** VMFMERGE, VMFREMOV, and VMFZAP ignore any records beginning with an unrecognized keyword. This allows products to define other keywords that product execs can use.

The following is an example of a product parameter file.

```
Base 250
Delta 251
Merge 252
ZAP 253
Run 254
```

Figure 220. Example of a Product Parameter File

**Note:** You can specify more than one address on these disks. This may be valuable if you want to keep a record of the disks which have different levels of your system which you created from different MERGE disks or DELTA disks. The maximum number of disks you can access is ten.

During service processing, the execs read the product parameter file and access the disks that are needed. For example, VMFMERGE and VMFREMOV access the DELTA and MERGE disks; VMFZAP accesses the BASE, MERGE, and ZAP disks.

## How VMFMERGE, VMFREMOV, and VMFZAP Use the PPF

- VMFMERGE, VMFREMOV, and VMFZAP get the minidisk information from the parameter file and access the needed disks using file modes E-N. Once these execs stop processing, the disk hierarchy is restored.
- VMFMERGE and VMFREMOV only access the first virtual address following the MERGE keyword. However, these execs access multiple DELTA disks in the order in which they are listed in the parameter file. Input files are taken from the first minidisk where they are found; output files are directed to the first single accessed MERGE disk.
- VMFZAP only accesses the first virtual address following the ZAP keyword. However, this exec accesses multiple MERGE and BASE disks in the order in which they are listed in the parameter file (the MERGE disks come before the BASE disks). Input files are taken from the first minidisk where they are found; output files are directed to the first single accessed ZAP disk.

In order to use VMFZAP, you **must** have an A disk accessed Read-Write. This disk **must not** be the ZAP disk, MERGE disk, or BASE disk. That is, the virtual address of your A disk **must not** appear on the ZAP, MERGE, or Base records of your VMFPARM file.

## Merge Log

---

A merge log is a file maintained by the VMFMERGE and VMFREMOV EXECs. This file is a log of changes (PTFs) which have been merged or superseded. A merge log exists for each product and is shipped on all Program Update Tapes (PUTs) and on the product tape. Note the file name of the log is the product identifier, and the file type must be VMFMGLOG.

VMFMERGE and VMFREMOV update the merge log on the disk where it currently exists. (The merge log must be on the MERGE disk.) VMFMERGE and VMFREMOV stop processing if they cannot find a merge log.

The merge log has an entry for each PTF indicating the PTF number and its status. Status can be:

### Merged

means the change is included in the code.

### Superseded

means the change is no longer needed because some later fix has replaced it.

In addition, each entry has information on the date and time the change was processed, and a list of the elements (TEXTs, execs, and so forth) the change affects.

Every time a change is processed, a new entry for that change is added to the merge log. This is referred to as the "history" of the change. The merge log is a record of the history of PTFs applied to and removed from a product. To find out the current status of a change, you need to read your merge log file starting at the bottom. Find the PTF in question. The first noncommented entry for that PTF indicates the status of the change.

For VMFREMOV, the merged entry is commented out and another entry is added to the end of the merge log indicating the PTF has been removed.

**Note:** If you change the merge log in any way, either of the following could happen:

- A change may not be merged.
- A change may be merged but not removed.

Figure 221 on page 761 shows an example of a merge log produced by VMFMERGE<sup>1</sup> and updated by VMFREMOV.

```

:entry.UV00001 Merged      12/13/21 13:09:42 DSILGN TEXT DSISRP TEXT
:entry.UV00005 Merged      12/15/21 06:28:17 DSIXSD TEXT DSIZVSIN TEXT
                        DSIREP TEXT DSIMCB TEXT DSITVB COPY
                        DSIMLG TEXT

:entry.UV00008 Merged      12/15/21 06:28:17 DSILMODE EXEC
*entry.UV00007 Merged      12/24/22 11:28:17 DSILMODE EXEC
:entry.UV00006 Merged      01/12/22 17:56:39 DSIXXX TEXT DSIMNT TEXT
:entry.Z00002 Superseded  01/12/22 17:56:39 By UV00006
:entry.Z00003 Superseded  01/12/22 17:56:39 By UV00006
*entry.UV00007 Removed    2/24/22 11:28:17 DSILMODE EXEC

```

Figure 221. Example of a Merge Log Produced by VMFMERGE

**Note:**

1. Comment records may be included in the merge log file.
2. Each comment record must begin with an asterisk (\*) in column 1.
3. You cannot put a comment record in the middle of an entry which spans more than one line in the file. (You should put all comment records at the beginning of the merge log file.)
4. The VMFREMOV command may also insert comment records in the merge log.
5. Blank lines are allowed between entries in the file, and are ignored.

## ZAP Log

The ZAP log is a file maintained by the VMFZAP EXEC. This file is a log of applied ZAPs. A ZAP log exists for each product and is built and maintained by the VMFZAP EXEC. Each entry has information on the date and time the change was processed, and a list of the elements (TEXT files) the change affects. Note the file name of the log is the product identifier and the file type must be VMFZPLOG.

The ZAP log contains information about TEXT files currently affected by ZAPs applied to a given product. This information includes the file name(s) of the ZAP control files(s), the time and date the ZAP was applied, and the modules affected by the ZAP. To remove ZAPs which may be superseded by service you applied, VMFZAP first uses this list of affected modules to erase the files which were previously ZAPped. The currently wanted ZAPs, which are not superseded by service you applied, are then applied and VMFZAP creates a new version of the ZAP log. VMFZAP erases the old ZAP before it writes the new one.

The ZAP log is on the ZAP disk. New ZAP logs are always created on the ZAP disk. You must not move the ZAP log from the ZAP disk.

The ZAP log has an entry for each ZAP indicating the ZAP number and its status. Status can be:

**Zapped**

means the change is included in the code.

An example of a ZAP log follows.

```

* ZAP log for Product 5664175
:entry.Z00001 Zapped      12/31/22 12:34:56 DSINME TEXT
:entry.Z00004 Zapped      12/31/22 12:34:56 DSIMNT TEXT
:entry.Z00005 Zapped      12/31/22 12:34:56 DSIMNT TEXT

```

Figure 222. Example of a ZAP Log (5664175 VMFZPLOG)

**Note:** Comment records may be included in the ZAP log. Such records contain an asterisk (\*) in column 1. Also, blank lines are allowed anywhere in the file and are ignored.

<sup>1</sup> The merge log loaded from the PUT tape has a different time and date stamp.

## Reqby Log

---

A 'required by' log (reqby log) is a file maintained by the VMFMERGE and VMFREMOV EXECs. The reqby log resides on the MERGE disk. This file is a log of all dependent PTFs of each PTF which has a merged or superseded entry in the merge log. A dependent PTF is one which either has a given PTF as a prerequisite or corequisite. A reqby log exists for each product and is shipped on all Program Updates Tapes (PUTs) and on the product tape. Note the file name of the log is the product identifier and the file type must be VMFREQBY.

The reqby log is not required when you issue VMFMERGE or VMFREMOV. If the log does not exist, VMFMERGE or VMFREMOV automatically creates one using the information in the existing merge log and all service control files.

The reqby log contains two types of entries: a comment entry and dependent entries.

A comment entry contains an asterisk (\*) in column 1. Only the first non-blank line in the reqby log should be a comment line. VMFMERGE and VMFREMOV ignore all other comments and eliminate the comments whenever the reqby log is processed. If the first non-blank line is not a comment, one is automatically created the next time VMFMERGE or VMFREMOV processes the reqby log.

The dependent entries are identified by an `:entry` tag followed by a particular PTF and its dependents. Dependent entries can overflow to the next line (or lines) whenever there are more dependent changes associated with the PTF than fit on one line. You can have as many overflow lines as needed to list all the dependent changes. Blank lines are allowed anywhere in the file and are ignored.

**Note:** If you change the reqby log in any way, either of the following could happen:

- A change may not be merged.
- A change may be merged but not removed.

Figure 223 on page 762 shows an example of a reqby log.

```
* 5664167 VMFREQBY
:entry.ptf1 pft2 ptf3
:entry.ptf2 ptf1
```

Figure 223. Example of a Reqby Log (5664167 VMFREQBY)

In the above example, PTF1 and PTF2 are corequisites; and PTF1 is a prerequisite of PTF3.

## Service Log

---

The service log is the merge log with the ZAP log appended on the end. It contains information about service that has been applied to your product. A service log exists for each product and is built by the product build execs.<sup>2</sup> The build exec copies the merge log for the product and appends the ZAP log to it. Note the file name of the service log is the product identifier and the file type must be VMFSVLOG.

The format of the service log is a combination of the formats of the merge log (see [Figure 221 on page 761](#)) and the ZAP Log (see [Figure 222 on page 761](#)). By viewing the service log, you can determine whether a PTF or ZAP has been applied to your product.

An example of a service log follows.

---

<sup>2</sup> The service log resides on the "run-time" disk.



```

:entry.UV00001 Merged      12/13/21 13:09:42 DSILGN TEXT DSISRP TEXT
:entry.UV00005 Merged      12/15/21 06:28:17 DSIXSD TEXT DSIZVSIN TEXT
                        DSIREP TEXT DSIMCB TEXT DSITVB COPY
                        DSIMLG TEXT

*:entry.UV00007 Merged      12/15/21 06:28:17 DSILMODE EXEC
:entry.UV00006 Merged      01/12/22 17:56:39 DSIXXX TEXT DSIMNT TEXT
:entry.Z00002 Superseded   01/12/22 17:56:39
:entry.Z00003 Superseded   01/12/22 17:56:39
*:entry.UV00007 Removed     2/25/22 06:28:17 DSILMODE EXEC
* ZAP log for Product 5664175 created by MAINTvrm 12/31/21 12:34:56
:entry.Z00001 Zapped       12/31/21 12:34:56 DSINMME TEXT
:entry.Z00004 Zapped       12/31/21 12:34:56 DSIMNT TEXT
:entry.Z00005 Zapped       12/31/21 12:34:56 DSIMNT TEXT

```

Figure 224. Example of a Service Log (5664175 VMFSVLOG)

**Note:** Comment records may be included in the service log. Such records contain an asterisk (\*) in column 1. Also, blank lines are allowed anywhere in the file and are ignored.

## Apply List

The apply list is a file that lists PTFs to be applied to a product. There is an apply list shipped on a Program Update Tape (PUT). That file contains the names of all PTFs in the Delta file of the PUT. (The file name of that apply list is *prodid*.) The order of the PTFs in the list is not significant. Apply lists must be on a DELTA disk.

You can build and then maintain the file yourself. There is an apply list for each product which has PTFs applied to it. If no PTFs are applied to a product, you do not need to create an apply list.

Any apply lists you maintain should not have the *prodid* as the file name. Each time you load the Delta files from the tape, the *prodid* apply list on the DELTA disk will be replaced with the *prodid* apply list from the tape. If you want to change the supplied apply list, make a copy of the supplied apply list (give it a different file name) and change the "copied" version of the apply list.

The first word on each line in the file is the file name of a PTF. Only one PTF name should be placed on a line, the remaining data on a line is treated as a comment. If the first character on the line is an asterisk (\*), the whole line is treated as a comment. In addition, blank lines are allowed anywhere in the file and are ignored.

An example of an apply list follows. Note you can specify the file name<sup>3</sup>—in this example the *prodid* is used—but the file type must be APPLIST.

```

* apply list for Product 5664175
UV00006 PTF for maintenance program
UV00003
*UV00009

```

Figure 225. Example of an Apply List (5664175 APPLIST)

## Remove List

The remove list is a file that lists the PTFs you want to remove from a product. You must build and maintain the file yourself. Remove lists must be on a DELTA disk.

The first word on each line in the file is the file name of a PTF. Only one PTF name can be placed on a line. The remaining data on a line is treated as a comment. If the first character on the line is an asterisk (\*) the whole line is treated as a comment. Blank lines are allowed anywhere in the file and are ignored.

An example of a remove list follows. Note you can specify the file name—in this example the *prodid* is used—but the file type must be REMLIST.

<sup>3</sup> The exception is you cannot have a file name of EXCLUDE.

```

* remove listfor Product 5664175
UV00001 this is a comment for ptf1
UV00004

*UV00005 this is a comment for ptf5

```

Figure 226. Example of a Remove List (5664175 REMLIST)

## Exclude List

An exclude list is a file listing PTFs to be excluded from a product.

There is an exclude list shipped as part of the Delta File on the Program Update Tape (PUT). This list contains the names of PTFs known to be in error. (The file name of that exclude list is *prodid*.) The VMFMERGE EXEC uses the combination of the exclude list you have and the *prodid* EXCLIST to obtain the names of the PTFs to exclude from the product. Exclude lists must be on a DELTA disk.

You can build and maintain the file yourself. Any exclude lists you maintain should not have the *prodid* as the file name. Each time you load the Delta files from the tape, the *prodid* exclude list on the DELTA disk will be replaced with the *prodid* exclude list from the tape. If you want to change the supplied exclude list, make a copy of the supplied exclude list (give it a different file name) and change the "copied" version of the exclude list.

The first word on each line in the file is the file name of a PTF. Only one PTF name should be placed on a line, the remaining data on a line is treated as a comment. If the first character on the line is an asterisk (\*) the whole line is treated as a comment. In addition, blank lines are allowed anywhere in the file and are ignored.

An example of an exclude list follows. Note you can specify the file name—in this example the *prodid* is used—but the file type must be EXCLIST.

```

* exclude list for Product 5664175
UV00002 Exclude PTF from CP ACCESS command.
UV00013
*UV00016

```

Figure 227. Example of an Exclude List (5664175 EXCLIST)

## ZAP List

A ZAP list is a file listing ZAPs to be applied to a product. You must build and maintain the file yourself.

The VMFZAP EXEC uses this file during its processing. If no ZAPs are applied, you do not need to create a ZAP List. ZAP lists must be on a disk listed on the BASE, MERGE, or ZAP entry record of the VMFPARM file for the product.

The first word on each line in the file is the file name of a ZAP. Only one ZAP name should be placed on a line; the remaining data on a line is treated as a comment. If the first character on the line is an asterisk (\*), the whole line is treated as a comment. In addition, blank lines are allowed anywhere in the file and are ignored.

If you want to "back off" an unwanted ZAP, comment that ZAP out in the ZAP list by placing an asterisk (\*) in the first column of the line containing the ZAP file name you want to "back off". Then run VMFZAP.

An example of a ZAP list follows. Note the file name must be the *prodid*, and the file type must be ZAPLIST.

```

* Zap List for Product 5664175
Z00001 Zap for CP ACCESS command.
Z00004
Z00005
*Z00016

```

Figure 228. Example of a Zap List (5664175 ZAPLIST)

## Object Code Service Processing

---

There are many ways to apply service to components of your system that are object code maintained. This section describes, in a very generic way, how to:

- Apply
  - Emergency fixes (using ZAPs)
  - Corrective Service
  - Preventive Service
- Merge Service
- Remove Service
- Prevent regression
- Remove a fix-in-error

## Applying Emergency Fixes Using ZAPs

---

When you need an emergency fix, you usually call in the problem and receive a fix over the phone in the form of a ZAP.

Suppose there is a problem with module A. Here is what you need to do.

1. First, call the IBM Support Center and report the problem.
2. You will be given a ZAP (for example, Z00005) to apply to module A.
3. You must create a ZAP control file (Z00005 ZAP).

For more information about valid records in the ZAP control file, see the description of the ZAP command in *z/VM: CMS Commands and Utilities Reference*.

4. You must create a ZAP list if you do not already have one for the product. If you already have one, you must append this new entry.

- **Creating a ZAP list**

Create a new file (on a disk accessed in the VMFPARM file) called *prodid* ZAPLIST. Enter the ZAP name (Z00005) as the first word on the line. (The rest of the information on each line is treated as a comment.)

- **Adding to an existing ZAP list**

On a new line at the bottom of the file, add the ZAP name (Z00005).

Do not delete any ZAP names from the file unless you no longer want to apply them. The VMFZAP EXEC erases all text files that have been previously zapped for a product and then reapplies all ZAPs found in the ZAP list. Thus, you need to make sure the ZAP list you create or add to contains all the ZAPs you want to apply.

See [Figure 228 on page 764](#) for an example of a ZAP list.

5. Run the VMFZAP EXEC using the *prodid* parameter.

VMFZAP erases all previously applied TEXT files from the ZAP disk. Then all the ZAPs listed in the *prodid* ZAPLIST (including Z00005) are applied to the product.

6. Finally, run the product-supplied exec that builds the executable version of the product.

## Applying Corrective Service to Object Code

---

Corrective service is the application of a Program Temporary Fix (PTF) or an IBM Change Team supplied fix to correct a problem.

Corrective fixes can be:

- **Fixes on a PTF tape** provided by Program Support Services (PSS) or by the Change Teams. These fixes are the result of closed valid APARs.
- **ZAPs** provided by the Change Teams over the phone.
- **Relief fixes** provided by the Change Teams on a tape prior to APAR closure to fix severe problems.

Suppose you order PTF UV00007 for your product and that UV00007 affects modules A and B. Here is what you need to do:

1. Backup your existing system and verify the backup copy you have is good.
2. Use VMFPLC2 to load the first physical file on the corrective service tape. (In the example, corrective service tape is shipped by PSS.) The first tape file contains information for you in the cover letter and in the Program Identification Number (PIN) pages.

For information on VMFPLC2, see *z/VM: CMS Commands and Utilities Reference*.

3. Use VMFPLC2 again to load the second physical file on the tape. This tape file contains the following for UV00007:
  - A service control file
  - SCFs for any prerequisites
  - The actual fixes
  - The prerequisites

PTF UV00007 contains fixes for modules A and B, and these fixes are on the tape as CMS files A TXT00007 and B TXT00007.

The fixes and the SCFs must be loaded to the delta disk. The other files on the tape can be loaded to any other disk.

4. Create an apply list named FIX007 APPLIST. (You can specify any file name, but the file type **must** be APPLIST.) You should save these lists so you can use them later when you apply a PUT.

It is in this apply list you specify the fixes you want to merge into the product. List one PTF per line in the file.

**Note:** You must list prerequisite PTFs before the PTFs that need them.

See [Figure 225 on page 763](#) for an example of an apply list.

5. Create an exclude list named FIX007 EXCLIST. (You can specify any file name, but the file type **must** be EXCLIST.)

It is in this exclude list you specify the fixes you want to exclude from the product. List one PTF per line in the file.

**Note:** If you have no fixes you want to have excluded from the product, you **must** create a null exclude list.

See [Figure 227 on page 764](#) for an example of an exclude list.

6. Run the VMFMERGE EXEC with the parameters *prodid* PTFLIST FIX007. VMFMERGE will merge UV00007 into the product.
7. Now run the VMFZAP EXEC for the product. (The latest ZAP list for this product must still be available.) Issuing VMFZAP with the latest ZAP list cleans up superseded ZAPs and prevents regression of the corrective service just applied by old ZAPs.

**Note:** If ZAP Z00005 was superseded by PTF UV00007, the ZAP will not be re-applied. If ZAP Z00005 was not superseded by PTF UV00007, the ZAP will be re-applied.

8. Finally, run the product-supplied service exec (or VMSERV) to build the executable version of the product.

## Applying Preventive Service to Object Code

---

Preventive service is the application of Program Temporary Fixes (PTFs) available on a Program Update Tape (PUT) to avoid known problems.

VMSERV is an exec procedure included on the PUT to help you when you apply service. Suppose you receive a PUT for your product. Here is what you need to do:

1. Use VMFPLC2 to load the first physical file on the preventive service tape. This file contains the PUT DOCUMENT. You should read this document because it contains information about the PUT.

For information on VMFPLC2, see [z/VM: CMS Commands and Utilities Reference](#).

2. Run VMSERV to apply the PUT. You are given the choice of loading the merge file, the delta file, or both. (It is fastest to load just the merge file. However, if you know there is a bad PTF in the merge file you need to load the delta file. Whether you should load the merge file or the delta file depends upon the severity of the bad PTF.) In this case, load the merge file. This gives you a new version of the merge log that includes all the PTFs that are now part of your product. See [Figure 221 on page 761](#) for an example of a merge log.

3. Run VMFMERGE using the apply lists that were saved when corrective fixes were merged.

If you apply any corrective service to your product, you should keep the apply lists you used. When you run VMFMERGE during preventive service, you use these same apply lists. This prevents you from losing any corrective fixes that were already merged, but are not included on the PUT.

For example, assume that FIX007 APPLIST is the only apply list used. You run VMFMERGE with the parameter PFTLIST FIX007. If UV00007 is included in the PUT a message is displayed explaining UV00007 has already been merged. If UV00007 is **not** included in the PUT a message is displayed telling you UV00007 will be merged at this time.

4. Now run VMFZAP EXEC for the product. (You must still have the latest ZAP list for this product available.) Issuing VMFZAP with the latest ZAP list cleans up superseded ZAPs and prevents regression of the corrective service just applied by old ZAPs.

**Note:** If ZAP Z00005 was superseded by PTF UV00007, the ZAP will not be reapplied. If ZAP Z00005 was not superseded by PTF UV00007, the ZAP will be reapplied.

5. Finally, run the product-supplied service exec (or VMSEV) to build the executable version of the product.

## Merge Service

---

### Merging a Single PTF (No Dependents or Supersedes)

To merge a PTF (for example, UV00002) that does not have any corequisite and is not a prerequisite of any other PTF, here's the process:

**Note:** You must access the minidisk containing the **prodid** VMFPARM file before you proceed.

- You must have a merge log and service control file(s). You may also have a user exclude list and a reqby log. Assume you have the merge log, exclude list, and service control files for product 5664167, as shown in [Figure 229 on page 767](#), [Figure 230 on page 767](#), [Figure 231 on page 768](#), and [Figure 232 on page 768](#).

```
:entry.UV00001 Merged 12/13/21 13:09:42 ELEM1 TEXT ELEM2 TEXT
```

Figure 229. Merge Single PTF—Sample Merge Log for 5664167

```
*EXCLUDE list for product 5664167
```

Figure 230. Merge Single PTF—Sample Exclude List for 5664167

```

:ptf.UV00001
:prodid.5664167
:changes.
:element.ELEM1 TEXT
:replace.TXTP0001
:element.ELEM2 TEXT
:replace.TXTP0002
:echanges.
:apartext.ELEM1 service fix
:apartext.ELEM2 service fix

```

Figure 231. Merge Single PTF—Sample SCF for UV00001

```

:ptf.UV00002
:prodid.5664167
:changes.
:element.ELEM3 TEXT
:replace.TXTP0003
:echanges.
:apartext.ELEM3 service fix

```

Figure 232. Merge Single PTF—Sample SCF for UV00002

- To merge UV00002, enter the following command:

```
vmfmerge 5664167 ptf UV00002
```

- VMFMERGE:

1. Reads the merge log and finds UV00002 is not already merged or superseded.
2. Reads the exclude list and finds UV00002 is not excluded either.
3. Reads the service control files and finds ELEM3 TEXT is the element affected by UV00002.  
ELEM3 TEXT's replacement file (TXTP0003) is copied from the delta disk to the merge disk.  
TXTP0003 replaces the element and adds service history to ELEM3 TEXT.
4. Updates the merge log to show UV00002 has been merged.

The resulting merge log looks as follows. (The service control files are not changed.)

```

:entry.UV00001 Merged 12/13/21 13:09:42 ELEM1 TEXT ELEM2 TEXT
:entry.UV00002 Merged 06/23/22 04:18:20 ELEM3 TEXT

```

Figure 233. Merge Single PTF—Changed merge log for 5664167

## Merging Multiple PTFs (with Dependents and Supersedes)

To merge more than one PTF (for example, UV00004 and UV00005), each having a combination of prerequisites, corequisites, and supersedes, here is the process:

- You must have a merge log and service control files. Assume you have the merge log and service control files shown in the following figures.

```

:entry.UV00002 Merged 05/03/22 13:09:42 ELEM3 TEXT

```

Figure 234. Merge Multiple PTFs—Sample Merge Log for 5664167

```

:ptf.UV00001
:prodid.5664167
:coreq.UV00003
:changes.
:element.ELEM1 TEXT
:replace.TXTP0001
:element.ELEM2 TEXT
:replace.TXTP0002
:echanges.
:apartext.ELEM1 service fix
:apartext.ELEM2 service fix

```

Figure 235. Merge Multiple PTFs—Sample SCF for UV00001

```

:ptf.UV00002
:prodid.5664167
:changes.
:element.ELEM3 TEXT
:replace.TXTP0003
:echanges.
:apartext.ELEM3 service fix

```

Figure 236. Merge Multiple PTFs—Sample SCF for UV00002

```

:ptf.UV00003
:prodid.5664167
:coreq.UV00001
:changes.
:element.ELEM4 TEXT
:replace.TXTP0004
:element.ELEM5 TEXT
:replace.TXTP0005
:echanges.
:apartext.ELEM4 service fix
:apartext.ELEM5 service fix

```

Figure 237. Merge Multiple PTFs—Sample SCF for UV00003

```

:ptf.UV00004
:prodid.5664167
:prereq.UV00001
:changes.
:element.ELEM6 TEXT
:replace.TXTP0006
:echanges.
:apartext.ELEM6 service fix

```

Figure 238. Merge Multiple PTFs—Sample SCF for UV00004

```

:ptf.UV00005
:prodid.5664167
:sup.UV00002
:changes.
:element.ELEM3 TEXT
:replace.TXTP003A
:echanges.
:apartext.ELEM3 service fix

```

Figure 239. Merge Multiple PTFs—Sample SCF for UV00005

- To merge UV00004 and UV00005 using one command, you need to create an apply list (see page “ZAP List” on page 764 for more details). Your apply list may look like [Figure 240 on page 769](#).

```

* This is my own apply list
UV00004 put on UV00004
UV00005 put on UV00005

```

Figure 240. MYLIST APPLIST

- After you create the apply list, enter the following command:

```
vmfmerge 5664167 ptflist mylist
```

- VMFMERGE:
  1. Looks for an apply list with a file name of MYLIST.
  2. Reads the merge log and finds UV00004 and UV00005 are not merged or superseded. VMFMERGE also reads the exclude list and finds UV00004 and UV00005 are not excluded either.
  3. Reads the service control files and finds:
    - UV00001 is a prerequisite of UV00004 and must be merged.
    - UV00003 is a corequisite of UV00001 and it too must be merged.
    - UV00002 is superseded by UV00005.

- Determines ELEM1 TEXT, ELEM2 TEXT, ELEM3 TEXT, ELEM4 TEXT, ELEM5 TEXT, and ELEM6 TEXT are the elements these PTFs affect.

VMFMERGE copies these elements' replacement files from the DELTA disk and replaces the appropriate files on the MERGE disk. In addition, service history for these elements is added to the files.

- Updates the merge log to show UV00003, UV00001, UV00004, and UV00005 are merged and UV00002 is superseded.

The resulting merge log is shown in [Figure 241 on page 770](#) and the reqby log is shown in [Figure 242 on page 770](#). (The service control files are not changed.)

```
:entry.UV00002 Merged      05/03/22 09:12:42 ELEM3 TEXT
:entry.UV00003 Merged      09/25/21 12:43:17 ELEM4 TEXT ELEM5 TEXT
:entry.UV00001 Merged      09/25/21 12:43:17 ELEM1 TEXT ELEM2 TEXT
:entry.UV00004 Merged      09/25/22 12:43:17 ELEM6 TEXT
:entry.UV00005 Merged      09/25/22 12:43:17 ELEM3 TEXT
:entry.UV00002 Superseded  09/25/22 12:43:17 By UV00005
```

Figure 241. Merge Multiple PTFs—Sample Changed Merge Log for 5664167

```
* reqby log for 5664167
:entry.UV00001 UV00003 UV00004
:entry.UV00003 UV00001
```

Figure 242. Merge Multiple PTFs—Changed Reqby Log for 5664167

## Remove Service

### Removing a Single PTF (No Dependents or Supersedes)

To remove a PTF (for example, UV00008) that does not have any corequisites and is not a prerequisite of any other PTF, use the following process.

- You must have a merge log and service control files. Assume you have the merge log and service control files shown in the following figures.

```
:entry.UV00001 Merged      12/13/21 13:09:42 ELEM1 TEXT
:entry.UV00004 Merged      12/15/21 06:28:17 ELEM4 TEXT ELEM5 TEXT
:entry.UV00005 Merged      12/15/21 06:28:17 ELEM5 TEXT ELEM1 TEXT
:entry.UV00006 Merged      01/12/22 17:56:39 ELEM6 TEXT ELEM1 TEXT
:entry.UV00001 Superseded  01/12/22 17:56:39 By UV00006
:entry.Z00001  Superseded  01/12/22 17:56:39 By UV00006
:entry.UV00007 Merged      01/12/22 17:56:39 ELEM6 TEXT
:entry.UV00008 Merged      01/12/22 17:56:39 ELEM7 TEXT
:entry.UV00009 Merged      01/12/22 17:56:39 ELEM8 TEXT
```

Figure 243. Remove Single PTF—Merge Log for 5664167

```
:ptf.UV00001
:prodid.5664167
:changes.
:element.ELEM1 TEXT
:replace.TXTP0001
:echanges.
```

Figure 244. Remove Single PTF—SCF for UV00001



```

:ptf.UV00004
:prodid.5664167
:prereq.UV00001
:changes.
:element.ELEM4 TEXT
:replace.TXTP0004
:element.ELEM5 TEXT
:replace.TXTP0004
:echanges.
:apartext.PP00004 - Split DMKABN for new base register
:apartext.PP00009 - Update CP loadlist to add DMKNEW

```

*Figure 245. Remove Single PTF—SCF for UV00004*

```

:ptf.UV00005
:prodid.5664167
:prereq.UV00004
:coreq.UV00006
:changes.
:element.ELEM5 TEXT
:replace.TXTP0005
:echanges.

```

*Figure 246. Remove Single PTF—SCF for UV00005*

```

:ptf.UV00006
:prodid.5664167
:coreq.UV00005
:sup.UV00001 Z00001
:changes.
:element.ELEM1 TEXT
:replace.TXTP0006
:element.ELEM6 TEXT
:replace.TXTP0006
:echanges.

```

*Figure 247. Remove Single PTF—SCF for UV00006*

```

:ptf.UV00007
:prodid.5664167
:prereq.UV00006
:changes.
:element.ELEM6 TEXT
:replace.TXTP0007
:echanges.

```

*Figure 248. Remove Single PTF—SCF for UV00007*

```

:ptf.UV00008
:prodid.5664167
:prereq.UV00007
:changes.
:element.ELEM7 TEXT
:replace.TXTP0008
:echanges.

```

*Figure 249. Remove Single PTF—SCF for UV00008*

```

:ptf.UV00009
:prodid.5664167
:prereq.UV00004
:changes.
:element.ELEM8 TEXT
:replace.TXTP0009
:echanges.

```

*Figure 250. Remove Single PTF—SCF for UV00009*

- To remove UV00008, enter the following command:

```
vmfiremov 5664167 ptf UV00008
```

- VMFREMOV:

1. Uses the merge log and the service control files to build the reqby log shown in the following figure. Notice there is an entry for each PTF which is a prerequisite or corequisite of another PTF which has a entry of "merged" in the merge log.

```
* reqby log for 5664167
:entry.UV00005 UV00006
:entry.UV00006 UV00005 UV00007
:entry.UV00001 UV00004
:entry.UV00004 UV00005 UV00009
:entry.UV00007 UV00008
```

Figure 251. Remove Single PTF—Sample Reqby Log for 5664167

2. Reads the reqby log and finds UV00008 does not have any dependent PTFs that must also be removed.
3. Reads the merge log and finds ELEM7 TEXT is the only element affected by UV00008.
4. Erases ELEM7 TEXT from the MERGE disk. Because, there are no other previously merged PTFs affected, the text deck for ELEM7 is taken from the BASE disk with the next build.
5. Updates the merge log to show UV00008 is removed and is no longer merged.
6. Removes UV00007 from the reqby log because there are no longer any dependents (for example, UV00008) for that PTF.

The resulting merge log is shown in [Figure 252 on page 772](#) and the resulting reqby log is shown in [Figure 256 on page 773](#). (The service control files are not changed.)

```
:entry.UV00001 Merged      12/26/21 13:09:42 ELEM1 TEXT
:entry.UV00004 Merged      12/28/21 06:28:17 ELEM4 TEXT ELEM5 TEXT
:entry.UV00005 Merged      12/28/21 06:28:17 ELEM5 TEXT
:entry.UV00006 Merged      01/22/22 17:56:39 ELEM6 TEXT ELEM1 TEXT
:entry.UV00001 Superseded  01/22/22 17:56:39 By UV00005
:entry.Z00001 Superseded  01/22/22 17:56:39 By UV00005
:entry.UV00007 Merged      01/22/22 17:56:39 ELEM6 TEXT
*entry.UV00008 Merged      01/22/22 17:56:39 ELEM7 TEXT
:entry.UV00009 Merged      01/22/22 17:56:39 ELEM8 TEXT
*entry.UV00008 Removed    02/02/22 17:56:39
```

Figure 252. Remove Single PTF—Changed Merge Log for 5664167

The changed entries in the merge log are now comments; therefore, they are preceded by an asterisk (\*).

```
* reqby log for 5664167
:entry.UV00005 UV00006
:entry.UV00006 UV00005 UV00007
:entry.UV00001 UV00004
:entry.UV00004 UV00005 UV00009
```

Figure 253. Remove Single PTF—Changed Reqby Log for 5664167

The reqby log no longer contains the line :entry.UV00007 UV00008.

## Removing Multiple PTFs (with Dependents and Supersedes)

To remove multiple PTFs (for example, UV00005 and UV00009), each having a combination of prerequisites, corequisites, and supersedes, use the following process:

- You must have a merge log and service control files. Assume you have the same merge log shown in [Figure 252 on page 772](#), the reqby log shown in [Figure 253 on page 772](#), and the service control files shown in [Figure 244 on page 770](#) through [Figure 250 on page 771](#).
- To remove UV00005 and UV00009 using one command, you need to create a remove list. Your remove list may look like [Figure 254 on page 773](#).

```
* This is my own remove list
UV00005 take off UV00005
UV00009 take off UV00009
```

Figure 254. MYLIST REMLIST

- After you create the remove list, enter the following command:

```
vmfremov 5664167 ptflist mylist
```

- VMFREMOV:
  1. Looks for the file MYLIST REMLIST on one of the DELTA disks, because you entered the keyword **ptflist** followed by a listname (**mylist**).
  2. Reads the reqby log and finds that to remove UV00005, UV00006 and UV00007 must also be removed.  
  
Because UV00001 and Z00001 are superseded by UV00005 (which now needs to be removed), UV00001's status is changed back to merged and Z00001 is no longer superseded.  
  
UV00009 has no dependents, so it can be removed without removing any other PTFs.
  3. Erases from the MERGE disk, or replaces with other levels from the DELTA disk, the following elements: ELEM5 TEXT, ELEM1 TEXT, ELEM6 TEXT, and ELEM8 TEXT.
  4. Updates the merge log to show UV00005, UV00006, UV00007, and UV00009 are removed and are no longer merged. In addition, UV00001 is no longer superseded and now has a status of merged. Z00001 is no longer superseded.
  5. Updates the reqby log by removing any entries for the removed PTFs and eliminates any dependents for those PTFs.

The resulting merge log is shown in Figure 255 on page 773, and the resulting reqby log is shown in Figure 256 on page 773. (The service control files are not changed.)

```
:entry.UV00001 Merged      01/13/22 13:09:42 ELEM1 TEXT
:entry.UV00004 Merged      01/13/22 06:28:17 ELEM4 TEXT ELEM5 TEXT
*entry.UV00005 Merged      01/13/22 06:28:17 ELEM5 TEXT ELEM1 TEXT
*entry.UV00006 Merged      02/12/22 17:56:39 ELEM6 TEXT ELEM1 TEXT
*entry.UV00001 Superseded  02/12/22 17:56:39 By UV00005
*entry.Z00001 Superseded  02/12/22 17:56:39 By UV00005
*entry.UV00007 Merged      02/12/22 17:56:39 ELEM6 TEXT
*entry.UV00008 Merged      02/12/22 17:56:39 ELEM7 TEXT
*entry.UV00009 Merged      02/12/22 17:56:39 ELEM8 TEXT
*entry.UV00008 Removed    02/22/22 05:56:39
*entry.UV00007 Removed    02/22/22 05:56:39
*entry.UV00006 Removed    02/22/22 05:56:39
*entry.UV00005 Removed    01/23/22 06:28:17
*entry.UV00009 Removed    02/22/22 05:56:39
```

Figure 255. Remove Multiple PTFs—Changed Merge Log for 5664167

The changed entries in the merge log are now comments; therefore, they are preceded by an asterisk (\*).

```
* reqby log for 5664167
:entry.UV00001 UV00004
```

Figure 256. Remove Multiple PTFs—Changed Reqby Log for 5664167

## Prevent Regression

There are precautions you can take to ensure the level of service applied to your product does not regress. You should:

- Save apply lists. Whenever you merge corrective fixes to your product, you **must** save the apply lists you use. You use these lists after you apply preventive service.

## VMFMERGE EXEC

- Reapply corrective fixes. If you have merged any corrective fixes into your product, you should run VMFMERGE after applying the PUT. You should do this for each apply list you used when applying corrective service. You should apply corrective fixes in chronological order, but you do not have to.

**Note:** You can create one large apply list that contains all the other apply lists and run VMFMERGE using just the large apply list.

- Reapply ZAPs. You can run VMFZAP over and over without harming your product. Each time you do, it erases all ZAP TEXT files that were previously ZAPped. Then all the ZAPs (except those that have been superseded) listed in *prodid* ZAPLIST are reapplied.

You should run VMFZAP just before you run the product supplied exec that builds the executable version of the product.

## Removing a Fix-in-Error

Let's assume that:

PUT 1 contains PTF 10, which affects Module A

PUT 2 contains PTF 11, which affects Module A

PUT 2 contains PTF 12, which affects Module B

PUT 2 contains PTF 13, which affects Module C

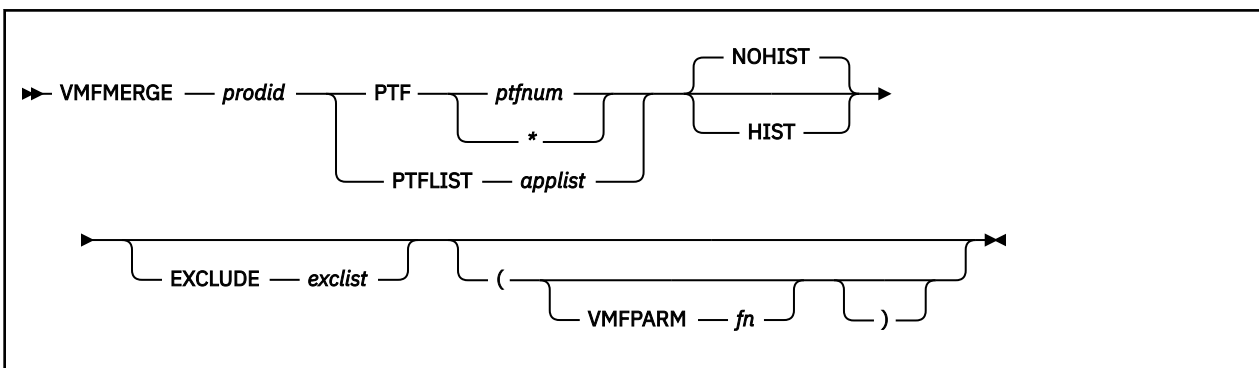
PUT 3 contains PTF 14, which affects Module B and Module C

PUT 4 contains PTF 15, which affects Module A

Now, after you apply PUT 4, suppose you find an error in your product after it has been built. When you report the error to your Support Center, they tell you the file names of the PTF(s) in error that must be "backed off". For example, if PTF 12 is bad and you want to remove it, here's what you need to do:

1. Run VMFREMOVE specifying the *prodid* and PTF12. VMFREMOVE removes PTF12 and PTF14.
2. Run VMFZAP specifying the *prodid* to apply any previously applied ZAPs. If the ZAP list has not been changed, then VMFZAP reapplies the ZAPs which were superseded by the changes just removed.
3. Finally, run the product-supplied exec that builds the executable version of the product.

## VMFMERGE EXEC



### Purpose

The VMFMERGE EXEC procedure applies PTFs (program temporary fixes) from the DELTA disk to the MERGE disk.

**Do not use this procedure to service any of the base components of z/VM. Use this procedure when applying PTFs to System Network Architecture (SNA) products.**

VMFMERGE requires a service control file (*ptfnum* SCF) for each requested PTF and its requisites. The service control file contains instructions for applying the PTF. To use VMFMERGE, you must access the

minidisk containing the *prodid* VMFPARM file. This file identifies the minidisks VMFMERGE must access to service each product.

VMFMERGE requires the following files to be on the DELTA disks:

*prodid* VMFPARM  
*prodid* VMFGLOG  
*prodid* APPLIST or *applist* APPLIST  
*prodid* EXCLIST or *exclist* EXCLIST  
*prodid* SCF

## Operands

### ***prodid***

is the product identifier for the product you specify.

You cannot specify a *prodid* of EXCLUDE, because EXCLUDE is a keyword for this exec.

### **PTF**

applies a single PTF or all PTFs for a product.

### ***ptfnum***

is the PTF file name. If you specify a PTF file name, VMFMERGE applies a single PTF. You cannot specify a file name of EXCLUDE, because EXCLUDE is a keyword for this exec.

### **\***

indicates you want to apply all PTFs for a product. If you enter an asterisk (\*) instead of a PTF file name, you apply all PTFs for the product, as they are listed in the apply list (*prodid* APPLIST) supplied on the service tape.

**Note:** If there is an exclude list file (*prodid* EXCLIST) on the service tape, any PTF listed in that file is not applied.

### **PTFLIST**

applies the selected PTFs that are listed in the apply list file (*applist* APPLIST).

### ***applist***

is the file name of the apply list file. The file type is APPLIST.

**Note:** If you specify a file name of EXCLUDE, you cannot use the EXCLUDE option to specify an Exclude List.

If there is an exclude list file (*prodid* EXCLIST) on the service tape, any PTF listed in that file is not applied.

### **NOHIST**

does not include APAR text lines from the SCF files as comments in the text decks processed. This option is effective unless you specifically override it with the HIST option. NOHIST is the default.

### **HIST**

includes APAR text comments from the SCF files in the text decks processed. The APAR text lines from the SCF files are included at the beginning of each text deck as comments. This option is effective for the duration of the command. All text decks processed during the VMFMERGE process will include APAR text entries as comments.

### **EXCLUDE**

excludes the selected PTFs listed in the user-specified exclude list file (*exclist* EXCLIST).

PTFs listed in *prodid* EXCLIST are also excluded, in addition to those in the user-specified exclude list. If you use *prodid* EXCLIST as the name of your exclude list, VMFMERGE ignores any other *prodid* EXCLIST file (including the one that may be supplied on the service tape) during processing. Therefore, you should use another name when you create an exclude list.

**Note:** If you specify EXCLUDE as the apply list file name, you cannot use the EXCLUDE option to specify an exclude list.

***exclist***

is the name of a user-specified exclude list.

## Options

**VMFPARM**

indicates you want to use a VMFPARM file other than the default one, which is *prodid* VMFPARM.

***fn***

is the file name of the VMFPARM file to be used.

## Usage Notes

1. You can issue VMFMERGE to process a single PTF, a list of PTFs, or all PTFs on the input disk for a product.
2. Merged PTFs cannot be excluded, but they can be superseded.
3. Superseded PTFs cannot be merged nor excluded.
4. PTFs in the Exclude List can be superseded.
5. If VMFMERGE has the HIST operand specified, and LKED does not have the LET option specified, error message IEW0222 will not occur and there will not be linkage editor errors due to comments (invalid card).

## VMFMERGE Processing

VMFMERGE is a service process that processes PTFs. The exec procedure:

1. Uses the parameter file (*prodid* VMFPARM) to determine the virtual address of the MERGE and DELTA disks.
2. Checks the merge log to insure the PTF you select is not already merged or superseded.
3. Reads the service control file to get the prerequisite and corequisite PTFs and the elements affected by this PTF.
  - If the service control file for any of the prerequisite or corequisite PTFs is missing, processing for the current PTF stops.
  - If a prerequisite or corequisite PTF is in the exclude list, the current PTF is not merged.
  - If a prerequisite or corequisite PTF has been superseded, that prerequisite or corequisite is not merged.
  - If the requisite is not within this product, the system displays a message indicating the requisite PTF must be merged at some later time.
  - When you merge a PTF that is a requisite of a change in another product, be sure to note this requisite information. There is no automatic way to tell you of this cross-product requisite if at a later time you remove the change that is a requisite of a change in another product.
4. Does the necessary COPY/RENAME from the DELTA disk to the MERGE disk for each element in the prerequisite and corequisite chain that was not superseded or already merged.

**Note:** Temporary files are created during this COPY/RENAME process to insure system integrity. These files are erased during normal VMFMERGE processing.
5. Adds service history to the element if it is a text deck. Element history consists of text deck comments containing:
  - The PTF or APAR number
  - A time and date stamp
  - Any APAR text information that was in the service control file SCF.

**Note:** A temporary file (\$APARTXT \$VMFMERG) is created during this history process to ensure system integrity. VMFMERGE erases this file during normal processing.

6. Updates the reqby log to reflect all the requisite relationships of all the merged and superseded PTFs.
7. Marks in the merge log any ZAPs and PTFs that are superseded by this PTF. Those ZAPs and PTFs are never applied.
8. Puts entries in the merge log to show which PTFs have been merged.
9. Updates the service-level apply status table (*prodid* SRVAPPS).

## Messages and Return Codes

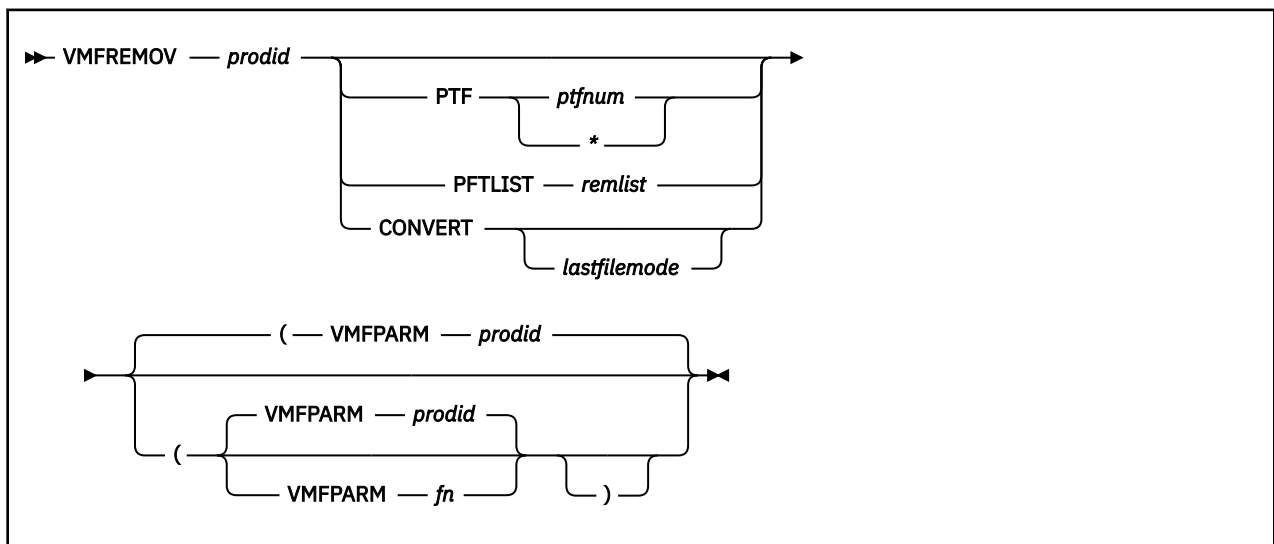
Use the HELP Facility to view the message explanation online or see the appropriate messages documentation. To display information on a specific message, for example DMS002E, enter:

```
help msg dms002e
```

If you are unfamiliar with the z/VM HELP Facility, enter help to display the main HELP Menu. For information on the HELP command, enter:

```
help cms help
```

## VMFREMOV EXEC



## Purpose

The VMFREMOV EXEC procedure removes PTFs applied by the VMFMERGE EXEC procedure.



**Attention:** Do not use this procedure to service any of the base components of z/VM. Use this procedure when applying PTFs to System Network Architecture (SNA) products that are not in VMSES/E format.

## Operands

### *prodid*

is the product identifier for the product.

### **Note:**

If you want to create the 'required by' log and you do **not** want to remove any changes, use the CONVERT keyword.

### PTF

removes the specified PTF.

#### *ptfnum*

is the file name of a PTF.

#### \*

removes the PTFs listed in a remove list file (*prodid* REMLIST). This file must already exist on a DELTA disk.

### PTFLIST

removes the selected PTFs that are in a remove list file (*remlist* REMLIST)

#### *remlist*

is the name of the remove list file. This file must already exist on a DELTA disk.

### CONVERT

calls the tool that creates the 'required by' log file (if one does not already exist), but does **not** remove any changes.

#### *lastfilemode*

specifies the file mode of the last DELTA disk. VMFREMOV assumes the merge disk is always accessed as E and that all other DELTA disks are accessed as consecutive file modes between F and the *lastfilemode*. If you do not specify *lastfilemode*, the file modes for the MERGE and DELTA disks are determined from the information in the VMFPARM file. Use the *lastfilemode* parameter only if you **know** you have the correct disks accessed as the proper modes.

## Options

### VMFPARM

identifies the VMFPARM file to use.

#### *prodid*

is the default file name for the VMFPARM file. *prodid* is the product identifier for the product.

#### *fn*

is the file name of the VMFPARM file to use.

## VMFREMOV Processing

VMFREMOV is a service process that removes PTFs applied by VMFMERGE. The exec procedure:

1. Obtains data from the merge log and the service control files to build the 'required by' log if one does not already exist. The 'required by' log contains a list of all dependent PTFs that must be removed if their requisite PTF is removed.
  - If a service control file for any of the PTFs is missing, then processing continues. However, the 'required by' log will be incomplete if the missing service control file contains requisites. If any SCFs were missing, processing ends after VMFREMOV completes the build of the 'required by' log. No PTFs are removed.
2. Checks the merge log to insure the PTF to be removed is currently merged.
3. Reads the 'required by' log for the list of dependent PTFs to remove.
4. Removes a PTF (for example, UV00007) that may supersede other PTFs (for example, UV00005).

**Note:** The other PTF (UV00005) is no longer superseded and its status remains as it was prior to the merge of the primary PTF (UV00007), that is, merged, superseded, or no status. If the prior status of UV00005 is no status, then VMFREMOV removes its dependents.
5. Copies, from the DELTA disk to the MERGE disk, each element affected by the PTF being removed if previous service for an element is merged. If the element has not been merged, VMFREMOV erases the element from the MERGE disk.





## Options

### VMFPARM

indicates the VMFPARM file is to be used in place of the default file, *prodid* VMFPARM.

#### *prodid*

is the default file name for the VMFPARM file. *prodid* is the product identifier for the product that is to be updated.

#### *fn*

is the file name of the VMFPARM file to use.

## Usage Notes

1. To use VMFZAP, you **must** have a read/write file accessed as mode A. This file mode **must not** be the ZAP minidisk, merge minidisk, or base minidisk. That is, if file mode A is a minidisk, the virtual address **must not** appear on the ZAP, Merge, or Base records of your VMFPARM file.
2. VMFZAP accesses the necessary minidisks using file mode letters E-N. The merge minidisk is accessed ahead of the base minidisk. When VMFZAP processing stops, your search hierarchy is restored.

## VMFZAP Processing

The VMFZAP EXEC:

1. Uses the product parameter file (*prodid* VMFPARM) to determine the virtual addresses of the ZAP, merge, and base minidisks.
2. Reads the ZAP Log file (*prodid* VMFZPLOG) and builds a list of TEXT file names with ZAPs applied to them.
3. Erases all TEXT files in this list from the ZAP minidisk. ZAPs are applied to the first version of these TEXT files found on the other accessed minidisks.
4. Erases the ZAP Log.
5. Reads the merge log file (*prodid* VMFMGLOG) and builds a list of ZAPs that are currently superseded.
6. Reads the ZAP list file (*prodid* ZAPLIST) for the names of all ZAPs you want to apply.
7. Checks each ZAP name to see if it is superseded. If it is not superseded, VMFZAP reads the control file for that ZAP (*zapname* ZAP).

The control file for a ZAP may contain information for updating more than one TEXT file. VMFZAP separates this information by TEXT file name and processes the ZAP of each TEXT file in the order they are listed in the control file.

8. Checks each TEXT file name to make sure it resides on some minidisk other than the ZAP minidisk. It is an error if the TEXT file already resides on the ZAP minidisk.

If the TEXT file exists on another minidisk, VMFZAP writes a temporary file called \$\$VMFZAP ZAP to the ZAP minidisk containing the ZAP control records for the current TEXT file.

9. Copies the TEXT file to the ZAP minidisk under a temporary name using the format VMF\$T*n* TEXT, where *n* is a number determined by how many TEXT files are affected by the ZAP currently being processed. When this temporary file has been successfully updated, it is renamed to its original name on the ZAP minidisk.
10. Invokes the ZAPTEXT EXEC and passes the \$\$VMFZAP name and the VMF\$T*n* file name to be updated.
 

See [“ZAPTEXT EXEC” on page 781](#) for more information about the ZAPTEXT EXEC.
11. Updates the ZAP log with the information about the TEXT file that was just updated.
12. Restores your minidisk search hierarchy after all ZAPs in the ZAP list have been processed.

## Messages and Return Codes

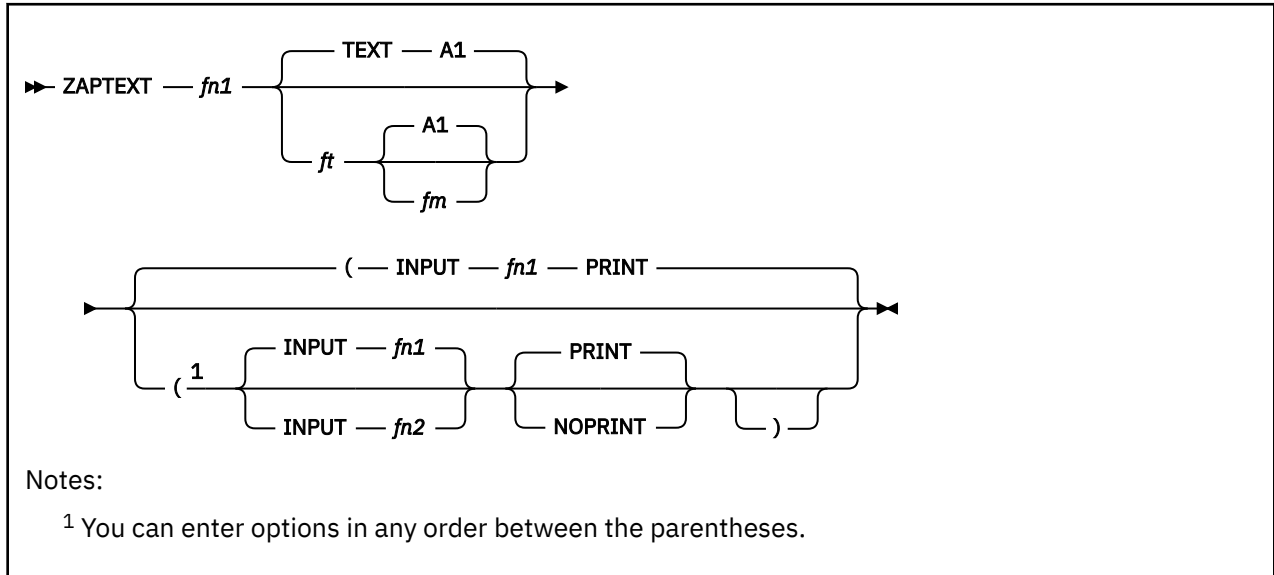
Use the HELP Facility to view the message explanation online or see the appropriate messages documentation. To display information on a specific message, for example DMS002E, enter:

```
help msg dms002e
```

If you are unfamiliar with the z/VM HELP Facility, enter help to display the main HELP Menu. For information on the HELP command, enter:

```
help cms help
```

## ZAPTEXT EXEC



## Purpose

The ZAPTEXT EXEC modifies or dumps individual text files. Use ZAPTEXT like the ZAP service program, but only for text files, not for MODULEs, TXTLIBs, and LOADLIBs. (Use ZAP to process MODULEs, TXTLIBs, and LOADLIBs). ZAPTEXT uses the same control information as ZAP and can also use the EXPAND control record. Your A-disk must be accessed read/write to use ZAPTEXT.

## Operands

### *fn1*

is the file name of the text file you want to change. This file must be inactive (not pre-opened) at the time ZAPTEXT is run or called. ZAPTEXT cannot process a file locked in another user's directory.

### TEXT

is the default file type for the text file.

### *ft*

is the file type of the text file you want to change.

### A1

is the default file mode for the text file.

### *fm*

is the file mode of the text file you want to change. The file mode must specify a read/write disk.

## Options

### INPUT

identifies the file that has the ZAP control records. INPUT is the default. This file must:

- Have a file type of ZAP
- Be a fixed 80-byte sequential file that resides on any file mode
- Not be either pre-opened or locked in another user's directory

### *fn1*

is the default file name for the input file that contains the ZAP control records.

### *fn2*

is the file name for the input file that contains the ZAP control records. If you do not specify *fn2*, it defaults to whatever was specified as *fn1* when you entered the ZAPTEXT command.

### PRINT

prints all output produced by ZAPTEXT. The system also displays on the terminal error messages, commands in error, and control records in error. PRINT is the default.

### NOPRINT

displays all output on the terminal and does not print anything.

## Input Control Records

ZAPTEXT uses the same input control records as the ZAP service program, with the addition of the EXPAND control record. The ZAP service program ignores any EXPAND control records. For information about the control records that both ZAP and ZAPTEXT can use, see the description of the ZAP module in *z/VM: CMS Commands and Utilities Reference*. Use the ZAP control records with ZAPTEXT according to ZAP's TXTLIB conventions.

**EXPAND Control Records:** The EXPAND control record lets you increase the size of a named control section in the text file. The format of the EXPAND control record is:



### *csect*

specifies the symbolic name of a control section whose length you want to increase.

### *size*

specifies the decimal number of bytes for the system to add to the control section length. The system initializes the added bytes to binary zero. The maximum number of bytes for each control section is 4095.

### **Control Record Usage Notes:**

1. Each EXPAND control record can have multiple entries, but you must separate them with commas. Do not spill an entry onto the next line.
2. The system processes all EXPAND control records before any other control records, regardless of their position in the control file.
3. The effective length of the expansion, which is the actual number of bytes added to the control section, may be greater than the length that you specify for the expansion. This may occur if, after the specified expansion, the system must add padding bytes to align the next control section or common area.
4. When you increase a control section's size, it could affect the offset address of any following control section. This is important when you determine values for BASE, REP, and VER control records. Use the effective expansion lengths when you are determining control section offsets.

## Messages and Return Codes

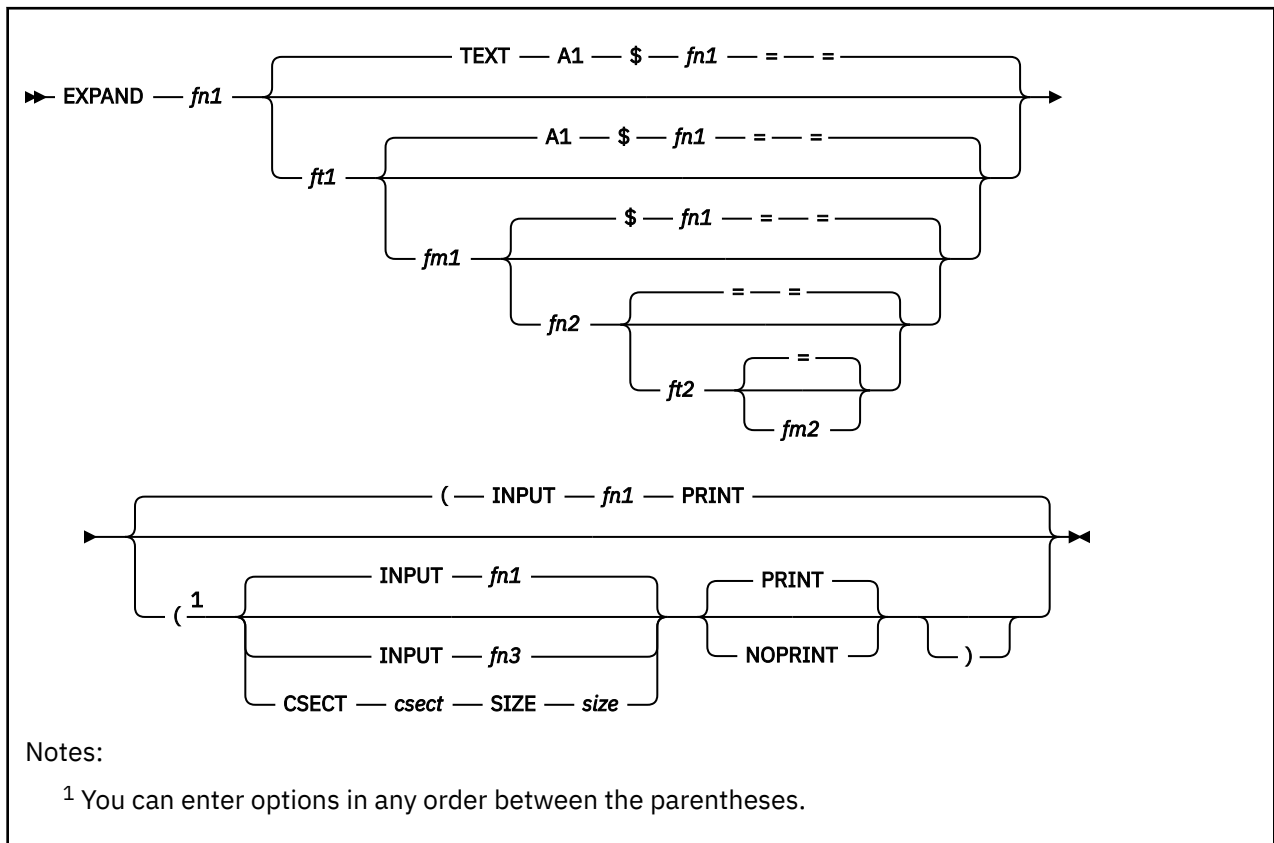
Use the HELP Facility to view the message explanation online or see the appropriate messages documentation. To display information on a specific message, for example DMS002E, enter:

```
help msg dms002e
```

If you are unfamiliar with the z/VM HELP Facility, enter help to display the main HELP Menu. For information on the HELP command, enter:

```
help cms help
```

## EXPAND Command



## Purpose

ZAPTEXT EXEC calls the EXPAND command if you specify an EXPAND control record in the ZAP control file. Use the EXPAND command to add space to a program in object deck form. The system creates object decks when you assemble or compile a source program. This is especially useful when you do not have the source code for a program or the program does not have a patch area.

**Note:** EXPAND can add extra space only at the end of named control sections (CSECTs). EXPAND cannot expand private code (unnamed CSECT) and common areas (named or unnamed).

If EXPAND finds more than one END card in the object deck, it will treat the text located between END cards as separate text. EXPAND will only add space to the text associated with the CSECT being expanded and its END card.

Do not increase the length of a program beyond its design limitations. For example, if you add space to a control section beyond the range of its base register addressability, that space is unusable.

### Operands

#### *fn1*

is the file name of the input text file that the system expands.

#### TEXT

is the default file type of the input text file that the system expands.

#### *ft1*

is the file type of the input text file that the system expands.

#### A1

is the default file mode of the input file.

#### *fm1*

is the file mode of the input text file that the system expands.

**Note:** The input text file must have valid object deck information, like that created by an assembler or compiler. The file must not be pre-opened or locked in another user's directory.

EXPAND assumes the input text file follows OS/VSE standards and the OS/VSE Linkage Editor will accept it without error. The system does a limited check for errors. If the input file is invalid, the system might not expand the text file correctly.

#### *\$fn1*

is the default file name of the output text file created by the system. *fn1* is truncated to 7 characters before '\$' is appended.

#### *fn2*

is the file name of the output text file created by the system. You can specify an equal sign (=) to tell ZAPTEXT to use the same file type as was specified for *fn1*

#### *ft2*

is the file type of the output text file created by the system. You can specify an equal sign (=) to tell ZAPTEXT to use the same file type as was specified for *ft1*

#### *fm2*

is the file mode of the output text file created by the system. You can specify an equal sign (=) to tell ZAPTEXT to use the same file type as was specified for *fm1*. *fm2* must be a read/write file mode. You cannot specify an asterisk (\*).

### Options

#### INPUT

identifies an EXPAND input file that contains EXPAND control records. If you do not specify INPUT, the file name defaults to the name of the text file you are expanding (*fn1*). The file type must be EXPAND. The system searches all accessed file modes for this file. The file must not be pre-opened or locked in another user's directory.

**Note:** Do not specify this option with the CSECT or SIZE options.

#### *fn1*

is the default file name of the input file.

#### *fn3*

is the file name of the input file that contains EXPAND control records.

#### CSECT

specifies the symbolic name of a control section whose length the system will increase. If you specify CSECT, you must also specify SIZE.

Do not specify CSECT with the INPUT option.

#### *csect*

is the symbolic name of a control section.

**SIZE**

specifies the decimal number of bytes the system adds to the control section length. The system initializes the added bytes to binary zeros. If you specify SIZE, you must also specify CSECT.

**Note:** Do not specify SIZE with the INPUT option.

***size***

is the decimal number of bytes added by the system to the control section length. The maximum number of bytes for each control section is 4095.

**PRINT**

prints on the printer all output that EXPAND produces. In addition, the system displays error messages, commands in error, and control records in error at the terminal. PRINT is the default.

**NOPRINT**

does not print any output on the printers, and instead displays it at the terminal.

**Input and Output Files**

- After the system expands each CSECT you specified, the system issues a message that indicates:
  - The number of bytes added to the control section.
  - Whether the number of bytes is greater than the length that you specify for the expansion. This may occur if, after the specified expansion, the system must add padding bytes to align the next control section or common area.
  - The offset, relative to the start of the specified control section where the expansion began.
- If the system finds an error during processing, it stops the update and does not do the expansions.

**Messages and Return Codes**

Use the HELP Facility to view the message explanation online or see the appropriate messages documentation. To display information on a specific message, for example DMS002E, enter:

```
help msg dms002e
```

If you are unfamiliar with the z/VM HELP Facility, enter help to display the main HELP Menu. For information on the HELP command, enter:

```
help cms help
```





## Notices

---

This information was developed for products and services offered in the US. This material might be available from IBM in other languages. However, you may be required to own a copy of the product or product version in that language in order to access it.

IBM may not offer the products, services, or features discussed in this document in other countries. Consult your local IBM representative for information on the products and services currently available in your area. Any reference to an IBM product, program, or service is not intended to state or imply that only that IBM product, program, or service may be used. Any functionally equivalent product, program, or service that does not infringe any IBM intellectual property right may be used instead. However, it is the user's responsibility to evaluate and verify the operation of any non-IBM product, program, or service.

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

*IBM Director of Licensing  
IBM Corporation  
North Castle Drive, MD-NC119  
Armonk, NY 10504-1785  
US*

For license inquiries regarding double-byte character set (DBCS) information, contact the IBM Intellectual Property Department in your country or send inquiries, in writing, to:

*Intellectual Property Licensing  
Legal and Intellectual Property Law  
IBM Japan Ltd.  
19-21, Nihonbashi-Hakozakicho, Chuo-ku  
Tokyo 103-8510, Japan*

INTERNATIONAL BUSINESS MACHINES CORPORATION PROVIDES THIS PUBLICATION "AS IS" WITHOUT WARRANTY OF ANY KIND, EITHER EXPRESS OR IMPLIED, INCLUDING, BUT NOT LIMITED TO, THE IMPLIED WARRANTIES OF NON-INFRINGEMENT, MERCHANTABILITY OR FITNESS FOR A PARTICULAR PURPOSE. Some jurisdictions do not allow disclaimer of express or implied warranties in certain transactions, therefore, this statement may not apply to you.

This information could include technical inaccuracies or typographical errors. Changes are periodically made to the information herein; these changes will be incorporated in new editions of the publication. IBM may make improvements and/or changes in the product(s) and/or the program(s) described in this publication at any time without notice.

Any references in this information to non-IBM websites are provided for convenience only and do not in any manner serve as an endorsement of those websites. The materials at those websites are not part of the materials for this IBM product and use of those websites is at your own risk.

IBM may use or distribute any of the information you provide in any way it believes appropriate without incurring any obligation to you.

Licensees of this program who wish to have information about it for the purpose of enabling: (i) the exchange of information between independently created programs and other programs (including this one) and (ii) the mutual use of the information which has been exchanged, should contact:

*IBM Director of Licensing  
IBM Corporation  
North Castle Drive, MD-NC119  
Armonk, NY 10504-1785  
US*

Such information may be available, subject to appropriate terms and conditions, including in some cases, payment of a fee.

The licensed program described in this document and all licensed material available for it are provided by IBM under terms of the IBM Customer Agreement, IBM International Program License Agreement or any equivalent agreement between us.

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

Information concerning non-IBM products was obtained from the suppliers of those products, their published announcements or other publicly available sources. IBM has not tested those products and cannot confirm the accuracy of performance, compatibility or any other claims related to non-IBM products. Questions on the capabilities of non-IBM products should be addressed to the suppliers of those products.

Statements regarding IBM's future direction or intent are subject to change or withdrawal without notice, and represent goals and objectives only.

This information may contain examples of data and reports used in daily business operations. To illustrate them as completely as possible, the examples include the names of individuals, companies, brands, and products. All of these names are fictitious and any similarity to actual people or business enterprises is entirely coincidental.

#### COPYRIGHT LICENSE:

This information may contain sample application programs in source language, which illustrate programming techniques on various operating platforms. You may copy, modify, and distribute these sample programs in any form without payment to IBM, for the purposes of developing, using, marketing or distributing application programs conforming to the application programming interface for the operating platform for which the sample programs are written. These examples have not been thoroughly tested under all conditions. IBM, therefore, cannot guarantee or imply reliability, serviceability, or function of these programs. The sample programs are provided "AS IS", without warranty of any kind. IBM shall not be liable for any damages arising out of your use of the sample programs.

## Programming Interface Information

---

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

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

**PI**

<...Programming Interface information...>

**PI end**

## Trademarks

---

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

The registered trademark Linux<sup>®</sup> is used pursuant to a sublicense from the Linux Foundation, the exclusive licensee of Linus Torvalds, owner of the mark on a world-wide basis.

## Terms and Conditions for Product Documentation

---

Permissions for the use of these publications are granted subject to the following terms and conditions.

## Applicability

These terms and conditions are in addition to any terms of use for the IBM website.

## Personal Use

You may reproduce these publications for your personal, noncommercial use provided that all proprietary notices are preserved. You may not distribute, display or make derivative work of these publications, or any portion thereof, without the express consent of IBM.

## Commercial Use

You may reproduce, distribute and display these publications solely within your enterprise provided that all proprietary notices are preserved. You may not make derivative works of these publications, or reproduce, distribute or display these publications or any portion thereof outside your enterprise, without the express consent of IBM.

## Rights

Except as expressly granted in this permission, no other permissions, licenses or rights are granted, either express or implied, to the publications or any information, data, software or other intellectual property contained therein.

IBM reserves the right to withdraw the permissions granted herein whenever, in its discretion, the use of the publications is detrimental to its interest or, as determined by IBM, the above instructions are not being properly followed.

You may not download, export or re-export this information except in full compliance with all applicable laws and regulations, including all United States export laws and regulations.

IBM MAKES NO GUARANTEE ABOUT THE CONTENT OF THESE PUBLICATIONS. THE PUBLICATIONS ARE PROVIDED "AS-IS" AND WITHOUT WARRANTY OF ANY KIND, EITHER EXPRESSED OR IMPLIED, INCLUDING BUT NOT LIMITED TO IMPLIED WARRANTIES OF MERCHANTABILITY, NON-INFRINGEMENT, AND FITNESS FOR A PARTICULAR PURPOSE.

## IBM Online Privacy Statement

---

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

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

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

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

- The section entitled **IBM Websites** at [IBM Privacy Statement](https://www.ibm.com/privacy) (<https://www.ibm.com/privacy>)
- [Cookies and Similar Technologies](https://www.ibm.com/privacy#Cookies_and_Similar_Technologies) ([https://www.ibm.com/privacy#Cookies\\_and\\_Similar\\_Technologies](https://www.ibm.com/privacy#Cookies_and_Similar_Technologies))



# Bibliography

---

This topic lists the publications in the z/VM library. For abstracts of the z/VM publications, see [z/VM: General Information](#).

## Where to Get z/VM Information

---

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

## z/VM Base Library

---

### Overview

- [z/VM: License Information](#), GI13-4377
- [z/VM: General Information](#), GC24-6286

### Installation, Migration, and Service

- [z/VM: Installation Guide](#), GC24-6292
- [z/VM: Migration Guide](#), GC24-6294
- [z/VM: Service Guide](#), GC24-6325
- [z/VM: VMSES/E Introduction and Reference](#), GC24-6336

### Planning and Administration

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

### Customization and Tuning

- [z/VM: CP Exit Customization](#), SC24-6269
- [z/VM: Performance](#), SC24-6301

### Operation and Use

- [z/VM: CMS Commands and Utilities Reference](#), SC24-6260
- [z/VM: CMS Primer](#), SC24-6265
- [z/VM: CMS User's Guide](#), SC24-6266
- [z/VM: CP Commands and Utilities Reference](#), SC24-6268

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

## Application Programming

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

## Diagnosis

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

## z/VM Facilities and Features

---

### Data Facility Storage Management Subsystem for z/VM

- [z/VM: DFSMS/VM Customization](#), SC24-6274
- [z/VM: DFSMS/VM Diagnosis Guide](#), GC24-6275
- [z/VM: DFSMS/VM Messages and Codes](#), GC24-6276
- [z/VM: DFSMS/VM Planning Guide](#), SC24-6277

- [z/VM: DFSMS/VM Removable Media Services](#), SC24-6278
- [z/VM: DFSMS/VM Storage Administration](#), SC24-6279

## Directory Maintenance Facility for z/VM

- [z/VM: Directory Maintenance Facility Commands Reference](#), SC24-6281
- [z/VM: Directory Maintenance Facility Messages](#), GC24-6282
- [z/VM: Directory Maintenance Facility Tailoring and Administration Guide](#), SC24-6283

## Open Systems Adapter

- [Open Systems Adapter-Express Customer's Guide and Reference](#) (<https://www.ibm.com/support/pages/node/6019492>), SA22-7935
- [Open Systems Adapter-Express Integrated Console Controller User's Guide](#) (<https://www.ibm.com/support/pages/node/6019810>), SC27-9003
- [Open Systems Adapter-Express Integrated Console Controller 3215 Support](#) ([https://www.ibm.com/docs/en/SSLTBW\\_2.1.0/com.ibm.zos.v2r1.ioa/ioa.htm](https://www.ibm.com/docs/en/SSLTBW_2.1.0/com.ibm.zos.v2r1.ioa/ioa.htm)), SA23-2247
- [Open Systems Adapter/Support Facility on the Hardware Management Console](#) ([https://www.ibm.com/docs/en/SSLTBW\\_2.1.0/com.ibm.zos.v2r1.ioa/ioa.htm](https://www.ibm.com/docs/en/SSLTBW_2.1.0/com.ibm.zos.v2r1.ioa/ioa.htm)), SC14-7580

## Performance Toolkit for z/VM

- [z/VM: Performance Toolkit Guide](#), SC24-6302
- [z/VM: Performance Toolkit Reference](#), SC24-6303

## RACF Security Server for z/VM

- [z/VM: RACF Security Server Auditor's Guide](#), SC24-6305
- [z/VM: RACF Security Server Command Language Reference](#), SC24-6306
- [z/VM: RACF Security Server Diagnosis Guide](#), GC24-6307
- [z/VM: RACF Security Server General User's Guide](#), SC24-6308
- [z/VM: RACF Security Server Macros and Interfaces](#), SC24-6309
- [z/VM: RACF Security Server Messages and Codes](#), GC24-6310
- [z/VM: RACF Security Server Security Administrator's Guide](#), SC24-6311
- [z/VM: RACF Security Server System Programmer's Guide](#), SC24-6312
- [z/VM: Security Server RACROUTE Macro Reference](#), SC24-6324

## Remote Spooling Communications Subsystem Networking for z/VM

- [z/VM: RSCS Networking Diagnosis](#), GC24-6316
- [z/VM: RSCS Networking Exit Customization](#), SC24-6317
- [z/VM: RSCS Networking Messages and Codes](#), GC24-6318
- [z/VM: RSCS Networking Operation and Use](#), SC24-6319
- [z/VM: RSCS Networking Planning and Configuration](#), SC24-6320

## TCP/IP for z/VM

- [z/VM: TCP/IP Diagnosis Guide](#), GC24-6328
- [z/VM: TCP/IP LDAP Administration Guide](#), SC24-6329
- [z/VM: TCP/IP Messages and Codes](#), GC24-6330

- *z/VM: TCP/IP Planning and Customization*, SC24-6331
- *z/VM: TCP/IP Programmer's Reference*, SC24-6332
- *z/VM: TCP/IP User's Guide*, SC24-6333

## Prerequisite Products

---

### Device Support Facilities

- Device Support Facilities (ICKDSF): User's Guide and Reference ([https://www.ibm.com/servers/resourcelink/svc00100.nsf/pages/zosv2r5gc350033/\\$file/ickug00\\_v2r5.pdf](https://www.ibm.com/servers/resourcelink/svc00100.nsf/pages/zosv2r5gc350033/$file/ickug00_v2r5.pdf)), GC35-0033

### Environmental Record Editing and Printing Program

- Environmental Record Editing and Printing Program (EREP): Reference ([https://www.ibm.com/servers/resourcelink/svc00100.nsf/pages/zosv2r5gc350152/\\$file/ifc2000\\_v2r5.pdf](https://www.ibm.com/servers/resourcelink/svc00100.nsf/pages/zosv2r5gc350152/$file/ifc2000_v2r5.pdf)), GC35-0152
- Environmental Record Editing and Printing Program (EREP): User's Guide ([https://www.ibm.com/servers/resourcelink/svc00100.nsf/pages/zosv2r5gc350151/\\$file/ifc1000\\_v2r5.pdf](https://www.ibm.com/servers/resourcelink/svc00100.nsf/pages/zosv2r5gc350151/$file/ifc1000_v2r5.pdf)), GC35-0151

## Related Products

---

### z/OS

- *Common Programming Interface Communications Reference* (<https://publibfp.dhe.ibm.com/epubs/pdf/c2643999.pdf>), SC26-4399
- z/OS and z/VM: Hardware Configuration Definition Messages ([https://www.ibm.com/servers/resourcelink/svc00100.nsf/pages/zosv2r5sc342668/\\$file/cbdlm100\\_v2r5.pdf](https://www.ibm.com/servers/resourcelink/svc00100.nsf/pages/zosv2r5sc342668/$file/cbdlm100_v2r5.pdf)), SC34-2668
- z/OS and z/VM: Hardware Configuration Manager User's Guide ([https://www.ibm.com/servers/resourcelink/svc00100.nsf/pages/zosv2r5sc342670/\\$file/eequ100\\_v2r5.pdf](https://www.ibm.com/servers/resourcelink/svc00100.nsf/pages/zosv2r5sc342670/$file/eequ100_v2r5.pdf)), SC34-2670
- z/OS: Network Job Entry (NJE) Formats and Protocols ([https://www.ibm.com/servers/resourcelink/svc00100.nsf/pages/zosv2r5sa320988/\\$file/hasa600\\_v2r5.pdf](https://www.ibm.com/servers/resourcelink/svc00100.nsf/pages/zosv2r5sa320988/$file/hasa600_v2r5.pdf)), SA32-0988
- z/OS: IBM Tivoli Directory Server Plug-in Reference for z/OS ([https://www.ibm.com/servers/resourcelink/svc00100.nsf/pages/zosv2r5sa760169/\\$file/glpa300\\_v2r5.pdf](https://www.ibm.com/servers/resourcelink/svc00100.nsf/pages/zosv2r5sa760169/$file/glpa300_v2r5.pdf)), SA76-0169
- z/OS: Language Environment Concepts Guide ([https://www.ibm.com/servers/resourcelink/svc00100.nsf/pages/zosv2r5sa380687/\\$file/ceea800\\_v2r5.pdf](https://www.ibm.com/servers/resourcelink/svc00100.nsf/pages/zosv2r5sa380687/$file/ceea800_v2r5.pdf)), SA38-0687
- z/OS: Language Environment Debugging Guide ([https://www.ibm.com/servers/resourcelink/svc00100.nsf/pages/zosv2r5ga320908/\\$file/ceea100\\_v2r5.pdf](https://www.ibm.com/servers/resourcelink/svc00100.nsf/pages/zosv2r5ga320908/$file/ceea100_v2r5.pdf)), GA32-0908
- z/OS: Language Environment Programming Guide ([https://www.ibm.com/servers/resourcelink/svc00100.nsf/pages/zosv2r5sa380682/\\$file/ceea200\\_v2r5.pdf](https://www.ibm.com/servers/resourcelink/svc00100.nsf/pages/zosv2r5sa380682/$file/ceea200_v2r5.pdf)), SA38-0682
- z/OS: Language Environment Programming Reference ([https://www.ibm.com/servers/resourcelink/svc00100.nsf/pages/zosv2r5sa380683/\\$file/ceea300\\_v2r5.pdf](https://www.ibm.com/servers/resourcelink/svc00100.nsf/pages/zosv2r5sa380683/$file/ceea300_v2r5.pdf)), SA38-0683
- z/OS: Language Environment Runtime Messages ([https://www.ibm.com/servers/resourcelink/svc00100.nsf/pages/zosv2r5sa380686/\\$file/ceea900\\_v2r5.pdf](https://www.ibm.com/servers/resourcelink/svc00100.nsf/pages/zosv2r5sa380686/$file/ceea900_v2r5.pdf)), SA38-0686
- z/OS: Language Environment Writing Interlanguage Communication Applications ([https://www.ibm.com/servers/resourcelink/svc00100.nsf/pages/zosv2r5sa380684/\\$file/ceea400\\_v2r5.pdf](https://www.ibm.com/servers/resourcelink/svc00100.nsf/pages/zosv2r5sa380684/$file/ceea400_v2r5.pdf)), SA38-0684
- z/OS: MVS Program Management Advanced Facilities ([https://www.ibm.com/servers/resourcelink/svc00100.nsf/pages/zosv2r5sa231392/\\$file/ieab200\\_v2r5.pdf](https://www.ibm.com/servers/resourcelink/svc00100.nsf/pages/zosv2r5sa231392/$file/ieab200_v2r5.pdf)), SA23-1392
- z/OS: MVS Program Management User's Guide and Reference ([https://www.ibm.com/servers/resourcelink/svc00100.nsf/pages/zosv2r5sa231393/\\$file/ieab100\\_v2r5.pdf](https://www.ibm.com/servers/resourcelink/svc00100.nsf/pages/zosv2r5sa231393/$file/ieab100_v2r5.pdf)), SA23-1393



## **XL C++ for z/VM**

- [XL C/C++ for z/VM: Runtime Library Reference, SC09-7624](#)
- [XL C/C++ for z/VM: User's Guide, SC09-7625](#)



# Index

## Special Characters

:ALTCNTRL. [628](#)  
:APARPFX. [630](#)  
:APPID. [627](#)  
:AXLIST. [628](#)  
:BCOMPNAME. [627](#)  
:BLD section, product parameter file [641](#)  
:BLD. [642](#)  
:BLDID. [627](#)  
:BLDREQ record (build lists) [150](#)  
:CKAUX. [629](#)  
:CKGEN. [629](#)  
:CKSDI. [629](#)  
:CKVV. [629](#)  
:CNTRL. [628](#)  
:CNTRLOP section, product parameter file [625](#)  
:CNTRLOP. [626](#)  
:COMPLST. [623](#)  
:COREQ [583](#), [714](#)  
:DABBV sction, product parameter file [644](#)  
:DABBV. [644](#)  
:DCL section, product parameter file [631](#)  
:DCL. [632](#)  
:DEPS [577](#), [589](#)  
:DREQ [595](#), [664](#), [688](#)  
:DREQDEPS [589](#)  
:EBLD. [642](#)  
:ECNTRLOP. [630](#)  
:EDABBV. [644](#)  
:EDCL. [632](#)  
:EMDA. [633](#)  
:EOBJNAME record (build lists) [152](#)  
:EP2P. [636](#)  
:ERECINS. [637](#)  
:ERECSER. [640](#)  
:EXCLIST. [628](#)  
:FORMAT record (build lists) [146](#)  
:GBLDREQ record (build lists) [150](#)  
:GGLOBAL record (build lists) [151](#)  
:GLOBAL record (build lists) [151](#)  
:GOBJPARM record (build lists) [152](#)  
:HARDREQ [583](#), [714](#)  
:IFREQ [583](#), [595](#), [664](#), [688](#), [714](#)  
:LIBNAME record (build lists) [147](#)  
:LOG. [627](#)  
:MDA section, product parameter file [633](#)  
:MDA. [633](#)  
:NLS. [628](#)  
:NPRE [595](#), [664](#), [688](#)  
:OBJNAME records (build lists) [148](#)  
:OPTIONS record (build lists) [149](#)  
:OUTREQS [577](#)  
:OVERLST. [624](#)  
:P2P. [636](#)  
:PARTID record (build lists) [149](#)  
:PREREQ [583](#), [595](#), [664](#), [687](#), [714](#)

:PRODESC. [627](#)  
:PRODID [595](#)  
:PRODID. [624](#), [655](#)  
:PTFPFX. [630](#)  
:PTFREQS [595](#)  
:RECID. [627](#)  
:RECINS section, product parameter file [637](#)  
:RECINS. [637](#)  
:RECSER section, product parameter file [639](#)  
:RECSER. [640](#)  
:RECVALL. [627](#)  
:REQ [595](#), [664](#), [688](#)  
:RETAIN. [630](#)  
:SETUP. [628](#)  
:SLVI. [628](#)  
:SUBHARDREQ [583](#)  
:SUBIF [583](#), [595](#)  
:SUBREQ [583](#), [595](#)  
:SUP [583](#), [595](#), [664](#), [688](#), [714](#)  
:SUPBY [577](#), [589](#)  
:UPDTID. [629](#)  
:USEREXIT. [630](#)  
:VERSION. [627](#)  
./DELETE [646](#)  
./INSERT [646](#)  
\$DASD\$ CONSTS file [367](#)  
\$VMFINS \$MSGLOG [14](#)

## A

accessing  
    minidisks read-only, VMFSETUP [502](#), [634](#)  
action (F10), Make Override Panel [37](#)  
action bar  
    help, Make Override Panel [42](#)  
    using [37](#)  
add  
    a new release of a product [63](#)  
    a single product [64](#)  
    local AUX file entries to Software Inventory [188](#)  
    local modifications [101](#)  
    products to your system [49](#)  
    several products [50](#), [65](#)  
    single product [49](#)  
Add Segment Definition Panel [514](#)  
additional syntax notation [142](#)  
APAR Abstract(s) query output [210](#)  
APAR, definition [89](#)  
APARFIX command (ServiceLink) [101](#)  
APARS contained in PTFs [181](#)  
appid SRVAPPS [716](#)  
appid VVTlvld [721](#)  
appliance servers  
    SMAPI [362](#)  
apply  
    a PTF [110](#)  
    service [105](#)

- apply enablement status
  - [DELETED 691](#)
  - [DISABLED 691](#)
  - [DISABSYNC 691](#)
  - [ENABLED 691](#)
  - [ENABSYNC 691](#)
  - [INSTALLED 691](#)
- apply list
  - containing requisites of a PTF [190](#)
  - example [133](#)
  - overview [133](#)
  - SNA service [763](#)
  - syntax [133](#)
- apply missing service
  - status table (SRVAPPS) [716](#)
- apply processing, sample queries [178](#), [181](#)
- apply status
  - [APPLIED 691](#), [716](#)
  - [DELETED 691](#)
  - [REMOVED 716](#)
  - [SUPED 691](#), [717](#)
- apply status table [111](#)
- APPLY string [106](#)
- ASMAHL
  - use with VMFASM [300](#)
  - use with VMFHLASM [394](#)
- assembly functions [117](#)
- assignments
  - Make Override Panel function keys [35](#)
  - Query NSS Map Window function keys [521](#)
  - Segment Definition Panel function keys [519](#)
  - Segment Map Panel function keys [511](#)
  - VMFINFO panel function keys [200](#), [201](#)
  - VMFVIEW EXEC function keys [618](#)
- associative stem input and output [529](#)
- asterisk in input field, Make Override Panel [34](#)
- auxiliary control file syntax [120](#)

## B

- BASE disks [106](#)
- beginning again, Make Override Panel [39](#)
- BITMAP files
  - structure [365](#)
- BLD record, build product definition section, PPF [642](#)
- blidid SRVBLDS [717](#)
- build
  - example [77](#)
  - products with VMFINS [77](#)
  - recovering from errors [411](#)
  - the serviced parts [105](#)
  - using VMFINS BUILD
    - example [77](#)
    - using the PPF operand [77](#)
  - using VMFINS, overview [77](#)
  - when to perform [77](#)
- BUILD disks [106](#)
- build list
  - additional syntax notation [142](#)
  - comments [152](#)
  - file type [141](#)
  - format 1
    - example [144](#)
    - overview [141](#)

- build list (*continued*)
  - format 1 (*continued*)
    - syntax [142](#)
  - format 2
    - object names [148](#)
    - overview [141](#)
    - wildcard objects [148](#)
  - format 2 and 3
    - general information [146](#)
    - specifying object parameters [148](#)
    - specifying part options [149](#)
    - syntax [144](#)
  - format 3
    - object names [148](#)
    - overview [141](#)
  - overview [112](#), [141](#)
  - build list option, definition [326](#)
  - build processing, sample queries [178](#), [183](#)
  - build product definition
    - BLD record [642](#)
    - example [643](#)
    - in the product parameter file [641](#)
    - syntax [641](#)
  - Build Requirements Query output [219](#)
  - build requisite processing [112](#)
  - build status
    - service-level
      - [BUILDALL 719](#)
      - [BUILT 719](#)
      - [BYPASSED 719](#)
      - [DELETE 719](#)
      - [DELETED 719](#)
      - [ERROR 719](#)
      - [MANUAL 719](#)
      - [SERVICED 719](#)
    - system-level
      - [BUILT 693](#)
      - [DELETE 693](#)
      - [DELETED 693](#)
      - [ERROR 693](#)
      - [SUPED 693](#)
  - build status information
    - service-level software inventory [173](#)
    - system-level software inventory [165](#)
  - build status table [112](#)
  - BUILD string [106](#)
  - build target, definition [326](#)
  - building
    - APPLY list containing requisites of a PTF [190](#)
    - callable services libraries [326](#)
    - CMS/DOS phase libraries [331](#)
    - CSLLIBs [326](#)
    - DOS libraries [331](#)
    - EXCLUDE list of dependents of a PTF [191](#)
    - generated objects [333](#)
    - LOAD libraries [341](#)
    - LOADLIBs [341](#)
    - MACLIBs [344](#)
    - MACRO libraries [344](#)
    - nuclei [352](#)
    - objects serviced by complete replacement [337](#)
    - saved segments [355](#)
    - TXTLIBs [359](#)
  - built, status of [179](#)

bypassing a tape file during receive [638](#)  
bypassing build lists [642](#)

## C

calculating DASD space requirements [33](#), [367](#)  
callable services libraries, building [326](#)  
cancel (F12)  
    Make Override Panel [37](#)  
    VMFINFO Panel [201](#)  
catalog of parts, VMSES PARTCAT [703](#)  
CD-ROM format, VM/ESA installation [753](#)  
Change Segment Definition Panel [513](#)  
changing  
    command defaults [48](#)  
    defaults, product installation [53](#), [68](#)  
    defaults, product migration [33](#)  
    entry format, Make Override Panel [46](#)  
    minidisk to SFS entry format [46](#)  
    product installation defaults [33](#), [53](#), [68](#)  
    SFS Directory File Pool Name [47](#)  
    Software Inventory to SFS directory [221](#)  
    the default system name [660](#)  
    the Software Inventory default file name [660](#)  
    the Software Inventory defaults [660](#)  
    VMFINS make override prompts [48](#)  
checking  
    for matching AUX files and version vector tables [188](#)  
CHKAPARS EXEC [234](#)  
choosing operand for VMFINS [17](#)  
circumventing a problem through z/VM service [89](#)  
circumvention, definition [89](#)  
closer look at  
    the VMFAPPLY EXEC [109](#)  
    the VMFBLD EXEC [112](#)  
    the VMFREC EXEC [108](#)  
CMS UPDATE command [122](#)  
CMS ZAPTEXT EXEC [781](#)  
CMS/DOS phase libraries, building [331](#)  
colon (:) on Make Override Panel [34](#)  
command  
    defaults, VMFINS, changing [48](#)  
    requirements for using VMSES/E [227](#)  
    syntax, reading [227](#)  
    understanding syntax diagrams [227](#)  
command (F2), Make Override Panel [35](#)  
command syntax, understanding [227](#)  
comments  
    format 1 build list [143](#)  
    in build lists [152](#)  
    product parameter file [621](#)  
Compare Table Contents Panel [218](#)  
Compare Table Contents Query output [218](#)  
comparing two Software Inventory tables  
    building apply list of PTFs received and not applied [189](#)  
    creating apply list from two SRVAPPS tables [190](#)  
    using VMFSIM COMPTBL [189](#)  
compname, definition [19](#)  
component area  
    source product parameter file [624](#)  
    usable form PPF [654](#)  
Component Name - Help Panel [202](#)  
confirmation, new PPF override file [41](#)  
conflict (F11), Make Override Panel [37](#)

console listing, VMFINS [14](#)  
console spooling, VMFINS [14](#)  
consolidating levels of the database [114](#)  
control file  
    AUX record [119](#)  
    auxiliary [120](#)  
    comments in [119](#)  
    extensions [119](#)  
    how VMSES/E uses [117](#)  
    MACS record [118](#)  
    secondary files [117](#)  
    syntax [118](#)  
    what is it? [117](#)  
    when used [118](#)  
control file extension [119](#)  
control options section  
    source PPF [625](#)  
    usable form PPF [625](#)  
control parameters for VMSES/E execs, PPF [625](#)  
controlling VMFINS prompts for creating overrides [48](#)  
converting DASD space requirements [33](#), [367](#)  
COR  
    corrective tape [92](#)  
    descriptor file [126](#)  
    document [125](#)  
correcting a problem through z/VM service [89](#)  
corrective  
    service (object code) [765](#)  
    tape (COR) [92](#)  
corrective service  
    installing [95](#)  
create text decks [124](#)  
creating  
    apply list from two SRVAPPS tables, example [190](#)  
    callable services libraries [326](#)  
    CMS/DOS phase libraries [331](#)  
    CSLLIBs [326](#)  
    DDR image files [330](#)  
    DOSLIBs [331](#)  
    executable modules [347](#)  
    executable modules, CPLINK or BIND [349](#)  
    generated objects [333](#)  
    LOAD libraries [341](#)  
    LOADLIBs [341](#)  
    MACLIBs [344](#)  
    MACRO libraries [344](#)  
    MODULES [347](#)  
    nuclei [352](#)  
    objects serviced by complete replacement [337](#)  
    objects with VMFBLD [325](#)  
    saved segments [355](#)  
    test copy with VMFBLD [308](#)  
    TEXT libraries [359](#)  
    text objects serviced by complete replacement [340](#)  
    TXTLIBs [359](#)  
    updated source files [122](#)  
CSLLIBs, creating [326](#)  
current working file, Make Override Panel [41](#)  
customer local modifications [101](#)  
customer local service [101](#)

## D

DABBV record, product parameter file [644](#)

DASD space requirements, converting [367](#)  
 data records, product parameter file [621](#)  
 data structure  
   of Software Inventory tables [659](#)  
   product parameter file [621](#)  
 database, VMSES/E [105](#)  
 DCL record, variable declarations section, PPF [632](#)  
 deciding operands to use, VMFINS [17](#)  
 defaults  
   changing  
     Make Override Panel, example [68](#)  
     product installation defaults [33](#)  
     SFS directory name [47](#)  
     Software Inventory disk [660](#)  
     Software Inventory to SFS directory [221](#)  
     system name [660](#)  
     VMFINS command option defaults [48](#)  
   SFS directory name [46](#)  
   Software Inventory [660](#)  
 define  
   build processing for a product, PPF [641](#)  
   file type abbreviations for products [644](#)  
   layout of installation tape for product, PPF [637](#)  
   layout of service tapes and envelopes for products, PPF [639](#)  
 delete and insert data, updateable sections of PPF [645](#)  
 delete override control record [646](#)  
 deleting  
   a record in :MDA section of PPF, example [656](#)  
   build lists from product parameter files [642](#)  
   example [81](#)  
   objects from build lists [146](#)  
   products [81](#)  
   recovering from errors [416](#)  
   with the PPF operand [81](#)  
 delimiters, Software Inventory syntax diagrams [659](#)  
 DELTA string [106](#)  
 description of APARs received [181](#)  
 description of products or components received [178](#)  
 description table [109](#)  
 determining  
   all parts serviced by a PTF [182](#)  
   all service applied to a part [183](#)  
   if a product can be deleted [30](#)  
   if a product can be installed [24](#)  
   if a product can be migrated [27](#)  
   local modifications requiring rework [124](#)  
   objects requiring manual building [184](#)  
   objects to build [184](#)  
   parts serviced by an APAR [183](#)  
   prerequisites for a product, example [192](#)  
   the current service level, example [189](#)  
   the dependents of a product, example [193](#)  
   the PTFs dependent on a PTF, example [193](#)  
   which operand to use [17](#)  
 diagrams, syntax [227](#)  
 disk management  
   VMFQMDA EXEC [116](#)  
   VMFSETUP EXEC [116](#)  
 DOS libraries, building [331](#)  
 DOSLIBs, creating [331](#)

## E

ellipsis following an option, Make Override Panel [34](#)  
 emergency service [101](#)  
 entering TDATA statements [526](#)  
 erasing all entries, Make Override Panel [39](#)  
 error recovery  
   build [411](#)  
   delete [416](#)  
   install [432](#)  
   migrate [440](#)  
 errors  
   during build, recovering [411](#)  
   during delete, recovering [416](#)  
   during install, recovering [432](#)  
   during migrate, recovering [440](#)  
 ESO (Expanded Service Option) [93](#)  
 established VMFINS command option defaults [48](#), [139](#)  
 example  
   :MDA section of source PPF [635](#)  
   :MDA section of usable form PPF [635](#)  
   apply list [133](#)  
   build  
     with the PPF operand [77](#)  
   build product definition section, PPF [643](#)  
   COR descriptor file [126](#)  
   default SFS directory name, Make Override Panel [46](#)  
   delete  
     with the PPF operand [81](#)  
   deleting records, override PPF (:MDA section) [656](#)  
   exclude list [134](#)  
   F4=Expand Dirid [46](#)  
   file type abbreviations extensions section, PPF [644](#)  
   format 1 build list [144](#)  
   format 2 build lists [152](#)  
   format 3 build list [154](#)  
   help, Make Override Panel [42](#)  
   inserting a record, override PPF (:MDA section) [655](#)  
   install  
     products with VMFINS [54](#)  
     with the PPF operand [54](#)  
   loadable unit section, PRODPART file [665](#)  
   Make Override Panel [34](#)  
   override PPF header area [653](#)  
   overrides to product parameter files [655](#)  
   product contents directory, COR tape [127](#)  
   product contents directory, Install/RSU tape [127](#)  
   product saved segment build list [153](#)  
   program level file [128](#)  
   RSU descriptor file [126](#)  
   select data file [129](#)  
   select data file for saved segments [130](#)  
   split-screen XEDIT session [70](#)  
   system saved segment build list [153](#)  
   tape descriptor files [126](#)  
   updating a record, multi-level override to PPF [657](#)  
   updating a record, single-level override to PPF [656](#)  
   variable declarations section, PPF [632](#), [636](#)  
   version vector table [722](#)  
   view-screen session [74](#)  
   VMFNLS LANGLIST file [155](#)  
   VMFSGMAP, displaying a segment definition record [523](#)  
   VMFSGMAP, displaying the Segment Map [522](#)  
   VMSESE PROFILE [132](#)

- exclude list
  - example [134](#)
  - of dependents of a PTF [191](#)
  - overview [134](#)
  - SNA service [764](#)
  - syntax [134](#)
- EXEC statements, format 1 build lists [143](#)
- EXEC2 statements, format 1 build lists [143](#)
- EXECs, other VMSES/E [113](#)
- exit (F3)
  - Make Override Panel [36](#)
  - VMFINFO Panel [200](#)
- EXPAND command [783](#)
- expand dirid (F4)
  - example [46](#)
  - Make Override Panel [36](#)
- Expanded Service Option [93](#)

**F**

- F key help, Make Override Panel [44](#)
- figures
  - :BLD section, source PPF [643](#)
  - :CONTRLOP section, source PPF [630](#)
  - :DABBV section, source PPF [644](#)
  - :DCL section, source PPF [632](#)
  - :MDA section, source PPF [635](#)
  - :MDA section, usable form PPF [635](#)
  - :P2P section, source PPF [636](#)
  - :RECINS section, source PPF [639](#)
  - :RECSER section, source PPF [641](#)
- apply algorithm [110](#)
- apply list [133](#)
- apply list syntax [133](#)
- build list syntax, format 1 [142](#)
- build list, format 1 [144](#)
- building with VMFINS, PPF operand
  - \$VMFINS \$MSGLOG file, example [79](#)
  - VMFINS CONSOLE file, example [79](#)
- command syntax
  - VMFINS BUILD [405](#)
  - VMFINS DELETE [412](#)
  - VMFINS INSTALL [425](#)
- component area, source PPF [624](#)
- component area, usable form PPF [655](#)
- control file structure [122](#)
- control file structure using version vector tables [123](#)
- COR descriptor file, example [126](#)
- deleting with VMFINS, PPF operand
  - \$VMFINS \$MSGLOG file, example [84](#)
  - 5654A22C ERASE file, example [83](#)
  - VMFINS CONSOLE file, example [85](#)
- exclude list [134](#)
- exclude list syntax [134](#)
- format 2 and 3 build list syntax [144](#)
- format 2 build list [152](#)
- format 3 build list [154](#)
- header area, override PPF [653](#)
- header area, source PPF [624](#)
- High-Level Overview of VMFINS Command [9](#)
- installing with VMFINS, PPF operand
  - \$VMFINS \$MSGLOG file [58](#)
  - 5654A22C PLANINFO file [56](#)
  - printing the Memo-to-Users [54](#)

- figures (*continued*)
  - installing with VMFINS, PPF operand (*continued*)
    - VMFINS CONSOLE file [59](#)
    - VMFINS PRODLIST file [54](#)
  - Introducing VMSES/E [1](#)
  - Make Override Panel
    - action bar, using to save PPF override files [38](#)
    - changing installation settings for migration [68](#)
    - changing to SFS directory entry input [46](#)
    - default SFS directory name [46](#)
    - entering a SFS directory name, Expand Dirid window [47](#)
    - example [34](#)
    - Expand Dirid window [46](#)
    - file options [38](#)
    - function key assignments [35](#)
    - getting basic instructions for [45](#)
    - help on F6 function key [44](#)
    - help on function keys window [44](#)
    - help on help window [43](#)
    - help options [42](#)
    - new override file name [41](#)
    - returning to Make Override Panel after saving PPF override [41](#)
    - save as.... window [40](#)
    - saving PPF override with a new name [40](#)
    - selecting a file option [39](#)
    - selecting a help option [42](#)
    - selecting help on F6=Mdisk or SFS Dir function key [44](#)
    - using the action bar [37](#)
  - override PPF, header area [653](#)
  - PLANINFO file, examples
    - created by a VMFINS DELETE with the PLAN option [31](#)
    - created by a VMFINS INSTALL with the PLAN option [25](#)
    - created by a VMFINS MIGRATE with the PLAN option [28](#)
    - created during a VMFINS MIGRATE [28](#)
    - created during VMFINS DELETE [31](#)
  - PPF entry in VMFINS PRODLIST file [18](#)
  - PRODPART file [675](#)
  - PRODPART file, header section syntax [661](#)
  - product contents directory, COR tape [127](#)
  - product contents directory, Install/RSU tape [127](#)
  - product contents directory, PUT tape [127](#)
  - product parameter file relationship [159](#), [622](#)
  - product saved segment build list [153](#)
  - program level file [128](#)
  - RSU descriptor file, example [126](#)
  - saved segment build parameters syntax [517](#)
  - select data file for saved segments [130](#)
  - select data file syntax [128](#)
  - selecting the Memo-to-Users for printing [22](#)
  - serviceable part to usable form relationship [90](#)
  - Servicing a CMS MODULE by Module Replacement [91](#)
  - Servicing a CMS MODULE by Text Replacement [91](#)
  - Software Inventory data structure [525](#), [659](#)
  - source PPF overall syntax [623](#)
  - source PPF, BLD section override syntax [651](#)
  - source PPF, BLD section syntax [641](#)
  - source PPF, CNTRLOP section override syntax [646](#)
  - source PPF, CONTRLOP section syntax [626](#)

figures (*continued*)

- source PPF, DABBV section override syntax [652](#)
- source PPF, DABBV section syntax [644](#)
- source PPF, DCL section override syntax [647](#)
- source PPF, DCL section syntax [631](#)
- source PPF, MDA section override syntax [648](#)
- source PPF, MDA section syntax [633](#)
- source PPF, RECINS section override syntax [649](#)
- source PPF, RECINS section syntax [637](#)
- source PPF, RECSEB section override syntax [650](#)
- source PPF, RECSEB section syntax [639](#)
- system saved segment build list [153](#)
- system-level and service-level Software Inventories [12](#)
- tailoring product files, migrate
  - customized files message screen [69](#)
  - CUTC prefix command example [71](#)
  - file after CUTC and PLACE prefix commands, example [72](#)
  - final tailorings phase screen, example [72–76](#)
  - PLACE prefix command example [71](#)
  - Restore Tailorings Phase screen [69](#)
  - split-screen XEDIT session [70](#)
  - view-screen session, sample [74](#)
- tape descriptor files [126](#)
- Updating a CMS MODULE Serviced by Updates [91](#)
- Updating a MACRO Using Parts Serviced by Updates Only [91](#)
- usable form PPF, component area [655](#)
- usable form PPF, overall syntax [654](#)
- VMFINS command defaults
  - VMFINS DEFAULTS file, example [48](#), [139](#)
- VMFINS PRODLIST file [18](#)
- VMFINS PRODLIST file updated by LIST operand and PLAN option [21](#)
- VMFMGRD EXEC example [114](#)
- VMFNLS LANGLIST file [155](#)
- VMFNLS LANGLIST file syntax [154](#)
- VMFPPF EXEC example [115](#)
- VMSES/E Installation/Service Tool [7](#), [225](#)
- VMSES/E Overview [3](#)
- VMSESE PROFILE [132](#)
- VMSESE PROFILE syntax [131](#)
- figuring DASD space requirements [33](#), [367](#)
- file (F5), VMFINFO Panel [201](#)
- file input to VMFSIM [527](#)
- FILE option (VMFSIM QUERY) [527](#)
- file options, Make Override Panel [38](#)
- file syntax
  - \$PTFPART file
    - header section [705](#)
    - parts definition section [708](#)
    - requisite section [706](#)
  - additional notation [142](#)
  - apply list [133](#)
  - auxiliary control file (AUX) [120](#)
  - control file [118](#)
  - control file extension [119](#)
  - exclude list [134](#)
  - file type abbreviation table (VM SYSABRVT) [702](#)
  - format 1 build list [142](#)
  - format 2 build list [144](#)
  - format 3 build list [144](#)
  - parts catalog (VMSES PARTCAT) [703](#)
  - PRODPART file

file syntax (*continued*)

- PRODPART file (*continued*)
  - header section [661](#)
  - loadable unit section [662](#)
  - overview [660](#)
  - parts section [665](#)
  - product parameters section [667](#)
  - saved segment definitions section [670](#)
- product parameter file [621](#)
- saved segment data file (SEGDATA) [677](#)
- select data file [128](#)
- service-level apply status table (appid SRVAPPS) [716](#)
- service-level build status table (bldid SRVBLDS) [717](#)
- service-level description table (prodid SRVDESCT) [712](#)
- service-level production status table (prodid SRVPROD) [720](#)
- service-level receive status table (prodid SRVRECS) [715](#)
- service-level requisite table (prodid SRVREQT) [713](#)
- Software Inventory tables [659](#)
- source product parameter file
  - BLD Section [641](#)
  - BLD section override [651](#)
  - CNTRLOP Section [626](#)
  - CNTRLOP section override [646](#)
  - DABBV Section [644](#)
  - DABBV section override [652](#)
  - DCL Section [631](#)
  - DCL section override [647](#)
  - MDA Section [633](#)
  - MDA section override [648](#)
  - overview [623](#)
  - RECINS Section [637](#)
  - RECINS section override [649](#)
  - RECSEB Section [639](#)
  - RECSEB section override [650](#)
- system-level apply status table (VM SYSAPPS) [690](#)
- system-level build status table (VM SYSBLDS) [692](#)
- system-level description table (VM SYSDESCT) [684](#)
- system-level receive status table (VM SYSRECS) [688](#)
- system-level requisite table (VM SYSREQT) [686](#)
- understanding how to read it [227](#)
- usable form product parameter file [654](#)
- version vector table (appid VVTIvld) [721](#)
- VMFNLS LANGLIST [154](#)
- VMSESE PROFILE [131](#)
- file type abbreviation table (VM SYSABRVT) [170](#)
- file type abbreviations extensions section
  - example [644](#)
  - in the product parameter file [644](#)
  - syntax [644](#)
- File Type Abbreviations Panel [219](#)
- File Type Abbreviations Query output [220](#)
- file type, build list [141](#)
- files
  - created during VMFINS INSTALL and MIGRATE [13](#)
  - for your information and use (VMFINS commands) [14](#)
  - input and output [735](#)
  - on installation media [13](#)
  - retailoring [68](#)
  - SERVICE \$PRODS [135](#)
- files, moving into BFS [336](#)
- finding out which products are on installation media [18](#)
- first install
  - with PPF operand [54](#)



- fixes using ZAPs (object code) [765](#)
- format 1 build list
  - comments [143](#)
  - example [144](#)
  - EXEC and EXEC2 statements [143](#)
  - loader cards [143](#)
  - overview [141](#)
  - serviceable parts records [143](#)
  - syntax [142](#)
- format 2 and 3 build lists
  - specifying object parameters [148](#)
  - specifying part options [149](#)
- format 2 build list
  - object names [148](#)
  - overview [141](#)
  - syntax [144](#)
  - wildcard objects [148](#)
- format 3 build list
  - object names [148](#)
  - overview [141](#)
  - syntax [144](#)
- function keys
  - getting help on, Make Override Panel [44](#)
  - Make Override Panel [35](#)
  - Query NSS Map Window [521](#)
  - Segment Definition Panel [519](#)
  - Segment Map Panel [511](#)
  - VMFINFO Panel [200](#)
  - VMFVIEW EXEC [618](#)

## G

- GENCPBLS EXEC [237](#)
- general help
  - on the Make Override Panel [45](#)
  - on the VMFINFO Panel [201](#)
- general information
  - format 2 and 3 build lists [146](#)
  - Make Override Panel [34](#)
  - VMFINFO command [200](#)
  - VMFSGMAP panels [507](#)
- generated objects, building [333](#)
- get
  - help [42](#)
  - help on help [43](#)
  - information for decisions [18](#)
  - list of products on installation media [18](#)
  - list of products that can be migrated [18](#)
  - Memo-to-Users [21](#)
  - online help
    - Make Override Panel [45](#)
    - split-screen XEDIT session [70](#)
    - view-screen session [74](#)
  - products installed with VMFINS [49](#)
  - started with VMFINS [10](#)

## H

- header area
  - override product parameter file [653](#)
  - source product parameter file [623](#)
- help
  - examples, Make Override Panel [42](#)

- help (*continued*)
  - F keys, Make Override Panel [44](#)
  - F6=mdisk to sfs dir [46](#)
  - general, Make Override Panel [34](#)
  - general, VMFINFO Panel [200](#)
  - mdisk to sfs dir (F6) [46](#)
  - on function keys window, Make Override Panel [44](#)
  - on function keys, Make Override Panel [44](#)
  - on help, Make Override Panel [43](#)
  - online
    - Make Override Panel [45](#)
    - split-screen XEDIT session [70](#)
    - view-screen session [74](#)
    - VMFINFO panel [201](#)
  - options, Make Override Panel [42](#)
- help (F1)
  - Make Override Panel [35](#)
  - VMFINFO Main Panel [200](#)
- HELP, online [227](#)
- HELP, using online [227](#)
- highlighting, Make Override Panel [34](#)
- how to build products using VMFINS [77](#)
- how to delete using VMFINS [81](#)
- how to get VMFINS CONSOLE file [14](#)
- how to install using VMFINS [54](#)
- how to migrate using VMFINS [63](#)
- how VMFINS MIGRATE works with the ADD option [63](#)
- how VMFINS MIGRATE works with the REPLACE option [65](#)
- how VMFSIM processes tagged data [527](#)
- how VMSES/E uses control files [117](#)

## I

- IBM local service [101](#)
- identifiers
  - for message modules [749](#)
- identifying
  - MACLIBs [124](#)
  - requisites, specifying [662](#)
  - saved segments to be built [354](#)
  - symbolic strings of minidisks or SFS directories, PPF [633](#)
  - system objects to be built [354](#)
  - variables and values assigned, PPF [631](#)
- identifying the latest version of a part
  - determining the current service level [189](#)
  - using VMFSIM GETLVL [189](#)
- INFO operand
  - output file [18](#)
  - using [18](#)
- initializing Software Inventory tables [195](#)
- input file for VMFBLD command with LIST operand [311](#)
- input files for VMSES/E EXECs [735](#)
- input to other commands, LIST operand [20](#)
- input to VMFSIM from a file [527](#)
- inputs to VMFSIM [177](#), [526](#)
- insert and delete data, updateable sections of PPF [645](#)
- insert override control record [646](#)
- inserting a record in :MDA section of PPF, example [655](#)
- install
  - and replacing products with VMFINS [51](#)
  - and replacing several products with VMFINS [52](#)
  - changing product installation defaults [53](#)
  - errors, recovering [432](#)

install (*continued*)  
examples [54](#)  
local service and modifications [101](#)  
new products [49](#)  
product, recommended approach [49](#)  
several products [50](#)  
single product [49](#)  
using PPF operand, example [54](#)  
using VMFSIM EXEC during [61](#)  
with the PPF operand, example [54](#)  
with VMFINS [49](#)

installation  
of corrective service [95](#)  
scenarios [54](#)

installation descriptor file [126](#)

installation media  
products on [19](#)

INSTFPP EXEC [723](#)

introduction  
override product parameter file [160](#)  
software inventory [11](#)  
source product parameter file [159](#)  
temporary product parameter file [160](#)  
to the product parameter file [159](#)  
to the VMFINS EXEC [9](#)  
to VMSES/E [9](#)  
usable form product parameter file [160](#)

## L

latest level serviceable parts  
creating executable modules [347](#)  
creating executable modules with CPLINK or BIND [349](#)  
creating text objects serviced by complete replacement  
[340](#)

LIST operand, VMFBLD command input file [311](#)

LIST operand, VMFINS command [20](#)

listing dependents  
for products [193](#)  
for PTFs [192](#)  
of a product, example [193](#)  
PTFs on another PTF [193](#)

listing requisites  
determining prerequisites for a product [192](#)  
for products [192](#)  
for PTFs [191](#)

loadable unit, definition [662](#)

loader cards, format 1 build list [143](#)

LOADLIBs, creating [341](#)

local modifications  
adding local replacement files to Software Inventory  
[194](#)  
adding local source update modifications to Software  
Inventory [194](#)  
adding to the Software Inventory [188](#), [193](#)  
installing [101](#)  
procedure overview [102](#)  
requiring rework, determining [124](#)  
tracking number [89](#)  
using the LOGMOD option [193](#), [298](#), [388](#), [395](#), [449](#), [534](#)  
using the VMFSIM LOGMOD command [193](#)  
VMFREPL, used with local mods replacement parts [494](#)

local service  
definition [101](#)

local service (*continued*)  
electronically received [101](#)  
preparation [101](#)  
LOCAL string [106](#)  
LOCALMOD EXEC [245](#)

## M

MACLIBs  
creating [344](#)  
identifying [124](#)

MACS records [118](#)

main panel, VMFINFO command [203](#)

maintenance-level information (service-level software) [172](#)

Make Override Panel  
asterisk as input [34](#)  
colon preceding input field [34](#)  
default SFS directory name [46](#)  
ellipsis following an option [34](#)  
example [34](#)  
Expand Dirid window [46](#)  
F1=Help [35](#)  
F12=Cancel [35](#)  
file options [38](#)  
function keys [35](#)  
general instructions [34](#)  
getting online help [45](#)  
highlighting [34](#)  
more: [34](#)  
naming conventions [34](#)  
new override file name [41](#)  
period preceding input field [34](#)  
using [33](#)

make override prompts, changing VMFINS [48](#)

manage  
disks for the service database [116](#)  
objects [116](#)  
product parameter files [115](#)  
saved segments [116](#)

MDA record, source product parameter file [634](#)

mdisk or sfs dir (F6)  
getting help [46](#)  
overview [36](#)

Memo-to-Users  
printing [21](#)  
using while applying service [127](#)

merge function  
example [114](#)  
VMFMFRDSK EXEC [114](#)  
with VMSES/E [114](#)

merge log (SNA service) [760](#)

message examples, notation used in [xxxi](#)

message logs  
viewing [117](#)

messages  
module identifiers [749](#)

MIGPvrn [682](#)

migrating  
new products, overview [63](#)  
recovering from errors [440](#)  
retailoring product files after [68](#)  
several products [67](#)  
several products, overview [65](#)

migration



panels (*continued*)

- PTF/APAR Query Panel [206](#)
- Query NSS Map Window [520](#)
- Query Output - APAR Abstract(s) [210](#)
- Query Output - Build Requirements [219](#)
- Query Output - Compare Table Contents [218](#)
- Query Output - File Type Abbreviations [220](#)
- Query Output - Minidisk/Directory Access [217](#)
- Query Output - Object Build Dependencies [214](#)
- Query Output - Object Build Requisites [214](#)
- Query Output - Object Part Handler/Target [215](#)
- Query Output - Object Status [213](#)
- Query Output - Part Service History [216](#)
- Query Output - Part Service Level [215](#)
- Query Output - Product Dependencies [205](#)
- Query Output - Product Description [204](#)
- Query Output - Product Requisites [205](#)
- Query Output - Product Status [204](#)
- Query Output - PTF Dependencies/Superseding [209](#)
- Query Output - PTF Requisites/Supersedes [208](#)
- Query Output - PTF Serviceable Parts [210](#)
- Query Output - PTF Status [207](#)
- Query Output - PTF User Memo [209](#)
- Segment Definition Panel [513](#)
- Segment Map Panel [507](#)
- Serviceable Parts/Usable Forms Query Panel [211](#)
- VMFINFO [199](#)
- VMFINFO Main Panel [203](#)

parameters, object

- used by VMFBDCOM [339](#)

part handlers

- VMFBDBFS [336](#)
- VMFBDCLB [326](#)
- VMFBDCOM [337](#)
- VMFBDCPY [340](#)
- VMFBDDDR [330](#)
- VMFBDDLDB [331](#)
- VMFBDGEN [333](#)
- VMFBDLLB [341](#)
- VMFBDMLB [344](#)
- VMFBDMOD [347](#)
- VMFBDNUC [352](#)
- VMFBDPMD [349](#)
- VMFBDSBR [354](#)
- VMFBDSSEG [355](#)
- VMFBDSSE [362](#)
- VMFBDTLB [359](#)
- VMFBLD [325](#)
- VMFREC [108](#)

part options

- definition [326](#)
- specifying in build lists [149](#)
- used by VMFBDBFS [337](#)
- used by VMFBDCLB [328](#)
- used by VMFBDCOM [339](#)
- used by VMFBDCPY [341](#)
- used by VMFBDDLDB [332](#)
- used by VMFBDGEN [335](#)
- used by VMFBDLLB [342](#)
- used by VMFBDMLB [346](#)
- used by VMFBDMOD [348](#)
- used by VMFBDNUC [354](#)
- used by VMFBDPMD [351](#)

part options (*continued*)

- used by VMFBDSBR [355](#)
- used by VMFBDSSEG [357](#)
- used by VMFBDTLB [361](#)
- Part Service History Query output [216](#)
- Part Service Level Query output [215](#)
- parts catalog [703](#)
- parts records, format 1 build lists [143](#)
- patch facility
  - controlling patches [728](#)
  - HCPLDR compatibility
    - local service to ASSEMBLE files [733](#)
    - local service to TEXT files [732](#)
  - overview [728](#)
- patch update files [121](#)
- path to choose, VMFINS [17](#)
- period (.) on Make Override Panel [34](#)
- PF key assignments, default (VMFVIEW) [618](#)
- picking the correct VMFINS operand [17](#)
- plan
  - for a Product Service Upgrade [462](#)
  - results [24](#)
  - to delete a product with VMFINS [30](#)
  - to install a product with VMFINS [24](#)
  - to migrate a product with VMFINS [27](#)
  - with VMFINS [23](#)
- PLAN option, using [23](#)
- PLANINFO file
  - for delete [31](#)
  - for install [25](#)
  - for migrate [28](#)
- PPF Fileid - Help Panel [202](#)
- PPF management
  - example [115](#)
  - VMFOVER EXEC [115](#)
  - VMFPPF EXEC [115](#)
  - with VMSES/E [115](#)
- PPF operand
  - overview [20](#)
  - using
    - to build [77](#)
    - to delete [81](#)
    - to install [54](#)
- PPF, override syntax [653](#)
- ppfname ERASE file [15](#), [30](#)
- ppfname, definition [13](#)
- preface [xxix](#)
- prerequisites
  - determining for PTF [191](#)
- primary service tasks
  - applying service [105](#)
  - building the serviced parts [105](#)
  - overview [105](#)
  - receiving service [105](#)
- printing, memo-to-users [21](#)
- processing
  - build requisites [112](#)
  - by product format [11](#)
  - install tapes with VMFINS [10](#)
  - multiple service tapes [108](#)
  - product parameter files [622](#)
  - products with VMFINS [17](#)
- PROD operand
  - overview [20](#)

PROD operand (*continued*)  
   using [20](#)  
 prodid \$APPLIST [15](#)  
 prodid \$PPF File [13](#)  
 prodid SRVDE SCT [712](#)  
 prodid SRVPROD [720](#)  
 prodid SRVREQ T [713](#)  
 prodid, definition [13](#)  
 PRODPART file  
   example [675](#)  
   file syntax [660](#)  
   header section [661](#)  
   loadable unit section [662](#)  
   overview [166](#), [660](#)  
   parts section [665](#)  
   product parameters section [666](#)  
   saved segment definitions section [670](#)  
 product  
   applying service to [95](#)  
   product and service structure [89](#)  
   product contents directory [127](#)  
   product dependencies query, VMFINFO [205](#)  
   product dependency query [203](#)  
   product description query [203](#)  
   product files, retailing [68](#)  
   product information (system-level software) [165](#)  
   product installation parameters for changing [33](#)  
   product parameter file  
     BLD record [642](#)  
     build product definition section [641](#)  
     comments [621](#)  
     component area  
       in source PPF [624](#)  
       in usable form PPF [654](#)  
     control options section [625](#)  
     DABBV record [644](#)  
     data records [621](#)  
     data structure [621](#)  
     deleting build lists [642](#)  
     file structure [621](#)  
     file type abbreviations extensions section [644](#)  
     general description [159](#)  
     header area  
       in override PPF [653](#)  
       in source PPF [623](#)  
     minidisk/directory assignments section [633](#)  
     override area  
       examples [655](#)  
       in override PPF [653](#)  
       in source PPF [644](#)  
     override control records, examples [655](#)  
     override PPF, examples [655](#)  
     override syntax [653](#)  
     processing [622](#)  
     product parameter file, temporary syntax [654](#)  
     receive installation tape definition section [637](#)  
     receive service media definition section [639](#)  
     syntax [621](#)  
     updating a record  
       multiple-level override [657](#)  
       single-level override [656](#)  
     usable form, syntax [654](#)  
     variable declarations section [631](#)  
   product parameter file (SNA service) [759](#)  
   product queries, VMFINFO [203](#)  
   product requisites query, VMFINFO [205](#)  
   product service upgrade (PSU) [92](#)  
   product service upgrade, using [97](#)  
   product status query [203](#)  
   product, definition [662](#)  
   production mode, moving from test mode to [48](#)  
   production status  
     BUILT [720](#)  
     PUT2PROD [720](#)  
   products  
     adding [49](#)  
     on installation media [18](#), [19](#)  
     processing with VMFINS [17](#)  
     replacing [51](#), [65](#)  
   program level file  
     example [128](#)  
     overview [127](#)  
   program temporary fix (PTF)  
     definition [89](#)  
     within service structure [90](#)  
   prompts, changing VMFINS [48](#)  
   providing input to VMFSIM [177](#)  
   PSU (Product Service Upgrade)  
     effect on tailored files [97](#)  
     overview [92](#)  
     planning for use [97](#)  
     procedure [92](#)  
   PTF  
     determining prerequisites for [191](#)  
   PTF (program temporary fix)  
     contents [90](#)  
     definition [89](#)  
   PTF Dependencies/Superseding Query output [209](#)  
   PTF information (service-level software) [172](#)  
   PTF Parts File [109](#)  
   PTF Parts file (\$PTFPART) [705](#)  
   PTF Requisites/Supersedes Query output [208](#)  
   PTF Serviceable Parts query output [210](#)  
   PTF Status Query output [207](#)  
   PTF User Memo query output [209](#)  
   PTF/APAR Query Panel [206](#)  
   PUT2PROD EXEC [258](#)

**Q**  
 query the Software Inventory  
   is a specific APAR applied? [185](#)  
   using VMFSIM QUERY [177](#)  
   what are the direct requisites of a specific PTF? [185](#)  
   what do I have installed that relates to a specific product? [179](#)  
   what is the service history for a part? [186](#)  
   what options are coded in the PPF? [179](#)  
   what parts are affected by a specific PTF? [186](#)  
   which PTFs are applied? [185](#)  
   which PTFs have a direct requisite of a specific PTF? [184](#)  
   querying APARS applied [182](#)  
   querying multiple tables with FILE option [527](#)  
   querying PTFs applied [182](#)  
   querying status of an object [183](#)

## R

read-only access with VMFSETUP [634](#)  
reading  
  syntax diagrams [142](#), [227](#)  
  the Make Override Panel [34](#)  
  the VMFINFO Panel [200](#)  
receive installation tape definition section, PPF [637](#)  
receive processing, sample queries [177](#), [180](#)  
receive service definition section, PPF [639](#)  
receive status  
  COMMITTED [716](#)  
  DELETE [689](#), [691](#)  
  DELETED [689](#)  
  RECEIVED [689](#), [715](#)  
receive status of all PTFs on an APAR [181](#)  
receive status table [109](#)  
receiving output from VMFSIM [177](#)  
receiving service [105](#)  
receiving VMFINS CONSOLE file [14](#)  
recid SRVRECS [715](#)  
RECINS record, source product parameter file [637](#)  
recommendations  
  installing single product with VMFINS [49](#)  
  way to migrate a single product using VMFINS [64](#)  
  way to migrate several products using VMFINS [65](#)  
  way to replace an existing product with VMFINS [51](#)  
  way to replace an existing product with VMFINS and  
  keep tailorings [66](#), [67](#)  
recovering  
  from an unsuccessful build [411](#)  
  from an unsuccessful delete [416](#)  
  from an unsuccessful migration [440](#)  
  from errors during install [432](#)  
  service-level Software Inventories [196](#)  
recovering the system-level Software Inventory [195](#)  
recovery procedures  
  CHKAPARS EXEC [236](#)  
  during a VMFINS BUILD [411](#)  
  during a VMFINS DELETE [416](#)  
  during a VMFINS INSTALL [432](#)  
  during a VMFINS MIGRATE [440](#)  
  LOCALMOD EXEC [248](#)  
  PUT2PROD EXEC [260](#)  
  SERVICE EXEC [268](#)  
  VMFAPPLY EXEC [295](#)  
  VMFASM EXEC [302](#)  
  VMFBLD EXEC [325](#)  
  VMFCNVT EXEC [368](#)  
  VMFCOPY EXEC [372](#)  
  VMFERASE EXEC [380](#)  
  VMFHASM EXEC [393](#)  
  VMFHLASM EXEC [401](#)  
  VMFINFO EXEC [403](#)  
  VMFMGRD SK EXEC [447](#)  
  VMFNLS EXEC [455](#)  
  VMFOVER EXEC [458](#)  
  VMFPPF EXEC [460](#)  
  VMFQMDA EXEC [471](#)  
  VMFQOBJ EXEC [476](#)  
  VMFREC EXEC [483](#)  
  VMFREM EXEC [491](#)  
  VMFREPL EXEC [493](#)  
  VMFSETUP EXEC [505](#)

recovery procedures (*continued*)  
  VMFSGMAP EXEC [524](#)  
  VMFSIM CHKLVL [537](#)  
  VMFSIM COMPTBL [542](#)  
  VMFSIM GETLVL [549](#)  
  VMFSIM INIT [555](#)  
  VMFSIM LOGMOD [561](#)  
  VMFSIM MODIFY [566](#)  
  VMFSIM QUERY [572](#)  
  VMFSIM SRVDEP [578](#), [590](#)  
  VMFSIM SRVREQ [584](#)  
  VMFSIM SYSDEP [578](#), [590](#)  
  VMFSIM SYSREQ [596](#)  
  VMFSUFIN EXEC [601](#)  
  VMFVIEW EXEC [619](#)  
RECSE record, source product parameter file [640](#)  
refreshing Make Override Panel [39](#)  
regenerating parts locally [117](#)  
relationship between serviceable parts and usable form [90](#)  
remove list (SNA service) [763](#)  
removing all inputs, Make Override Panel [39](#)  
REP statement, patch update file  
  REP statement [122](#)  
replacement maintained parts  
  VMFREPL, used with local mods [494](#)  
replacement objects  
  used by VMFBDCOM [339](#)  
replacing  
  a single product [51](#)  
  several products [52](#)  
replacing products  
  several products [67](#)  
  single product [66](#)  
  while keeping tailorings [65](#)  
  with VMFINS MIGRATE [51](#), [65](#)  
reqby log (SNA) [762](#)  
required by log (reqby log) [762](#)  
requirements for using VMFINS [10](#)  
requisite processing, VMFBLD [112](#)  
requisite table [109](#)  
requisites missing, install or migrate [24](#)  
response examples, notation used in xxxi  
restrictions on using stem input and output [530](#)  
results, planning [24](#)  
retailoring product files [68](#)  
retrieve (F9), Make Override Panel [37](#)  
rework local service [102](#)  
RSU (Recommended Service Upgrade)  
  document [125](#)  
RSU descriptor file [126](#)

## S

sample queries, apply processing [181](#)  
save  
  changes on Make Override Panel [38](#)  
  PPF override files [38](#)  
save as... (F5), Make Override Panel [36](#)  
saved segment data file [113](#)  
saved segment definition record [514](#)  
saved segments  
  building [355](#)  
  identifying to be built [354](#)  
scenarios

scenarios (*continued*)

- build
  - using the PPF operand [77](#)
- delete
  - using the PPF operand [81](#)
- install
  - using VMFINS INSTALL [54](#)
  - with the PPF operand [54](#)

SEGBLD \$SELECT, example [130](#)

SEGDATA file

- example [682](#)
- overview [166](#)

segment management

- VMFSGMAP EXEC [116](#)
- with VMSES/E [116](#)

segments (saved), building [355](#)

select data file

- contents [128](#)
- example [129](#)
- for saved segments [130](#)
- overview [111](#), [113](#)
- saved segments, example [130](#)
- syntax [128](#)

select level of serviceable part, no version vector table [123](#)

selecting

- a VMFINS operand [17](#)
- the serviceable part version [123](#)

service

- applying to product [95](#)
- corrective
  - installing [95](#)

SERVICE \$PRODS [135](#)

service application using PSU procedure [97](#)

service concepts,z/VM [89](#)

service control file (SNA service) [758](#)

SERVICE EXEC [261](#)

service log (SNA) [762](#)

service media definition, receive processing, PPF [639](#)

service procedures, object code

- applying corrective service [765](#)
- applying fixes using ZAPs [765](#)
- applying preventive service
  - merging a single PTF [767](#)
  - merging multiple PTFs [768](#)
  - removing a single PTF [770](#)
  - removing multiple PTFs [772](#)

merge service [765](#)

prevent regression [773](#)

remove service [765](#)

removing fix-in-error [774](#)

service queries, VMFINFO [203](#)

service supported by VMSES/E [92](#)

service tape formats [754](#)

service tasks, primary [105](#)

service to a product, definition [89](#)

service types, VMSES/E supported [92](#)

service with VMSES/E, overview [105](#)

service-level

- apply status table (SRVAPPS) [175](#)
- build status
  - BUILDALL [719](#)
  - BUILT [719](#)
  - BYPASSED [719](#)
  - DELETE [719](#)

service-level (*continued*)

- build status (*continued*)
  - DELETED [719](#)
  - ERROR [719](#)
  - MANUAL [719](#)
  - SERVICED [719](#)
- build status table (SRVBLDS) [175](#), [717](#)
- description table (SRVDE SCT) [173](#), [712](#)
- production status table (SRVPROD) [720](#)
- receive status table (SRVRECS) [174](#), [715](#)
- requisite table (SRVREQT) [174](#), [713](#)
- Software Inventory [163](#), [704](#)
- Software Inventory tables [704](#)

serviceable parts

- creating by complete replacement [340](#)
- creating executable module [347](#)
- creating executable module with CPLINK [349](#)
- restoring DDR image files [330](#)
- serviced by updates only, servicing usable forms [91](#)
- version selection [123](#)
- within service structure [90](#)

serviceable parts records, format 1 build lists [143](#)

Serviceable Parts/Usable Forms Query Inputs and Outputs [211–213](#)

Serviceable Parts/Usable Forms Query Panel [211](#)

ServiceLink APARFIX command [101](#)

SERVMGR EXEC [269](#)

SERVMGR INITIALIZE [270](#)

SERVMGR MANAGED [288](#)

SERVMGR REMOVE [285](#)

SERVMGR SRVLVL [279](#)

SERVMGR SYSTEM [272](#)

several products

- installing [50](#)
- migrating [65](#)
- replacing using VMFINS INSTALL [52](#)
- replacing using VMFINS MIGRATE [67](#)

SFS directory name, Make Override Panel [46](#)

simple stem input and output [529](#)

single product

- install
  - with PPF operand, example [54](#)
- installing [49](#)
- migrating [64](#)
- replacing [66](#)
- replacing using VMFINS INSTALL command [51](#)

SMAPI

- appliance servers [362](#)

SNA, service products

- apply list [763](#)
- exclude list [764](#)
- merge log [760](#)
- product parameter file [759](#)
- remove list [763](#)
- reqby log [762](#)
- service control file [758](#)
- service log [762](#)
- ZAP list [764](#)
- ZAP log [761](#)

SNTINFO EXEC [733](#)

software inventory

- introduction [11](#)

Software Inventory

- \$PTFPART file [705](#)

Software Inventory (*continued*)

- changing the defaults [660](#)
- changing to SFS directory [221](#)
- default
  - changing [660](#)
  - changing to SFS directory [221](#)
  - file name [660](#)
  - minidisk [660](#)
  - system name [660](#)
- files, system-level [659](#)
- levels [163](#)
- parts catalog [703](#)
- PRODPART file [660](#)
- product parts file (PRODPART) [660](#)
- PTF parts file [705](#)
- service-level [704](#)
- service-level apply status table (SRVAPPS) [716](#)
- service-level build status table (SRVBLDS) [717](#)
- service-level description table (SRVDESCRIPT) [712](#)
- service-level production status table (SRVPROD) [720](#)
- service-level receive status table (SRVRECS) [715](#)
- service-level requisite table (SRVREQT) [713](#)
- SRVAPPS file [716](#)
- SRVBLDS file [717](#)
- SRVDESCRIPT file [712](#)
- SRVPROD file [720](#)
- SRVRECS file [715](#)
- SRVREQT file [713](#)
- SYSABRVT file [702](#)
- SYSAPPS file [690](#)
- SYSBLDS file [692](#)
- SYSDESCRIPT file [684](#)
- SYSTEMO file [685](#)
- SYSRECS file [688](#)
- SYSREQT file [686](#)
- system-level [659](#)
- system-level apply status table (SYSAPPS) [690](#)
- system-level base APAR table (SYSAPARS) [701](#)
- system-level build status table (SYSBLDS) [692](#)
- system-level description table (SYSDESCRIPT) [684](#)
- system-level file type abbreviation table (SYSABRVT) [702](#)
- system-level memo table (SYSTEMO) [685](#)
- system-level Product Inventory table (SYSSUF) [696](#)
- system-level receive status table (SYSRECS) [688](#)
- system-level requisite table (SYSREQT) [686](#)
- system-level restart table (SYSREST) [696](#)
- system-level Service Update Facility table (SYSSUF) [693](#)
- version vector table [721](#)
- VMSES PARTCAT file
  - example [704](#)

Software Inventory syntax [659](#)

Software Inventory tables, data structure [659](#)

source files, creating updated [122](#)

source product parameter file

- component area [624](#)
- header area [623](#)
- introduction [13](#)
- MDA record [634](#)
- override area [644](#)
- override control records [645](#)
- overview [159](#)
- RECINS record [637](#)
- RECSER record [640](#)

source product parameter file (*continued*)

- syntax [623](#)
- tag extensions [645](#)
- source, product installation parameters [33](#)
- space requirements, converting [367](#)
- specifying
  - object parameters [148](#)
  - part options [149](#)
- split-screen XEDIT session
  - example [70](#)
  - PF keys [70](#)
  - using PF keys [70](#)
- spooling the console [14](#)
- SRVAPPS file [716](#)
- SRVBLDS file [717](#)
- SRVDESCRIPT file [712](#)
- SRVPROD file [720](#)
- SRVRECS file [715](#)
- SRVREQT file [713](#)
- stand-alone dump utility [362](#)
- starting over, Make Override Panel [39](#)
- status information, build (system-level software) [165](#)
- status of APARS received [180](#)
- status of applied products [178](#)
- status of products received [178](#)
- status of received PTFs [180](#)
- stem
  - associative [529](#)
  - restrictions [530](#)
  - REXX [528](#)
  - simple [529](#)
  - variables (VMFSIM QUERY) [528](#)
- strings [106](#), [634](#), [638](#), [640](#), [643](#)
- structure of data, Software Inventory tables [525](#), [659](#)
- structure of file, product parameter file [621](#)
- supported tape formats [10](#)
- supporting information (system-level software) [166](#)
- syntax
  - auxiliary control files [120](#)
  - control files [118](#)
  - diagrams, how to read them [227](#)
  - format 1 build lists [142](#)
  - format 2 and 3 build lists [144](#)
  - national language support table (VMFNLS LANGLIST) [154](#)
  - notation [227](#)
  - other files used by VMSES/E [128](#)
  - product parameter files [621](#)
  - Software Inventory tables [659](#)
  - TDATA statements [526](#)
- syntax diagrams, how to read [xxix](#)
- SYSABRVT file [702](#)
- SYSAPARS [701](#)
- SYSAPPS file [690](#)
- SYSBLDS file [692](#)
- SYSDESCRIPT file [684](#)
- SYSMOD [699](#)
- SYSTEMO file [685](#)
- SYSPIINV [696](#)
- SYSRECS file [688](#)
- SYSREQT file [686](#)
- SYSREST [696](#)
- SYSSUF [693](#)
- SYSTEM disks [106](#)



- system information (system-level software) [165](#)
- system objects
  - identifying those to be built [354](#)
- system-level
  - apply status table (VM SYSAPPS) [168](#)
  - build status
    - BUILT [693](#)
    - DELETE [693](#)
    - DELETED [693](#)
    - ERROR [693](#)
    - SUPED [693](#)
  - build status table (VM SYSBLDS) [168](#)
  - description table (VM SYSDSCT) [166](#)
  - receive status table (VM SYSRECS) [167](#)
  - requisite table (VM SYSREQT) [167](#)
  - Software Inventory tables [166](#)
- system-level Base APAR table [113](#)
- system-level Product Inventory table [113](#)
- system-level Software Inventory
  - overview [163](#), [164](#), [659](#)

## T

- tables
  - build list format by VMFBLD part handler [141](#)
  - COR descriptor file, example [126](#)
  - function key assignments, make override panel [35](#)
  - function key assignments, VMFINFO panels [200](#), [201](#)
  - methods of installation and service [5](#)
  - PF keys for split-screen XEDIT session, VMFINS MIGRATE [70](#)
  - PF keys for view-screen session, VMFINS MIGRATE [74](#)
  - product contents directory [127](#)
  - product contents directory, COR tape [127](#)
  - product contents directory, Install/RSU tape [127](#)
  - RSU descriptor file, example [126](#)
  - tape descriptor files [126](#)
  - VMFINS processing by product format [11](#)
  - VMSES/E database [105](#), [106](#)
- tables (system-level Software Inventory)
  - VM SYSABRVT [170](#)
  - VM SYSAPARS [701](#)
  - VM SYSAPPS [168](#)
  - VM SYSBLDS [168](#)
  - VM SYSDSCT [166](#)
  - VM SYSPINV [696](#)
  - VM SYSRECS [167](#)
  - VM SYSREQT [167](#)
  - VM SYSREST [696](#)
  - VM SYSSUF [693](#)
- tables in the parts catalog [170](#)
- tag extensions
  - :BLD [651](#)
  - :CNTRLOP [646](#)
  - :DABBV [652](#)
  - :DCL [647](#)
  - :MDA [648](#)
  - :RECINS [649](#)
  - :RECSER [650](#)
  - source product parameter file [645](#)
- tagged data [527](#)
- tagged data statements (TDATA), VMFSIM [526](#)
- tailable file [68](#)
- tailored file [68](#)

- tape descriptor file [125](#)
- tape document [125](#)
- tape formats
  - processed by VMFINS [10](#)
  - VM/ESA system delivery offering (SDO) [753](#)
  - VMSES/E [753](#)
- TASK string [106](#)
- TDATA [527](#)
- TDATA statements, VMFSIM [526](#)
- temporary files for VMSES/E EXECs [735](#)
- temporary product parameter file
  - syntax [654](#)
  - using [160](#)
- test mode to production mode moves [48](#)
- text replacement, servicing usable forms [91](#)
- tools for system generation tasks
  - online HELP [227](#)
  - syntax conventions [229](#)
- trademarks [788](#)
- TXTLIBs, building [359](#)
- types of product parameter files
  - PPF types [159](#)
- types of requisites, system-level requisite table [687](#)
- types of service supported by VMSES/E [92](#)

## U

- understanding
  - help options, Make Override Panel [42](#)
  - syntax diagrams [227](#)
- updated source files, creating [122](#)
- updates [90](#)
- updates to service usable forms [91](#)
- updating
  - a record in :RECSER section of PPF, example [656](#)
  - a record, multiple-level override, PPF [657](#)
  - a record, single-level override, PPF [656](#)
  - PTF level information [124](#)
  - SRVBLDS after manual build processing [187](#)
- updating the Software Inventory
  - adding a product to the system-level Software Inventory [186](#)
  - updating the SRVBLDS table after manual build processing [187](#)
  - with VMFSIM MODIFY [186](#)
- usable form product parameter file
  - overview [160](#)
  - syntax [654](#)
- usable forms
  - overview [90](#)
  - serviced by module replacement [91](#)
  - serviced by parts serviced by updates only [91](#)
  - serviced by text replacement [91](#)
  - serviced by updates [91](#)
- using
  - F4=Expand Dirid function key [46](#)
  - file input for VMFSIM commands [527](#)
  - function keys, Make Override Panel [35](#)
  - function keys, Query NSS Map Window [521](#)
  - function keys, Segment Definition Panel [519](#)
  - function keys, Segment Map Panel [511](#)
  - function keys, VMFINFO Panel [200](#)
  - LIST operand
    - to install several products [50](#)



## VMFINFO (continued)

- Minidisk/Directory Access Query output [217](#)
- Miscellaneous Queries Panel [216](#)
- Object Build Dependencies Query output [214](#)
- Object Build Requisites Query output [214](#)
- Object Part Handler/Target Query output [215](#)
- Object Status Query output [213](#)
- Part Service History Query output [216](#)
- Part Service Level Query output [215](#)
- PPF Fileid - Help Panel [202](#)
- product dependencies query [205](#)
- product dependency query [203](#)
- product description query [203](#), [204](#)
- product queries [203](#)
- product requisite query [203](#)
- product requisites query [205](#)
- product status query [203](#), [204](#)
- PTF Dependencies/Superseding query output [209](#)
- PTF Requisites/Supersedes query output [208](#)
- PTF Serviceable Parts query output [210](#)
- PTF status query output [207](#)
- PTF User Memo query output [209](#)
- PTF/APAR Query Panel [206](#)
- service queries [203](#)
- Serviceable Parts/Usable Forms Query Panel [211](#)
- Serviceable Parts/Usable Forms Query Panel, inputs/outputs [211–213](#)
- using the panels [199](#)
- where does the information come from? [199](#)

## VMFINFO EXEC [402](#)

### VMFINS

- adding a new release of a product [63](#)
- adding products [49](#)
- BUILD
  - command syntax [405](#)
  - example [77](#)
  - recovering from errors [411](#)
- building products [77](#)
- command defaults, changing [48](#)
- command syntax [404](#)
- DELETE
  - command syntax [412](#)
  - deleting products, overview [81](#)
  - example [81](#)
  - recovering from errors [416](#)
- deleting products [81](#)
- DISABLE
  - command syntax [417](#)
- ENABLE
  - command syntax [421](#)
- INSTALL
  - command syntax [426](#)
  - examples [54](#)
  - recovering from an unsuccessful installation [432](#)
- installing products [49](#)
- MIGRATE
  - adding products, overview [63](#)
  - command syntax [433](#)
  - recovering from errors [440](#)
  - replacing products, overview [65](#)
- migrating products [63](#)
- overview [9](#)
- processing products [17](#)
- processing, by product format [11](#)

## VMFINS (continued)

- replacing products [49](#)
- VMFINS BUILD
  - command syntax [405](#)
  - overview [77](#)
  - recovering from errors [411](#)
- VMFINS CONSOLE file [14](#)
- VMFINS DEFAULTS file [139](#)
- VMFINS DEFAULTS file, which one used [408](#), [414](#), [418](#), [422](#), [429](#), [438](#)
- VMFINS DELETE
  - command syntax [412](#)
  - example [81](#)
  - overview [81](#)
  - recovering from errors [416](#)
- VMFINS DISABLE
  - command syntax [417](#)
- VMFINS ENABLE
  - command syntax [421](#)
- VMFINS INSTALL
  - command syntax [426](#)
  - examples [54](#)
  - overview [49](#)
  - recovering from an unsuccessful installation [432](#)
- VMFINS MIGRATE
  - command syntax [433](#)
  - how it works with the ADD option [63](#)
  - how it works with the REPLACE option [65](#)
  - overview [63](#)
  - recovering from errors [440](#)
  - replacing a product [66](#)
- VMFINS override prompts, changing [48](#)
- VMFINS PRODLIST file [18](#)
- VMFINS, console spooling [14](#)
- VMFMERGE EXEC [774](#)
- VMFMRDSK EXEC [444](#)
- VMFNLS EXEC [448](#)
- VMFNLS LANGLIST file
  - example [155](#)
  - file syntax [154](#)
  - overview [154](#)
- VMFOVER EXEC [115](#), [456](#)
- VMFPPF EXEC [115](#), [458](#)
- VMFPSU EXEC [462](#)
- VMFQMDA EXEC [469](#)
- VMFQOBJ EXEC [116](#), [472](#)
- VMFREC EXEC
  - a closer look [108](#)
  - bypassing tape files [638](#), [640](#)
  - command syntax [477](#)
  - part handlers [108](#)
  - processing multiple service tapes [108](#)
  - software inventory files used [109](#)
- VMFREM EXEC
  - command syntax [485](#)
  - examples [489](#)
- VMFREMOV EXEC [777](#)
- VMFREMOV processing [778](#)
- VMFREPL EXEC
  - command syntax [493](#)
  - examples [497](#)
- VMFSETUP EXEC [500](#)
- VMFSETUP, accessing disks as read-only [634](#)
- VMFSETUP, accessing disks read-only [502](#)

## VMFSGMAP EXEC

- Add Segment Definition Panel [514](#)
- Change Segment Definition Panel [513](#)
- command syntax [506](#)
- examples [522](#)
- managing saved segments [116](#)
- Query NSS Map Window [520](#)
- Query NSS Map Window, function keys [521](#)
- saved segment build parameters syntax [517](#)
- saved segment definition record [514](#)
- Segment Definition Panel [513](#)
- Segment Definition Panel, function keys [519](#)
- Segment Map Panel [507](#)
- segment map record [508](#), [511](#)
- subcommands [521](#)

## VMFSIM CHKLVL [531](#)

## VMFSIM COMPTBL [538](#)

## VMFSIM EXEC

- associative stems [529](#)
- command syntax [525](#)
- how it processes tagged data [527](#)
- introduction [177](#)
- providing input to [526](#)
- querying multiple tables with the FILE option [527](#)
- receiving output from [526](#)
- simple stems [529](#)
- stem restrictions [530](#)
- tree structure format, associative stems [530](#)
- using during install [61](#)
- using file input [527](#)
- using the STEM variable [528](#)

## VMFSIM GETLVL [543](#)

## VMFSIM INIT [550](#)

## VMFSIM LOGMOD [556](#)

## VMFSIM MODIFY [562](#)

## VMFSIM QUERY [567](#)

## VMFSIM SRVDEP [573](#)

## VMFSIM SRVREQ [579](#)

## VMFSIM stem input (associative) [529](#)

## VMFSIM stem input (simple) [529](#)

## VMFSIM SYSDEP [585](#)

## VMFSIM SYSREQ [591](#)

## VMFSIM: tagged data (TDATA) [527](#)

## VMFSUFIN EXEC [597](#)

## VMFSUFTB EXEC [602](#)

## VMFUPDAT EXEC [604](#)

## VMFVIEW EXEC [615](#)

## VMFZAP EXEC [779](#)

## VMSBR \$SELECT, example [130](#)

## VMSES PARTCAT [703](#)

## VMSES/E

- database [105](#)
- input files [735](#)
- output files [735](#)
- temporary files [735](#)
- types of service supported [92](#)

## VMSES/E commands

- CHKAPARS EXEC [234](#)
- GENCPBLS EXEC [237](#)
- LOCALMOD EXEC [245](#)
- PUT2PROD EXEC [258](#)
- SERVICE EXEC [261](#)
- SERV MGR EXEC [269](#)
- SERV MGR INITIALIZE [270](#)

## VMSES/E commands (*continued*)

## SERV MGR MANAGED [288](#)

## SERV MGR REMOVE [285](#)

## SERV MGR SRVLVL [279](#)

## SERV MGR SYSTEM [272](#)

## VMFAPPLY EXEC [291](#)

## VMFASM EXEC [297](#)

## VMFBLD EXEC [305](#)

## VMFBTMAP EXEC [364](#)

## VMFCNVT EXEC [367](#)

## VMFCOPY EXEC [369](#)

## VMFENRPT [373](#)

## VMFERASE EXEC [377](#)

## VMFEXUPD EXEC [381](#)

## VMFHASM EXEC [387](#)

## VMFHLASM EXEC [394](#)

## VMFINFO EXEC [402](#)

## VMFINS BUILD [405](#)

## VMFINS DELETE [412](#)

## VMFINS DISABLE [417](#)

## VMFINS ENABLE [421](#)

## VMFINS EXEC [404](#)

## VMFINS INSTALL [426](#)

## VMFINS MIGRATE [433](#)

## VMFM RDSK EXEC [444](#)

## VMFNLS EXEC [448](#)

## VMFOVER EXEC [456](#)

## VMFPPF EXEC [458](#)

## VMFPSU EXEC [462](#)

## VMFQMDA EXEC [469](#)

## VMFQOBJ EXEC [472](#)

## VMFREC EXEC [477](#)

## VMFREM EXEC [485](#)

## VMFREPL EXEC [493](#)

## VMFSETUP EXEC [500](#)

## VMFSGMAP EXEC [506](#)

## VMFSIM CHKLVL [531](#)

## VMFSIM COMPTBL [538](#)

## VMFSIM EXEC [525](#)

## VMFSIM GETLVL [543](#)

## VMFSIM INIT [550](#)

## VMFSIM LOGMOD [556](#)

## VMFSIM MODIFY [562](#)

## VMFSIM QUERY [567](#)

## VMFSIM SRVDEP [573](#)

## VMFSIM SRVREQ [579](#)

## VMFSIM SYSDEP [585](#)

## VMFSIM SYSREQ [591](#)

## VMFSUFIN EXEC [597](#)

## VMFSUFTB EXEC [602](#)

## VMFUPDAT EXEC [604](#)

## VMFVIEW EXEC [615](#)

## VMSES/E Installation/Service Tool introduction [9](#)

## VMSESE profile

### description [131](#)

### example [132](#)

### syntax [131](#)

## W

what is z/VM product service? [89](#)

when to perform a build [77](#)

where to find more information [xxxii](#)

which VMFINS DEFAULTS file used [408](#), [414](#), [418](#), [422](#), [429](#),  
[438](#)  
which VMFINS operand to use [17](#)  
who can use VMFINS [10](#)  
wildcard objects  
    in format 2 build lists [148](#)  
working with  
    product files [68](#)  
    tailored files [68](#)

## Z

z/VM service concepts [89](#)  
ZAP list (SNA service) [764](#)  
ZAP log (SNA service) [761](#)  
ZAPs, applying fixes (object code) [765](#)  
ZAPTEXT EXEC [781](#)







Product Number: 5741-A09

Printed in USA

GC24-6336-73

