z/VM

Program Management Binder for CMS

Version 6 Release 3

SC24-6211-03

Note:
Before using this information and the product it supports, read the information in "Notices" on page 89.

This edition applies to version 6, release 3, modification 0 of IBM z/VM (product number 5741-A07) and to all subsequent releases and modifications until otherwise indicated in new editions.

This edition replaces SC24-6211-02.

© Copyright IBM Corporation 2001, 2013.
US Government Users Restricted Rights – Use, duplication or disclosure restricted by GSA ADP Schedule Contract with IBM Corp.

Contents

© Copyright IBM Corp. 2001, 2013

Figures

© Copyright IBM Corp. 2001, 2013

Tables

© Copyright IBM Corp. 2001, 2013

About This Document

This document provides information about the IBM® z/VM® Program Management Binder for CMS. Only the differences in usage and behavior from the *z/OS MVS Program Management Binder* are included.

Intended Audience

This document is for anyone who needs to generate executable program objects using the Program Management Binder of the z/VM Conversational Monitor System (z/VM CMS).

To use this document effectively, you should be familiar with *z/OS MVS Program Management Binder*.

Conventions Used in This Document

The following terminology is used in this document:

CMS Binder is used as an abbreviation of Program Management Binder for CMS.

z/OS MVS Program Management Binder is used to specifically reference z/OS V1R12 MVS Program Management Binder when discussing the binder's level of functionality.

Throughout this document, the term “file name” is used to refer to files in general, regardless of the specific file system in which they reside. The intended use should be clear from context, but it is often necessary to make a distinction between files that reside in the byte file system and files that reside on minidisks or in the shared file system (but not in the byte file system). For convenience, the byte file system will be referred to as “BFS files”, and the latter as “CMS files”.

The term “ddname” is used to refer to a data definition name. The relation of a ddname to one or more CMS files is achieved by using the FILEDEF command or for a BFS file by using the OPENVM PATHDEF CREATE command.

The term “FILEDEF” is used to refer to the data definition created by the use of the FILEDEF command.

The term “PATHDEF” is used to refer to the data definition created by the use of the OPENVM PATHDEF CREATE command.

The term “program module” is defined as the output of the binder; a collective term for program object and load module.

How to Read Syntax Diagrams

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.

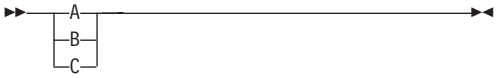
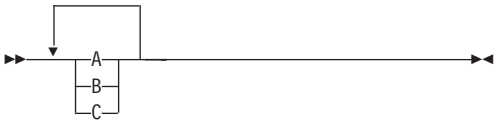
- The ►►— symbol indicates the beginning of the syntax diagram.

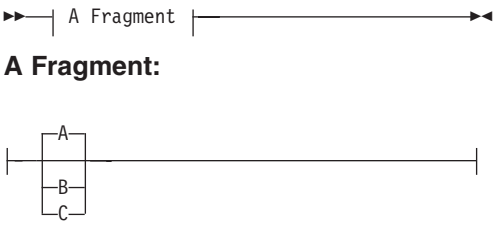
© Copyright IBM Corp. 2001, 2013

- 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.

Examples of the other syntax diagram conventions are shown in the following table.

Syntax Diagram Convention	Example
Keywords and Constants: 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 might have additional conventions for using all-uppercase or all-lowercase.	
Abbreviations: 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. In this example, you can specify KEYWO, KEYWOR, or KEYWORD.	
Symbols: You must specify these symbols exactly as they appear in the syntax diagram.	* Asterisk : Colon , Comma = Equal Sign - Hyphen () Parentheses . Period
Variables: A variable appears in highlighted lowercase, usually italics. In this example, *var_name* represents a variable that you must specify following KEYWORD.	

Syntax Diagram Convention	Example
Repetitions: An arrow returning to the left means that the item can be repeated. A character within the arrow means that you must separate each repetition of the item with that character. 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. Syntax notes might also be used to explain other special aspects of the syntax.	   **Notes:** 1 Specify *repeat* up to 5 times.
Required Item or Choice: When an item is on the line, it is required. In this example, you must specify A. 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.	 
Optional Item or Choice: When an item is below the line, it is optional. In this example, you can choose A or nothing at all. 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.	 
Defaults: 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. In this example, A is the default. You can override A by choosing B or C.	
Repeatable Choice: 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. In this example, you can choose any combination of A, B, or C.	

Syntax Diagram Convention	Example
Syntax Fragment: 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. In this example, the fragment is named "A Fragment."	

Message and Response Notation

This document might include examples of messages or responses. Although most examples are shown exactly as they would appear, some content may 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 an optional item that might be displayed.

{ } Braces enclose alternative items, 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

Within the text, references might be made to *z/OS MVS System Messages Vol 8 (IEF - IGD)*, SA22-7638.

For more information about z/VM functions, see the documents listed in the "Bibliography" on page 95.

Links to Other Documents and Web Sites

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

How to 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.

Use one of the following methods to send us your comments:

1. Send an email to mhvrcfs@us.ibm.com.
2. Go to IBM z/VM Reader's Comments (www.ibm.com/systems/z/os/zvm/zvmforms/webqs.html).
3. Mail the comments to the following address:
   IBM Corporation
   Attention: MHVRCFS Reader Comments
   Department H6MA, Building 707
   2455 South Road
   Poughkeepsie, NY 12601-5400
   U.S.A.
4. Fax the comments to us as follows:
   From the United States and Canada: 1+845+432-9405
   From all other countries: Your international access code +1+845+432-9405

Include the following information:

- Your name and address
- Your email address
- Your telephone or fax number
- The publication title and order number:
  z/VM V6.3 Program Management Binder for CMS
  SC24-6211-03
- The topic name or page number related to your comment
- The text of your comment

When you send comments to IBM, you grant IBM a nonexclusive right to use or distribute your comments in any way it believes appropriate without incurring any obligation to you.

IBM or any other organizations will use the personal information that you supply only to contact you about the issues that you submit to IBM.

If You Have a Technical Problem

Do not use the feedback methods listed above. Instead, do one of the following:

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

© Copyright IBM Corp. 2001, 2013

Summary of Changes

This document contains terminology, maintenance, and editorial changes. Technical changes are indicated by a vertical line to the left of the change. Some product changes might be provided through service and might be available for some prior releases.

SC24-6211-03, z/VM Version 6 Release 3

This edition includes changes to support the general availability of z/VM V6.3.

z/VM V6.3 supports Binder z/OS R13 Equivalency.

SC24-6211-02, z/VM Version 6 Release 2

This edition includes changes to support the general availability of z/VM V6.2.

z/VM V6.2 supports Binder z/OS R12 Equivalency:

- The 6.2 version of the CMS Binder is based on the 1.12 version of the z/OS Binder.
- The COMPAT option of the BIND command now includes a new suboption: ZOSV1R12. See "CMS BIND Command Options" on page 10 and Table 8 on page 62.
- The RMODE option of the BIND command now includes new suboptions. See "CMS BIND Command Options" on page 10.

© Copyright IBM Corp. 2001, 2013

Chapter 1. Introduction

This book provides information about the Program Management Binder for CMS. Only the differences in usage and behavior from the z/OS MVS Program Management Binder, as documented in *z/OS MVS Program Management: User's Guide and Reference* and *z/OS MVS Program Management: Advanced Facilities*, are included here.

In this section, we will briefly review both *z/OS MVS Program Management: User's Guide and Reference* and *z/OS MVS Program Management: Advanced Facilities* and highlight the Chapters and Appendices of interest to users of the Program Management Binder for CMS. First, we need to comment on some terms that are found throughout these books:

PDSE z/OS® DFSMS™ provides this data set organization to support libraries. It is superior to the original partitioned organization in not requiring compression and providing some support for long names. It is not supported on CMS.

PDS CMS OS Simulation does support PDS libraries in a limited way. The following types of supported libraries are relevant to the operation of the Program Management Binder for CMS:

LOADLIB
used as the target for both input and output of load modules

TXTLIB
input source of object decks

MACLIB
members may contain binder control statements

Program Object
z/OS MVS Program Management defines this executable format. It has many advantages over the OS load module format. For a description of how this format is used in CMS, see "Executable Formats" on page 61.

z/OS MVS Program Management: User's Guide and Reference

Summary of Chapters and Appendices

Chapter 1. Introduction
The only Program Management component that is utilized by Program Management Binder for CMS is Program Management Binder.

CMS equivalents of the z/OS DFSMS Utilities:

- IEBCOPY functionality is provided by the CMS LOADLIB command.
- IEHPROGM has no CMS equivalent.
- IEHLIST directory lists can be obtained using the `LIST` or `MAP` option of the relevant LOADLIB, TXTLIB or MACLIB command.

The Service Aid AMBLIST currently has no CMS equivalent. The `ZAP` CMS command provides similar functions to AMASPZAP, although it does not currently support the CMS extended modules or program objects.

© Copyright IBM Corp. 2001, 2013

Chapter 2. Creating Programs from Source Modules
This chapter provides a good overview of the bind process.

Chapter 3. Starting the Binder
The only part of this chapter that is relevant to the CMS environment is "3.3. Invoking the Binder from a Program."

Chapter 4. Defining Input to the Binder
This chapter is relevant to the CMS environment with the exception of the following sections:
- 4.1 Defining the Primary Input
- 4.2.3 Including Concatenated Data Sets
- 4.3.5 Searching the Link Pack Area

Chapter 5. Editing Data Within a Program Module
This chapter provides a good description of the editing services of the binder.

Chapter 6. Binder Options Reference
Reference this chapter in conjunction with "CMS BIND Command Options" on page 10.

Chapter 7. Binder Control Statement Reference
Read this chapter in conjunction with "Control Statements Summary" on page 45.

Chapter 8. Interpreting Binder Listings
This chapter describes each section of the listing produced by the binder.

Chapter 9. Binder Serviceability Aids
This chapter explains how to analyze, resolve, and diagnose problems while using the binder.

Appendix A. Using the Linkage Editor and Batch Loader
This appendix is not relevant to the CMS environment.

Appendix B. Summary of Program Management User Considerations
Only parts of this appendix are of interest to users of Program Management Binder for CMS.

Appendix C. Binder Return Codes
The applicable sections concern return codes and reason codes issued from the API.

Appendix D. Designing and Specifying Overlay Programs
None of this material is relevant because "Overlays" are not supported in the CMS environment.

z/OS MVS Program Management: Advanced Facilities

Summary of Chapters and Appendices

Chapter 1. Using the Binder Application Programming Interface
Read this chapter in conjunction with "API Considerations" on page 62.

Chapter 2. IEWBUFF - Binder API buffers interface assembler macro for generating and mapping data areas
Read this chapter in conjunction with "API Considerations" on page 62.

Chapter 3. IEWBIND - Binder regular API functions
Read this chapter in conjunction with "API Considerations" on page 62.

Chapter 4. IEWBFDAT - Binder Fast Data Access API functions
This chapter is not relevant because Fast Data Access is not supported in the CMS environment.

Chapter 5. iewbndd.so - Binder C/C++ API DLL functions
Read this chapter in conjunction with "C/C++ API" on page 74.

Chapter 6. Invoking the binder from another program
Read this chapter in conjunction with "Invoking the Binder from a Program" on page 4.

Chapter 7. Setting options with the regular binder API
Read this chapter in conjunction with "API Considerations" on page 62.

Chapter 8. User exits
This chapter describes the capabilities of the available user exits.

Appendix A. Object Module Input Conventions and Record Formats
This appendix describes both object and xobject input to the binder.

Appendix B. Load Module Formats
This appendix is not relevant to the CMS environment.

Appendix C. Generalized Object File Format (GOFF)
Generalized Object File Format (GOFF) is a supported input format in the CMS environment.

Appendix D. Binder API Buffer Formats
This appendix maps the data areas used by the GETC, GETD, GETE, GETN and PUTD API calls.

Appendix E. Data Areas
Most of this appendix is not relevant to the CMS environment.

Appendix F. Programming Example for the Binder API
This appendix provides a good illustration of how many of the API facilities might be used.

Appendix G. Using the Transport Utility (IEWTPORT)
None of this material is relevant because the "Transport Utility" is not supported in the CMS environment.

Overview

The following diagram summarizes the user interfaces to the CMS Binder.

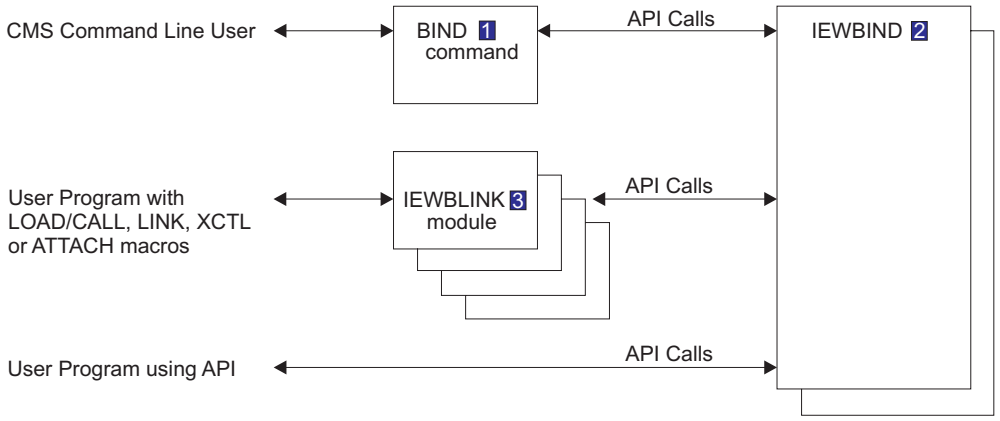


Figure 1. High-Level Interfaces to the VM Binder (Overview)

Notes on the diagram:

1 For command line users the BIND command provides bind and store processing which parallels the z/OS batch interface (IEWBLINK) or the z/OS TSO interface (LINK command).

2 IEWBIND in this diagram represents a composite of a binder API front end. This performs functional and environmental tasks to simplify the support of the binder operation under VM, the z/OS Binder linked as a CMS Module, and the CMS binder "cradle" (a collection of extensions to CMS OS Simulation to support, among other things, the program call (PC) instruction and the associated linkage stack).

3 User programs can invoke the binder using LOAD/CALL, LINK, XCTL or ATTACH macros using the module name IEWBLINK (bind and store).

Invoking the Binder from a Program

The module IEWBLINK may be given control by a program using the LINK, XCTL or ATTACH macroinstructions or the combination of the LOAD and CALL macroinstructions. IEWBLINK uses the INCLUDE function of the binder API to process the files associated with the ddname SYSLIN and attempts to bind and save any resulting program object to the target identified by the ddname SYSLMOD. Support for the alternate entry points IEWBLOAD, IEWBLODI, and IEWBLDGO is not provided by the CMS binder.

The Command Interface

The primary interface to the CMS Binder is the CMS **BIND** command. The BIND command exploits the binder's application programming interface (API) to request binder functions from the z/OS MVS Program Management Binder.

Control statements read by the BIND command are preprocessed so that `NAME` statements can be detected to control the bind process.

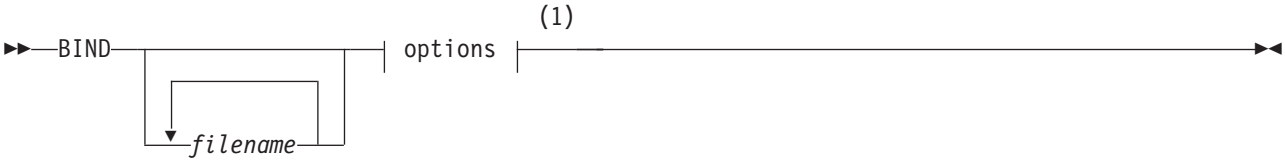
The API Front End

The CMS Binder API Front End previews all binder API calls (whether from the command interface or a user program) before any call is made to the z/VM binder code.

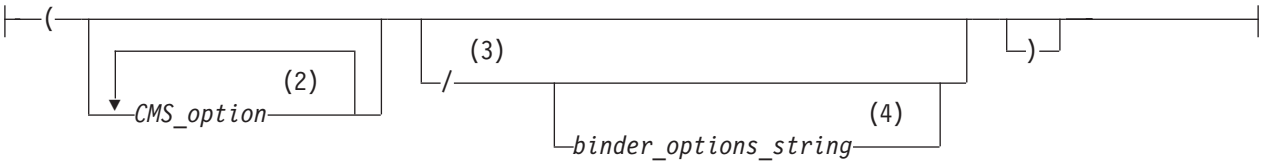
Preprocessing is performed according to the requested binder function code to set up an environment that allows the binder to operate as if it were running under z/VM, and thus process the requested function correctly.

The front end also preparses the options string for options that affect its processing.

Chapter 2. BIND



options:



Notes:

1 Options can be presented to the command interface using either a CMS or MVS style syntax.

2 CMS options may be entered in any order.

3 The "slash" delimiter is like any other token in this syntax: it must be preceded and followed by at least one blank to be recognized.

4 The internal syntax of *binder_options_string* is as described for the PARM field of a JCL EXEC statement in *z/OS MVS Program Management: User's Guide and Reference*.

Authorization

General User

Purpose

Use the BIND command to invoke the services of the CMS Binder to generate executable program objects. Input to the binder may consist of CMS or BFS files containing relocatable object code, binder control statements, or program objects previously generated by the binder. The binder can store a program object in either a CMS module file or a BFS file. An executable file that is produced by the binder may be used on a CMS system containing the program object loader in the same way as a conventional module file created by the GENMOD command.

Operands

filename
A list of file names can be specified to define the primary input to the binder. If primary input is to be specified in a load list file, then only one CMS record file system file name may be specified and the FILETYPE option must be used with a value of EXEC. The load list file must be a fixed record length of 80, have a file type of EXEC, and the input must consist of valid EXEC control words (that are ignored) and names of input text files in the following form:

```
&1 &2 filename [filetype]
```

© Copyright IBM Corp. 2001, 2013

The file name and file type (if specified) must not be more than eight characters in length. If no file names are specified then the BIND command checks for the presence of active FILEDEFs or PATHDEFs using the ddname SYSLIN. Refer to Usage Note 5 on page 29 for more information on using the SYSLIN ddname.

The file names in the list are separated by blanks. When the blanks are embedded within a quoted string, they are not treated as delimiters. Each file name is examined by the BIND command to see if it should be interpreted as representing a CMS record file system file (minidisk file or shared file system file, referred to as a "CMS file"), or a byte file system file (referred to as a "BFS file").

To be interpreted as representing a BFS file, a file name must either be completely enclosed in single or double quotation marks, or commence with a dot (.) or a forward slash (/). Otherwise the file name is interpreted as representing a CMS file, for which it may or may not be valid.

The following example shows how a list of files is interpreted when entering the BIND command:

```
bind file1 ./file2 /u/user1/file3 'file 4' "*file5" *file6
```

Where:

`file1` is interpreted as a CMS file.
`./file2` is interpreted as a BFS file.
`/u/user1/file3` would be interpreted as a BFS file.
`'file 4'` would be interpreted as a BFS file.
`"*file5"` would be interpreted as a BFS file.
`*file6` would be interpreted as a CMS file and subsequently found to be incorrect.

Each file name in the list is validated by the BIND command before any attempt to process it as primary input.

For CMS files, the file name must be no more than eight characters long, and must consist only of alphanumeric characters (**A-Z**, **a-z**, **0-9**), national characters (**$**, **#**, **@**), as well as the plus (**+**), hyphen (**-**), underscore (_) and colon (**:**) characters. If valid, the file name is assumed to represent a CMS file with the file type ID resolved to one of the following:

- as specified on the **FILETYPE** option, if present.
- SYSLIN
- List of file types defined in a CNTRL file; see the CTL option.
- TEXT
- MODULE
- as a member in the GLOBAL TXTLIB concatenation, unless the NOLIBE option is specified.

Note: The SYSLIN file type is intended to allow control statements for a program suite to be gathered into a single file that might share its name with one of the program TEXT files. Do not confuse this file type usage with the SYSLIN ddname described in 5 on page 29.

For BFS file names, the file name represents a BFS path, and is limited to 1023 characters. BFS paths are interpreted according to standard OpenExtensions rules, so that `./file2` and `'file 4'` are both relative paths referring to files in the current working directory, but `/u/user1/file3` is an absolute path.

Options

Options can be presented to the command interface using either a CMS style syntax, or as a string of options using the syntax described for the PARM field of a JCL EXEC statement in *z/OS MVS Program Management: User's Guide and Reference* ("MVS™ binder style"). In general, options specified in CMS style are converted to MVS binder style before being passed to the binder. Options specified in MVS binder style are not preprocessed in any way by the BIND command, but are simply passed to the binder appended to the converted CMS style options. For most options the effect is the same regardless of how the option is specified. The following is an example of the CMS style:

```
bind file1 (amode 24
```

It has the same effect as the following MVS binder style:

```
bind file1 (/ amode=24
```

Options specified in CMS style come in three flavors:

- Options that only affect the operation of the BIND command itself and are not converted or passed on to the binder in any way. The only options of this kind are CTL, FILETYPE, LIBE, and OUTPUT, and they are described completely in "CMS BIND Command Options" on page 10.
- Options that affect the operation of the BIND command itself and also cause related options to be generated and passed on to the binder. These options are DISK/PRINT, MSGLEVEL, SNAME, and TERM/TYPE, and they are described in detail in "CMS BIND Command Options" on page 10. They should not be specified in MVS binder style because the appropriate command level actions will not be taken and output could be processed incorrectly. Further information about the PRINT and TERM options that are generated in MVS binder style from these options can be found in *z/OS MVS Program Management: User's Guide and Reference*.
- Options that have no direct affect on the operation of the BIND command itself. These options are converted to MVS binder style format and are passed to the binder. These options are summarized in "CMS BIND Command Options" on page 10 and described further as noted in that table.

Notes:

1. Not all of the options available for the z/OS MVS Program Management Binder have CMS style BIND command equivalents. In general, those options that are not applicable to the CMS environment do not have equivalents. Refer to Table 2 on page 25 for a cross-reference of z/OS MVS Program Management Binder options to CMS style BIND command options.
2. In general, options cannot be abbreviated when specified on the BIND command. Exceptions to this rule are shown explicitly in the table.
3. The short forms of the options that start with the characters CMS are intended as a convenience for terminal entry. They should be avoided when coding execs or programs that invoke the BIND command in case some future change in z/OS MVS Program Management Binder options conflicts with the short form of the option.
4. Rudimentary syntax verification and validity checking of options specified in CMS style are performed by the command interface. If an invalid option specification is encountered, an error message is issued and the command terminates without the binder being invoked. The BIND command cannot, however, filter out all possible erroneous option specifications. So in some

situations, option strings may be generated and passed to the binder that cause the binder's option processor to issue an error message.

5. The default values for only six options are provided for the following in "CMS BIND Command Options":CTL, DISK/PRINT, FILETYPE, LIBE, MSGLEVEL, and OUTPUT. The effective default for all other options not specified on the command will be the product default, unless overridden by an installation or user default specified in the IEWBODEF module. For product defaults, see Table 9 on page 66 and Table 35. Setting Options With the Binder API in *z/OS MVS Program Management: Advanced Facilities*.

CMS BIND Command Options

Except for those options that are not documented elsewhere, the information about the CMS style option syntax provided here is intended only as a summarized reference. Definitive information should be obtained from the source referred to in individual option descriptions.

ALIGN2

ALIGN2 controls the page alignment (2 KB or 4 KB) of sections of text produced by the binder.

NOALIGN2 is equivalent to ALIGN2 NO.

For more information, refer to the description of the ALIGN2 option in *z/OS MVS Program Management: User's Guide and Reference*.

AMODE

AMODE sets the addressing mode of the saved program module.

For more information, refer to the description of the AMODE option in *z/OS MVS Program Management: User's Guide and Reference*.

Note: AMODE 64, although supported by the binder, is not currently supported by the CMS loader.

AUTO

See CMSAUTO.

CALL

CALL controls an automatic library call.

NOCALL and NCAL are equivalent to CALL NO.

For more information, refer to the description of the CALL option in *z/OS MVS Program Management: User's Guide and Reference*.

CASE

CASE controls case sensitivity for symbols.

MIXED is equivalent to CASE MIXED. UPPER is equivalent to CASE UPPER.

For more information, refer to the description of the CASE option in *z/OS MVS Program Management: User's Guide and Reference*.

CLEAN

See CMSCLEAN.

CMSAUTO

CMSAUTO controls use of TEXT files to resolve external references during a final automatic library call.

AUTO is equivalent to CMSAUTO. NOAUTO and NOCMSAUTO are equivalent to CMSAUTO NO.

For more information, refer to the description of the CMSAUTO option in "Setting Options With the Binder API" on page 62

CMSCLEAN

CMSCLEAN determines the setting of an attribute that controls the removal of the program module from storage at the end of execution.

CLEAN is equivalent to CMSCLEAN. NOCLEAN and NOCMSCLEAN are equivalent to CMSCLEAN NO.

For more information, refer to the description of the CMSCLEAN option in "Setting Options With the Binder API" on page 62

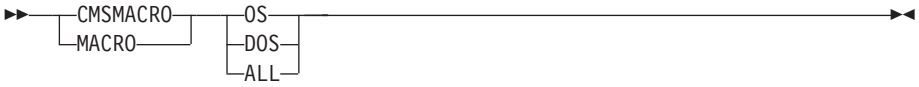
CMSINCL

CMSINCL controls the use of TEXT, extended format MODULE and TXTLIB files for input when no FILEDEF or PATHDEF matches the ddname.

INCL is equivalent to CMSINCL. NOINCL and NOCMSINCL are equivalent to CMSINCL NO.

For more information, refer to the description of the CMSINCL option in "Setting Options With the Binder API" on page 62

CMSMACRO



CMSMACRO determines the setting of an attribute that indicates whether the program contains OS or DOS macros.

MACRO is equivalent to CMSMACRO.

For more information, refer to the description of the CMSMACRO option in "Setting Options With the Binder API" on page 62

CMSSTR

CMSSTR determines the setting of an attribute that controls deletion of previously loaded OS programs.

STR is equivalent to CMSSTR. NOCMSSTR and NOSTR are equivalent to CMSSTR NO.

For more information, refer to the description of the CMSSTR option in "Setting Options With the Binder API" on page 62

CMSSYSTEM

CMSSYSTEM determines the setting of an attribute that controls the storage protect key set at load time.

SYSTEM is equivalent to CMSSYSTEM. NOCMSSYSTEM and NOSYSTEM are equivalent to CMSSYSTEM NO.

For more information, refer to the description of the CMSSYSTEM option in "Setting Options With the Binder API" on page 62

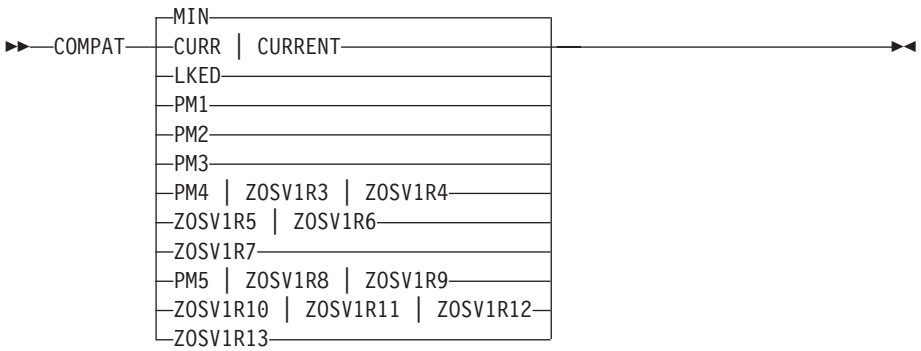
CMSXC

CMSXC determines the setting of an attribute that indicates the program module can execute only in XC virtual machines.

XC is equivalent to CMSXC. NOCMSXC and NOXC are equivalent to CMSXC NO.

For more information, refer to the description of the CMSXC option in "Setting Options With the Binder API" on page 62

COMPAT



COMPAT controls binder compatibility level.

COMPAT CURR is equivalent to COMPAT CURRENT.

COMPAT PM4, COMPAT ZOSV1R3, and COMPAT ZOSV1R4 are all equivalent.

COMPAT ZOSV1R5 is equivalent to COMPAT ZOSV1R6.

COMPAT PM5, COMPAT ZOSV1R8, and COMPAT ZOSV1R9 are all equivalent.

COMPAT ZOSV1R10, COMPAT ZOSV1R11, and COMPAT ZOSV1R12 are all equivalent.

For more information, refer to Table 8 on page 62 and to the description of the COMPAT option in *z/OS MVS Program Management: User's Guide and Reference*.

Notes: Unless SYSLMOD specifies a LOADLIB, both LKED and PM1 will result in the generation of a standard format CMS module. They differ in that LKED specifies that certain binder processing options are to work in a manner compatible with the linkage editor:

1. Where conflicts exist between the AMODE or RMODE of individual entry points or sections and the value specified in the AMODE or RMODE option, the option specification will prevail.

2. If a section is encountered in a module with a lower reusability than that specified on the REUS option, the reusability of the module is automatically downgraded. An information message is issued and the return code remains unchanged.

COMPRESS

Use this option to compress additional data that the binder stores with the executable program. This has no effect on the program size during execution, but can reduce the disk storage required to hold it. This option allows you to control whether the binder will attempt compression. You might want to prohibit compress in some cases.

For more information, refer to the description of the COMPRESS option in *z/OS MVS Program Management: User's Guide and Reference*.

CTL

CTL specifies a control file which may be used to define the following to the BIND command:

- Additional primary input file types of the form TXT*xxxxx*; these are inserted between SYSLIN and TEXT in the standard hierarchy.
- One or more TXTLIBs which will temporarily replace the global TXTLIB concatenation for the duration of the BIND command; these TXTLIBs are specified as parameters on one or more cards that start *BIND TXTS.
- Additional CMS style binder options which are inserted into the command line options at the point of the CTL option; these options are specified on one or more cards that start *BIND OPTS. The CTL option cannot be specified on a *BIND OPTS card.

Note: The file type of the control file must be CNTRL.

The CMS commands that use control files (UPDATE, VMFLOAD, PRELOAD, and XEDIT) ignore binder specific commands that appear as comments. For example, the lines beginning with *BIND in the sample DMSVM CNTRL file below would be ignored:

```
TEXT  MACS DMSGPI DMSOM IXXOM OSMACRO OSPSI HCPGPI HCPOM1 OSVSAM
TEXT  MACS OSMACRO1
PAT   AUXPAT   TX$  * LOCAL PATCHES
LCIXX AUXLIXX  TXC  * REXX AUX File and VVTLIXX Level for Local Mods
LCL   AUXLCL        * CMS AUX File and VVTLCL Level for Local Mods
CMS   AUXIXX   TXC  * REXX AUX FILE and VVTIXX Level for PTF service
TEXT  AUXVM         * CMS AUX FILE and VVTVM Level for PTF service
*BIND TXTS txtlib1 txtlib2 ...
*BIND OPTS option1 option2 ...
```

DISK

DISK directs binder SYSPRINT output to a disk file. If OUTPUT CMS is in effect, then output is directed to *fn* **SYSPRINT** *fm*, where *fn* and *fm* are the default file name and file mode respectively. If OUTPUT BFS is in effect, then output is directed to ***./fn*.i**, where *fn* is the default file name.

PRINT directs binder SYSPRINT output to the virtual printer (the command interface issues `FILEDEF SYSPRINT PRINTER`).

NOPRINT suppresses binder SYSPRINT output.

DISK NO, NODISK, and NOPRINT are all equivalent to PRINT NO.

For more information, refer to the description of the PRINT option in *z/OS MVS Program Management: User's Guide and Reference*.

Notes:

1. If none of these options are specified, then the default is DISK YES.
2. If a valid FILEDEF or PATHDEF for SYSPRINT already exists when the BIND command is invoked, then PRINT and DISK are ignored and SYSPRINT is directed to the FILEDEF/PATHDEF destination.
3. Refer to the OUTPUT option for a discussion of OUTPUT CMS and OUTPUT BFS.
4. Refer to Usage Note 3 on page 28 for a discussion of default file names and file modes.

DLL See DYNAM.

DYNAM

DYNAM controls whether a module being bound is to be enabled for dynamic linking.

DLL YES is equivalent to DYNAM DLL. DLL NO and NODLL are equivalent to DYNAM NO.

For more information, refer to the description of the DYNAM option in *z/OS MVS Program Management: User's Guide and Reference*.

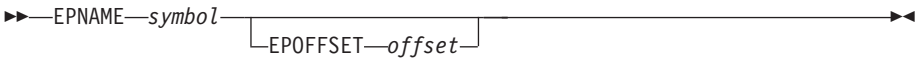
EDIT

EDIT controls retention of external symbol data, which is required for program modules to be editable.

NOEDIT and NE are equivalent to EDIT NO.

For more information, refer to the description of the EDIT option in *z/OS MVS Program Management: User's Guide and Reference*.

EPNAME



EPNAME specifies the program entry point as a symbol of up to 1024 characters, and an optional entry point offset as a decimal value. Enclose *symbol* in quotation marks if it contains blanks.

For more information, refer to the description of the EP option in *z/OS MVS Program Management: User's Guide and Reference*.

Note: EPOFFSET is only valid immediately after the EPNAME *symbol*.

FILETYPE

FILETYPE defines a file type for primary input processing. The specified file type is added to the top of the file type resolution hierarchy for CMS primary input files. There is no default value for FILETYPE.

For example, suppose that you had files called PROG1 TEXT and PROG1 GOFF containing text and goff data respectively for PROG1, and no other files with the file name PROG1. Entering `bind prog1` would cause PROG1 TEXT to be read as primary input and processed by the binder. To process PROG1 GOFF instead, you could enter `bind prog1 (ft goff`.

If primary input is to be specified in a load list file, then only one CMS record file system file name may be specified and the FILETYPE option must be used with a value of EXEC. Refer to the description of the *filename* parameter of more information.

FILL

FILL specifies a hex value defining a byte to be used to fill uninitialized areas of the program object.

For more information, refer to the description of the FILL option in *z/OS MVS Program Management: User's Guide and Reference*.

GID

GID specifies the Group ID attribute to be set for program objects and sidedecks when they are written to the BFS. To set Group ID attributes, you must have superuser authority or be the owner of the file or directory.

For more information, refer to the description of the GID option in *z/OS MVS Program Management: User's Guide and Reference*.

HOBSET

HOBSET controls whether the high-order bit in each V-type address constant is to be set according to the AMODE of the target symbol.

NOHOBSET is equivalent to HOBSET NO.

For more information, refer to the description of the HOBSET option in *z/OS MVS Program Management: User's Guide and Reference*.

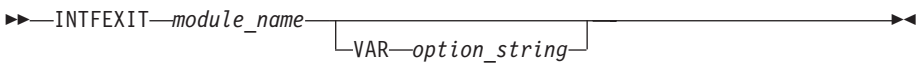
INCL See CMSINCL.

INFO

When the INFO option is specified, the binder produces a report listing the PTF level for all binder sections to which maintenance has been applied. This report appears at the end of the binder SYSPRINT or SYSLOUT data set, prior to the message summary report.

For more information, refer to the description of the INFO option in *z/OS MVS Program Management: User's Guide and Reference*.

INTFEXIT



INFEXIT specifies an interface validation exit, and optionally an option string to pass to it. The exit routine must exist as a module file on an accessed disk or directory.

For each adjacent series of EXITS options (INTFEXIT, MSGEXIT, and SAVEXIT), including any associated VAR options (for INTFEXIT and MSGEXIT), a single binder EXITS option is passed to the binder.

For more information, refer to the description of the EXITS option for the interface validation exit in *z/OS MVS Program Management: User's Guide and Reference*.

Note: VAR is only valid immediately after INTFEXIT (or MSGEXIT).

LET

LET controls the severity level for the acceptability of errors.

LET YES and LET are equivalent to LET 8. LET NO and NOLET are equivalent to LET 4.

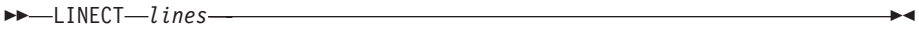
For more information, refer to the description of the LET option in *z/OS MVS Program Management: User's Guide and Reference*.

LIBE

LIBE controls whether the BIND command searches the text libraries defined by a previously issued GLOBAL TXTLIB command for unresolved references. The default is LIBE.

NOLIBE is equivalent to LIBE NO.

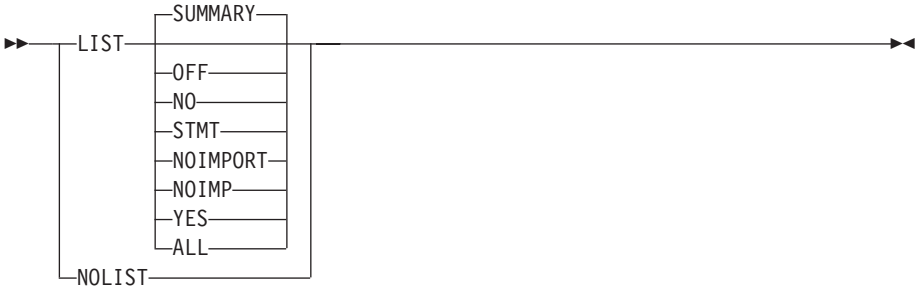
LINECT



LINECT specifies the number of lines per page of the binder output listings.

For more information, refer to the description of the LINECT option in *z/OS MVS Program Management: User's Guide and Reference*.

LIST

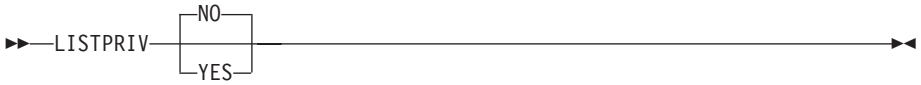


LIST controls the contents of the output listings.

LIST YES and LIST are equivalent to LIST SUMMARY. LIST NO and NOLIST are equivalent to LIST OFF. LIST NOIMP is equivalent to LIST NOIMPORT.

For more information, refer to the description of the LIST option, and "Chapter 8. Interpreting Binder Listings" in *z/OS MVS Program Management: User's Guide and Reference*.

LISTPRIV



LISTPRIV obtains a list of unnamed (private code) sections. Unnamed sections are sections that were input to the binder with no name (the name consists of blanks).

For more information, refer to the description of the LISTPRIV option in *z/OS MVS Program Management: User's Guide and Reference*.

MACRO

See CMSMACRO

MAP

MAP controls whether or not a module map is produced.

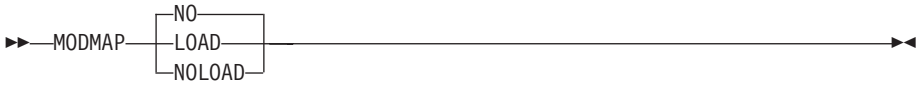
NOMAP is equivalent to MAP NO.

For more information, refer to the description of the MAP option in *z/OS MVS Program Management: User's Guide and Reference*.

MIXED

See CASE.

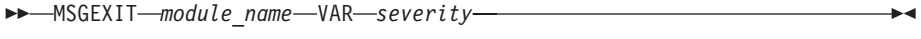
MODMAP



You can build a map of the module contents in a separate section as part of the module being bound by coding the MODMAP option.

For more information, refer to the description of the MODMAP option in *z/OS MVS Program Management: User's Guide and Reference*.

MSGEXIT



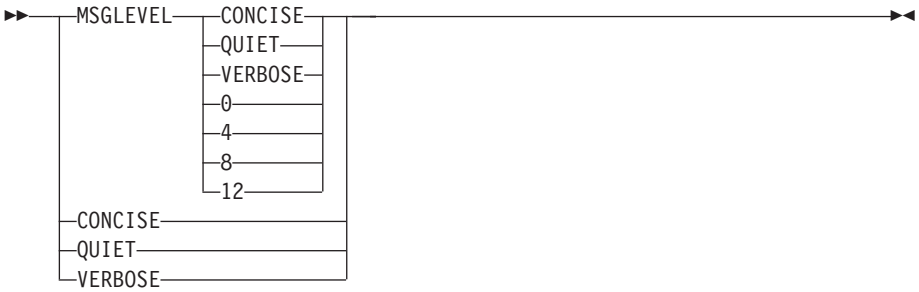
MSGEXIT specifies a message processing exit and the minimum severity of messages to be passed to the exit. The exit routine must exist as a module file on an accessed disk or directory.

For each adjacent series of EXITS options (INTFEXIT, MSGEXIT, and SAVEXIT), including any associated VAR options (for INTFEXIT and MSGEXIT), a single binder EXITS option is passed to the binder.

For more information, refer to the description of the EXITS option for the message exit in *z/OS MVS Program Management: User's Guide and Reference*.

Note: VAR is only valid immediately after MSGEXIT (or INTFEXIT).

MSGLEVEL



MSGLEVEL specifies the minimum severity level of messages to be issued.

BIND command level message suppression can be in one of two modes (VERBOSE or QUIET) which control whether informational command interface messages are issued. The command options CONCISE, VERBOSE and QUIET may be specified either as a keyword or as a value on the MSGLEVEL option, and they relate to the command mode and to the binder MSGLEVEL option as summarized in Table 1:

•

Table 1. Message Level Options Passed to the Binder

Specified	Cmd Level	Bind Level
—	QUIET	—
CONCISE	QUIET	—
QUIET	QUIET	4
VERBOSE	VERBOSE	0
0	VERBOSE	0
4	QUIET	4
8	QUIET	8
12	QUIET	12

Notes:

1. The "Specified" column shows what is specified as a value for the MSGLEVEL option in CMS style.
2. The "Cmd Level" column shows the resulting message suppression mode setting.
3. The "Bind Level" column shows the MSGLEVEL option value passed to the binder (where a dash means nothing is passed).

The severity level of messages is summarized as follows:

0 Informational

4 Warning

8 Error

12 Severe

For more information, refer to the description of the MSGLEVEL option in *z/OS MVS Program Management: User's Guide and Reference*.

OL

OL determines the setting of an attribute that controls how the program can be brought into virtual storage.

NOOL is equivalent to OL NO.

For more information, refer to the description of the OL option in *z/OS MVS Program Management: User's Guide and Reference*.

OPTIONS

OPTIONS specifies the ddname for a data set containing binder options to be used during current processing.

For more information, refer to the description of the OPTIONS option in *z/OS MVS Program Management: User's Guide and Reference*.

OUTPUT

OUTPUT controls the file system target for output files.

If OUTPUT CMS is specified or the OUTPUT option is omitted and the first primary input file is a CMS file, then output files are written as CMS files.

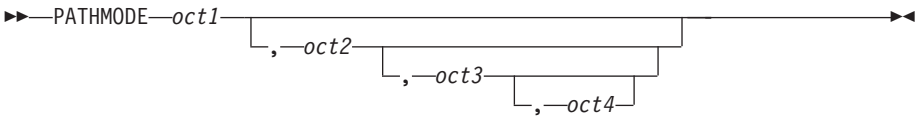
If OUTPUT BFS is specified or the OUTPUT option is omitted and the first primary input file is a BFS file, then output files are written as BFS files. There is no default value for OUTPUT.

Note: The actual output files produced are also affected by:

- The use of the DISK and PRINT options
- Any pre-existing FILEDEFs or PATHDEFs for ddnames used by the binder; refer to Usage Note 5 on page 29 for more information.
- The use of the DYNAM option and whether or not its use causes a side file to be generated by the binder
- The presence of NAME statements in the primary input

Refer to Usage Note 3 on page 28 for a discussion of default file names and file modes.

PATHMODE



PATHMODE is used to set OpenExtensions file attributes for program objects and sidedecks by specifying a string of 4 digits in the range 0-7.

If fewer than four digits are specified, the string is padded to the right with zeros. If more than four digits are specified, the excess digits are discarded.

For more information, refer to the description of the PATHMODE option in *z/OS MVS Program Management: User's Guide and Reference*.

PRINT

See DISK.

REUS

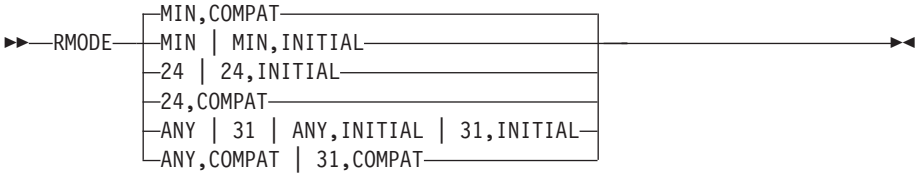
REUS specifies the reusability characteristics of the program module.

REUS YES and SERIAL are equivalent to REUS SERIAL. REUS NO and NOREUS are equivalent to REUS NONE.

For more information, refer to the description of the REUS option in *z/OS MVS Program Management: User's Guide and Reference*.

Note: Additional negative alternative forms (such as NORENT and NOREFR, which are supported by the z/OS MVS Program Management Binder) are not supported by the BIND command because they create an ambiguous syntax.

RMODE



RMODE sets the residence mode.

For more information, refer to the description of the RMODE option in *z/OS MVS Program Management: User's Guide and Reference*.

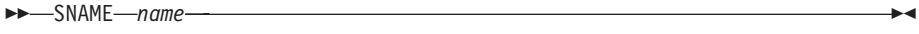
SAVEXIT

SAVEXIT specifies a save exit. The exit routine must exist as a module file on an accessed disk or directory.

For each adjacent series of EXITS options (INTFEXIT, MSGEXIT, and SAVEXIT), including any associated VAR options (for INTFEXIT and MSGEXIT), a single binder EXITS option is passed to the binder.

For more information, refer to the description of the EXITS option save exit in *z/OS MVS Program Management: User's Guide and Reference*.

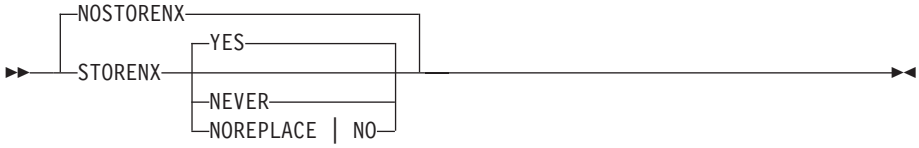
SNAME



SNAME specifies a name of 1 to 1024 characters in length for saving the program module. Enclose *name* in quotation marks if it contains blanks.

For more information, refer to the description of the SNAME option in *z/OS MVS Program Management: User's Guide and Reference*.

STORENX



The STORENX option specifies the conditions under which the binder is to store a nonexecutable program module.

For more information, refer to the description of the STORENX option in *z/OS MVS Program Management: User's Guide and Reference*.

STR See CMSSTR.

STRIPCL

The STRIPCL option allows you to remove unneeded classes from a program object or load module. For a class to be eligible for removal, in addition to having the removable attribute:

- It must not be a binder-owned class (a class whose name starts with a B_).
- It must not contain any RLD entries.

For more information, refer to the description of the STRIPCL option in *z/OS MVS Program Management: User's Guide and Reference*.

STRIPSEC

The STRIPSEC option allows you to remove unneeded sections from a program object or load module.

For more information, refer to the description of the STRIPCL option in *z/OS MVS Program Management: User's Guide and Reference*.

SYSTEM
See CMSSYSTEM.

TERM

TERM directs binder SYSTERM output to the console (the command interface issues `FILEDEF SYSTERM TERMINAL`).

NOTERM causes binder SYSTERM output to be suppressed.

TYPE is equivalent to TERM. TYPE NO, NOTYPE, and NOTERM are all equivalent to TERM NO.

For more information, refer to the description of the TERM option in *z/OS MVS Program Management: User's Guide and Reference*.

Note: If there is a valid pre-existing FILEDEF or PATHDEF for SYSTERM when the BIND command is invoked, then TERM is ignored and SYSTERM is directed to the FILEDEF/PATHDEF destination.

TYPE See TERM.

UID

UID specifies the User ID attribute to be set for program objects and sidedecks when they are written to the BFS. To set User ID attributes, you must have superuser authority.

For more information, refer to the description of the UID option in *z/OS MVS Program Management: User's Guide and Reference*.

UPCASE

UPCASE determines if unresolved function references that are marked as renameable and that are not imported are set to uppercase if they are eight characters or less in length. This occurs during the final automatic library call.

NOUPCASE is equivalent to UPCASE NO.

For more information, refer to the description of the UPCASE option in *z/OS MVS Program Management: User's Guide and Reference*.

UPPER
See CASE.

VAR See INTFEXIT, MSGEXIT, and SAVEXIT.

WKSABOVE

WKSABOVE specifies in units of 1 KB the amount of space available for binder processing above the 16 MB line (minimum 1024 KB).

For more information, refer to the description of the WKSPACE option in *z/OS MVS Program Management: User's Guide and Reference*. The WKSABOVE specification corresponds to the WKSPACE *value2* specification.

WKSBELOW

WKSBELOW specifies in units of 1 KB the amount of space available for binder processing below the 16 MB line (minimum 96 KB).

For more information, refer to the description of the WKSPACE option in *z/OS MVS Program Management: User's Guide and Reference*. The WKSBELOW specification corresponds to the WKSPACE *value1* specification.

XC See CMSXC.

XREF

XREF controls printing of a cross-reference table.

NOXREF is equivalent to XREF NO.

For more information, refer to the description of the XREF option in *z/OS MVS Program Management: User's Guide and Reference*.

Table 2. z/OS MVS Program Management Binder to BIND Command Options Cross-reference

z/OS MVS Program Management Binder option	BIND command option
AC	not available

Table 2. z/OS MVS Program Management Binder to BIND Command Options Cross-reference (continued)

z/OS MVS Program Management Binder option	BIND command option
ALIASES	not available
ALIGN2	ALIGN2
AMODE	AMODE
CALL	CALL
CALLIB	not available **Note:** CALLIB has no effect if specified in MVS binder style, but may be specified as a parameter on the SETOPT control statement.
CALLERID	not available
CASE	CASE, UPPER, MIXED
COMPAT	COMPAT
COMPRESS	COMPRESS
DC	not available
DCBS	not available
DYNAM	DYNAM, DLL
EDIT	EDIT
EP	EPNAME, EPOFFSET
EXITS	INTFEXIT, MSGEXIT, SAVEXIT, VAR
EXTATTR	not available
FETCHOPT	not available
FILL	FILL
GID	GID
HOBSET	HOBSET
INFO	INFO
LET	LET
LINECT	LINECT
LIST	LIST
LISTPRIV	LISTPRIV
LNAME	not available
MAP	MAP
MAXBLK	not available
MODLIB	not available **Note:** MODLIB has no effect if specified in MVS binder style, but may be specified as a parameter on the SETOPT control statement.
MODMAP	MODMAP
MSGLEVEL	MSGLEVEL
NAME	not available
OL	OL
OPTIONS	OPTIONS

Table 2. z/OS MVS Program Management Binder to BIND Command Options Cross-reference (continued)

z/OS MVS Program Management Binder option	BIND command option
OVLY	not available
PATHMODE	PATHMODE
PRINT	PRINT, DISK
RES	not available
REUS	REUS, RENT, REFR
RMODE	RMODE
SCTR	not available
SNAME	SNAME
SIZE	not available
SSI	not available
STORENX	STORENX
STRIPCL	STRIPCL
STRIPSEC	STRIPSEC
TERM	TERM
TEST	not available
TRAP	not available
UID	UID
UPCASE	UPCASE
WKSPACE	WKSBELOW, WKSABOVE
XCAL	not available
XREF	XREF

Usage Notes

1. The OpenExtensions c89 command can also be used to invoke the BIND command. Refer to *z/VM: OpenExtensions Commands Reference* for more information.
2. **Primary Input Processing**

   Primary input processing by the BIND command proceeds according to the following rules:

   - Each primary input file is processed separately and in the order specified on the BIND command.
   - CMS files are processed according to their file type and their record format and length:
     - Files that are fixed format with a record length of 80 can contain control statements, ordinary object code (OBJ), extended object code (XOBJ, as processed by the C prelinker), or generalized object code (GOFF). These files are preprocessed to check for the presence of NAME statements and buffered in a work file.
     - Files that are fixed format with a record length other than 80 are rejected with message DMSBCO1609E.
     - Files that are variable format are passed to the binder by issuing a FILEDEF command and generating an INCLUDE statement in the work

file that refers to the ddname used on the FILEDEF. The ddnames used are generated by the BIND command and are of the form INCL*nnnn*, where *nnnn* is a numeric character sequence that increments from 0000 for each required FILEDEF. The only valid variable format input is either GOFF or an extended format MODULE.

- BFS files are processed according to whether or not they contain a program object:
  - Files containing program objects are passed to the binder as is by generating an **INCLUDE** statement in the work file referring to the path for the file.
  - All other files are assumed to contain control statements, ordinary object code (OBJ), extended object code (XOBJ, as processed by the C prelinker), or generalized object code (GOFF) in simulated card image format. They are read 80 bytes at a time, preprocessed to check for the presence of `NAME` statements, and buffered in the work file.
- Whenever a `NAME` statement is found, the binder include API is invoked to process all the data buffered to the work file, including any generated `INCLUDE` statements, and the resulting data is bound into a program object and saved using the name specified on the `NAME` statement.
- If no `NAME` statements have been found when end-of-file is reached on the last primary input file, then the binder include API is invoked to process all the data buffered to the work file, including any generated `INCLUDE` statements, and the resulting data is bound and saved as though a `NAME` statement specifying the default file name with the replace option were found.
- If any further input is found after the last `NAME` statement processed, then when the end-of-file is reached on the last primary input file, the binder include API is invoked to process all the data buffered to the work file, including any generated `INCLUDE` statements. The resulting data is bound and saved using a generated name of the form **TEMPNAM***n*, where *n* is a numeric character in the range 0-9. If all possible temporary names have been used, the program object is not saved.

Note: The default file name and file mode are discussed in 3.

3. **Default File Name and File Mode Determination**

   The BIND command determines a default file name and file mode that might be used to determine the placement of output files. The determination of the default file name and file mode depends on the location of the first primary input file.

   When the first primary input file is a CMS file the default file name is the uppercase form of the specified file name. The default file mode will be the file mode associated with the first primary input file if it is accessed in R/W mode, or else it will be the first available R/W file mode.

   When the first primary input file is a BFS file and **OUTPUT BFS** is in effect, the default file name is the file name as specified on the BIND command, subject to the file name extension convention discussed below. The default file mode is the first available R/W file mode.

   When the first primary input file is a BFS file and **OUTPUT CMS** is in effect, the default file name is the file name as specified on the BIND command. It is converted to uppercase unless the specified file name is not a valid CMS file name. If the specified file name is not a valid CMS file name, then a warning message is issued and the default file name is set to "$BINDER$." The default file mode is the first available R/W file mode in any case.

To see how the default file name and file mode are used by the BIND command, refer to Usage Notes 2 on page 27 and 5.

4. **BFS file name extension convention**

   The BIND command recognizes and uses a convention for BFS file name extensions based on what is used by the c89 command. The convention is only meaningful when the default file names are used. The following extensions are recognized or used:

   .i Signifies a listing file. This extension is appended to the default file name when the BIND command generates a BFS file name for SYSPRINT.

   .m Signifies a program object file. This extension is appended to the default file name when the BIND command generates a BFS file name for SYSLMOD, unless the first primary input file used a **.o** extension, in which case the default file name is used unextended.

   .o Signifies an object file. If this extension is used on the first primary input file, then the default file name becomes the name without the extension.

   .x Signifies a side file. This extension is appended to the default file name when the BIND command generates a BFS file name for SYSDEFSD.

5. **FILEDEF and PATHDEF Usage**

   The BIND command uses the CMS Binder implementation of the binder API to control binder processing, but ultimately the program being run is the z/OS MVS Program Management Binder. Because of this, input and output to the binder is controlled by ddnames that are manipulated using the CMS FILEDEF and OPENVM PATHDEF CREATE commands. In general, the BIND command provides suitable defaults for the task at hand, but an understanding of the ddname requirements of the binder makes it easier to use and facilitates the performance of more advanced binding tasks.

   See also “FILEDEF/PATHDEF - Relationship with DD Statement” on page 58 and “Binder Input and Output” on page 57 for general information about the CMS Binder and file specification.

   The BIND command uses the following ddnames:

   SYSLIN

   SYSLIN is the default ddname for primary input to the binder. The BIND command only uses the SYSLIN ddname if no file names are specified as arguments, in which case all active FILEDEFs and PATHDEFs for SYSLIN are used. FILEDEFs are processed in the order that the FILEDEF commands were entered, and all the FILEDEFs are processed before any PATHDEF is processed. For example:

   ```
   filedef syslin disk file1 text a (concat
   openvm pathdef create syslin ./file3
   filedef syslin disk file2 text a (concat
   ```

   This would result in file1, file2 and file3 (in that order) being processed by the binder.

   Note: Only one PATHDEF at a time can be active for a given ddname.

   SYSUT1

   SYSUT1 is used as a work file for staging input to the binder. A FILEDEF is always issued for this ddname, regardless of any pre-existing FILEDEF or PATHDEF. The command issued is

```
filedef sysut1 disk fn sysut1 fm
```

where *fn* and *fm* are the default file name and file mode.

SYSUT1 is used for staging input files that might contain control statements so that NAME control statements can be detected. When a NAME statement is encountered the command interface uses include and bind workmod API calls to process the contents of SYSUT1, and then issues a save workmod API call to save the bound program object.

options
Any ddname may be used for the options file. The options file is read by the binder if the **OPTIONS** option is specified.

SYSLIB
SYSLIB is used by the binder to resolve external references when automatic library call processing is invoked. The command interface does not provide a default FILEDEF for this ddname.

Note: The library used to resolve external references can be altered using the CALLIB option. When using the BIND command to invoke the binder, the CALLIB option is only effective if set with a SETOPT control statement. It has no effect if specified as an MVS binder style option.

include
Any ddname may be used for files referred to by include statements.

SYSPRINT
SYSPRINT is used by the binder for diagnostic output. The SYSPRINT target is determined as follows:

- If the **NOPRINT** option is specified, SYSPRINT output is not produced and no FILEDEF or PATHDEF for SYSPRINT is issued.
- If the **NOPRINT** option is not specified and a FILEDEF or PATHDEF for SYSPRINT exists when the BIND command is issued, then the pre-existing FILEDEF or PATHDEF is used.
- If the **DISK** option is specified and no FILEDEFs or PATHDEFs for SYSPRINT exist when the BIND command is issued, then the BIND command issues an appropriate FILEDEF or PATHDEF for SYSPRINT. If **OUTPUT CMS** is in effect, then the following is issued:

  ```
  filedef sysprint disk fn SYSPRINT fm
  ```

  where *fn* and *fm* are the default file name and file mode respectively. If **OUTPUT BFS** is in effect, then the following is issued:

  ```
  openvm pathdef create sysprint ./fn.i
  ```

  where *fn* is the default file name.

  Notes:

  a. Refer to the **OUTPUT** option for a discussion of **OUTPUT CMS** and **OUTPUT BFS**.
  b. Refer to Usage Note 3 on page 28 for a discussion of default file name and file modes.

- If the **PRINT** option is specified and no FILEDEFs or PATHDEFs for SYSPRINT exist when the BIND command is issued, then the BIND command issues a FILEDEF to direct sysprint to the virtual printer:

```
filedef sysprint printer
```

Note: If more than one FILEDEF or PATHDEF exists, or a FILEDEF exists that specifies the **CONCAT** option or is not to disk, printer, or terminal, then the BIND command considers the specification to be in error and terminates without invoking the binder.

SYSLMOD

SYSLMOD determines where program objects created by the binder are placed. The SYSLMOD target is determined as follows:

- If neither a FILEDEF nor a PATHDEF for SYSLMOD exists when the BIND command is invoked, then the SYSLMOD target used depends on whether **OUTPUT CMS** or **OUTPUT BFS** is in effect, and on the presence of `NAME` statements in the primary input stream.
  - If **OUTPUT CMS** is in effect and no `NAME` statements are found in primary input, then the single program object produced by the binder is placed in a file called *fn* **MODULE** *fm*, where *fn* and *fm* are the default file name and file mode respectively.
  - If **OUTPUT CMS** is in effect and `NAME` statements are found in primary input, then a program object with file type MODULE is created for each `NAME` statement found using the symbol specified on the statement as the file name.
  - If **OUTPUT BFS** is in effect and no `NAME` statements are found in primary input, then the single program object produced by the binder is placed in a file called *fn* or *fn***.m** in the current working directory, where *fn* is the default file name. (The **.m** extension is appended if the first primary input file is a BFS file without a **.o** extension. This helps avoid inadvertently overwriting an input text file with an output program object file.)
  - If **OUTPUT BFS** is in effect and `NAME` statements are found in primary input, then a program object is created in the current working directory for each `NAME` statement found using the symbol specified on the statement as the file name.

Table 3. Program Module Output Determination when No SYSLMOD is Predefined

OUTPUT Option	First File Name Specified on BIND Command	NAME Statement	SNAME Option	Output Module
none	*fn*	none	none	*fn* MODULE *dfm*
			sname	*sname* MODULE *dfm*
		name	none	*name* MODULE *dfm*
			sname	*name* MODULE *dfm*
	directory/file	none	none	*./file* or *./file* .m
			sname	*./sname*
		name	none	*./name*
			sname	*./name*

Table 3. Program Module Output Determination when No SYSLMOD is Predefined (continued)

OUTPUT Option	First File Name Specified on BIND Command	NAME Statement	SNAME Option	Output Module
CMS	*fn ft fm*	none	none	*fn* MODULE *dfm*
			sname	*sname* MODULE *dfm*
		name	none	*name* MODULE *dfm*
			sname	*name* MODULE *dfm*
	directory/file	none	none	*file* MODULE *dfm*
			sname	*sname* MODULE *dfm*
		name	none	*name* MODULE *dfm*
			sname	*name* MODULE *dfm*
BFS	*fn ft fm*	none	none	*./fn*
			sname	*./sname*
		name	none	*./name*
			sname	*./name*
	directory/file	none	none	*./file* or *./file* .m
			sname	*./sname*
		name	none	*./name*
			sname	*./name*

Notes about the table:

1. This is a decision table. To use it, start in the left-hand column and work to the right choosing the row according to the values that apply to your situation. The cell in the right-most column that you arrive at describes the output program module file that will be created.
2. *dfm* represents the default file mode determined by the BIND command.
3. For BFS input that results in a CMS module file output, if the file name is used for the module name it is changed to uppercase, checked for validity, and may be truncated.
4. The SNAME option value can be set as a BIND command option, or with a SETOPT control statement in the input stream.

- If a FILEDEF for SYSLMOD exists when the BIND command is invoked, it must specify a file type of either MODULE or LOADLIB.
  - If the file type is MODULE and no `NAME` statements are found in primary input, then the single program object produced by the binder is placed in the specified file.
  - If the file type is MODULE and `NAME` statements are found in primary input, then a program object with file type MODULE is created for each `NAME` statement found using the symbol specified on the statement as the file name.
  - If the file type is LOADLIB and no `NAME` statements are found in primary input, then the single program object produced by the binder is placed in the specified file using the default file name as a member name.
  - If the file type is LOADLIB and `NAME` statements are found in primary input, then a load module is created for each `NAME` found using the symbol specified on the statement as the member name.

  The following table summarizes the relationship between the SYSLMOD specification and the output program object file when a

FILEDEF for SYSLMOD exists prior to invoking the BIND command. The OUTPUT option does not affect these scenarios.

Table 4. Program Module Output Determination (SYSLMOD FILEDEF Exists)

SYSLMOD Specification	NAME Statement	SNAME Option	Output Module
fn MODULE *fm*	none	none	*fn* MODULE *fm*
		sname	*sname* MODULE *fm*
	name	none	*name* MODULE *fm*
		sname	*name* MODULE *fm*
fn LOADLIB *fm*	none	none	*fn* LOADLIB(*infile1*) *fm*
		sname	*fn* LOADLIB(*sname*) *fm*
	name	none	*fn* LOADLIB(*name*) *fm*
		sname	*fn* LOADLIB(*name*) *fm*
fn LOADLIB(*member*) *fm*	none	none	*fn* LOADLIB(*member*) *fm*
		sname	*fn* LOADLIB(*sname*) *fm*
	name	none	*fn* LOADLIB(*name*) *fm*
		sname	*fn* LOADLIB(*name*) *fm*

Notes about the table:

1. This is a decision table. To use it, start in the left-hand column and work to the right choosing the row according to the values that apply to your situation. The cell in the right-most column that you arrive at describes the output program module file that is created.
2. *infile1* represents the first file name specified on the BIND command
3. The SNAME option value can be set as a BIND command option or using a SETOPT control statement in the input stream.

- If a PATHDEF for SYSLMOD exists when the BIND command is invoked, the directory or file it specifies must exist.
  - If it refers to a file and no `NAME` statements are found in primary input, then the single program object produced by the binder replaces the named file.
  - If it refers to a file and `NAME` statements are found in primary input, then a program object is created in the directory containing the file specified by the PATHDEF for each `NAME` statement found using the symbol specified on the statement as the file name.
  - If it refers to a directory and no `NAME` statements are found in primary input, then the single program object produced by the binder is placed in the directory using the default file name.
  - If it refers to a directory and `NAME` statements are found in primary input, then a program object is created in the directory for each `NAME` statement found using the symbol specified on the statement as the file name.

  The following table summarizes the relationship between the SYSLMOD specification and the output program object file when a PATHDEF for SYSLMOD exists prior to invoking the BIND command. The OUTPUT option does not affect these scenarios.

Table 5. Program Module Output Determination (SYSLMOD PATHDEF Exists)

SYSLMOD Specification	NAME Statement	SNAME Option	Output Module
directory	none	none	*directory/infile1*
		sname	*directory/sname*
	name	none	*directory/name*
		sname	*directory/name*
directory/file	none	none	*directory/file*
		sname	*directory/sname*
	name	none	*directory/name*
		sname	*directory/name*

Notes about the table:

1. This is a decision table. To use it, start in the left-hand column and work to the right choosing the row according to the values that apply to your situation. The cell in the right-most column that you arrive at describes the output program module file that is created.
2. *infile1* represents the first file name specified on BIND command.
3. The expected behavior is dependent on the existence of the specified path. If the directory or file does not exist the message IEW2785S is issued.
4. The SNAME option value can be set as a BIND command option or using a SETOPT control statement in the input stream.

Notes:

a. Program objects are saved with an implicit disposition of replace if no `NAME` statements are encountered in the primary input stream.

b. If the symbol name to be used for a file name exceeds the maximum length allowed for the target file system, the name is truncated to the maximum length and an error message issued.

c. The default file name is always uppercase when **OUTPUT CMS** is in effect, and mixed case when **OUTPUT BFS** is in effect, and `NAME` statement symbols may be mixed case, but whichever is the source of the file name used, the case of the actual symbol, and therefore file name used, depends on the **CASE** option setting.

d. The character set supported for BFS file names is not the same as that for binder symbol names. Because the BIND command always uses a symbol name (an "SNAME") when saving program objects, SYSLMOD PATHDEFs to files with names that contain characters that are not valid in binder symbol names should not be used.

e. If more than one FILEDEF or PATHDEF exists, or a FILEDEF exists that specifies the **CONCAT** option or is not to disk, then the BIND command considers the specification to be in error and terminates without invoking the binder.

f. When using the BIND command to invoke the binder, the MODLIB option can only be specified as a parameter on the SETOPT control statement. It is not a valid CMS style option and is ignored if specified as an MVS binder style option.

SYSTERM

SYSTERM is used by the binder for diagnostic messages. The SYSTERM target is determined as follows:

- If the **NOTERM** option is specified, SYSTERM output is not produced and no FILEDEF or PATHDEF for SYSTERM is issued.
- If the **NOTERM** option is not specified and a FILEDEF or PATHDEF for SYSTERM exists when the BIND command is issued, then the preexisting FILEDEF or PATHDEF is used.
- If the **TERM** option is specified and no FILEDEFs or PATHDEFs for SYSTERM exist when the BIND command is issued, then the BIND command issues a FILEDEF to direct SYSTERM to the console:

  ```
  filedef systerm terminal
  ```

  Note: If more than one FILEDEF or PATHDEF exists, or a FILEDEF exists that specifies the **CONCAT** option or is not to disk, printer, or terminal, then the BIND command considers the specification to be in error and terminates without invoking the binder.

SYSDEFSD

SYSDEFSD is used by the binder to save any side file generated when the **DYNAM=DLL** option is used. The SYSDEFSD target is determined as follows:

- If neither a FILEDEF or a PATHDEF for SYSDEFSD exists when the BIND command is invoked, then the SYSDEFSD target used depends on whether **OUTPUT CMS** or **OUTPUT BFS** is in effect.

  If **OUTPUT CMS** is in effect, then the BIND command issues a FILEDEF for SYSDEFSD:

  ```
  filedef sysdefsd disk fn sysdefsd fm
  ```

  where *fn* and *fm* are the default file name and file mode respectively.

  If **OUTPUT BFS** is in effect, then the BIND command issues a PATHDEF for SYSDEFSD:

  ```
  openvm pathdef create sysdefsd ./fn.x
  ```

  where *fn* is the default file name.

  Note: If `NAME` statements are found in primary input and more than one side file is generated, then only the last side file generated remains in the file when the bind process has completed.
- If a FILEDEF for SYSDEFSD exists when the BIND command is invoked, then it is used as the side file target.

  Note: If `NAME` statements are found in primary input and more than one side file is generated, then only the last side file generated remains in the file when the bind process has completed.
- If a PATHDEF for SYSDEFSD exists when the BIND command is invoked, the directory or file it specifies must exist.

  Notes:

  a. If the PATHDEF refers to a file and `NAME` statements are found in primary input and more than one side file is generated, then only the last side file generated remains in the file when the bind process has completed.

  b. If the PATHDEF refers to a directory and no `NAME` statements are found in primary input, then if a side file is produced by the binder, it is placed in the directory using the same name as is used to save the program object. This means that if the

SYSDEFSD PATHDEF specifies the same directory that the program object is saved into, the side file overwrites the program object.

c. If the PATHDEF refers to a directory and `NAME` statements are found in primary input, then any side files produced are created in the directory using the symbol specified on the associated statement as the file name. This means that if the SYSDEFSD PATHDEF specifies the same directory as the program objects are saved into, the side files overwrite their corresponding program objects.

d. The only way to successfully write multiple side files in a single bind process is to issue a PATHDEF for SYSDEFSD that specifies a directory that is different from the directory (normally the current working directory) into which the program objects are saved.

6. **Resolution of External References**

   The following flowchart summarizes the external reference resolution process for the BIND command and how it is affected by various option specifications. For more details about the resolution process, see Figure 4 on page 71.

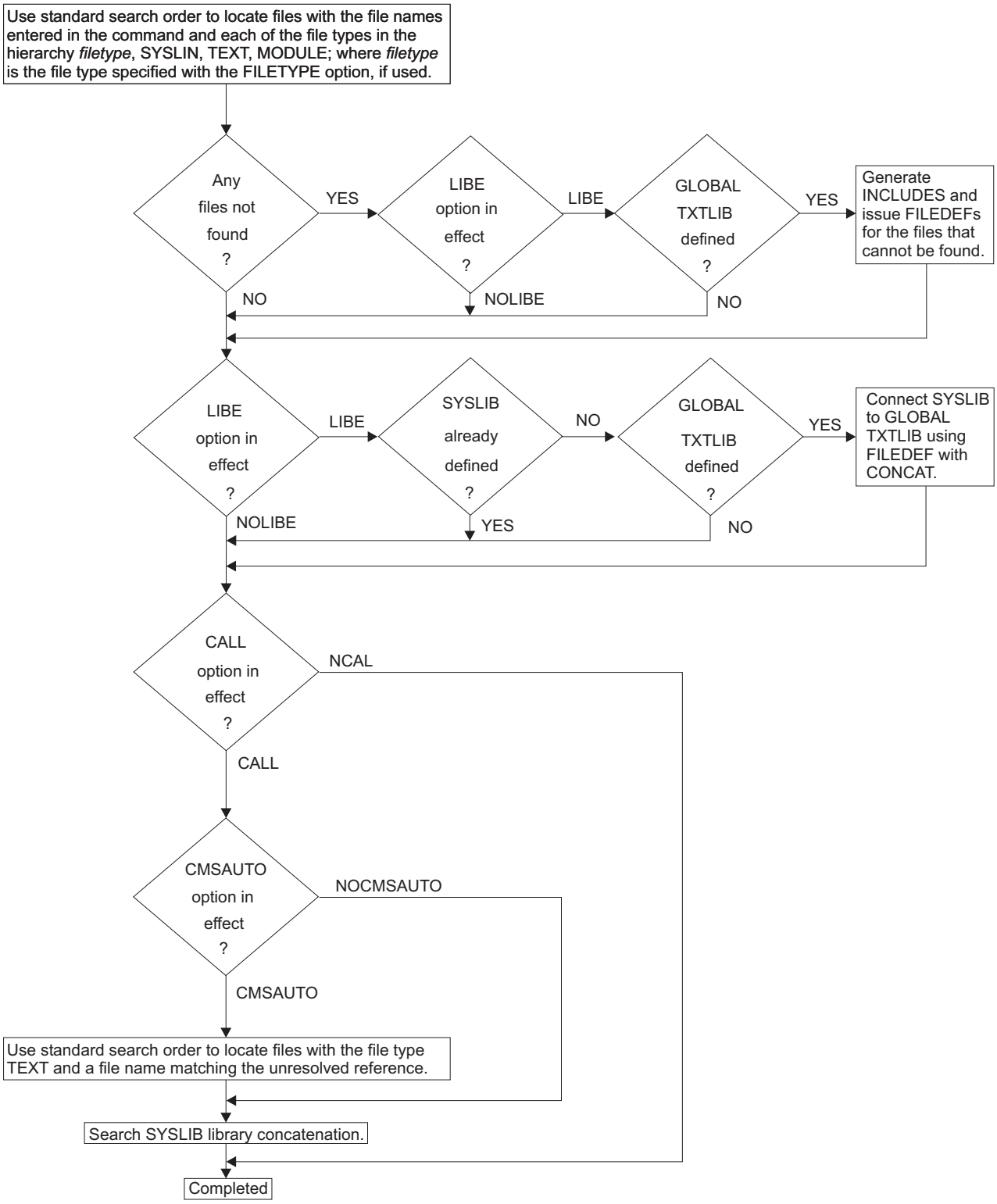


Figure 2. External Reference Resolution Process for the BIND Command

Examples

In some cases the behavior described may vary from what you would see on your own system. For example, if your installation defaults (established by the

IEWBODEF CSECT, see "Defining Installation Defaults" on page 88) specify NO as the value for the CALL option, the external references examples do not operate as described.

Examples of Using the BIND Command from the CMS Ready Prompt

1. **Binding a Single Text Deck with No External References**

   Assume you have a file called FILE1 TEXT on your A-disk containing object code produced by a language compiler. Enter the following:

   ```
   bind file1
   ```

   This example binds the text file and produces a program module called FILE1 MODULE on your A-disk. A listing file called FILE1 SYSPRINT is also produced.

2. **Binding a Single Text Deck with External References**

   Suppose FILE2 is another program with external references resolved by members of text libraries called MYLIB TXTLIB and PRODLIB TXTLIB. Enter the following:

   ```
   bind file2
   ```

   This example results in error messages from the binder because no SYSLIB is available for autocall processing and because of the unresolved external references. Enter the following:

   ```
   global txtlib mylib prodlib
   bind file2 ( /nocmsauto
   ```

   This example binds the text file and resolves the external references to produce a program object called FILE2 MODULE on your A-disk. A listing file called FILE2 SYSPRINT is also produced.

   Notes:

   a. The binder searches text libraries in the GLOBAL TXTLIB concatenation because the LIBE option is in effect by default. This automatic search is not done if the NOLIBE option is specified.

   b. The NOCMSAUTO option ensures that the external references are satisfied from the SYSLIB concatenation and not by TEXT files in the standard search order; see "CMSAUTO" on page 63.

3. **Specifying binder options**

   The CMS Binder command interface allows you to specify binder options either as "mapped" options, or as native binder options in an options string. The CMS system attribute is implemented using the CMS Binder API CMSSYSTEM option, which is mapped as the SYSTEMoption. So if you also wanted to set addressing and residence mode attributes for FILE1, you could enter any of the following:

   ```
   bind file1 (system amode 31 rmode any
   bind file1 ( / cmssytem=yes,amode=31,rmode=any
   bind file1 (system / amode=31,rmode=any
   ```

   Note: Options appearing before the slash (/) separator use a CMS-style syntax, defined elsewhere in this manual, while options appearing after the slash use binder JCL parameter string syntax described in *z/OS MVS Program Management: User's Guide and Reference*.

4. **Binding Several Text Decks with One Command**

Suppose you want to bind FILE1 and FILE2 into a single module. Enter the following:

```
filedef syslib disk mylib txtlib *
bind file1 file2 (noauto
```

This example binds the two text files and resolves the external references to produce a program module called FILE1 MODULE on your A-disk. A listing file called FILE1 SYSPRINT is also produced.

Notes:

a. Only one text library is required to resolve external references. Therefore, it can be specified directly on a FILEDEF for SYSLIB and no GLOBAL TXTLIB command is required.

b. The first file name specified on the BIND command is used as a default for determining the names of the output files.

5. **Creating Several Modules Using Control Statements**

In a single bind process using control statements we can initiate binds of arbitrary complexity. Assume a file called FILE1 SYSLIN has been created containing:

```
INCLUDE FILE1
NAME FILE1(R)
INCLUDE FILE1
INCLUDE FILE2
NAME FILE2(R)
SETOPT PARM(CMSAUTO=YES)
INCLUDE FILE3
NAME FILE3(R)
```

Entering the following:

```
global txtlib mylib prodlib
bind file1 (cmsauto no
```

This example binds the various text files and produces three program objects called FILE1 MODULE, FILE2 MODULE and FILE3 MODULE on your A-disk. A listing file called FILE1 SYSPRINT is also produced.

Notes:

a. The arguments of the INCLUDE statements are resolved to the required text files by the binder. If however there were FILEDEFs or PATHDEFs in effect with these names, then the files referred to by the FILEDEFs/PATHDEFs would have been included instead. This behavior is controlled by the option `CMSINCL`; see "CMSINCL" on page 64 for a description of the option.

b. The NAME statement causes a program object to be created with the specified name and a file type of MODULE. A program object module file is produced each time a NAME statement is encountered.

c. External references in FILE1 and FILE2 are resolved from the GLOBAL TXTLIB concatenation, but the SETOPT control statement changed the effective CMSAUTO option for FILE3 so that TEXT files in the standard search order are used before members from the GLOBAL TXTLIB concatenation.

6. **Binding a BFS File from the CMS Ready Prompt**

If your input file resides in the byte file system, you can bind it by specifying the BFS file name in a way recognized by the BIND command:

```
bind 'file1.o' "file2.o" ./file3.o /u/myself/file4.o (output cms
```

If the current working directory is /u/myself, the preceding command binds files file1.o, file2.o, file3.o and file4.o from that directory and produces a program module called FILE1 MODULE on your A-disk.

Notes:

a. This example illustrates the four different ways of indicating to the BIND command that a primary input file name represents a BFS path: enclosing in single quotation marks, enclosing in double quotation marks, beginning the file name with a "dot" (typically the start of a "dot-slash" combination to indicate the current working directory), and beginning the file name with a "slash" to indicate an absolute path name. File names enclosed in quotation marks may be either absolute or relative paths.

b. The output file name is FILE1 because the first primary input file was a BFS file with a .o extension. See 3 on page 28 for more information.

c. If the output cms option was not specified, the program module created is saved as file1 in the current working directory.

Examples of Using the BIND Command from the OPENVM Shell

The BIND command itself functions in exactly the same way from the shell as from the CMS ready prompt. The main usage differences relate to the way in which the shell treats certain characters, and the use of BFS files.

Note: It may be useful to have both single and double quotation marks available to enclose strings when working in the shell. The double quotation mark is often set by default as the CMS terminal escape character (use cms query terminal from the shell to check this). You can disable the terminal escape character to make the double quotation mark available by entering: cms terminal escape off from the shell. The following examples all assume the double quotation mark character is available for use.

1. **Binding a Single Text Deck with No External References**

   Assume you have a file called file1.o in your current working directory containing object code produced by a language compiler. Enter the following:

   ```
   cms 'bind "file1.o"'
   ```

   This example binds the text file and produces a program object called file1 in your current working directory. A listing file called file1.i is also produced.

   Notes:

   a. This example illustrates some of the extension naming conventions used by the BIND command, which are compatible with those used by the c89 command. If the input file does not have the .o extension (file1, for example), then a .m extension is added to the program object name (file1.m). See 3 on page 28 for more information.

   b. The single quotation marks around the CMS command argument are not required, but ensure that no shell substitution occurs so that the enclosed string is passed to CMS exactly as you type it. In subsequent examples where options are specified, they also prevent a shell parsing syntax error.

   c. The double quotation marks around the BIND command argument tell the BIND command that the enclosed string is a BFS path. Alternative specifications are ''file1.o'', ./file1.o, or an absolute path name.

2. **Binding a Single Text Deck with External References**

Suppose file2.o is another program with external references resolved by files in a subdirectory of the current working directory called mytxt. Enter the following:

```
cms 'openvm pathdef syslib ./mytxt'
cms 'bind "file2.o" (nocmsauto'
```

This example binds the text file and resolves the external references to produce a program object called file2 in your current working directory.

Notes:

a. It is not possible to concatenate PATHDEFs. If autocall resolution from multiple sources is required then this can be achieved using the autocall statement.

b. An archive library can also be specified as the target of the SYSLIB PATHDEF.

3. **Binding Several Text Decks with One Command**

   Suppose you want to bind file1.o and file2.o (from Fred's home directory) into a single module. Enter the following:

   ```
   cms 'openvm pathdef syslib ./mytxt'
   cms 'bind ./file1.o /u/fred/file2.o (nocmsauto'
   ```

   This example binds the two text files and resolves the external references from the mytxt directory to produce a program object called file1 in your current working directory. A listing file called file1.i is also produced.

   Note: The first file name specified on the BIND command is used to generate a default for determining the names of the output files.

4. **Binding Several Text Decks Using Control Statements**

   Another way to effect the results of the previous example is to use a file of control statements to control the bind process. Assume a file called file1.syslin has been created containing:

   ```
   setopt parm(nocmsauto)
   include './file1.o'
   include './file2.o'
   name file1(r)
   ```

   Enter the following:

   ```
   cms 'openvm pathdef syslib ./mytxt'
   cms 'bind ./file1.syslin'
   ```

   This example binds the two text files and resolves the external references from the mytxt directory to produce a program object called file1 in your current working directory. A listing file called file1.syslin.i is also produced.

   Notes:

   a. file1.syslin must be in simulated card image format. An appropriate shell command to create such a file is :

      ```
      cms 'x file1.syslin (nametype bfs bfsline 80'
      ```

   b. The arguments of the include statements are resolved to the required text files by the binder.

   c. The name statement causes a program object to be created with the specified name. The replace option is redundant here because BFS files are always replaced.

d. Note that the listing file produced gets a "double-extension" because the input file did not use the **.o** extension convention.

5. **Binding a CMS File from the Shell**

   If your input file resides on a minidisk or in the shared file system, you can bind it by specifying the file name in a way recognized by the BIND command as representing a CMS file:

   ```
   cms 'bind file1 (output bfs'
   ```

   The preceding command would bind FILE1 TEXT and produce a program object called `file1.m` in your current working directory.

   Note: If the `output bfs` option is not specified, the program object created is saved as FILE1 MODULE on your A-disk.

Messages and Return Codes

To display information on a specific error message enter HELP MSG and the message identifier; for example:

```
HELP MSG DMS1616W
```

DMS002E **[Input|Overlay] {File[(s)]|Dataset|Note} [***fn* **[***ft* **[***fm***|***dirname***]]] not found[:** *pathname***]**

DMS006E **No read/write {disk|filemode|***filemode* **filemode } accessed [for** *fn ft***]**

DMS108S **More than** *nn* **libraries specified**

DMS179E **Missing or invalid MACS card in control file** *fn ft fm*

DMS183E **Invalid {CONTROL|AUX} file control card**

DMS234E **Error in LOAD LIST file** *fn ft fm*

DMS252E **Invalid {filename** *fn***|file ID|directory id}**

DMS389E **Invalid operandtype:** *operand*

DMS1600I *diagnostic information*

DMS1604E **Error accessing primary input file** *filename* **[***filetype filemode***] (***service* **Return Code** *rtncode* **[Reason Code** *rsncode***])**

DMS1605E **Primary input path refers to a directory** *path*

DMS1606E **No primary input specified**

DMS1608E **Incorrect** *dddef* **for** *ddname* **(***reason***)**

DMS1609E **Primary input file** *fileid* **not valid: fixed format record length not 80**

DMS1610I **Primary input file found** *filename* **[***filetype filemode***]**

DMS1611W **Unable to extend the default module file name**

DMS1612W **Default file name reset from** *oldname* **to** *newname*

DMS1613W **Short record found in BFS file** *filename*

DMS1614I **Temporary name generated for save process TEMPNAM***n*

DMS1615W **Unable to generate temporary name for save process**

DMS1616W **LIBE option was specified but no GLOBAL TXTLIB is defined**

DMS1617W **LIBE option was specified but SYSLIB is already defined as a PATHDEF**

DMS1618W **LIBE option was specified but the SYSLIB FILEDEF that exists is not for file type TXTLIB**

DMS1619W **LIBE option was specified, but the SYSLIB FILEDEF that exists was defined without the CONCAT option**

DMS1621W **Binder function** *function* **completed with return code** *retcode* **and reason code** *rsncode*

DMS1681E **Unable to load user MESSAGE exit** *name* **CC/RC(***cc/rc***)**

DMS1699E **Unexpected error in** *module* **(***symptom***[,***symptom2***]) [Return Code** *rtncode* **[Reason Code** *rsncode***]]**

DMS1900I **All TEMPNAMes have been used. The module cannot be saved**

DMS1901I **No module name was specified. Module was saved using TEMPNAM***n*

DMS1902E **Symbol** *symbol* **has been truncated at the first embedded blank**

DMS1903E **Expected control statement continuation was not found**

DMS1904E **Unmatched quote in current control statement stream**

DMS1905S **Duplicate module** *module* **found**

DMS2141E **Missing quote or quote specification is not valid**

DMS2513E **Extended plist is required.**

Chapter 3. Binder Control Statements

Control Statements Syntax

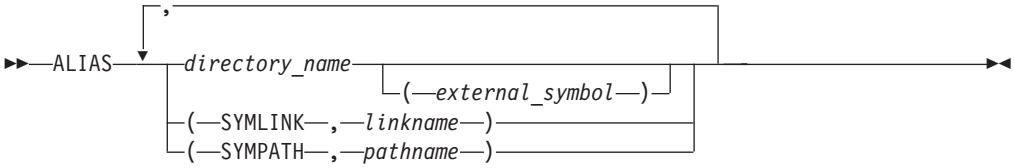
Binder control statements are 80 byte records. Nothing should be written preceding the operation, which must begin in or after column 2. Placing an asterisk (*) in column 1 of a control statement causes the binder to treat that line as a comment. Each binder control statement specifies an operation and one or more operands which may be written up to column 71 or continued on subsequent records. Binder control statements are placed before, between or after object modules. For more details see "7.1 Binder Syntax Conventions" in *z/OS MVS Program Management: User's Guide and Reference*.

Control Statements Summary

For each control statement that is relevant to the CMS environment, this section gives a brief description and then documents any differences in the processing of the control statement between CMS Binder and z/OS MVS Program Management Binder.

Note: To obtain complete descriptions for control statements, this section should be read in conjunction with "Chapter 7. Binder Control Statement Reference" in *z/OS MVS Program Management: User's Guide and Reference*.

ALIAS



The ALIAS statement specifies one or more additional names for the primary entry point, and can also specify names of alternate entry points. Symbolic links to program objects in the BFS can also be created by using a combination of SYMLINK and SYMPATH parameters. These entries can be repeated in any order, and alias entries can be divided up among separate ALIAS statements as desired except that there must be at least one SYMPATH specification following a given SYMLINK or group of SYMLINKs. A SYMPATH specification applies to all SYMLINK specifications that precede it, back to the first previous SYMPATH.

directory_name
specifies an alternate name for the program module.

externalsymbol
specifies the name of the entry point to be used when the program is executed using the associated alias.

linkname
is a path that, when concatenated to the SYSLMOD path, provides the symbolic link path name.

© Copyright IBM Corp. 2001, 2013

pathname
is a path used as the contents of the symbolic link.

Notes:

1. Aliases are supported only when the program module is written to either the BFS or a LOADLIB.
2. Alternate entry points are supported only when the program module is written to a LOADLIB.
3. Refer to "7.2 ALIAS Statement" in *z/OS MVS Program Management: User's Guide and Reference* for more information and examples regarding symbolic links.

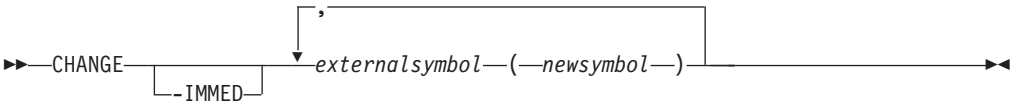
AUTOCALL

The AUTOCALL control statement prompts the binder to perform incremental (or immediate) autocall using only the given library as the search library to resolve symbol references.

library
specifies either the name of a DD statement that describes a text library or the path for an OpenExtensions file or archive library file.

Note: If the CMSINCL option is in effect when the AUTOCALL statement is processed, then a file name *filename* may be substituted for *library* and then the binder attempts to use the first text library *filename* **TXTLIB** found in the standard search order.

CHANGE



The CHANGE statement causes an external symbol to be replaced by the symbol in parentheses following the external symbol..

-IMMED
causes CHANGE to operate against the modules that have already been included in the module being built rather than against the next input module.

externalsymbol
specifies a control section name, a common area name, an entry name, an external reference or a pseudoregister name that is to be changed.

newsymbol
is the name to which the external symbol is to be changed.

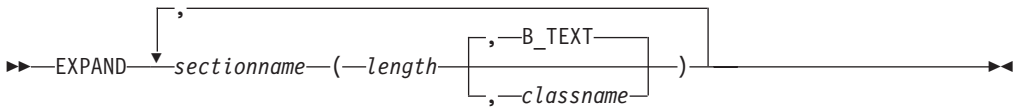
For more information, refer to the description of the CHANGE control statement in *z/OS MVS Program Management: User's Guide and Reference*.

ENTRY

The ENTRY statement specifies the symbolic name of the instruction to be the main entry point for the program module.

externalsymbol
specifies a control section name or an entry name.

EXPAND



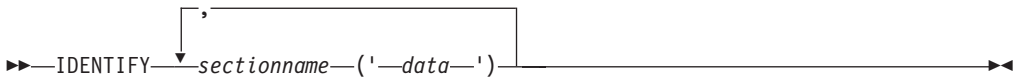
The EXPAND statement lengthens control sections or named common areas by a specified number of bytes.

sectionname
symbolic name of a common area or control section with a length that is to be increased.

length
the decimal number of bytes to be added.

classname
the name of the text class to be expanded.

IDENTIFY

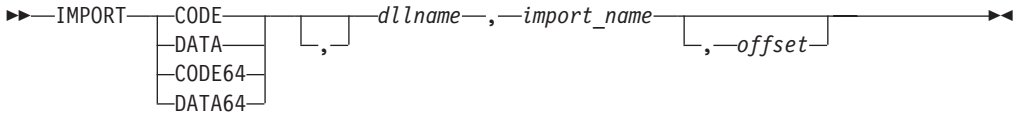


The IDENTIFY statement specifies that any data you supply should be entered into the CSECT identification records (IDR) for a particular control section.

sectionname
is the symbolic name of the control section to be identified.

data
specifies up to 80 EBCDIC characters of identifying information for program objects, and up to 40 characters for load modules.

IMPORT



The IMPORT statement specifies an external symbol name to be imported and the name of the DLL module where it can be found. An imported symbol is one that is expected to be dynamically resolved.

CODE
the *import_name* must represent the name of a code section or entry point.

DATA
the *import_name* must represent the name of a variable or data type definition to be imported.

CODE64
equivalent to **CODE** but specified when using 64-bit addressing mode.

DATA64
equivalent to **DATA** but specified when using 64-bit addressing mode.

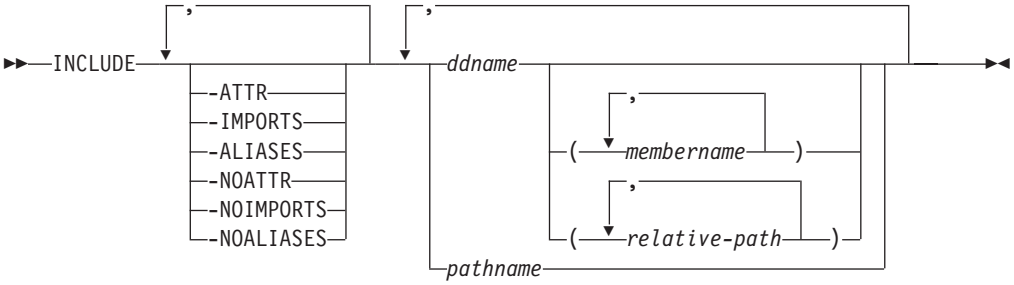
dllname
the name of the DLL module that contains the import_name to be imported. If it is an OpenExtensions file, the file name is limited to 255 bytes. Otherwise, the limit is eight bytes.

import_name
represents a function or method definition, or a variable or data type definition. The *import_name* can be up to 32767 bytes in length.

offset
consists of up to 8 hexadecimal characters. The offset is stored with the DLL information for an imported function. This is primarily for the use of LE (Language Environment®).

Note: CODE64 and DATA64, although supported by the binder, are not currently supported by the CMS loader.

INCLUDE



The INCLUDE statement specifies sequential data sets, library members, extended format modules, or OpenExtensions files that are to be sources of additional input for the binder.

Note: If options that contradict one another are specified, the last valid option specified will be used. For example, if both -ATTR and -NOATTR are specified in that order, the binder will honor the -NOATTR option.

-ATTR
specifies that module attributes should be copied from the input module and be applied to the module being built by the binder.

-IMPORTS
specifies that dynamic resolution information (if any) will be copied from the input module. Starting in z/VM 5.2, this is the default.

-ALIASES
specifies that the aliases of the input module be copied in and used as aliases for the output module.

-NOATTR
specifies that module attributes will not be copied from the input module.

-NOIMPORTS
specifies that dynamic resolution information (if any) will not be copied from the input module.

-NOALIASES
specifies that the aliases of the input will not be copied from the input module.

ddname
the name of a data definition that defines a sequential data set, a partitioned data set, an extended format module, or an OpenExtensions file to be used as additional input to the binder.

For a partitioned data set, at least one member name must also be specified. If only a single member is to be included, its member name can be specified in the FILEDEF rather than on the control statement. When the source is an OpenExtensions file, the PATHDEF command must contain the full or partial path name of the file to be included. If a partial path name is provided, it must be completed using a *relative-path* expression following the *ddname*.

membername
the name of, or an alias for, a member of the library defined by the *ddname*.

pathname
the absolute or relative path name of an OpenExtensions file which can be up to 255 bytes.

relative-path
If the referenced *ddname* specifies a path, then *relative-path* will be appended to that path name.

Notes:

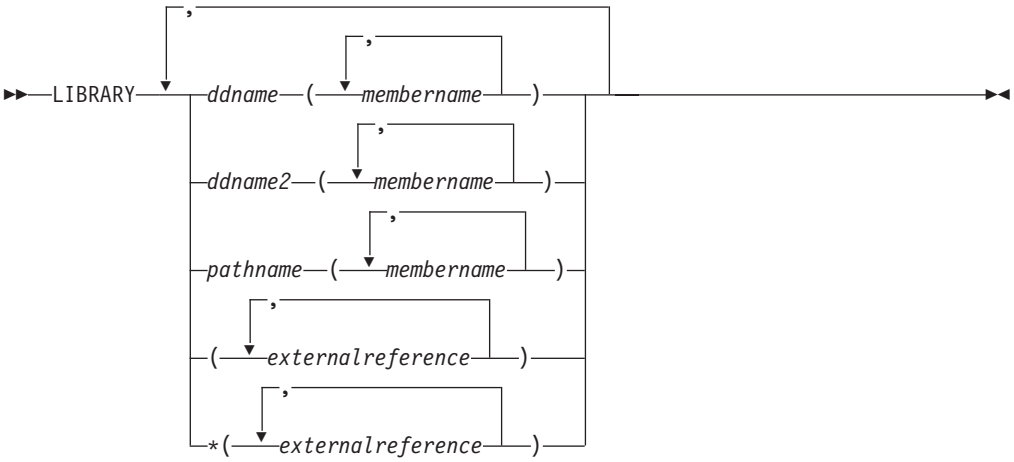
1. The path name may be enclosed in single quotation marks, but it **must** start with a / in the case of an absolute path name, and either a ./ or a ../ in the case of a relative path name.
2. If the CMSINCL option is in effect when the INCLUDE statement is processed, then file name *filename* may be substituted for any *ddname* and the binder attempts to use the first TEXT file, extended MODULE or TXTLIB library found in the standard search order, whichever is appropriate. So, if *membername* is specified, the binder tries to include member *membername* from text library *filename* **TXTLIB**. If no *membername* is specified, the binder attempts to include *filename* **TEXT** first; if none is found, then the binder includes *filename* MODULE, but only if it is an extended format module.
3. A partitioned data set may be one of the following:
   - A PDS (not PDSE) on an attached MVS disk
   - A LOADLIB, which is created by either the CMS binder or the LKED command
   - A TXTLIB, which is created by the TXTLIB command

- A C/370™ CMS text library, which is created using the C370LIB command. C/370 CMS text libraries have a file type of TXTLIB and also have the same structure as TXTLIBs with the addition of a C370LIB-directory member(s) @@DC370$ and/or @@DC390$.

INSERT

The INSERT statement is not applicable in the CMS environment. Overlays are not supported in CMS and the INSERT repositions a section in an overlay structure.

LIBRARY



The LIBRARY statement can be used to specify:

- Additional automatic call libraries that contain modules used to resolve external references found in the program.
- Restricted no-call: External references that are not to be resolved by an automatic library call during the current binder job step.
- Never-call: External references that are not to be resolved by an automatic library call during this or any subsequent binder job step.

When LIBRARY statements identify additional libraries that can be used, the following search order is applied during final autocall:

1. The library or libraries associated with the first LIBRARY specification are searched. This may identify an OpenExtensions directory, an OpenExtensions archive, a partitioned data set, or a concatenation of partitioned data sets.
   - For an OpenExtensions directory, the file names and links in the directory are checked.
   - For an OpenExtensions archive or C370LIB PDS, all names that have been cataloged by the **ar** command or Object Library Utility are checked.
   - For other partitioned data sets, only the member names and aliases are checked.
   - If specific names are listed in the LIBRARY specification, only those names can be used for resolution; otherwise any name can be used.
2. Libraries associated with other LIBRARY specifications are searched in the order the specifications were provided within a LIBRARY statement and the order in which the LIBRARY statements were provided.
3. The SYSLIB concatenation is searched.

4. If unresolved symbols remain, the search is restarted from step 1 on page 50. It is repeated until all symbols are resolved in a complete pass through all libraries.

ddname
The name of a DD statement that defines a library from which the listed symbols will be included during automatic library call.

membername
Usually, the name of or an alias for a member of the specified library. If the *ddname* points to an OpenExtensions archive, the names in parentheses can be any external symbols indexed by the **ar** command. If the *ddname* points to a C370LIB, the names in parentheses can be any external symbols defined by the special C370LIB directory. Conversely, if member names are used for a C370LIB, the binder looks at the members only if there are unresolved symbols whose name matches the member name.

Here is an example: A C370LIB (*ddname* MYC3LIB) contains a member named FOO within which there is an external entry FooSez, and that FooSez is in the special C370LIB directory. Also, a program has an unresolved symbol FooSez.
- If the LIBRARY statement says MYC3LIB(F00), the symbol is not found.
- If it says MyC3LIB('FooSez'), it is resolved.
- If it says MYC3LIB(FOO) and the program also contains unresolved symbol FOO, both are resolved.
- Only those members specified are used to resolve references.

ddname2
The name of a DD statement that defines a library that may be used to resolve references during automatic library call. The DD statement can point to a PDS, PDSE, PDS/PDSE concatenation, OpenExtensions directory, or OpenExtensions archive library.

pathname
The name of a OpenExtensions archive library or directory that may be used to resolve references during automatic library cal. For a directory, the binder looks for files or links whose name matches the symbol to be resolved.

(*externalreference***)**
An external reference that can be unresolved after primary input processing. The external reference is not to be resolved by automatic library call.

***** Indicates never-call; the external references should never be resolved from an automatic call library. If the * (asterisk) is missing, the reference is left unresolved during the current binder job step but can be resolved in a subsequent step.

If all binder input modules containing references to a specific symbol were bound with never-call, that symbol is not resolved by automatic library call during this binder run. However, if one or more input modules do not indicate a symbol as never-call, the binder attempts to resolve the symbol from the automatic call library.

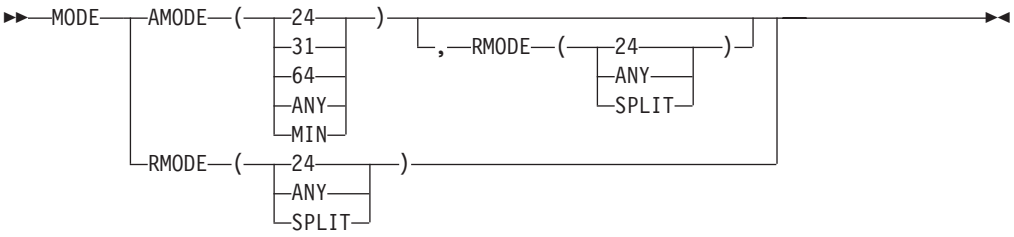
Placement: A LIBRARY statement can be placed before, between, or after object modules or other control statements.

Notes:

1. A member or external reference listed in a LIBRARY statement has no affect except when a matching name appears as an unresolved reference in the program.

2. For C370LIB or archives, the name may be any symbol listed in the archive or special C370LIB directory.
3. For a non-C370LIB PDS or PDSE, the name must be a member name or alias to be effective.
4. For an OpenExtensions directory, the name must be a file name or alias to be effective.
5. If the **NCAL** option is specified, the LIBRARY statement has no effect.
6. Members included by automatic library call are placed in the root segment of an overlay program, unless they are repositioned with an INSERT statement.
7. The LIBRARY control statement is not processed immediately. If the same symbol appears on more than one LIBRARY statement, only the last occurrence is used.
8. Specifying an external reference for restricted no-call or never-call by means of the LIBRARY statement prevents the external reference from being resolved by automatic inclusion of the necessary module from an automatic call library; it does not prevent the external reference from being resolved if the module necessary to resolve the reference is specifically included or is included as part of an input module.
9. The LIBRARY statement is not allowed in a data set that is included from an automatic call library.

MODE



The MODE statement can specify the addressing mode for all the entry points into the program module and the residence mode for the program module.

AMODE(24)
indicates that 24-bit addressing must be in effect.

AMODE(31)
indicates that 31-bit addressing must be in effect.

AMODE(64)
indicates that 64-bit addressing can be in effect. AMODE64 is not currently supported by the CMS loader.

AMODE(ANY)
indicates that either 24-bit or 31-bit addressing may be in effect.

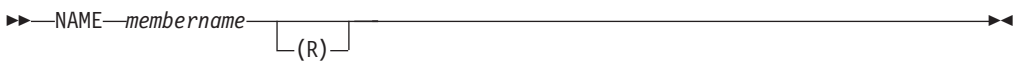
AMODE(MIN)
causes the most restrictive AMODE of all control sections within the program module to be assigned.

RMODE(24)
indicates that the module must reside below the 16 MB virtual storage line.

RMODE(ANY)
indicates that the module may reside anywhere in virtual storage.

RMODE(SPLIT)
indicates that the module is split into two class segments, one to be loaded below 16 MB and one to be loaded above the 16 MB virtual storage line. Split residency mode is not currently supported by the CMS loader.

NAME



The NAME statement specifies the name of the program module created from the preceding input modules.

membername
the name to be assigned to the program module.

(R)
indicates that this program module can replace an existing identically named module.

ORDER

The ORDER statement indicates the sequence in which control sections or named common areas appear in the program module.

sectionname
is the name of the section to be sequenced.

(P)
indicates that the starting address of the control section or named common area is to be on a page boundary. If the ALIGN2 option is in effect, sections are aligned on 2 KB boundaries.

OVERLAY

The OVERLAY statement is not applicable because overlays are not supported in CMS.

PAGE

The PAGE statement aligns a control section or named common area on a 4 KB page boundary in the program module.

sectionname
is the name of the section to be aligned on a page boundary. If the ALIGN2 option is in effect, sections are aligned on 2 KB boundaries.

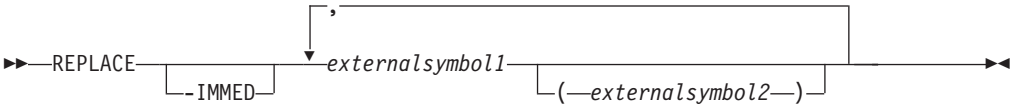
RENAME

The RENAME statement allows for the renaming of specific symbols. The rename requests take place only after the binder attempts to resolve the original names. The new names are then used during the binder's final autocall in order to resolve any references previously unresolved.

oldname
is the symbol to be renamed.

newname
is the symbol name to which the *oldname* should be changed.

REPLACE



The REPLACE statement is used to replace or delete external symbols.

-IMMED
causes REPLACE to operate against the modules that have already been included in the module being built rather than against the next input module.

externalsymbol1
specifies a control section, a common area, an entry point name, an external reference, or a pseudoregister name that is to be either replaced or deleted.

externalsymbol2
is the section, common area, or name with which *externalsymbol1* is to be replaced. If no *externalsymbol2* is specified, then the *externalsymbol1* is deleted.

For more information, refer to the description of the REPLACE control statement in *z/OS MVS Program Management: User's Guide and Reference*.

SETCODE

The SETCODE statement is not applicable in the CMS environment. A SETCODE assigns an authorization code to the program module and the authorization code is only used by the MVS Authorized Program Facility.

SETOPT

The SETOPT statement allows you to set options at the module level rather than the BIND command or dialog level. The options you specify are valid only until after the next NAME control statement is processed

parm
a string of parameter specifications entered using the MVS binder style syntax rather than the CMS style syntax.

Note: The following list of options cannot be specified or are ineffective on a SETOPT statement:

- CALLERID
- COMPAT
- EXITS
- FILETYPE
- LIBE
- LINECT
- LIST
- MSGLEVEL
- OUTPUT
- PRINT
- REFR
- RENT
- REUS (Only the old form)
- SIZE
- TERM
- TRAP
- WKSPACE

SETSSI

The SETSSI statement is not applicable to the CMS environment.

Chapter 4. The CMS Binder

Binder Input and Output

The following diagram shows the standard input and output processes.

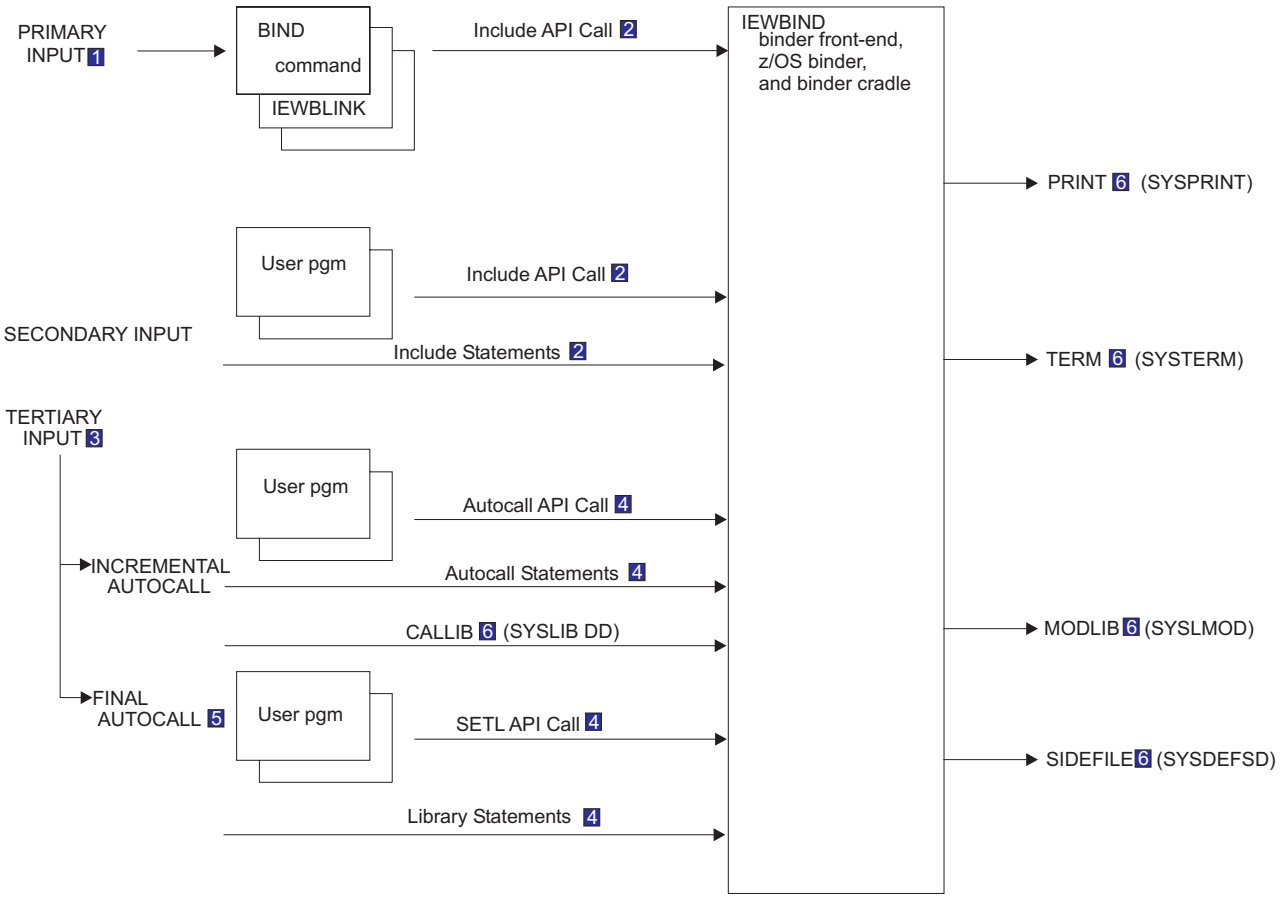


Figure 3. The VM Binders Input and Output

Notes on the diagram:

1. SYSLIN DD, or for the BIND command, it may instead be files specified in the command. This is the only input that can validly introduce a name statement.
2. Include can be specified with ddnames or path names.
3. Resolving external references.
4. These inputs may be libraries (TXTLIB, C370LIB, or LOADLIB), BFS archive files, or BFS directories and can be specified with a ddname or a path name.
5. Final automatic library call is done during the bind processing as part of either a BINDW API call or a SAVEW for an unbound object.
6. These input and output files have the BIND command's default ddname in parentheses.

© Copyright IBM Corp. 2001, 2013

FILEDEF/PATHDEF - Relationship with DD Statement

The CMS Binder's MVS heritage means that ddnames are used extensively to determine the target of I/O activity. The concepts of JCL and allocation are foreign to CMS, but OS Simulation provides the FILEDEF command to establish data definitions for OS ddnames. See *z/VM: CMS Commands and Utilities Reference* for details of syntax and usage.

The byte file system introduced with OpenExtensions can be accessed indirectly with a ddname. The OPENVM PATHDEF CREATE command can be used to establish a path definition for a ddname. See *z/VM: OpenExtensions Commands Reference* for details of syntax and usage.

Restrictions

The FILEDEF command allows the specification of block size. In the case of input, this option is redundant, and if block size is specified as anything other than 0 or the record length, the library cannot be opened.

FILEDEFs for an output LOADLIB have some restrictions on the options: The BLOCK value is ignored if the LOADLIB exists and the RECFM and LRECL values are always ignored.

It is possible to issue both a FILEDEF and a PATHDEF for the same ddname. However, this may lead to unpredictable results (usually the binder uses the path definitions).

Files

There are 14 types of input or output files that can be distinguished by how they are specified to the binder and how the binder uses them. For example, the PRINT file is related to a ddname for the duration of each invocation of the binder and is used by the binder as a listing data set for messages produced by the LIST, MAP and XREF options. Of the 14 files, three are purely for diagnostic output and they are discussed further in section "Diagnostic Information" on page 81. The binder uses the remaining files as follows:

PRIMARY
: Sequential data set, library member, or OpenExtensions file that is a source of primary input for the binder. Not used by the binder API interface.

OPTIONS
: Sequential data set, library member, or OpenExtensions file that contains binder options.

INCLUDE
: Sequential data set, library, or OpenExtensions file that is a source of secondary input for the binder.

AUTOCALL
: Library, OpenExtensions directory, or OpenExtensions archive searched to resolve symbol references during an incremental automatic library call.

LIBRARY
: OpenExtensions directory or OpenExtensions archive Library from which specific symbols are included during final automatic library call.

SETL Library or OpenExtensions file from which specific symbols are included during final automatic library call.

CALLIB
Library, OpenExtensions directory or OpenExtensions archive searched to resolve symbol references during final automatic library call.

PRINT
Sequential data set or OpenExtensions file that is the target for messages and listing produced by the LIST, MAP and XREF options.

TERM
Sequential data set or OpenExtensions file that is the target for messages issued during binder processing.

SIDEFILE
Sequential data set, library, OpenExtensions file, or OpenExtensions directory that is the target for the side file of a DLL module.

MODLIB
Sequential data set, library, OpenExtensions file, or OpenExtensions directory that is the target for the produced executable.

Byte file system files and directories may be identified to the binder in one of two ways:

- A PATHDEF relating the path to a ddname.
- Some APIs and control statements allow either a ddname or path name to be specified. They recognize it as a path name if it starts with a **/**, **./** or **../**. In the case of control statements, the path name may be enclosed in quotation marks.

The following table summarizes the manner in which each file may be specified to the binder.

Table 6. File Specification

FILE	Default ddname[1]	STARTD Filelist	Option	API call	Control Statement
PRIMARY	SYSLIN				
OPTIONS			X		
INCLUDE				X	X
AUTOCALL				X	X
LIBRARY					X
SETL				X	
CALLIB	SYSLIB	X	X	X [2]	
PRINT	SYSPRINT	X			
TERM	SYSTERM	X			
MODLIB	SYSLMOD	X	X	X [3]	
SIDEFILE	SYSDEFSD	X [4]			

Notes:

1. If the binder is invoked using the API, then these files have no default ddnames.
2. CALLIB may be overridden on the BINDW API call.
3. MODLIB may be overridden on the SAVEW API call.
4. SIDEFILE is the only STARTD FILELIST entry that may be a path name.
5. In addition to the files in the table, there are three diagnostic output files.

The following table summarizes the I/O types supported for each file.

Table 7. File Types

FILE	Physical Sequential	LOADLIB	TXTLIB or C370LIB	BFS File	BFS Directory	Archive
PRIMARY	X		X [1]	X		
OPTIONS	X			X		
INCLUDE	X	X	X	X		
AUTOCALL		X	X		X	X
LIBRARY		X	X			X
SETL		X	X	X		
CALLIB		X	X		X	X
PRINT	X			X [2]		
TERM	X			X [2]		
MODLIB	X	X		X	X	
SIDEFILE	X			X	X	

Notes:

1. Primary input from a TXTLIB is supported by appending the GLOBAL TXTLIB concatenation to the standard search order when the LIBE option is in effect.
2. These are only supported by relating a path name to the appropriate ddname with the OPENVM PATHDEF CREATE command.

Autocall with Archive Libraries

The binder also supports autocall from OpenExtensions archive libraries. These archive libraries may contain members that are object files – in OBJ, XOBJ and GOFF format and with special directory information similar to that contained in C370LIB object libraries. They may also contain members which are side files (of IMPORT control statements) or other files of control statements.

Archive libraries are created by the OpenExtensions **ar** command. Like C370LIBs, they may contain attributes used by the binder to select among variant routines with matching names. Unlike C370LIBs, archives cannot be concatenated.

Note: Archive libraries cannot be used as the target for INCLUDE statements.

While the **ar** command is typically used to create archive libraries of object files, it can also be used to create archive libraries of non-object files, or archive libraries containing a combination of object files and non-object files. In addition to processing archive library object file members during autocall, the binder can also process certain non-object file archive library members. Those members must have the following characteristics:

- Members that are side files (containing IMPORT control statements). To be recognized, an IMPORT statement must be the very first statement in the file, in the format produced by the binder when it writes to SYSDEFSD.
- Members that are files specifically identified as containing binder control statements. To be recognized, the first statement must contain the string *! in the first two columns, followed by the string IEWBIND INCLUDE. These two strings may be separated by blanks, but must be contained in a single statement.

For the binder to process these non-object files, one such file must be positioned as the very first member of the archive library (excluding the symbol table member, _.SYMDEF). The binder then processes that first member as if it had been explicitly included as binder input, and then includes any other such members that it can recognize in that archive library. The following additional points should be noted:

- This processing is performed only during autocall processing of an archive library and only when there are still unresolved symbols.
- If the archive library also contains members that are object files, it is still processed to attempt to resolve symbols using those object file members. If the archive library contains neither object file members nor non-object file members with the characteristics described here, the binder reports an error when attempting to process that archive library.
- As is the case for object files, these non-object files must be composed of statements that are exactly 80 bytes long, with no newline terminator.
- Processing of non-object files during autocall does not change the binder precedence for resolving symbols. Just as when a side file is explicitly included, the IMPORT information will only be used to resolve a symbol dynamically if it is still unresolved after all static resolution is complete.

See *z/VM: OpenExtensions Commands Reference* for more information about using the **ar** utility to create archive libraries and how to position members within them.

Executable Formats

The following executable formats may be produced using the CMS Binder:

OS Load Module
: A member of a CMS LOADLIB, which can be executed using the OSRUN command

Module
: A standard CMS module that may be produced by the GENMOD command

Extended Module
: A CMS module file that contains an MVS program object

Program Object
: A standard program object identical to one that the MVS binder might write to the hierarchical file system can be written by the CMS binder to the byte file system.

All of the above objects except the standard CMS modules may be re-edited by the CMS binder.

Program Object Formats

There are currently eight program object formats defined: PM1, PM2, PM3, PM4, PM4SUB2, PM4SUB3, PM5, and PM5SUB2. PM1 format program objects cannot be created by the CMS binder, although the CMS loader is able to load all formats. If a program object is produced by the binder, then its format is controlled by the `COMPAT` option and the program management features utilized in the program object. See *z/OS MVS Program Management: User's Guide and Reference* "6.3.7 COMPAT: Binder Level Option" for details.

The following table summarizes how the executable format is determined based on the COMPAT option and the value of MODLIB[1].

Table 8. Module Output Formats

COMPAT Option	MODLIB[1]		
	File Type of LOADLIB	File Type of MODULE	OpenExtensions Path
LKED | PM1	LOADMOD[2]	MODULE[3]	MODULE[3]
PM2	LOADMOD[2]	PM2[4]	PM2[5]
PM3	LOADMOD[2]	PM3[4]	PM3[5]
PM4 | ZOSV1R3 | ZOSV1R4	LOADMOD[2]	PM4[4]	PM4[5]
ZOSV1R5 | ZOSV1R6	LOADMOD[2]	PM4SUB2[4]	PM4SUB2[5]
ZOSV1R7	LOADMOD[2]	PM4SUB3[4]	PM4SUB3[5]
PM5 | ZOSV1R8 | ZOSV1R9	LOADMOD[2]	PM5[4]	PM5[5]
ZOSV1R10 | ZOSV1R11 | ZOSV1R12	LOADMOD[2]	PM5SUB2[4]	PM5SUB2[5]

Notes:

1. The CMS BIND command uses a default ddname of SYSLMOD for MODLIB.
2. An OS loadmod with the same format as that created by LKED
3. A conventional format CMS module
4. A CMS extended module with an embedded program object
5. Program object written as an OpenExtensions file

API Considerations

In general, the application programming interface to the CMS binder is the same as for the program management binder as documented in *z/OS MVS Program Management: Advanced Facilities*. All extensions, restrictions, and other differences are documented in the rest of this section.

Version Number

The default version number generated by the IEWBIND and IEWBUFF macros is VERSION=1. It is recommended that you always explicitly code VERSION=6 if you use these macros, and if not using the macros, set the parameter list version field to 6.

Setting Options With the Binder API

Options may be set in a variety of ways while using the API:

- As part of a parameter string in the installation defaults module IEWBODEF.
- As an option keyword or value pair entry in the option list on the STARTD API call
- As an option keyword or value pair on the SETO API call
- As part of a parameter string on either the STARTD or SETO API calls
- As part of a parameter string read in from an options file
- As part of a parameter string on the SETOPT control statement
- As part of a parameter string passed as the value of the IEWBIND_OPTIONS environment variable when the ENVARS= parameter is used on the STARTD.

So with one or two exceptions, any option can be specified either as part of a parameter string (see *z/OS MVS Program Management: User's Guide and Reference* "Chapter 6. Binder Options Reference" for the syntax of this form and a description) or as a keyword or value pair (see Table 35 in *z/OS MVS Program Management: Advanced Facilities* "5.1 Setting Options With the Binder API").

In addition to the binder options specified in the *z/OS MVS Program Management: User's Guide and Reference*, the CMS binder provides seven CMS-specific options; two that affect the binder processing and five that provide CMS-specific attributes available on the GENMOD command. These attributes are saved into modules, extended modules, and program objects, but not into load modules in load libraries.

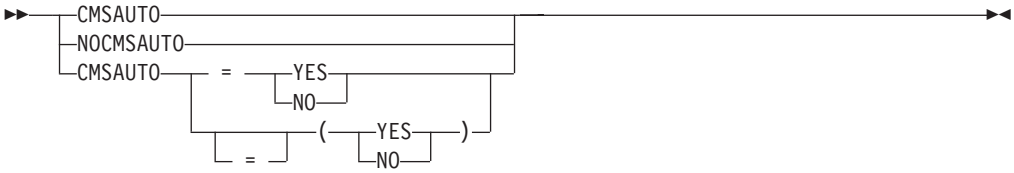
There are restrictions placed on the mixing of Program Management binder options with CMS-specific binder options:

- Installation defaults must be specified as two variable length parameter strings in module IEWBODEF; the first specifies the Program Management binder options and the second specifies the CMS-specific binder options. See section "Defining Installation Defaults" on page 88.
- An options file can specify both Program Management binder options and CMS-specific binder options, but not in the same record.
- Any SETOPT control statement can specify Program Management binder options or CMS-specific binder options, but not both in the same statement.

Each of the CMS-specific binder options is described below along with the syntax of its parameter string form. See Table 9 on page 66 for the keyword/variable pair forms (this table also shows the product defaults).

CMSAUTO

The CMSAUTO option is analogous to the AUTO option of the LOAD and INCLUDE CMS commands.



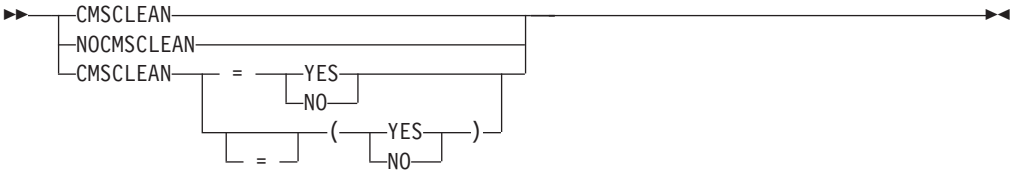
When **CMSAUTO** is in effect, object decks with file type of TEXT are used to resolve external references during **final** automatic library call before searching the default call library.

Notes:

1. TEXT files are not searched if a CALLIB ddname (usually SYSLIB) is specified that points to an OpenExtensions directory or archive.
2. TEXT files are not searched to resolve external references specified on SETL API calls and LIBRARY control statements.

CMSCLEAN

The CMSCLEAN option has the same effect as the CLEAN attribute option of the GENMOD CMS command.

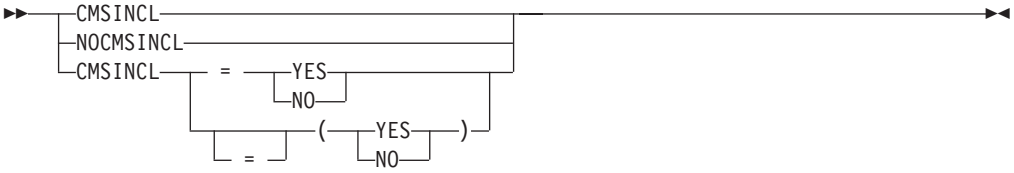


CMSCLEAN
indicates that the module is to be removed from storage at the end of its execution.

NOCMSCLEAN
indicates that the module is to remain in storage until end-of-command (Ready;).

CMSINCL

The CMSINCL option controls the use of TEXT, extended format MODULE and TXTLIB files for input when no FILEDEF or PATHDEF matches the ddname.

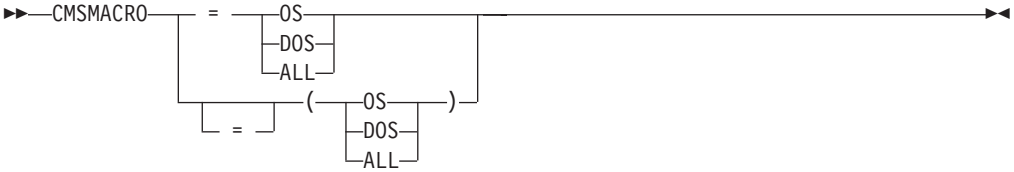


When **CMSINCL** is in effect, a file name can be substituted for a ddname that is used for input where the file in question has a file type of either TEXT, MODULE, or TXTLIB. This means that the user does not need to issue FILEDEFs for input TEXT files, extended format MODULEs and TXTLIB libraries that are introduced to the binder as a result of the following:

- INCLUDE control statements
- INCLUDE API calls
- AUTOCALL control statements
- AUTOCALL API calls
- LIBRARY control statements
- SETL API calls
- CALLIB file specification

CMSMACRO

The CMSMACRO option has the same effect as the OS, DOS or ALL attribute options of the GENMOD CMS command.



OS indicates that the program may contain OS macros, and therefore, should be executed only when CMS/DOS is not active.

DOS
indicates that the program contains VSE macros; CMS/DOS must be active (SET DOS ON must have been previously invoked) for this program to execute.

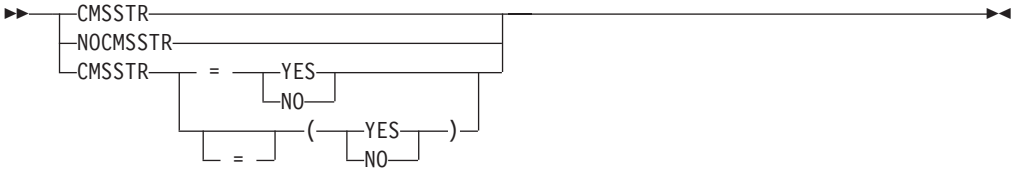
ALL
indicates that the program:

- Contains CMS macros and must be capable of running regardless of whether CMS/DOS is active.
- Contains no VSE or OS macros.
- Preserves and resets the DOS flag in the CMS nucleus.
- Does its own setting of the DOS flags.

The ALL option is primarily for use by CMS system programmers. CMS system routines are aware of which environment is active and preserve and reset the DOS flag in the CMS nucleus.

CMSSTR

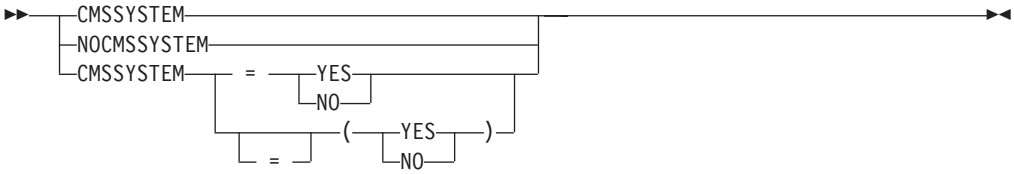
The CMSSTR option has the same effect as the STR attribute option of the GENMOD CMS command.



When a program bound with **CMSSTR** in effect is loaded by a LOADMOD command with the NOPRES option, the system deletes previously loaded non-OS programs.

CMSSYSTEM

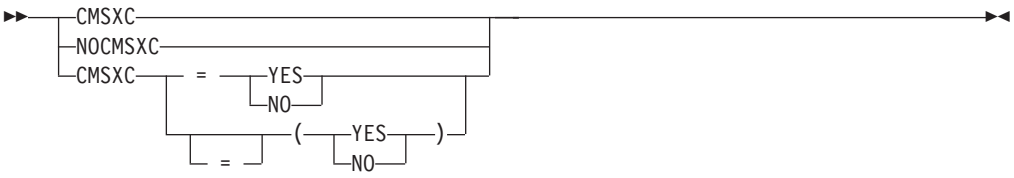
The CMSSYSTEM option has the same effect as the SYSTEM attribute option of the GENMOD CMS command.



CMSSYSTEM
indicates that when the MODULE is loaded, it is to have a storage protect key of zero.

CMSXC

The CMSXC option has the same effect as the XC attribute options of the GENMOD CMS command.



CMSXC
specifies that this module can execute only in XC virtual machines.

NOCMSXC
indicates that this module can execute in XA and XC virtual machines.

Keyword/Variable Form

Table 9. Setting CMS Options With the Binder API

Option Keyword	Description	Allowable Values	Product Default Value
CMSAUTO	Object decks with a file type of TEXT are used to resolve external references.	YES, NO	YES
CMSCLEAN	The module is to be cleaned from storage at the end of its execution.	YES, NO	YES
CMSINCL	FILEDEF assist for input TEXT, extended format MODULE and TXTLIB files	YES, NO	YES
CMSMACRO	Indicates the program's requirements in relation to the DOS flag.	OS, DOS, ALL	OS
CMSSTR	Deletes previously loaded non-OS programs with NOPRES option on the LOADMOD.	YES, NO	NO
CMSSYSTEM	Indicates that when the MODULE is loaded, it is to have a storage protect key of zero.	YES, NO	NO
CMSXC	Indicates the programs require XC features.	YES, NO	NO

Note: All of these option keywords may be truncated to five characters.

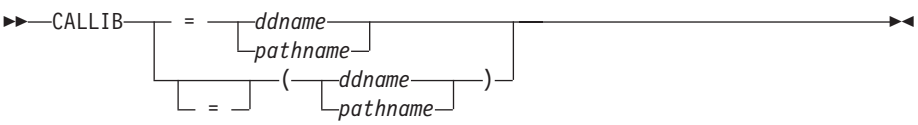
Comments on Some Program Management Binder Options:

AC This option is not relevant if the executable produced is a CMS module, an extended module, or a program object to be executed in CMS.

ALIASES
This option is not relevant because it only affects PDSEs.

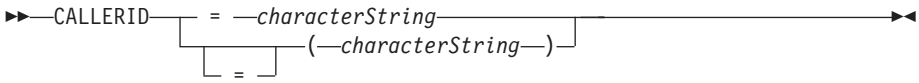
AMODE
AMODE64, although supported by the binder, is not currently supported by the CMS loader.

CALLIB
Can be specified as part of a parameter string using the following syntax:



The option's description in *z/OS MVS Program Management: Advanced Facilities* "5.1 Setting Options With the Binder API" specifies that *ddname* is the only allowable value. This is incorrect because a 1024 character *pathname* is also accepted. The *pathname* must resolve to either a directory or an archive file. CALLIB is limited to *ddname* if it is specified, not as an option, but as an entry in the STARTD FILELIST. See *z/OS MVS Program Management: Advanced Facilities* "3.25 STARTD: Start Dialog" for details.

CALLERID
The option can be specified as part of a parameter string using the following syntax:



characterString
a character string of up to 80 bytes to be printed at the top of each page of binder output listing.

COMPAT
See Table 8 on page 62 for the effect of the COMPAT option on the CMS binder.

DC This is NOT SUPPORTED in CMS. If specified in the installation defaults or on a SETOPT control statement, it may cause unpredictable results including the corruption of the target LOADLIB.

DCBS
This is NOT SUPPORTED in CMS. If specified in the installation defaults or on a SETOPT control statement, it may cause unpredictable results including the corruption of the target LOADLIB.

EXITS

- Although the EXITS option is listed in *z/OS MVS Program Management: Advanced Facilities* "5.1 Setting Options With the Binder API," the option cannot be specified as a keyword/value pair. It is only valid when specified in the parameter string on the STARTD API call.
- The description of the EXITS option in *z/OS MVS Program Management: User's Guide and Reference* "6.3.13 EXITS: Specify Exits to be Taken Option" is incorrect in describing the meaning of *variable* for the MESSAGE exit. It should read: "error severity level below which the binder does not call your exit."
- Exit routines may also be specified in the STARTD exit list. See *z/OS MVS Program Management: Advanced Facilities* "3.25 STARTD: Start Dialog" for details.

EXTATTR
Not applicable to the CMS environment.

FETCHOPT
Not applicable to the CMS environment.

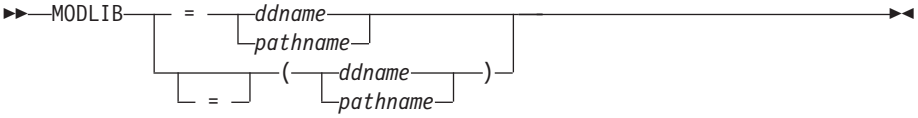
GID In addition to being applied to program objects created in the BFS, the group ID is also applied to sidedecks in the BFS. To set group IDs, you must have superuser authority or be the owner of the file or directory.

LNAME
The options LNAME and NAME are one and the same. LNAME may be

used in either the keyword/variable pair form or in a parameter string, whereas NAME is limited to the parameter string form. This option is only applicable to the LOADW function call, which is not currently supported in the CMS environment.

MAXBLK
This has limited effect in CMS because LOADLIBs are not blocked. The only effect may be on size of the records written.

MODLIB
Can be specified as part of a parameter string using the following syntax:



The option's description in *z/OS MVS Program Management: Advanced Facilities* "5.1 Setting Options With the Binder API" specifies that *ddname* is the only allowable value. This is incorrect because a 1024 character *pathname* is also accepted. MODLIB **is** limited to *ddname* if it is specified not as an option, but as an entry in the STARTD FILELIST. See *z/OS MVS Program Management: Advanced Facilities* "3.25 STARTD: Start Dialog" for details.

NAME
The options LNAME and NAME are one and the same. LNAME may be used in either the keyword/variable pair form or in a parameter string, whereas NAME is limited to the parameter string form. This option is only applicable to the LOADW function call which is not currently supported in the CMS environment.

OPTIONS
The OPTIONS option is supported by the application programming interface, but only as a parameter string option on the STARTD call, and with restrictions on mixing types of options. See details on page "Setting Options With the Binder API" on page 62.

OVLY
This option is not relevant if the executable produced is a CMS module, an extended module, or a program object to be executed in CMS.

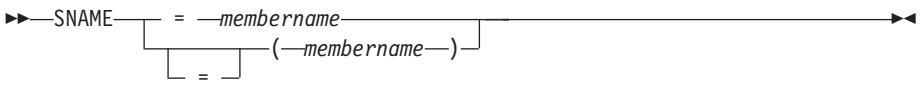
RMODE
The SPLIT value, although supported by the binder, is not currently supported by the CMS loader.

RES Not applicable to the CMS environment.

REUS
This option is not relevant if the executable produced is a CMS module, an extended module, or a program object to be executed in CMS.

SCTR
This option is not relevant if the executable produced is a CMS module, an extended module, or a program object to be executed in CMS.

SNAME
The option can be specified as part of a parameter string using the following syntax:



SIZE The use of this option is NOT SUPPORTED and is rejected with a return code of eight and a reason code of 83000111.

SSI This option is not relevant if the executable produced is a CMS module, an extended module, or a program object to be executed in CMS.

TEST This option is not relevant if the executable produced is a CMS module, an extended module, or a program object to be executed in CMS.

TRAP The TRAP option is not relevant if the executable produced is a CMS module, an extended module, or a program object to be executed in CMS.

UID In addition to being applied to program objects created in the BFS, the owner ID is also applied to sidedecks in the BFS. To set owner IDs, you must have superuser authority.

XCAL
This option is not relevant if the executable produced is a CMS module, an extended module, or a program object to be executed in CMS.

Invoking the Binder API

Setting the Invocation Environment

Only two of the program environment requirements, as described in *z/OS MVS Program Management: Advanced Facilities* "1.2.1 Setting the Invocation Environment," are relevant to the CMS binder. These are:

- In primary address space mode
- In 31-bit addressing mode

Loading the Binder

As described in *z/OS MVS Program Management: Advanced Facilities* "1.2.2 Loading the Binder": "The IEWBIND macro issues the LOAD macro for the IEWBIND entry point on the STARTD call". Considering IEWBIND is shipped as a module, the use of the LOAD macro requires that the compiler switch (COMPSWT) flag is set on. This is accomplished by using the COMPSWT macro to toggle the flag on and off around the STARTD function call.

API Function Calls

The API functions are described in *z/OS MVS Program Management: Advanced Facilities* "3.0 IEWBIND Function Reference." There are CMS-specific comments for the API calls described in the following:

BINDW

When reading the section "The processing rules for resolving external references are:" in "3.5.1 Processing Notes" of *z/OS MVS Program Management: Advanced Facilities*, take note of the following:

- Where SETL calls are mentioned, the same may be achieved using LIBRARY control statements.
- The RES option is not applicable in the CMS environment.
- The CALLIB used is determined in the following order:
  1. The CALLIB parameter if it was specified on the BINDW call

2. The most recent SETOPT control statement, or SETO API call, which specified the workmod token
3. The most recent SETO API call that specified the dialog token
4. The option list entry on the STARTD API call
5. The file list entry on the STARTD API call
6. The parameter string in the STARTD API call

If a CALLIB is used that resolves to a library, or a concatenation of libraries, **AND** the symbol is eight characters or less, **AND** the CMSAUTO option is in effect, then before looking at the CALLIB, the user's accessed disks or directories are searched for a file with a file name of *symbol* and a file type of TEXT. See Figure 4 on page 71 for a summary of the processing for final autocall.

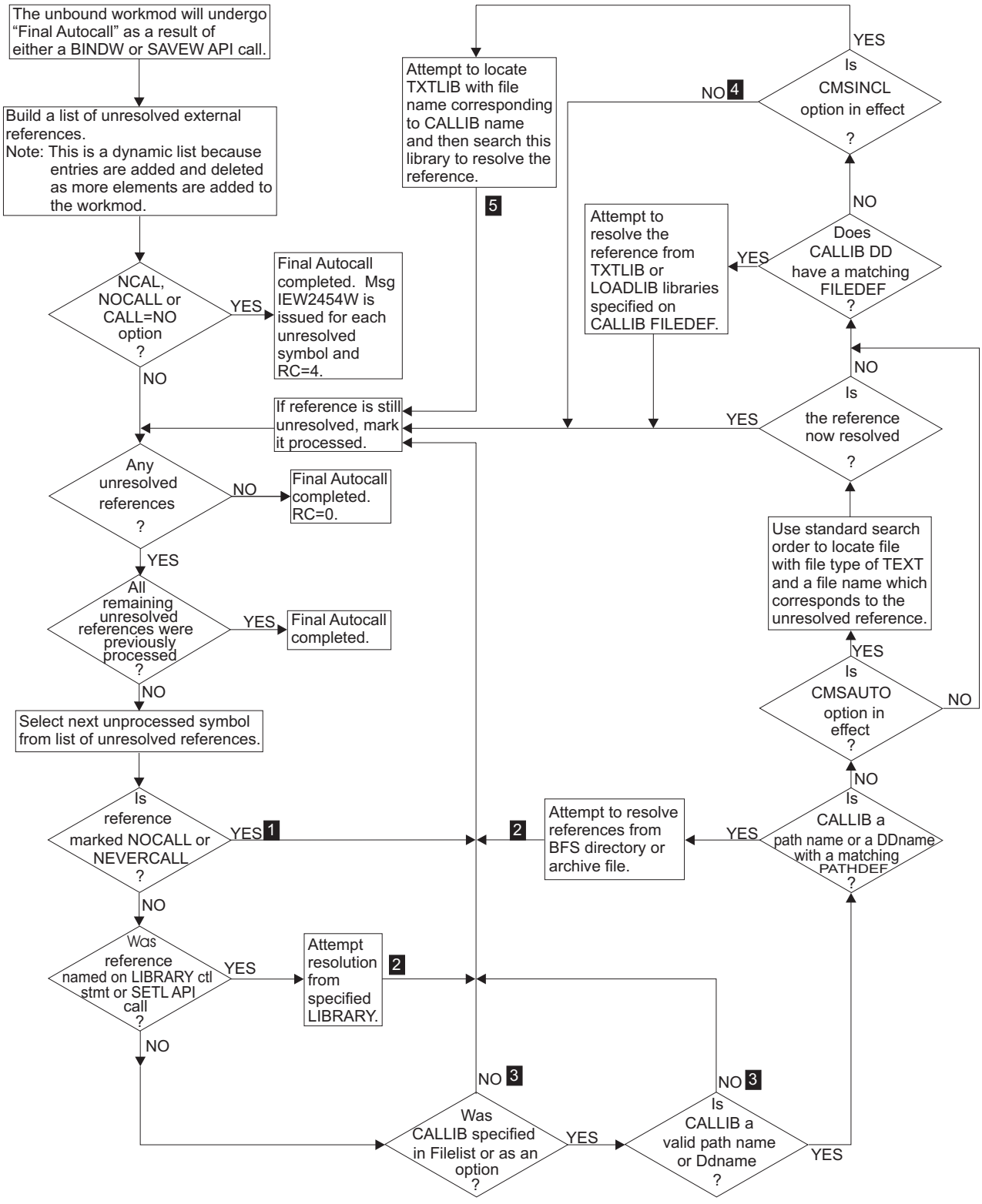


Figure 4. The API Perspective on Final Autocall

Notes on figure Figure 4.

1 If the symbol was designated 'restricted no call' or 'never call' on a LIBRARY control statement or SETLIB function call, then message

IEW2455W is issued. If the symbol was marked as 'never call' when previously bound, then message IEW2458W is issued. Return code is 4.

2 If the symbol is not resolved, then message IEW2456E is issued "MEMBER COULD NOT BE INCLUDED FROM THE DESIGNATED CALL LIBRARY" with a return code of 8.

3 Message IEW2457E is issued: "NO CALL LIBRARY SPECIFIED.." Return code is 8.

4 If CMSAUTO was specified, then message IEW2456E is issued; otherwise message IEW2457E is issued. Return code is 8.

5 If the symbol is not resolved, one of two messages are issued:

- IEW2457E is issued if no TXTLIB was located **AND** NOCMSAUTO was specified.
- Otherwise, IEW2456E is issued.

Return code is 8.

INCLUDE

INTYPE=TOKEN and INTYPE=POINTER are both unsupported.

ATTRIB=YES only applies to the z/OS MVS Program Management Binder attributes and not to the CMS-specific attribute options:

- CMSCLEAN
- CMSMACRO
- CMSSTR
- CMSSYSTEM
- CMSXC

LOADW

This function is not supported in the CMS environment.

SAVEW

The MODLIB used is determined in the following order:

1. The MODLIB parameter if it was specified on the SAVEW call
2. The most recent SETOPT control statement, or SETO API call, that specified a workmod token
3. The most recent SETO API call that specified a dialog token
4. The option list entry on the STARTD API call
5. The file list entry on the STARTD API call
6. The parameter string in the STARTD API call

The format of the executable produced by SAVEW is determined by the settings of MODLIB and the COMPAT option, see Table 8 on page 62. Table 10 on page 73 summarizes the output module or error message to be expected for various combinations of MODLIB and program object name specification.

Table 10. API-Level Module Output Determination

MODLIB Specification	SAVEW SNAME	SNAME Option	Output Module or Error Message
fn MODULE *fm*	none	none	*fn* MODULE *fm*
		sname	*sname* MODULE *fm*
	name	none	*name* MODULE *fm*
		sname	*name* MODULE *fm*
fn LOADLIB *fm*	none	none	IEW2639S
		sname	*fn* LOADLIB(*sname*) *fm*
	name	none	*fn* LOADLIB(*name*) *fm*
		sname	*fn* LOADLIB(*name*) *fm*
fn LOADLIB(*member*) *fm*	none	none	*fn* LOADLIB(*member*) *fm*
		sname	*fn* LOADLIB(*sname*) *fm*
	name	none	*fn* LOADLIB(*name*) *fm*
		sname	*fn* LOADLIB(*name*) *fm*
directory	none	none	IEW2812S
		sname	*directory*/*sname*
	name	none	*directory*/*name*
		sname	*directory*/*name*
directory/*file*	none	none	*directory*/*file*
		sname	*directory*/*sname*
	name	none	*directory*/*name*
		sname	*directory*/*name*

Notes about the table:

1. This is a decision table. To use it, start in the left-hand column and work to the right choosing the row according to the values that apply to your situation. The cell in the right-most column that you arrive at describes the output module that is created, or else gives the error message number that you would get.
2. The SNAME option value can be set in one of the following ways:
   - A parameter string or option value on a STARTD API call
   - From an options data set processed as the result of an OPTIONS option in a parameter string on a STARTD API call
   - From a dialog or workmod level SETO API call
   - From a SETOPT control statement in the input stream
3. When a LOADLIB is specified with a member, and an SNAME option is in effect but no SNAME is specified on the SAVEW API call, the SNAME option value is used as the member name. This is not what you would expect from reading *z/OS MVS Program Management: Advanced Facilities* (but is consistent with z/OS MVS Program Management Binder behavior).
4. When a directory is specified with a file, and an SNAME option is in effect, but no SNAME is specified on the SAVEW API call, the SNAME option value is used as the file name. This is not what you would expect from reading *z/OS MVS Program Management: Advanced Facilities* (but is consistent with z/OS MVS Program Management Binder behavior).
5. When a path is specified, the expected behavior is dependent on the existence of the specified path. If the directory or file does not exist, message IEW2785S is issued.

SETO

All option keywords may be truncated to three characters with the exception of the following, which require a minimum of five characters:

- CALLIB
- CMSCLEAN
- CMSMACRO
- CMSSTR
- CMSSYSTEM
- CMSXC

STARTD

The following lists of options can only be specified on a STARTD call. The first list of options is classified in the *z/OS MVS Program Management: Advanced Facilities* "5.1 Setting Options With the Binder API" as "environmental options":

- CALLERID
- COMPAT
- LINECT
- MSGLEVEL
- PRINT
- SIZE
- TERM
- TRAP
- WKSPACE

The second list can only appear in the parameter string on the STARTD function call:

- EXITS
- RENT
- REFR
- REUS (Only the old form)

The truncation of option keywords is the same as for SETO; see "SETO" with the addition of CALLERID, which must have a minimum of five characters.

STARTS

Overlays are not relevant in the CMS environment.

Exits

As described in *z/OS MVS Program Management: Advanced Facilities* "7. User Exits" with the following additions:

- If exits are specified using the option, then they must be found as MODULE files on one of the user's accessed disks or directories.
- The SAVE exit is invoked only if the target is a LOADLIB member.

C/C++ API

The following z/OS MVS Program Management Binder C/C++ API access functions and utilities functions are supported. For more information, see *z/OS MVS Program Management: Advanced Facilities* .

Note that CMS Binder does not support the following features:

- The _IEW_ZOSV1R12_ feature test macro

- The XPLINK DLL
- The __iew_modmap.h header file

The CMS Binder C/C++ API uses a single header file with the name IEWBAPI H, rather than __iew_api.h as in z/OS.

C/C++ API access functions:
- __iew_addA()
- __iew_alignT()
- __iew_alterW()
- __iew_autoC()
- __iew_bindW()
- __iew_closeW()
- __iew_getC()
- __iew_getD()
- __iew_getE()
- __iew_getN()
- __iew_import()
- __iew_includeName()
- __iew_includePtr()
- __iew_includeSmde()
- __iew_includeToken()
- __iew_insertS()
- __iew_loadW()
- __iew_openW()
- __iew_orderS()
- __iew_putD()
- __iew_rename()
- __iew_resetW()
- __iew_saveW()
- __iew_setL()
- __iew_setO()
- __iew_startS()

C/C++ API utilities functions:
- __iew_api_name_to_str()
- __iew_create_list()
- __iew_eod()
- __iew_get_reason_code()
- __iew_get_return_code()
- __iew_get_cursor()
- __iew_set_cursor()

Invoking the Binder from a Program

In general, the program invocation of the CMS binder version of IEWBLINK is the same as for the program management binder as documented in *z/OS MVS Program Management: User's Guide and Reference* "3.4 Invoking the Binder from a Program." All extensions, restrictions and other differences are documented in the rest of this section.

The module IEWBLINK can be given control by a program using the LINK, XCTL or ATTACH macroinstructions or the combination of the LOAD and CALL macroinstructions. IEWBLINK uses the INCLUDE function of the binder API to process any files associated with the ddname SYSLIN and attempts to bind and save any resulting program object to the target identified by the ddname SYSLMOD. Support for the alternate entry points IEWBLOAD, IEWBLODI, and IEWBLDGO is not provided by the CMS binder.

COMPSWT

Prior to issuing the LINK, LOAD, XCTL or ATTACH macroinstruction which causes the IEWBLINK module to be loaded, the compiler switch (COMPSWT) flag must be set on. This is accomplished by using the COMPSWT macro to toggle the flag on and off.

Aliases

None of the alternate aliases for IEWBLINK are supported by the CMS binder:

- IEWL
- HEWL
- HEWLH096
- LINKEDIT

Primary Input

SYSLIN is the default ddname for primary input to IEWBLINK. Multiple FILEDEFs for a ddname can be defined if the FILEDEF commands use the CONCAT option. In the case of IEWBLINK, all active FILEDEFs and PATHDEFs for SYSLIN are used. FILEDEFs are processed in the order that the FILEDEF commands are entered, and all the FILEDEFs are processed before any PATHDEF is processed. For example:

```
filedef syslin disk file1 text a (concat
openvm pathdef create syslin ./file3
filedef syslin disk file2 text a (concat
```

This would result in file1, file2 and file3 (in that order) being processed by IEWBLINK.

Note: Only one PATHDEF at a time can be active for a given ddname.

Setting Options for IEWBLINK

Options may be set in a variety of ways while using IEWBLINK:

- As part of an option string in the installation defaults module IEWBODEF
- As part of an option string passed as a parameter on the LINK, CALL, XCTL or ATTACH macroinstruction

- As part of an option string read in from an options file; the options file is defined on an OPTIONS option which may be passed on the invoking macroinstruction.
- As part of a parameter string on the SETOPT control statement

The general syntax rules for specifying options in an option string are described in *z/OS MVS Program Management: User's Guide and Reference* "Chapter 6. Binder Options Reference."

In addition to the binder options specified in the *z/OS MVS Program Management: User's Guide and Reference*, the CMS binder provides CMS-specific options; see page "Setting Options With the Binder API" on page 62 for the syntax and descriptions of these options.

There are restrictions placed on the mixing of Program Management binder options with CMS-specific binder options; see page "Setting Options With the Binder API" on page 62 for details.

Output Formats

The format of the executable produced by IEWBLINK is determined by the settings of SYSLMOD (MODLIB) and the COMPAT option, as shown in Table 8 on page 62. Table 11 summarizes the output module or error message to be expected for various combinations of SYSLMOD and program object name specification.

Table 11. IEWBLINK SYSLMOD Output Determination

SYSLMOD Specification	NAME Control Stmt	SNAME on SETOPT Stmt	Member in DD List	SNAME in Option List	Output Module or Error Message
fn MODULE *fm*	none	none	none	none	*fn* MODULE *fm*
				option	*option* MODULE *fm*
			ddlist	-	*ddlist* MODULE *fm*
		setopt	-	-	*setopt* MODULE *fm*
	name	-	-	-	*name* MODULE *fm*
fn LOADLIB *fm*	none	none	none	none	IEW2639S
				option	*fn* LOADLIB(*option*) *fm*
			ddlist	-	*fn* LOADLIB(*ddlist*) *fm*
		setopt	-	-	*fn* LOADLIB(*setopt*) *fm*
	name	-	-	-	*fn* LOADLIB(*name*) *fm*

Table 11. IEWBLINK SYSLMOD Output Determination (continued)

SYSLMOD Specification	NAME Control Stmt	SNAME on SETOPT Stmt	Member in DD List	SNAME in Option List	Output Module or Error Message
fn LOADLIB(*member*) *fm*	none	none	none	none	*fn* LOADLIB(*member*) *fm*
				option	*fn* LOADLIB(*option*) *fm*
			ddlist	-	*fn* LOADLIB(*ddlist*) *fm*
		setopt	-	-	*fn* LOADLIB(*setopt*) *fm*
	name	-	-	-	*fn* LOADLIB(*name*) *fm*
directory	none	none	none	none	IEW2812S
				option	*directory/option*
			ddlist	-	*directory/ddlist*
		setopt	-	-	*directory/setopt*
	name	-	-	-	*directory/name*
directory/file	none	none	none	none	*directory/file*
				option	*directory/option*
			ddlist	-	*directory/ddlist*
		setopt	-	-	*directory/setopt*
	name	-	-	-	*directory/name*

Notes about the table:

1. This is a decision table. Starting in the left-hand column, move to the right while choosing the row according to the values that apply to your situation. The cell in the rightmost column either describes the output module that is created or lists an error message number that you would get.
2. When a path is specified, the expected behavior is dependent on the existence of the specified path. If the directory or file does not exist, message IEW2785S is issued.

General Environmental Considerations

Concatenating Files

To define more than one library with the same ddname, use the CONCAT option of the FILEDEF command in conjunction with an appropriate GLOBAL command. You can concatenate CMS TXTLIB files with each other or with TXTLIB files on OS/MVS disks. Any library to be searched must be specified in the GLOBAL TXTLIB statement. Similarly, you can concatenate LOADLIBs by specifying CONCAT on a FILEDEF for a LOADLIB and having an active GLOBAL LOADLIB.

CMS files in the GLOBAL list do not require individual file definitions. The GLOBAL list determines the order in which the libraries are searched. For GLOBAL libraries, the file mode on a related concatenated FILEDEF is honored. If the file cannot be found on the specified CMS disk, an error message is issued during open processing for the ddname. If a file mode of '*' is used on FILEDEFs relating to GLOBAL libraries, the established search order finds the first occurrence

of the file and uses it. However, the use of file mode '*' can increase search time and degrade overall performance if there are many disks in the search order.

For example, set up two OS/MVS libraries and a CMS LOADLIB for use in an incremental autocall with a ddname of AUTOC1:

```
ACCESS 520 P        (520 is the address of the OS/MVS disk)
FILEDEF AUTOC1 DISK OSLIB LOADLIB P DSN SYS1 LIB1
FILEDEF AUTOC1 DISK OSLIB2 LOADLIB P DSN SYS1 LIB2 (CONCAT)
GLOBAL LOADLIB OSLIB OSLIB2 DMSGPI
```

It is advisable not to code the CONCAT option on the first FILEDEF command because this guarantees that it clears all previous FILEDEFs for that ddname.

Restrictions

Relocatability

1. The CMS Binder only generates relocatable standard format modules. This implies the following:
   - Fixed transient, variable transient, and variable non-relocatable modules cannot be generated.
   - The BIND command needs no equivalent to the LOAD command's RLDSAVE/NORLDSAVE option.
2. Two-byte adcons (Y-types and AL2) are not supported and do not appear in either standard or extended modules. This is because these are discarded with message IEW2633 unless the module output is directed to a LOADLIB.

Overlay Structures

The z/OS MVS Program Management Binder only supports overlay structures for:

`COMPAT(LKED)`

and

`COMPAT(PM1)`

Overlay structures are not supported in CMS.

Appendix A. Troubleshooting

This appendix contains information that can be used to analyze and resolve problems in the CMS Binder.

For messages issued by the CMS Binder, refer to *z/VM: CMS and REXX/VM Messages and Codes*.

The BIND Command DEBUG Option

The DEBUG option of the BIND command requests debugging messages to be issued by the command interface to indicate its progress, and stops the suppression of information messages related to the API calls made by the interface. It should only be used at the direction of your IBM service representative. The syntax of the DEBUG option is:

The effect of the DEBUG settings is as follows:

ALL Requests both the CMD and API debug options.

CMD Causes the command interface to issue debugging messages containing information that may be useful for command level problem diagnosis.

API Stops the suppression of information messages related to the API calls made by the command interface that may be useful for command or API level problem diagnosis. LIST ALL should be specified in conjunction with this option.

Note: The DEBUG option specifying ALL or CMD should be specified as the first option. This allows it to be found prior to the options processor being invoked. If it is not specified first, the debugging messages that would have been issued prior to options processing are not issued, and a warning message is issued during options processing to indicate this condition.

Diagnostic Information

The z/OS MVS Program Management Binder can generate up to four different output files that provide diagnostic information. Also, diagnostic options can be provided in an options file with the ddname IEWPARMS. The options file can be a sequential data set, a member of a maclib, a BFS file, or a concatenation of sequential data sets.

Table 12 shows the diagnostic input and output files that can be used by the z/OS MVS Program Management Binder.

Table 12. z/OS MVS Program Management Binder Diagnostic Input and Output Files

DD Name	Filelist Name	Type of File	Contents
SYSPRINT	PRINT	Output	z/OS MVS Program Management Binder processing messages. See "The SYSPRINT File" on page 82.

© Copyright IBM Corp. 2001, 2013

Table 12. z/OS MVS Program Management Binder Diagnostic Input and Output Files (continued)

DD Name	Filelist Name	Type of File	Contents
IEWDIAG	DIAG	Output	z/OS MVS Program Management Binder diagnostic messages are duplicated in SYSPRINT if it exists. See "The IEWDIAG File."
IEWGOFF	GOFF	Output	z/OS MVS Program Management Binder GOFF records produced by the binder when the input is Extended Object (XOBJ) module records. See "The IEWGOFF File" on page 83.
IEWPARMS		Input	z/OS MVS Program Management Binder options, specified in MVS format (using commas and equals signs rather than spaces). See "The IEWPARMS File" on page 83.
IEWTRACE	TRACE	Output	z/OS MVS Program Management Binder Internal Trace. See "The IEWTRACE File" on page 83.

Note: These files are the ddnames which:

- For SYSPRINT, is the default unless using the API.
- For IEWPARMS, is fixed.
- In all other cases these ddnames are fixed unless using the API, in which case they are defaults and can be overridden by providing an entry in the STARTD file list.

The SYSPRINT File

The SYSPRINT file is where the z/OS MVS Program Management Binder writes the Binder processing messages.

The SYSPRINT file can be allocated to one of the following:
- A FILEDEF to a file on an accessed CMS minidisk or SFS directory
- A FILEDEF to the virtual printer
- A FILEDEF to the terminal
- A PATHDEF to a BFS file (not a BFS directory)

No SYSPRINT records are produced if NOPRINT option is set or when the API is being used if either a FILEDEF or a PATHDEF for SYSPRINT cannot be located.

The IEWDIAG File

The IEWDIAG file is where the z/OS MVS Program Management Binder writes the Binder diagnostic messages. If there is a SYSPRINT file, these messages are duplicated there.

The IEWDIAG file can be allocated to one of the following:
- A FILEDEF to a file on an accessed CMS minidisk or SFS directory
- A FILEDEF to the virtual printer
- A FILEDEF to the terminal
- A PATHDEF to a BFS file (not a BFS directory)

If a FILEDEF or a PATHDEF cannot be located, then no IEWDIAG records are produced.

The IEWGOFF File

The IEWGOFF file is where the z/OS MVS Program Management Binder writes GOFF records when the input is Extended Object (XOBJ) module records.

The IEWGOFF file can be allocated to one of the following:
- A FILEDEF to a file on an accessed CMS minidisk or SFS directory
- A FILEDEF to the virtual printer
- A FILEDEF to the terminal

Note: Output to the terminal for the IEWGOFF file is not recommended.

If a FILEDEF cannot be located, then no IEWGOFF records are produced.

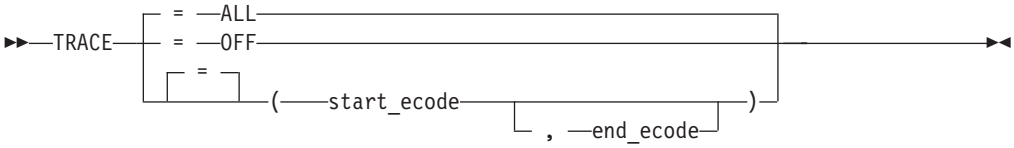
The IEWPARMS File

The IEWPARMS file contains z/OS MVS Program Management Binder options, specified in MVS format (using commas and equals signs rather than spaces). The existence of the IEWPARMS DDNAME is all that is required for it to be used as a source of options. The file can reside on a minidisk, SFS directory or in the BFS. The ddname is specified using either of the following commands:

- FILEDEF
- PATHDEF

The IEWTRACE File

The IEWTRACE file is where the z/OS MVS Program Management Binder writes the Binder internal trace. If a FILEDEF cannot be located, then no IEWTRACE records are produced. The amount of trace output can be controlled with the TRACE option:



ALL
indicates that tracing of all components is required. This is the default when the IEWTRACE DD is allocated.

OFF
indicates that no tracing is to occur even if the ddname IEWTRACE is allocated.

start_ecode
the event code which will cause tracing to start.

end_ecode
the event code which will stop tracing.

The IEWTRACE file can be allocated to one of the following:
- A FILEDEF to a file on an accessed CMS minidisk or SFS directory
- A FILEDEF to the virtual printer
- A FILEDEF to the terminal

Note: Output to the terminal for the IEWTRACE file is not recommended.

Unexpected messages

In the following message descriptions the original explanations and system actions from the relevant product documentation have been included in parentheses, followed by explanations and actions applicable to the CMS Binder. Refer to the relevant product documentation for complete information.

z/OS MVS Program Management Binder (IEW) Messages

Refer to *z/OS MVS System Messages, Vol 8 (IEF-IGD)* for further details of Binder specific messages (messages with a prefix of `IEW`).

IEW2730S **INVALID RECFM FOR DDNAME** *ddname* **AND CONCATENATION NUMBER** *number***.**

Explanation: (A sequential data set was found to have an incorrect record format. The DCB must specify either RECFM=F or RECFM=U, or RECFM=V for GOFF modules.)

This is what you get if you try to bind a standard format CMS module (for example, bind fred (ft(module), where fred is an ordinary module)

System action: (Processing for the data set terminates.)

No special action is taken by the CMS Binder.

User response: Reissue the bind command for a valid input file.

IEW2796S **FILE ASSOCIATED WITH DDNAME** *ddname* **CANNOT BE WRITTEN. HFS WRITE ISSUED RETURN CODE** *return code* **AND REASON CODE** *reason code***.**

Explanation: (UNIX® System Services write() failed with the indicated return and reason codes.)

Issued following DMS1905S when the write routine (DMSBX2WR) determines that the module being saved already exists and the replace option was not specified.

System action: (Processing for the file ends.)

No special action is taken by the CMS Binder.

User response: Refer to the user response for DMS1905S.

Note: The "posix" return and reason codes returned with this message are generated by DMSBX2WR and documented in *z/VM: OpenExtensions Callable Services Reference*.

Language Environment (CEE) Messages

Refer to *z/OS: Language Environment Run-Time Messages* for further details of Language Environment messages (messages with a prefix of `CEE`).

CEE3534S **The requested function is not supported.**

Explanation: (The CEEPPOS service was invoked with a function that is not recognized. No action was taken.)

The version of Language Environment that is running does not support CMS program objects. So a request for an LE program object service has been rejected.

System action: (Unless the condition is handled the default action is to terminate the enclave.)

The deferred class loader cannot be invoked on behalf of LE without the LE program object service.

User response: Provide access to LE version 1.8 or higher to provide direct support for CMS program objects.

OpenExtensions Return and Reason Codes

The CMS Binder utilizes the POSIX Callable Services to process the Program Object data. Any data errors detected with the Program Object data generates an error message indicating the nature of the error. This is followed by the z/OS MVS Program Management Binder error message:

```
IEW2796S DF16 FILE ASSOCIATED WITH DDNAME /fd CANNOT BE WRITTEN. HFS WRITE
         ISSUED RETURN CODE rc AND REASON CODE rsn.
```

For return codes and reason codes in the range 83000000-8300FFFF, refer to *z/VM: OpenExtensions Callable Services Reference* for further details of these OpenExtensions return codes (rc) and reason codes (rsn) that relate to the CMS Binder.

Common Problems

A large number and variety of problems may occur with the CMS Binder. When using the CMS BIND command, the most likely errors are in the allocation or specification of included or autocall objects.

Incorrect SYSLIB

An incorrect or missing SYSLIB FILEDEF may result in unexpected z/OS MVS Program Management Binder messages, particularly where an object being bound is required to resolve external references. For instance, the following FILEDEF for SYSLIB with the CONCAT keyword:

```
FILEDEF SYSLIB DISK SCEELKED TXTLIB * (CONCAT PERM
```

And no GLOBAL TXTLIB specified causes the following messages when the z/OS MVS Program Management Binder attempts to open the SYSLIB:

```
IEW2715S D806 JOB FILE CONTROL BLOCK (JFCB) CANNOT BE FOUND FOR DDNAME SYSLIB.
         RDJFCB MACRO ISSUED RETURN CODE  4.
IEW2453E 920D UNABLE TO PROCESS LIBRARY SYSLIB DURING AUTOCALL PROCESSING.
IEW2456E 9207 SYMBOL CEEBETBL UNRESOLVED.  MEMBER COULD NOT BE INCLUDED FROM
         THE DESIGNATED CALL LIBRARY.
IEW2456E 9207 SYMBOL CEEROOTA UNRESOLVED.  MEMBER COULD NOT BE INCLUDED FROM
         THE DESIGNATED CALL LIBRARY.
IEW2456E 9207 SYMBOL PRINTF UNRESOLVED.  MEMBER COULD NOT BE INCLUDED FROM THE
         DESIGNATED CALL LIBRARY.
IEW2456E 9207 SYMBOL EDCINPL UNRESOLVED.  MEMBER COULD NOT BE INCLUDED FROM THE
         DESIGNATED CALL LIBRARY.
IEW2456E 9207 SYMBOL CEESG003 UNRESOLVED.  MEMBER COULD NOT BE INCLUDED FROM
         THE DESIGNATED CALL LIBRARY.
DMSBCP1621W Binder function BINDW completed with return code 8 and reason code 83000320
```

The solution is to remove the CONCAT keyword on the FILEDEF or to issue a GLOBAL command for one or more valid TXTLIBs.

Storage

Insufficient virtual storage may give a variety of symptoms:

- Trying to run the Binder with insufficient contiguous space for the executables:

  ```
  DMSFRO159E Insufficient storage available to satisfy free storage
             request from 0113A11E
  DMSMOD109S Virtual storage capacity exceeded
  DMSBCP1621W Binder function STARTD completed with return code 16 and
              reason code 02CDCC12
  ```

- Binding a program module with insufficient working storage may cause termination with any of a number of messages:

  ```
  IEW2971T C406 INSUFFICIENT DATASPACE STORAGE WAS AVAILABLE TO CONTINUE
           BINDER PROCESSING.
  DMSBCP1621W Binder function BINDW completed with return code 16 and
              reason code 83000050
  ```

  or

  ```
  IEW2900T E913 BINDER ABNORMAL TERMINATION  4F5A2900
  IEW2134S 0005 INVALID DIALOG TOKEN 8054900001D1B638 PASSED.
  DMSBCP1621W Binder function SAVEW completed with return code 16 and
              reason code 83EE2900
  ```

CMS OpenExtensions Problems

Here is a selection of the most common problems with BFS output:

- SYSLMOD to a directory without write access:

```
IEW2777S DF03 FILE ASSOCIATED WITH DDNAME SYSLMOD CANNOT BE WRITTEN
         BECAUSE HFS HAS DENIED WRITE ACCESS TO THE FILE.
DMSBCP1621W Binder function SAVEW completed with return code 12 and
            reason code 83000602
```

- SYSDEFSD to a directory without write access:

```
IEW2777S DF03 FILE ASSOCIATED WITH DDNAME SYSDEFSD CANNOT BE WRITTEN
         BECAUSE HFS HAS DENIED WRITE ACCESS TO THE FILE.
DMSBCP1621W Binder function SAVEW completed with return code 12 and
            reason code 83000424
```

- SYSPRINT to a directory:

```
IEW2765S ED43 FILE ASSOCIATED WITH DDNAME SYSPRINT CANNOT BE OPENED.
         HFS OPEN ISSUED RETURN CODE  123 AND REASON CODE  100.
DMSBCP1621W Binder function STARTD completed with return code 12 and
            reason code 83000200
```

- SYSTERM to a directory:

```
IEW2765S ED43 FILE ASSOCIATED WITH DDNAME SYSTERM CANNOT BE OPENED.
         HFS OPEN ISSUED RETURN CODE  123 AND REASON CODE 100.
DMSBCP1621W Binder function STARTD completed with return code 12 and
            reason code 83000203
```

- SYSPRINT to a file without write access:

```
IEW2765S ED43 FILE ASSOCIATED WITH DDNAME SYSPRINT CANNOT BE OPENED.
         HFS OPEN ISSUED RETURN CODE  111 AND REASON CODE  0.
DMSBCP1621W Binder function SAVEW completed with return code 12 and
            reason code 83000200
```

- SYSTERM to a file without write access:

```
IEW2765S ED43 FILE ASSOCIATED WITH DDNAME SYSTERM CANNOT BE OPENED.
         HFS OPEN ISSUED RETURN CODE  111 AND REASON CODE  0.
DMSBCP1621W Binder function SAVEW completed with return code 12 and
            reason code 83000203
```

Appendix B. Customization and Set Up

The following sections describe the setup required to use the Program Management Binder for CMS and the ways it can be customized.

Virtual Storage Requirements

The storage required for the binder can be determined by adding the sum of the sizes of the executables to the working storage. The approximate sizes of the executables (all RMODE ANY) are as follows:

BIND	75K
IEWBIND	57K
IEWBINDM	1771K

The amount of working storage required by the binder is directly related to the number of pieces being bound together (not just the text size itself, but the number of CSECTs, load modules, RLDs, and so on, being combined). The recommended practice is to provide working storage that is twice the text size.

Providing Dataspace Support

The CMS Binder makes use of dataspaces if the user is authorized to create dataspaces. When a user is not authorized to create dataspaces, or is unable to create a dataspace of the required size, the z/OS MVS Program Management Binder continues and uses storage from the CMS user's primary address space. The use of the user's storage rather than dataspace storage is likely to cause problems when using the CMS Binder to bind large objects.

The use of dataspaces by the CMS Binder requires a user to be able to create `PRIVATE` (non-shared) dataspaces.

The `DIRMAINT XCONFIG` command is used to define a user's authority to create dataspaces. The use of the `DIRMAINT` command must be performed by a person authorized to use the `XCONFIG` subcommand. Refer to *z/VM: Directory Maintenance Facility Commands Reference* and *z/VM: CP Planning and Administration*.

The following example of the DIRMAINT XCONFIG command can be used to provide the required dataspace support:

```
DIRMAINT XCONFIG ADDRSPACE MAXNUMBER 20 TOTSIZE 64M NOSHARE
DIRMAINT XCONFIG ACCESSLIST ALSIZE 40
```

The number used in the above `XCONFIG` commands should be reviewed according to installation requirements. The minimum values that should be used are:

MAXNUMBER	4
TOTSIZE	16M
ALSIZE	20

Note: The use of VM dataspaces is restricted to a virtual machine running in XC mode. When a user is running a virtual machine in XA or ESA mode, dataspaces are not available.

© Copyright IBM Corp. 2001, 2013

Defining Installation Defaults

Default values for the CMS Binder can be tailored to the need of your installation by replacing the data-only module IEWBODEF.

The customization of the z/OS MVS Program Management Binder default options is described in the section "Establishing Installation Defaults" in *z/OS MVS Program Management: User's Guide and Reference*. There are also a number of CMS-specific binder options. So an additional section has been added to the IEWBODEF module that contains the installation default options that relate to CMS. A halfword length field preceding each of the strings contains the length of each of the z/OS MVS Program Management Binder and CMS-specific binder option strings. The module is initially supplied as two null strings with each halfword string length set to zero.

The following example shows how to set the z/OS MVS Program Management Binder installation default options LET and MSGLEVEL, and the CMS Binder option CMSAUTO:

```
IEWBODEF CSECT
IEWBODEF AMODE 31
IEWBODEF RMODE ANY
         DC    AL2(MVSEND-MVSPARMS)
MVSPARMS EQU   *
         DC    C'LET(4),'
         DC    C'MSGLEVEL(4)'
MVSEND   EQU   *
         DC    AL2(CMSEND-CMSPARMS)
CMSPARMS EQU   *
         DC    C'CMSAUTO(NO)'
CMSEND   EQU   *
         END
```

Once this code has been assembled, it must be linked into the relocatable module IEWBODEF using either LOAD/GENMOD or the CMS Binder.

Note: When the BIND command is used to create the IEWBODEF module, specify the COMPAT LKED option to create a standard CMS module.

Notices

This information was developed for products and services offered in the U.S.A.

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
Armonk, NY 10504-1785
U.S.A.

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

The following paragraph does not apply to the United Kingdom or any other country where such provisions are inconsistent with local law:
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 states 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.

© Copyright IBM Corp. 2001, 2013

IBM may use or distribute any of the information you supply 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:

Site Counsel
IBM Corporation
2455 South Road
Poughkeepsie, NY 12601-5400
U.S.A.

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.

Any performance data contained herein was determined in a controlled environment. Therefore, the results obtained in other operating environments may vary significantly. Some measurements may have been made on development-level systems and there is no guarantee that these measurements will be the same on generally available systems. Furthermore, some measurement may have been estimated through extrapolation. Actual results may vary. Users of this document should verify the applicable data for their specific environment.

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.

All 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 the names and addresses used by an actual business enterprise 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.

Privacy Policy Considerations

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 IBM Online Privacy Policy at http://www.ibm.com/privacy and the IBM Online Privacy Statement at http://www.ibm.com/privacy/details, in particular the section entitled “Cookies, Web Beacons and Other Technologies”, and the IBM Software Products and Software-as-a-Service Privacy Statement at http://www.ibm.com/software/info/product-privacy.

Programming Interface Information

This book documents intended Programming Interfaces that allow the customer to write programs to obtain services of the z/OS MVS Program Management Binder.

Trademarks

IBM, the IBM logo, and ibm.com are trademarks or registered trademarks of International Business Machines Corp., registered in many jurisdictions worldwide. Other product and service names might be trademarks of IBM or other companies. A current list of IBM trademarks is available on the web at IBM copyright and trademark information - United States (www.ibm.com/legal/us/en/copytrade.shtml).

Glossary

For a list of z/VM terms and their definitions, see *z/VM: Glossary*.

The z/VM glossary is also available through the online z/VM HELP Facility, if HELP files are installed on your z/VM system. For example, to display the definition of the term "dedicated device", issue the following HELP command:

```
help glossary dedicated device
```

While you are in the glossary help file, you can do additional searches:

- To display the definition of a new term, type a new HELP command on the command line:

  ```
  help glossary newterm
  ```

  This command opens a new help file inside the previous help file. You can repeat this process many times. The status area in the lower right corner of the screen shows how many help files you have open. To close the current file, press the Quit key (PF3/F3). To exit from the HELP Facility, press the Return key (PF4/F4).
- To search for a word, phrase, or character string, type it on the command line and press the Clocate key (PF5/F5). To find other occurrences, press the key multiple times.

  The Clocate function searches from the current location to the end of the file. It does not wrap. To search the whole file, press the Top key (PF2/F2) to go to the top of the file before using Clocate.

© Copyright IBM Corp. 2001, 2013

Bibliography

See the following publications for additional information about z/VM. For abstracts of the z/VM publications, see *z/VM: General Information*, GC24-6193

Where to Get z/VM Information

z/VM product information is available from the following sources:

- z/VM V6.3 Information Center (publib.boulder.ibm.com/infocenter/zvm/v6r3/)
- IBM: z/VM Internet Library (www.ibm.com/vm/library/)
- IBM Publications Center (www.ibm.com/e-business/linkweb/publications/servlet/pbi.wss)
- *IBM Online Library: z/VM Collection*, SK5T-7054

z/VM Base Library

Overview

- *z/VM: General Information*, GC24-6193
- *z/VM: Glossary*, GC24-6195
- *z/VM: License Information*, GC24-6200

Installation, Migration, and Service

- *z/VM: Installation Guide*, GC24-6246
- *z/VM: Migration Guide*, GC24-6201
- *z/VM: Service Guide*, GC24-6247
- *z/VM: VMSES/E Introduction and Reference*, GC24-6243

Planning and Administration

- *z/VM: CMS File Pool Planning, Administration, and Operation*, SC24-6167
- *z/VM: CMS Planning and Administration*, SC24-6171
- *z/VM: Connectivity*, SC24-6174
- *z/VM: CP Planning and Administration*, SC24-6178
- *z/VM: Getting Started with Linux on System z*, SC24-6194
- *z/VM: Group Control System*, SC24-6196
- *z/VM: I/O Configuration*, SC24-6198
- *z/VM: Running Guest Operating Systems*, SC24-6228
- *z/VM: Saved Segments Planning and Administration*, SC24-6229
- *z/VM: Secure Configuration Guide*, SC24-6230
- *z/VM: TCP/IP LDAP Administration Guide*, SC24-6236
- *z/VM: TCP/IP Planning and Customization*, SC24-6238
- *z/OS and z/VM: Hardware Configuration Manager User's Guide*, SC33-7989

Customization and Tuning

- *z/VM: CP Exit Customization*, SC24-6176
- *z/VM: Performance*, SC24-6208

Operation and Use

- *z/VM: CMS Commands and Utilities Reference*, SC24-6166
- *z/VM: CMS Pipelines Reference*, SC24-6169
- *z/VM: CMS Pipelines User's Guide*, SC24-6170
- *z/VM: CMS Primer*, SC24-6172
- *z/VM: CMS User's Guide*, SC24-6173
- *z/VM: CP Commands and Utilities Reference*, SC24-6175
- *z/VM: System Operation*, SC24-6233
- *z/VM: TCP/IP User's Guide*, SC24-6240
- *z/VM: Virtual Machine Operation*, SC24-6241
- *z/VM: XEDIT Commands and Macros Reference*, SC24-6244
- *z/VM: XEDIT User's Guide*, SC24-6245
- *CMS/TSO Pipelines: Author's Edition*, SL26-0018

Application Programming

- *z/VM: CMS Application Development Guide*, SC24-6162
- *z/VM: CMS Application Development Guide for Assembler*, SC24-6163
- *z/VM: CMS Application Multitasking*, SC24-6164
- *z/VM: CMS Callable Services Reference*, SC24-6165
- *z/VM: CMS Macros and Functions Reference*, SC24-6168
- *z/VM: CP Programming Services*, SC24-6179
- *z/VM: CPI Communications User's Guide*, SC24-6180

© Copyright IBM Corp. 2001, 2013

- *z/VM: Enterprise Systems Architecture/Extended Configuration Principles of Operation*, SC24-6192
- *z/VM: Language Environment User's Guide*, SC24-6199
- *z/VM: OpenExtensions Advanced Application Programming Tools*, SC24-6202
- *z/VM: OpenExtensions Callable Services Reference*, SC24-6203
- *z/VM: OpenExtensions Commands Reference*, SC24-6204
- *z/VM: OpenExtensions POSIX Conformance Document*, GC24-6205
- *z/VM: OpenExtensions User's Guide*, SC24-6206
- *z/VM: Program Management Binder for CMS*, SC24-6211
- *z/VM: Reusable Server Kernel Programmer's Guide and Reference*, SC24-6220
- *z/VM: REXX/VM Reference*, SC24-6221
- *z/VM: REXX/VM User's Guide*, SC24-6222
- *z/VM: Systems Management Application Programming*, SC24-6234
- *z/VM: TCP/IP Programmer's Reference*, SC24-6239
- *Common Programming Interface Communications Reference*, SC26-4399
- *Common Programming Interface Resource Recovery Reference*, SC31-6821
- *z/OS: IBM Tivoli Directory Server Plug-in Reference for z/OS*, SA76-0148
- *z/OS: Language Environment Concepts Guide*, SA22-7567
- *z/OS: Language Environment Debugging Guide*, GA22-7560
- *z/OS: Language Environment Programming Guide*, SA22-7561
- *z/OS: Language Environment Programming Reference*, SA22-7562
- *z/OS: Language Environment Run-Time Messages*, SA22-7566
- *z/OS: Language Environment Writing Interlanguage Communication Applications*, SA22-7563
- *z/OS MVS Program Management: Advanced Facilities*, SA22-7644
- *z/OS MVS Program Management: User's Guide and Reference*, SA22-7643

Diagnosis

- *z/VM: CMS and REXX/VM Messages and Codes*, GC24-6161
- *z/VM: CP Messages and Codes*, GC24-6177
- *z/VM: Diagnosis Guide*, GC24-6187
- *z/VM: Dump Viewing Facility*, GC24-6191
- *z/VM: Other Components Messages and Codes*, GC24-6207
- *z/VM: TCP/IP Diagnosis Guide*, GC24-6235
- *z/VM: TCP/IP Messages and Codes*, GC24-6237
- *z/VM: VM Dump Tool*, GC24-6242
- *z/OS and z/VM: Hardware Configuration Definition Messages*, SC33-7986

z/VM Facilities and Features

Data Facility Storage Management Subsystem for VM

- *z/VM: DFSMS/VM Customization*, SC24-6181
- *z/VM: DFSMS/VM Diagnosis Guide*, GC24-6182
- *z/VM: DFSMS/VM Messages and Codes*, GC24-6183
- *z/VM: DFSMS/VM Planning Guide*, SC24-6184
- *z/VM: DFSMS/VM Removable Media Services*, SC24-6185
- *z/VM: DFSMS/VM Storage Administration*, SC24-6186

Directory Maintenance Facility for z/VM

- *z/VM: Directory Maintenance Facility Commands Reference*, SC24-6188
- *z/VM: Directory Maintenance Facility Messages*, GC24-6189
- *z/VM: Directory Maintenance Facility Tailoring and Administration Guide*, SC24-6190

Open Systems Adapter/Support Facility

- *zEnterprise System, System z10, System z9 and eServer zSeries: Open Systems Adapter-Express Customer's Guide and Reference*, SA22-7935
- *System z9 and eServer zSeries 890 and 990: Open Systems Adapter-Express Integrated Console Controller User's Guide*, SA22-7990
- *System z: Open Systems Adapter-Express Integrated Console Controller 3215 Support*, SA23-2247
- *System z10: Open Systems Adapter-Express3 Integrated Console Controller Dual-Port User's Guide*, SA23-2266

Performance Toolkit for VM

- *z/VM: Performance Toolkit Guide*, SC24-6209
- *z/VM: Performance Toolkit Reference*, SC24-6210

RACF Security Server for z/VM

- *z/VM: RACF Security Server Auditor's Guide*, SC24-6212
- *z/VM: RACF Security Server Command Language Reference*, SC24-6213
- *z/VM: RACF Security Server Diagnosis Guide*, GC24-6214
- *z/VM: RACF Security Server General User's Guide*, SC24-6215
- *z/VM: RACF Security Server Macros and Interfaces*, SC24-6216
- *z/VM: RACF Security Server Messages and Codes*, GC24-6217
- *z/VM: RACF Security Server Security Administrator's Guide*, SC24-6218
- *z/VM: RACF Security Server System Programmer's Guide*, SC24-6219
- *z/VM: Security Server RACROUTE Macro Reference*, SC24-6231

Remote Spooling Communications Subsystem Networking for z/VM

- *z/VM: RSCS Networking Diagnosis*, GC24-6223
- *z/VM: RSCS Networking Exit Customization*, SC24-6224
- *z/VM: RSCS Networking Messages and Codes*, GC24-6225
- *z/VM: RSCS Networking Operation and Use*, SC24-6226
- *z/VM: RSCS Networking Planning and Configuration*, SC24-6227

Prerequisite Products

Device Support Facilities

- *Device Support Facilities: User's Guide and Reference*, GC35-0033

Environmental Record Editing and Printing Program

- *Environmental Record Editing and Printing Program (EREP): Reference*, GC35-0152
- *Environmental Record Editing and Printing Program (EREP): User's Guide*, GC35-0151

Index

© Copyright IBM Corp. 2001, 2013

D

E

F

G

H

I

L

M

N

O

P

R

S

T

U

V

W

X

Product Number: 5741-A07

Printed in USA

SC24-6211-03