

IBM Configuration Management Version Control

SC09-1635-01

**Commands Reference**

Version 2 Release 2





## **Commands Reference**

Version 2 Release 2

**Note!**

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

**Second Edition (Dec 1993)**

This edition applies to Version 2, Release 2, Modification Level 0, of IBM Configuration Management Version Control/6000 (Program 5765-207), IBM Configuration Management Version Control for HP systems (Program 5765-202), IBM Configuration Management Version Control for Sun systems (Program 5622-063), and to all subsequent releases and modifications until otherwise indicated in new editions. Make sure you are using the correct edition for the level of the product.

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

A form for readers' comments is provided at the back of this publication. If the form has been removed, address your comments to:

IBM Canada Ltd. Laboratory  
Information Development  
2G/345/1150/TOR  
1150 Eglinton Avenue East  
North York, Ontario, Canada. M3C 1H7

You can also send your comments by facsimile (attention: RCF Coordinator), or you can send your comments electronically to IBM. See "Communicating Your Comments to IBM" for a description of the methods. This page immediately precedes the Readers' Comment Form at the back of this publication.

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

© **Copyright International Business Machines Corporation 1993. All rights reserved.**

Note to U.S. Government Users — Documentation related to restricted rights — Use, duplication or disclosure is subject to restrictions set forth in GSA ADP Schedule Contract with IBM Corp.

---

# Contents

<b>Notices</b> . . . . .	ix
Trademarks and Service Marks . . . . .	ix
<b>About This Book</b> . . . . .	xi
Who Should Read This Book . . . . .	xi
What You Should Know . . . . .	xi
How To Use This Book . . . . .	xi
Highlighting Conventions . . . . .	xii
Related Publications . . . . .	xii
<b>Changes and Additions for CMVC Version 2</b> . . . . .	xv
Changes of Command Syntax for OS/2 Client . . . . .	xvi
<b>Chapter 1. General Command Information</b> . . . . .	1
Flags . . . . .	2
Action Flags . . . . .	2
Attribute Flags . . . . .	2
Flag Arguments . . . . .	2
Using Standard Input for Arguments . . . . .	4
Environment Variables . . . . .	4
Methods of Setting Environment Variables . . . . .	6
Authority Requirements . . . . .	6
Base Authority . . . . .	6
Superuser Privilege . . . . .	6
Implicit Authority . . . . .	7
Explicit Authority . . . . .	7
Restricted Authority . . . . .	7
How to Read Syntax Statements . . . . .	8
A Note on Examples . . . . .	8
<b>Chapter 2. Access</b> . . . . .	9
Syntax . . . . .	9
Action Flags . . . . .	9
Attribute Flags . . . . .	10
Examples . . . . .	10
Related Information . . . . .	11
<b>Chapter 3. Approval</b> . . . . .	13
Syntax . . . . .	13
Action Flags . . . . .	14
Attribute Flags . . . . .	14
Examples . . . . .	15
Related Information . . . . .	16
<b>Chapter 4. Approver</b> . . . . .	17
Syntax . . . . .	17
Action Flags . . . . .	17
Attribute Flags . . . . .	18
Examples . . . . .	18
Related Information . . . . .	18

<b>Chapter 5. Component (Componen)</b>	19
Syntax	20
Action Flags	20
Attribute Flags	21
Examples	21
Related Information	22
<b>Chapter 6. Coreq</b>	23
Syntax	23
Action Flags	23
Attribute Flags	24
Examples	24
Related Information	24
<b>Chapter 7. Defect</b>	25
Syntax	25
Action Flags	26
Attribute Flags	28
Examples	29
Related Information	31
<b>Chapter 8. Environment (Environ)</b>	33
Syntax	33
Action Flags	33
Attribute Flags	34
Examples	34
Related Information	35
<b>Chapter 9. Feature</b>	37
Syntax	37
Action Flags	38
Attribute Flags	40
Examples	41
Related Information	42
<b>Chapter 10. File</b>	43
Syntax	44
Action Flags	45
Attribute Flags	47
Examples	49
Related Information	51
<b>Chapter 11. Fix</b>	55
Syntax	55
Action Flags	56
Attribute Flags	57
Examples	57
Related Information	58
<b>Chapter 12. Host (Hostcmd)</b>	59
Syntax	59
Action Flags	59
Attribute Flags	60
Examples	60

Related Information . . . . .	61
<b>Chapter 13. Level</b> . . . . .	63
Syntax . . . . .	64
Action Flags . . . . .	64
Attribute Flags . . . . .	65
Examples . . . . .	67
Related Information . . . . .	68
<b>Chapter 14. LevelMember (Levelmem)</b> . . . . .	69
Syntax . . . . .	69
Action Flags . . . . .	69
Attribute Flags . . . . .	70
Examples . . . . .	70
Related Information . . . . .	70
<b>Chapter 15. Migrate</b> . . . . .	73
Syntax . . . . .	73
Action Flags . . . . .	74
Attribute Flags . . . . .	74
Examples . . . . .	75
Related Information . . . . .	75
<b>Chapter 16. Notify</b> . . . . .	77
Syntax . . . . .	77
Action Flags . . . . .	77
Attribute Flags . . . . .	78
Examples . . . . .	78
Related Information . . . . .	79
<b>Chapter 17. Release</b> . . . . .	81
Syntax . . . . .	82
Action Flags . . . . .	82
Attribute Flags . . . . .	83
Examples . . . . .	84
Related Information . . . . .	85
<b>Chapter 18. Report</b> . . . . .	87
Syntax . . . . .	87
Action Flags . . . . .	88
Attribute Flags . . . . .	88
Examples . . . . .	89
Related Information . . . . .	92
<b>Chapter 19. Size</b> . . . . .	95
Syntax . . . . .	95
Action Flags . . . . .	95
Attribute Flags . . . . .	96
Examples . . . . .	97
Related Information . . . . .	97
<b>Chapter 20. Test</b> . . . . .	99
Syntax . . . . .	99
Action Flags . . . . .	99

Attribute Flags . . . . .	100
Examples . . . . .	100
Related Information . . . . .	101
<b>Chapter 21. Track . . . . .</b>	<b>103</b>
Syntax . . . . .	103
Action Flags . . . . .	104
Attribute Flags . . . . .	105
Examples . . . . .	106
Related Information . . . . .	106
<b>Chapter 22. User . . . . .</b>	<b>107</b>
Syntax . . . . .	107
Action Flags . . . . .	107
Attribute Flags . . . . .	108
Examples . . . . .	108
Related Information . . . . .	109
<b>Chapter 23. Verify (Verifycm) . . . . .</b>	<b>111</b>
Syntax . . . . .	111
Action Flags . . . . .	111
Attribute Flags . . . . .	112
Examples . . . . .	112
Related Information . . . . .	113
<b>Appendix A. Report-Raw Output . . . . .</b>	<b>115</b>
AccessDownView . . . . .	115
AccessView . . . . .	115
ApprovalView . . . . .	116
ApproverView . . . . .	116
Authority . . . . .	116
Cfgcomproc . . . . .	117
Cfgrelproc . . . . .	117
ChangeView . . . . .	117
CompView . . . . .	117
Config . . . . .	118
DefectDownView . . . . .	118
DefectView . . . . .	119
EnvView . . . . .	120
FeatureDownView . . . . .	120
FeatureView . . . . .	121
FileView . . . . .	121
FilesOutView . . . . .	122
FixView . . . . .	122
HostView . . . . .	123
Interest . . . . .	123
LevelMemberView . . . . .	123
LevelView . . . . .	123
NoteView . . . . .	124
NotifyDownView . . . . .	124
NotifyUpView . . . . .	124
NotifyView . . . . .	125
ReleaseView . . . . .	125
SizeView . . . . .	125



TestView	126
TrackView	126
Users	127
VerifyView	127
<b>Glossary</b>	<b>129</b>



---

## Notices

References in this publication to IBM products, programs, or services do not imply that IBM intends to make these available in all countries in which IBM operates. Any reference to an IBM licensed program in this publication is not intended to state or imply that only IBM's licensed program may be used. Any functionally equivalent product, program or service that does not infringe any of IBM's intellectual property rights may be used instead of the IBM product, program, or service. Evaluation and verification of operation in conjunction with other products, except those expressly designated by IBM, is the user's responsibility.

IBM may have patents or pending patent applications covering subject matter in this document. The furnishing of this document does not give you any license to these patents. You can send license inquiries, in writing, to the IBM Director of Commercial Relations, IBM Corporation, Purchase, NY 10577, USA.

---

## Trademarks and Service Marks

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

AIX	IBM	IBMLink
Operating System/2	OS/2	PROFS

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

HP	Hewlett-Packard Company
INFORMIX	Informix Software, Inc.
Network File System	Sun Microsystems, Inc.
NFS	Sun Microsystems, Inc.
ORACLE	Oracle Corporation
OSF/Motif	Open Software Foundation, Inc.
PVCS Version Manager	INTERSOLV, Inc.
SoftBench	Hewlett-Packard Company
Sun	Sun Microsystems, Inc.
SYBASE	Sybase, Inc.



---

## About This Book

This book is part of the documentation library supporting the IBM\* Configuration Management Version Control (CMVC) licensed programs. It describes all of the CMVC *commands* you can use from the command line interface.

---

## Who Should Read This Book

All CMVC *users* who want to perform tasks by entering commands from a standard shell should read this book. These users include those who want to write shell scripts using CMVC commands, those who cannot access the CMVC graphical user interface (*GUI*) program because they work on ASCII (nongraphics) terminals, and those who want to use the command line interface as a fast path alternative to the GUI.

---

## What You Should Know

You should read the book *IBM CMVC Concepts*, SC09-1633, before you use CMVC. It introduces the fundamentals of the *configuration management*, *version control*, *change control*, and *problem tracking* features in the CMVC licensed programs. It also defines the concepts that are the foundation of CMVC *actions* and establishes their interrelationships.

You should be familiar with your operating system because you access the CMVC licensed programs through that *environment*.

---

## How To Use This Book

Read Chapter 1, “General Command Information” on page 1 for an overview of the CMVC commands you can issue from the command line interface and an explanation of the command syntax.

If you have used a previous version of CMVC, read “Changes and Additions for CMVC Version 2” on page xv to see the changes and additions made to CMVC commands.

The remaining chapters describe the commands in detail. Each chapter describes one command and includes:

- Description of the command and an overview of its purpose
- Syntax statements of the command, one statement per action flag
- Action flags you can use with the command
- Attribute flags that apply to the command
- Examples of CMVC commands
- Related information.

The appendix describes the fields for various CMVC *views* and *tables*.

A glossary is provided at the back of this book.

---

## Highlighting Conventions

The style conventions that are used in CMVC command syntax are described in “How to Read Syntax Statements” on page 8. The following highlighting conventions are also used in this book:

<b>Bold</b>	Commands, flags, files, directories, field names, and other items predefined by CMVC appear in bold. Valid abbreviations for commands are also in bold.
<i>Italic</i>	Arguments or options whose names or values must be supplied by you appear in italics. Italics are also used for emphasis, for the first occurrence in text of terms that appear in the glossary, and for titles of books.
Monotype	Examples of specific data values, examples of text that you might see displayed, messages, or information that you should type appear in monotype.

---

## Related Publications

The following books contain additional information about CMVC and are shipped with this product.

- *IBM CMVC Concepts*, SC09-1633, provides the basis for your understanding of CMVC. It describes in detail the concepts and processes involved in using CMVC.
- *IBM CMVC Server Administration and Installation*, SC09-1631 contains detailed information for planning, installing, customizing, operating, and maintaining the CMVC server.
- *IBM CMVC User's Guide*, SC09-1634, describes all CMVC *actions* as implemented in the *graphical user interface (GUI)* on the AIX, Sun-OS, and HP-UX platforms.
- *IBM CMVC User's Reference*, SC09-1597, contains the reference lists, tables, and *state* diagrams for CMVC. It also describes how the message-integrated CMVC uses the *Broadcast Message Server (BMS)* to fully integrate with other integrated development environment tools.
- *IBM CMVC Client Installation and Configuration*, SC09-1596, contains detailed information needed to install and configure the CMVC client on (*GUI*) on the AIX\*, Sun-OS\*\*, and HP-UX\*\* platforms.
- *NetLS Quick Start Guide*, SC09-1661, provides the information needed to set up the Network License System (NetLS) software to work with CMVC.
- *Managing Software Products with the Network License System*, SC09-1660, provides the information needed to manage the use of the NetLS software with CMVC.

These two books are shipped with the OS/2 workstation client for CMVC, and can be ordered separately:

- *IBM CMVC Client/2 Getting Started*, SC09-1599, contains detailed information about installing and configuring the OS/2 workstation client for CMVC.

- *IBM CMVC Client/2 User's Guide*, SC09-1783, contains step-by-step information on how to use the graphical user interface for the OS/2 workstation client.

For information on *databases* and operating systems used with CMVC, refer to your specific database or operating system documentation.





## Changes and Additions for CMVC Version 2

The table below shows those commands that have changed between CMVC Versions 1 and 2, and those commands that have been added to CMVC Version 2.

Version 1 Commands	Version 2 Additions/Changes
Access -delete	Added new flag: -inherited
Not applicable	Access -restrict {-login Name ...   -inherited} -authority Name -component Name -family Name -become Name [-verbose]
Component -create	Add new mandatory flag: -process Name
Component -modify	Add optional flag: -process Name
Component -view	Add optional flag: -processInfo
Not applicable	Defect -configInfo -raw -family Name [-become Name]
Not applicable	Defect -design Number ... [-remarks Text] -verbose -family Name [-become Name]
Defect -modify	Add optional flag: -name Number
Defect -open	Add optional flag: -name Number
Not applicable	Defect -review Number ... [-remarks Text] [-verbose] -family Name [-become Name]
Not applicable	Defect -size Number ... [-remarks Text] [-verbose] -family Name [-become Name]
Defect -view	Add optional flag: -processInfo
Not applicable	Feature -configInfo -raw -family Name [-become Name]
Feature -modify	Add optional flag: -name Number
Feature -open	Add optional flag: -name Number
Feature -view	Add optional flag: -processInfo
Not applicable	File -configInfo -raw -family Name [-become Name]
File -create	Add optional flag: -fmode Octal_number
File -modify	Add optional flag: -fmode Octal_number
Level -assign	Supports multiple level names
Level -check	Supports multiple level names
Level -commit	Supports multiple level names
Level -complete	Supports multiple level names
Level -create	Supports multiple level names
Level -delete	Supports multiple level names
Level -extract	Supports multiple level names. -node and -root are now required.
Level -modify	Supports multiple level names.
Level -view	Supports multiple level names.
Release -create	Supports multiple level names. +/-binding flags no longer supported. Add required flag: -process Name Add optional flags: [-approver Name] [-environment Name -tester Name]

<b>Version 1 Commands</b>	<b>Version 2 Additions/Changes</b>
Release -delete	Supports multiple level names.
Release -extract	Supports multiple level names.
Not applicable	Release -link Name ... -to Name -family Name [-date yy/mm/dd   -committed] [-defect Number ...   -feature Number ... ] [-become Name] [-verbose]
Release -modify	Supports multiple release names. Add optional flags: -process Name [-approver Name] [-environment Name -tester Name]
Release -recreate	Supports multiple release names. Add optional flags: [-approver Name] [-environment Name -tester Name]
Release -view	Supports multiple release names. Add optional flags: -processInfo
Not applicable	User -configInfo -raw -family Name [-become Name]

---

## Changes of Command Syntax for OS/2 Client

The following table lists the five commands that have been changed for the OS/2 client for CMVC, together with the standard CMVC commands:

<b>Standard CMVC Command</b>	<b>OS/2 Client Command</b>
Component	Componen
Environment	Environ
Host	Hostcmd
LevelMember	LevelMem
Verify	Verifycm

---

## Chapter 1. General Command Information

This chapter introduces all the CMVC commands that you can issue from the command line. The executable programs for these commands are installed in the **/usr/lpp/cmvc/bin** directory when you install the CMVC client code. For information about installation, see the book *IBM CMVC Client Installation and Configuration*. This chapter also outlines the *authority* required to issue commands and tells you how to use action and attribute flags with the commands, how to use CMVC environment variables, and how to read the command syntax found in this book.

The purpose of each CMVC command is shown in Figure 1.

---

Command	Purpose
<b>Access</b>	Identifies <i>explicit authority</i> for user IDs using component <i>access lists</i> .
<b>Approval</b>	Records approvers' opinions about proposed changes in a <i>release</i> on <i>approval records</i> .
<b>Approver</b>	Specifies <i>approvers</i> of changes for releases using <i>approver lists</i> .
<b>Component</b>	Creates and maintains a <i>component</i> hierarchy for project control and management.
<b>Coreq</b>	Identifies <i>tracks</i> as corequisites, that is, tracks that must be included in the same <i>level</i> .
<b>Defect</b>	Monitors the reporting, evaluation, and resolution of problems.
<b>Environment</b>	Specifies environments and <i>testers</i> for releases using <i>environment lists</i> .
<b>Feature</b>	Monitors the suggestion, evaluation, and implementation of design changes and enhancements.
<b>File</b>	Places <i>files</i> in the CMVC environment and allows users to work with them.
<b>Fix</b>	Monitors the status of file changes (fixes) made for a component using <i>fix records</i> .
<b>Host</b>	Identifies client access on the <i>host list</i> associated with a user ID.
<b>Level</b>	Defines and works with levels of file changes within a release.
<b>LevelMember</b>	Identifies tracks that must be included in or deleted from a level.
<b>Migrate</b>	Migrates all versions of SCCS text files into the CMVC environment.
<b>Notify</b>	Identifies notification interest for user IDs using component <i>notification lists</i> .
<b>Release</b>	Creates and maintains releases to group project-related files.
<b>Report</b>	Searches database tables for information on CMVC objects.
<b>Size</b>	Records sizing information for <i>defects</i> and <i>features</i> using <i>sizing records</i> .
<b>Test</b>	Records testers' opinions about test results using environment <i>test records</i> .
<b>Track</b>	Creates and maintains tracks to monitor the progress of changes in a release.
<b>User</b>	Creates user IDs and maintains information about the <i>owners</i> .
<b>Verify</b>	Indicates the outcome of defects and features using verification records.

---

Figure 1. Summary of CMVC Commands

---

## Flags

Two types of flags are associated with commands: action flags and attribute flags. You can type the names of flags in any order on the command line.

A flag is a negative (-) or a positive (+) symbol followed by a lowercase word on the command line. The symbols associated with each flag are not interchangeable.

You can abbreviate both action and attribute flags; however, the number of letters required to make a flag unique within a command depends on the names of all of the other flags (both action and attribute flags) associated with that command. Valid abbreviations for all flags appear in bold within their related command's action and attribute flag tables.

## Action Flags

Every command has action flags associated with it. These action flags represent the actions that you can perform for a command. When you use the command line to perform a CMVC action, you must specify one command and only one action flag. You do not have to type the action flag directly after the command.

For example, you can perform six actions using the **User** command. Each of these tasks requires one of the following action flags:

<b>-configInfo</b>	Displays configurable field properties for users
<b>-create</b>	Creates a new user ID
<b>-delete</b>	Deletes an existing user ID
<b>-recreate</b>	Recreates a previously deleted user ID
<b>-modify</b>	Changes information related to a user ID
<b>-view</b>	Displays current information for a user ID

## Attribute Flags

Some action flags have mandatory attribute flags associated with them; others have optional attribute flags.

For example, the **-login** and **-address** attribute flags are mandatory when you use the **-create** action flag for the **User** command. The other attribute flags are optional.

```
User -create -login billyb -address williams@vroom1 -name "William Bronson"
-area Dept450 +super
```

You get the same results if you rearrange the order of the flags and abbreviate some of them.

```
User -login billyb -name "William Bronson" -ad williams@vroom1 -create -ar
Dept450 +super
```

Syntax indicates the attribute flags that are mandatory.

## Flag Arguments

In most cases, you have to type additional information for an action or attribute flag. This additional information is an argument. The seven types of arguments, their format and their restrictions, are listed in Figure 2 on page 3.

Argument	Format	Example	Restrictions
<i>Date</i>	yy/mm/dd	93/04/29	You must use numbers separated by slashes. Blanks are not permitted.
<i>Name</i>	One alphanumeric string	42tool	You cannot use blanks, vertical bars ( ), or ASCII control characters. Nor can you use shell metacharacters unless they are quoted. <sup>1</sup>
<i>Name ...</i>	One or more alphanumeric strings	prod1 prod2 prod3	You cannot use vertical bars ( ), or ASCII control characters. Nor can you use shell metacharacters unless they are quoted. <sup>1</sup> Blanks are permitted to separate unique strings.
<i>Number</i>	Numeric string <sup>2</sup>	823	You must use numbers. Blanks are not permitted.
<i>Number ...</i>	One or more numeric strings <sup>3</sup>	411 1124 1 362	You must use numbers. Blanks are permitted to separate unique strings.
<i>Octal_Number</i>	Numeric string	750	You must use numbers from 0 to 7. Blanks are not permitted.
<i>Text</i>	Alphanumeric strings enclosed in quotations	"Not able to verify."	You cannot use vertical bars ( ), or ASCII control characters. Nor can you use shell metacharacters unless they are quoted. <sup>1</sup>

<sup>1</sup> For information on ASCII control characters and the shell you are using, refer to your operating system documentation.

<sup>2</sup> When used with a **-defect** or **-feature** flag, an alphanumeric string is acceptable and, consequently, the restrictions for *Name* apply.

<sup>3</sup> When used with a **-defect** or **-feature** flag, one or more alphanumeric strings are acceptable and, consequently, the restrictions for *Name ...* apply.

Figure 2. Flag Arguments and Syntax

If you specify a list of arguments for more than one flag, the action is performed for every possible combination of arguments. For example:

```
Track -create -defect 1 2 -release one two
```

creates four tracks, as shown in Figure 3.

Tracks	Defect	Release
1)	1	one
2)	1	two
3)	2	one
4)	2	two

Figure 3. Creating CMVC Tracks

One track is referenced by defect 1 and release one, another by defect 1 and release two, another by defect 2 and release one, and another by defect 2 and release two.

## Using Standard Input for Arguments

To specify an argument using standard input, use "-" as the argument type. You can specify only one flag per command in this way. In the following example of standard input from a keyboard, you can type the remarks flag argument directly from the keyboard:

```
Defect -open -component debugr -sev 3 -remarks -
```

Press Enter to create additional lines on which to type the text. When you are finished entering the text, press Enter to create a new line and then press Ctrl D to end standard input.

In the following example of standard input from a file, the **-remarks** argument is equivalent to the contents of the file you specified:

```
Defect -open -component debugr -sev 3 -remarks - < /tmp/defect.descr
```

**Note:** The shell interprets environment variables when used in arguments, including text. For example, in `-remarks "The sun is $path"`, the value of `$path` is substituted.

---

## Environment Variables

You can set environment variables to describe the CMVC environment in which you are working. The names of the CMVC environment variables, the purpose they serve, and the CMVC flag that is the equivalent to each environment variable are listed in Figure 4 on page 5.

Make sure you set your `CMVC_FAMILY` environment variable because this information is required with every command.

Environment Variable	Purpose	Flag
CMVC_FAMILY	Identifies the CMVC <i>family</i> you are working with.	<b>-family</b>
CMVC_BECOME	Identifies the user ID you want to issue CMVC commands from, if the user ID differs from your <i>login</i> . You assume the access authority of the user ID you specify.	<b>-become</b>
CMVC_RELEASE	Specifies a release.	<b>-release</b>
CMVC_COMPONENT	Specifies a component.	<b>-component</b>
CMVC_TOKEN_EXPIRY	Specifies the expiry time for a NetLS token.	<b>Not applicable</b>
CMVC_TOP	Specifies a directory prefix.	<b>-top</b>

Figure 4. CMVC Environment Variables and Flags

You can override the value you set for an environment variable by using the corresponding flag. In Figure 5, an X indicates the CMVC commands that use the CMVC environment variable values.

	CMVC_FAMILY	CMVC_RELEASE	CMVC_COMPONENT	CMVC_BECOME	CMVC_TOP
<b>Access</b>	X		X	X	
<b>Approval</b>	X	X		X	
<b>Approver</b>	X	X		X	
<b>Component</b>	X			X	
<b>Coreq</b>	X	X		X	
<b>Defect</b>	X		X	X	
<b>Environment</b>	X	X		X	
<b>Feature</b>	X		X	X	
<b>File</b>	X	X	X	X	X
<b>Fix</b>	X	X	X	X	
<b>Host</b>	X			X	
<b>Level</b>	X	X		X	
<b>LevelMember</b>	X	X		X	
<b>Migrate</b>	X	X	X	X	X
<b>Notify</b>	X		X	X	
<b>Release</b>	X		X	X	
<b>Report</b>	X			X	
<b>Size</b>	X	X	X	X	

Figure 5 (Part 1 of 2). CMVC Commands and Related Environment Variables

	CMVC_FAMILY	CMVC_RELEASE	CMVC_COMPONENT	CMVC_BECOME	CMVC_TOP
<b>Test</b>	X	X		X	
<b>Track</b>	X	X		X	
<b>User</b>	X			X	
<b>Verify</b>	X			X	

Figure 5 (Part 2 of 2). CMVC Commands and Related Environment Variables

## Methods of Setting Environment Variables

For methods of setting your environment variables, refer to your operating system documentation.

For example, you can use the following command to set the `CMVC_FAMILY` environment variable using the Korn shell:

```
export CMVC_FAMILY=familyName@hostname@portnumber
```

Where `hostname` is the name of the server host for the family, and `portnumber` is the TCP/IP port number assigned to the family. If you do not specify the port number, it is obtained from the system configuration database (for example, the file `/etc/services`). If you do not specify the server host, the family name resolves to the network address of the server either in the file `/etc/hosts` or in the host database (that is, the nameserver).

## Authority Requirements

Different authority requirements are attached to each of the actions in CMVC. Five types of authority control the actions you can or cannot perform.

### Base Authority

If you have a valid CMVC user ID, you can perform these unrestricted *base authority* actions:

- Open defects and features
- Modify the information associated with your user ID
- View information associated with any user ID
- Add notes to existing defects or features
- Generate reports

### Superuser Privilege

If you have been granted CMVC *superuser privilege* by a *family administrator* or someone else with superuser privilege, you can perform all possible actions in your CMVC family. There are five actions that only a superuser can do: create a user ID, create the first host list entry for a user ID, delete a user ID, recreate a user ID, and grant superuser privilege to a user ID.

**Note:** Superuser privilege is not related to the operating system superuser classification.



## Implicit Authority

*Implicit authority* allows you to perform actions on the basis of ownership. For example, if you open a defect, you become the *originator* of the defect and have the implicit ability to perform certain actions, such as, canceling the defect or verifying its outcome. Similarly, if you own a component, a release, or a feature, you have implicit authority related specifically to those roles as well. You have this implicit authority until you relinquish ownership of the object in question.

## Explicit Authority

Explicit access authority is specified for you for a component. Granting explicit authority provides control over who can perform specific actions for the component.

Your family administrator can group sets of actions according to access authority groups. You are assigned to one or more of these groups by a component owner. The authority groups you belong to for any given component are inherited for all descendant components unless they are restricted.

For specific actions used to create access authority groups, refer to the *IBM CMVC User's Reference*. For a list of the preconfigured authority groups shipped with this product, which your family administrator may choose to use, refer to the *IBM CMVC User's Reference*.

## Restricted Authority

You can be restricted from performing certain actions at a specific component by someone with AccessRestrict authority who wishes to control which users inherit authority from the *parent components*. The authority can be restricted for specific users with the authority group or for all users with the group. CMVC notifies you if your specific authority is restricted or if you are a subscriber of the AccessRestrict action for the component. If all inherited users are restricted for the component, then only those users who have subscribed to the AccessRestrict action are notified.

**Note:** *Restricted authority* is not inherited and does not affect implicit authority or superuser privilege.

When an action flag is described in this book, the authority required to perform the action is shown in the columns next to it. If explicit authority is required, the table indicates the specific action that must be included in your access authority group. Superuser privilege will be listed in the explicit authority column if only a superuser can perform that action.

For more information on authority requirements, refer to the book *IBM CMVC Concepts*.

---

## How to Read Syntax Statements

This manual uses the braces and brackets representation of command syntax. The style conventions listed in Figure 6 apply to the command syntax.

Style	Usage
<b>bold</b>	Items you must enter exactly as shown, such as commands, flags, and symbols. The valid abbreviations for commands and flags appear in bold, with the optional part of the command or flag shown in regular type. For example, <b>Access</b> or <b>+super</b> .
<i>Italic</i>	Arguments or options whose values you must supply. For example, <i>Name</i> or <i>Text</i> .
...	Parameter can be repeated on the command line. For example, <b>-login Name...</b> means that you can enter more than one argument for the <b>-login</b> flag.
[ ]	Optional parameters are enclosed in square brackets. For example, [ <b>-description</b> <i>Text</i> ].
{ }	There is more than one parameter choice, but one is required.
	Choose one parameter only. [a b] indicates that you can choose a or b, or neither a nor b. {a b} indicates that you must choose either a or b.
-	Standard input. See page 4 for discussion of using this method.

Figure 6. Reading Syntax Statements

---

## A Note on Examples

- No **-family** flag is shown. All examples assume that you have set a value for your CMVC\_FAMILY environment variable.
- All examples assume that information is entered on a single line. Use a backslash ( \ ) if you require more than one line.
- Some flags are abbreviated; however, most examples contain the full flag name for clarity.

---

## Chapter 2. Access

Use the **Access** command to create entries on the component access list, *delete* entries from this list and restrict authority for entries (including those normally inherited from ancestor components) on the list. Each entry associates a user ID and a preconfigured access authority group. The authority group specifies the set of actions a user ID has the authority to perform in relation to the component. For details of the access authority groups shipped with CMVC, refer to the *IBM CMVC User's Reference*. Existing authority groups can be modified and new ones can be defined by a family administrator.

A user ID can have more than one entry on the access list for a given component. A user ID inherits explicit authority from all ascendant components for that component and has the accumulation or superset of all authority groups defined for those ascendant components, unless the authority is restricted.

You cannot grant another user any access authority that is not defined for your own user ID.

### Syntax

The syntax statements for the **Access** command are:

```
Access -create    -login Name ... -authority Name -component Name
                  -family Name [-become Name] [-verbose]

Access -delete   {-login Name ... | -inherited } -authority Name
                  -component Name -family Name [-become Name]
                  [-verbose]

Access -restrict {-login Name ... | -inherited } -authority Name
                  -component Name -family Name [-become Name]
                  [-verbose]
```

### Action Flags

The action flags of the **Access** command and their required authority are listed in Figure 7.

Action Flag	Purpose	Implicit Authority	Explicit Authority
-create	Adds entries to a component access list.	Component owner	AccessCreate
-delete	Deletes entries from a component list or deletes the restriction on authority for entries on a component access list.	Component owner	AccessDelete
-restrict	Restricts authority (including inherited authority) for entries on a component access list.	Component owner	AccessRestrict

Figure 7. Access Action Flags

## Attribute Flags

The attribute flags of the **Access** command are listed in Figure 8.

Attribute Flag and Argument	Purpose
-authority <i>Name</i>	Specifies a preconfigured access authority group for the user ID.
-become <i>Name</i>	Specifies the CMVC user ID to validate your authority to perform this action, only if the CMVC user ID differs from your login. (Environment Variable: CMVC_BECOME)
-component <i>Name</i>	Specifies the component associated with the access list. (Environment Variable: CMVC_COMPONENT)
-family <i>Name</i>	Specifies the family for which this command is being issued. (Environment Variable: CMVC_FAMILY)
-inherited	Indicates all users in the CMVC family.
-login <i>Name ...</i>	Specifies one or more CMVC user IDs.
-verbose	Indicates that you want to see a confirmation message after you issue this command.

Figure 8. Access Attribute Flags

## Examples

The following are examples of **Access** command actions:

1. To give developer access authority to CMVC user ID maria, for the graphix component and all its descendents, type:

```
Access -create -login maria -authority developer -component graphix
```

This creates an entry on the access list associated with the graphix component giving user ID maria the authority to perform all actions included in the developer access authority group.

2. To give writer access authority to CMVC user IDs maria and john for the graphix component and all its descendents, type:

```
Access -create -login maria john -authority writer -component graphix
```

This creates two entries on the access list associated with the graphix component.

3. To remove writer access from user ID maria for the graphix component, type:

```
Access -delete -login maria -authority writer -component graphix
```

This deletes the entry from the access list.

4. To restrict the actions in the access authority group developer+ for the CMVC user ID richard in the graphix component, type:

```
Access -restrict -login richard -authority developer+ -component graphix
```

This creates an entry on the access list associated with the graphix component restricting the user ID richard the authority from performing all actions included in the developer+ access authority group.

5. To restrict the actions in the access authority group `releaselead` for all users inheriting the `releaselead` access authority group from all parents of the `confidential` component, type:

```
Access -restrict -inherited -authority releaselead -component
confidential
```

This creates an entry on the access list associated with the `confidential` component restricting all users who inherited the `releaselead` access authority group from performing actions in the `releaselead` access authority group.

6. To remove the restricted authority group `releaselead` for all users inheriting this access authority group from all parents of the `confidential` component, type:

```
Access -delete -inherited -authority releaselead -component confidential
```

This deletes the entry from the access list and permits users with the `releaselead` authority in parent components of the `confidential` component to perform all actions in the `releaselead` access authority group at the `confidential` component.

## Related Information

See commands: **Component**, **Report**.

Use the **Report** command to obtain more information on existing authority groups:

- **Report -vi** authority
- **Report -vi** authority -w "name ='developer' "
- **Report -vi** authority -w "action ='FileCheckIn' "

For a list of the access authority groups shipped with CMVC, refer to the *IBM CMVC User's Reference*.

See your family administrator, or refer to the book *IBM CMVC Server Administration and Installation* for information about configuring new access authority groups and modifying existing ones.



---

## Chapter 3. Approval

Use the **Approval** command to record on approval records approvers' opinions about proposed changes to files in a release. You can only use this command for a track that is in the approve state. The **Approval** command provides greater control over changes made to releases as final deadlines approach.

Approval records are created automatically every time a track is created for a release that has an approver list. You can also create additional approval records for a track, with the **Approval** command, without changing the approver list associated with the release. For information on changing the approver list, refer to Chapter 4, "Approver" on page 17. You can also use the **Approval** command to delete track approval records or assign them to other users.

Owners of an approval record must indicate on it whether they accept or reject the changes proposed by the track. An abstain option is available.

The state of the approval record controls whether the associated track can move to the fix state. When all approval records are in the accept or abstain state, the track moves automatically to the fix state. If one or more approval records is in the reject state, the track cannot move to the fix state.

### Syntax

The syntax statements for the **Approval** command are:

**Approval -abstain -release** *Name ... -family Name [ -become Name ]*  
{ **-defect** *Number ... -feature Number ...* }  
[ **-approver** *Name* ] [ **-verbose** ]

**Approval -accept -release** *Name ... -family Name [ -become Name ]*  
{ **-defect** *Number ... -feature Number ...* }  
[ **-approver** *Name* ] [ **-verbose** ]

**Approval -assign -to** *Name -release Name ... -family Name [ -verbose ]*  
{ **-defect** *Number ... -feature Number ...* }  
[ **-approver** *Name* ] [ **-become** *Name* ]

**Approval -create -approver** *Name -release Name ... -family Name*  
{ **-defect** *Number ... -feature Number ...* }  
[ **-become** *Name* ] [ **-verbose** ]

**Approval -delete -approver** *Name -release Name ... -family Name*  
{ **-defect** *Number ... -feature Number ...* }  
[ **-become** *Name* ] [ **-verbose** ]

**Approval -reject -release** *Name ... -family Name [ -become Name ]*  
{ **-defect** *Number ... -feature Number ...* }  
[ **-approver** *Name* ] [ **-verbose** ]

## Action Flags

The action flags of the **Approval** command and their required authority are listed in Figure 9.

Action Flag	Purpose	Implicit Authority	Explicit Authority
<b>-abstain</b>	Abstains from accepting or rejecting the proposed file changes for the specified track.	Owner of approval record	ApprovalAbstain
<b>-accept</b>	Approves the proposed file changes for the specified track.	Owner of approval record	ApprovalAccept
<b>-assign</b>	Assigns an existing approval record to another user ID. The owner of that user ID becomes the owner of the approval record.	Owner of approval record	ApprovalAssign
<b>-create</b>	Creates an approval record for a track. This action does not change the approver list. <b>Note:</b> To perform this action, the associated release's <i>process</i> must include the approval <i>subprocess</i> .	Track owner	ApprovalCreate
<b>-delete</b>	Deletes an existing approval record for a specified user ID and track.	N/A	ApprovalDelete
<b>-reject</b>	Rejects the proposed changes for the specified track and keeps the track in the approve state. This prevents the track from moving to fix. Tracks that are not approved can be canceled.	Owner of approval record	ApprovalReject

Figure 9. Approval Action Flags

## Attribute Flags

The attribute flags of the **Approval** command are listed in Figure 10.

Attribute Flag and Argument	Purpose
<b>-approver</b> <i>Name</i>	Specifies the user ID of the owner of the approval record.
<b>-become</b> <i>Name</i>	Specifies the CMVC user ID to validate your authority to perform this action, only if the CMVC user ID differs from your login. (Environment Variable: CMVC_BECOME)
<b>-defect</b> <i>Number ...</i>	Specifies the defects for which proposed changes need to be approved. Each defect and release combination identify a track.
<b>-family</b> <i>Name</i>	Specifies the family for which this command is being issued. (Environment Variable: CMVC_FAMILY)
<b>-feature</b> <i>Number ...</i>	Specifies the features for which proposed changes must be approved. Each feature and release combination identifies a track.

Figure 10 (Part 1 of 2). Approval Attribute Flags



Attribute Flag and Argument	Purpose
<code>-release Name ...</code>	Specifies the releases in which work is being tracked for a defect or a feature. (Environment Variable: CMVC_RELEASE)
<code>-to Name</code>	Specifies the user ID to which you want to reassign the approval record. The user ID you specify becomes the owner of the approval record.
<code>-verbose</code>	Indicates that you want to see a confirmation message after you issue this command.

Figure 10 (Part 2 of 2). Approval Attribute Flags

## Examples

The following are examples of **Approval** command actions:

1. The track referencing defect 147 in release 20graphix is in the approve state. To create an approval record so that jack must approve the proposed changes for that track, type:

```
Approval -create -release 20graphix -defect 147 -approver jack
```

An approval record is created and its owner is jack. When you specify a value for the **-release** flag, any existing value set for the CMVC\_RELEASE environment variable is ignored. This action can only be done for releases that have an approver list, even though the action does not modify the list.

Releases without approver lists result in tracks being created with an initial state of fix and only when a track is in the approve state can approval records be created or acted upon.

2. You own an approval record that is in the ready state. It refers to the changes proposed for a track created to monitor work for feature 179 in the release specified in your CMVC\_RELEASE environment variable. To indicate that you approve of the proposed changes for feature 179, type:

```
Approval -accept -feature 179
```

The approval record moves to the accept state. You do not have to specify the **-release** flag because the CMVC\_RELEASE environment variable was set.

3. You have superuser privilege. To delete the approval record owned by maria for a track addressing feature 2431 in release 10graphix, type:

```
Approval -delete -release 10graphix -feature 2431 -approver maria
```

The approval record is deleted. The track must still be in the approve state for a superuser to be able to delete the approval record.

4. You own two approval records, one for the work required to resolve defect 9122 in release 10graphix and the other for the work required to resolve the same defect in release 20graphix. To assign both of these approval records to pam, type:

```
Approval -assign -release 10graphix 20graphix -defect 9122 -to pam
```

The approval records are now owned by pam.

## Related Information

See commands: **Approver**, **Defect**, **Feature**, **Release**, **Report**, **Track**.

---

## Chapter 4. Approver

Use the **Approver** command to create entries on, and delete entries from, a release approver list. Each entry associates a user ID with a release, making the owner of the user ID an approver for any proposed changes to address defects or features in the specified release. The release approver list provides greater control over changes made to releases as final deadlines approach.

Every time a track is created for a release to address a defect or a feature, approval records are created for each of the user IDs on the approver list associated with that release (providing that the release's process includes the approval subprocess). Each approval record refers to one defect or feature in one release and is owned by one approver. Approvers must use the **Approval** command to accept or reject the proposed changes. Modifying an approver list does not change existing approval records.

Approval records that are accepted allow the track to move to the fix state. If one or more approvers rejects an approval record, the track cannot move to the fix state.

### Syntax

The syntax statements for the **Approver** command are:

```
Approver -create -login Name ... -release Name -family Name  
[ -become Name ] [ -verbose ]
```

```
Approver -delete -login Name ... -release Name -family Name  
[ -become Name ] [ -verbose ]
```

### Action Flags

The action flags of the **Approver** command and their required authority are listed in Figure 11.

Action Flag	Purpose	Implicit Authority	Explicit Authority
-create	Adds user IDs to a release approver list.	Release owner	ApproverCreate
-delete	Deletes user IDs from a release approver list.	Release owner	ApproverDelete

**Note:** You cannot delete the last entry in an approver list if it is associated with a release whose process includes the approval subprocess.

Figure 11. Approver Action Flags

## Attribute Flags

The action flags of the **Approver** command are listed in Figure 12.

Attribute Flag and Argument	Purpose
-become <i>Name</i>	Specifies the CMVC user ID to validate your authority to perform this action, only if the CMVC user ID differs from your login. (Environment Variable: CMVC_BECOME)
-family <i>Name</i>	Specifies the family for which this command is being issued. (Environment Variable: CMVC_FAMILY)
-login <i>Name ...</i>	Identifies CMVC user IDs as members of an approver list.
-release <i>Name</i>	Specifies the release with which the approver list is associated. (Environment Variable: CMVC_RELEASE)
-verbose	Indicates that you want to see a confirmation message after you issue this command.

Figure 12. Approver Attribute Flags

## Examples

The following are examples of **Approver** command actions:

1. To make the owners of user IDs `jack` and `smitty` approvers for any changes that may be proposed for the `10debugr` release, type:

```
Approver -create -login jack smitty -release 10debugr
```

Two entries are added to the approver list for the `10debugr` release. Approval records are created for `jack` and `smitty` when new tracks are created in reference to the `10debugr` release.

2. To delete the approver list entry identifying the owner of user ID `maria` as an approver for the `tools` release, type:

```
Approver -delete -login maria -release tools
```

`Maria` is deleted from the approver list.

3. To add user IDs `maria`, `john`, and `kevin` to the approver list associated with the release you have specified in your `CMVC_RELEASE` environment variable, type:

```
Approver -create -login maria john kevin
```

Three entries are made to the approver list for the release set in your environment variable, one for each of the user IDs you specified.

## Related Information

See commands: **Approval**, **Defect**, **Feature**, **Release**, **Report**, **Track**.

---

## Chapter 5. Component (Componen)

Use the **Component** command (or the **Componen** command for the OS/2 client) to create and maintain a component structure for project control and management. The component structure or hierarchy consists of a top level component called root. Every component below root is linked to one or more parent components and zero or more *child components*. Use **-link** and **-unlink** to redefine an existing component structure.

You can create, delete, and recreate components, modify their properties, or view information about them.

When you create a component, you become its owner and have implicit authority to define the access list and the notification list for that component. Although you have implicit authority to define the access list, you cannot add to that list until you have some level of authority defined on the access list. Therefore, when you first become the owner of a component, someone with enough authority must give you authority to create access for additional users. When you become the owner of a component, you may want to ask the component creator to give you access authority so that you can add other users to the access list. The access and notification list entries for a component apply to all descendant components via *inheritance*, unless access has been specifically restricted. As component owner, you also have implicit authority to manage that component and other objects relating to it.

When creating a component, you must specify a preconfigured process for the component using the **-process** flag. A process groups different combinations of CMVC subprocesses. CMVC subprocesses determine the states that apply to the defects and features associated with a component. For component processes, the DSR (design, size, review) and verify subprocesses can be specified for defects, features, or both. Processes are configured by your family administrator who can modify current processes or define new ones. For a list of the valid component processes and the CMVC subprocesses they include, use the **Report -view cfgcomproc** command. You can change the process for an existing component using the **-modify** flag. For more information on how CMVC subprocesses relate to the states of CMVC objects, refer to the book *IBM CMVC Concepts*.

You can delete a component only if there are no files, child components, releases, active features, active defects, or active sizing records associated with it. The component's access and notification lists are deleted when it is deleted, and the component is detached from its parents. You cannot reuse the name of a deleted component to create another component; however, you can recreate a deleted component.

When you recreate a deleted component, you have to create new access and notification lists for it. The original access and notification lists are not recreated; however, the recreated component does inherit the access and notification information from all of the components above it in the hierarchy.

## Syntax

The syntax statements for the **Component** command are:

```

Component -create   Name ... -parent Name -family Name
                    -process Name [ -owner Name ]
                    [ -description Text ] [ -become Name ] [ -verbose ]

Component -delete Name ... -family Name [ -become Name ]
                    [ -verbose ]

Component -link   Name ... -parent Name -family Name [ -become Name ]
                    [ -verbose ]

Component -modify Name ... -family Name { -process Name
                    -owner Name -name Name -description Text }
                    [ -become Name ] [ -verbose ]

Component -recreate Name ... -parent Name -family Name [ -become Name ]
                    [ -verbose ]

Component -unlink Name ... -parent Name -family Name [ -become Name ]
                    [ -verbose ]

Component -view   Name ... -family Name [ -long | -processInfo ]
                    [ -become Name ] [ -verbose ]
  
```

## Action Flags

The action flags of the **Component** command and their required authority are listed in Figure 13.

Action Flag and Arguments	Purpose	Implicit Authority	Explicit Authority
<b>-create</b> <i>Name ...</i>	Creates components with the specified names. (Component names must be unique within a family.)	Parent component owner	CompCreate for parent component
<b>-delete</b> <i>Name ...</i>	Deletes the specified components. (You cannot delete the <i>root component</i> .)	Component owner	CompDelete
<b>-link</b> <i>Name ...</i>	Attaches components to an existing component. The components you list with this flag become child components of the component you specify with the <b>-parent</b> attribute flag.	Component owner of component being linked	CompLink for component being linked
<b>-modify</b> <i>Name ...</i>	Modifies properties of the specified components.	Component owner	CompModify
<b>-recreate</b> <i>Name ...</i>	Recreates components as child components of the parent component. (Use the <b>-parent</b> flag to specify the parent.)	Component owner of new parent component	CompRecreate for parent component

Figure 13 (Part 1 of 2). Component Action Flags

Action Flag and Arguments	Purpose	Implicit Authority	Explicit Authority
-unlink <i>Name ...</i>	Detaches components from a parent component. (The components being unlinked must still be linked to at least one parent component.)	Component owner of component being unlinked	CompUnlink in component being unlinked
-view <i>Name ...</i>	Shows all current information for the specified components.	Component owner	CompView

Figure 13 (Part 2 of 2). Component Action Flags

## Attribute Flags

The attribute flags of the **Component** command are listed in Figure 14.

Attribute Flag and Argument	Purpose
-become <i>Name</i>	Specifies the CMVC user ID to validate your authority to perform this action, only if the CMVC user ID differs from your login. (Environment Variable: CMVC_BECOME)
-description <i>Text</i>	Specifies a description of the component.
-family <i>Name</i>	Specifies the family for which this command is being issued. (Environment Variable: CMVC_FAMILY)
-long	Displays more information for the specified components, including all child and parent components, in reference to the <b>-view</b> action flag.
-name <i>Name</i>	Specifies a new name for an existing component.
-owner <i>Name</i>	Specifies the user ID of the component owner.
-parent <i>Name</i>	Specifies the parent component. You must specify this with <b>-create</b> , <b>-link</b> , <b>-unlink</b> or <b>-recreate</b> action flags.
-processInfo	Displays the current process setting and associated CMVC subprocesses for the specified components when used with the <b>-view</b> action flag.
-process <i>Name</i>	Specifies a process when creating or modifying a component. Processes are configured by your family administrator. For a list of the valid component processes and the CMVC subprocesses they include, use the <b>Report -view cfgcomproc</b> command.
-verbose	Indicates that you want to see a confirmation message after you issue this command.

Figure 14. Component Attribute Flags

## Examples

The following are examples of **Component** command actions:

1. To create a new child component called docs for the existing graphix component, and to assign the IBM shipped process preship to it, type:

```
Component -create docs -parent graphix -description "Technical Info"
-process preship
```

The docs component is created with the preship process specified. It inherits the access and notification defined at the graphix component and at all components above it in the hierarchy provided access has not been restricted.

**Note:** The access and notification lists for the docs component do not show the inherited access and notification information. Additional access and notification can be defined by creating access and notification lists for the docs component.

2. To delete a component called archive01, type:

```
Component -delete archive01
```

The component called archive01 is deleted only if it has no child components, no releases, no associated files, no active defects, no active features, or no active sizing records referencing it.

3. To give pam ownership of the graphix component that you currently own, type:

```
Component -modify graphix -owner pam
```

Ownership is re-assigned.

4. To change the name, description, and process of the existing component called graphix, type:

```
Component -modify graphix -name graphix00 -description  
"Version 00 of graphix files" -process prototype
```

The component graphix is renamed to graphix00. The description indicates that this component refers to version 00 graphix files. The process for the component is changed to prototype.

5. To link two existing components, docs and etc, so that etc is the parent component to the docs component, type:

```
Component -link docs -parent etc
```

The component docs becomes a child component of the component etc. The docs component inherits access and notification information from the etc component. It does not lose existing access and notification information from its own access and notification lists.

6. To recreate the deleted tools component so that it exists as a child component of the graphix00 component, type:

```
Component -recreate tools -parent graphix00
```

The component tools is now a child component of graphix00. It inherits the access and notification information from the graphix00 component.

7. To view information about an existing component called debugr, type:

```
Component -view debugr
```

All information for the component called debugr is displayed.

## Related Information

See commands: **Access, File, Notify, Report.**



---

## Chapter 6. Coreq

Use the **Coreq** command to create and delete corequisite relationships between two or more tracks that are in the fix or integrate states. (A track is identified by a defect identifier and a release name or by a feature identifier and a release name.) The tracks you identify as corequisites must all apply to the same release to be compiled together. Tracks defined as prerequisites by CMVC must also be compiled together. For a discussion of *prerequisite tracks*, refer to the book *IBM CMVC Concepts*.

Identify corequisite relationships between tracks to indicate that work being done for a given feature or defect is dependent on changes to files associated with another defect or feature and must therefore be built together (committed together) so that the resulting code works correctly. This action ensures that a level that includes one or more groups of *corequisite tracks* cannot be committed unless all the tracks in the corequisite group are included in the level.

Once you identify two or more tracks as corequisites, you can add additional tracks to that corequisite group without identifying all of the tracks already in the group. You only have to specify one track from the existing group and the new track or tracks you want to add to the group. If you specify one track from each of two or more groups of corequisites, the associated groups are merged into one corequisite group.

When you delete one track from a corequisite group containing only two tracks, no corequisite group remains. You must have at least two tracks to create a corequisite group of tracks.

### Syntax

The syntax statements for the **Coreq** command are:

```
Coreq -create    { -defect Number ... -feature Number ... } -release Name  
                  -family Name [ -become Name ] [ -verbose ]  
  
Coreq -delete   { -defect Number ... -feature Number ... } -release Name  
                  -family Name [ -become Name ] [ -verbose ]
```

### Action Flags

The action flags of the **Coreq** command and their required authority are listed in Figure 15.

Action Flag	Purpose	Implicit Authority	Explicit Authority
<b>-create</b>	Creates a corequisite relationship between the specified tracks.  <b>Note:</b> To perform this action, the associated release's process must include the level subprocess.	Track owner of all specified tracks	CoreqCreate
<b>-delete</b>	Deletes the specified tracks from an existing group of corequisite tracks.	Track owner of all specified tracks	CoreqDelete

Figure 15. Coreq Action Flags

## Attribute Flags

The attribute flags of the **Coreq** command are listed in Figure 16.

Attribute Flag and Argument	Purpose
-become <i>Name</i>	Specifies the CMVC user ID to validate your authority to perform this action, only if the CMVC user ID differs from your login. (Environment Variable: CMVC_BECOME)
-defect <i>Number ...</i>	Specifies one or more defects that define corequisite tracks.
-family <i>Name</i>	Specifies the family for which this command is being issued. (Environment Variable: CMVC_FAMILY)
-feature <i>Number ...</i>	Specifies one or more features that define corequisite tracks.
-release <i>Name</i>	Specifies the release in which the tracks exist. (Environment Variable: CMVC_RELEASE)
-verbose	Indicates that you want a confirmation message after you issue this command.

Figure 16. Coreq Action Flags

## Examples

The following are examples of **Coreq** command actions:

1. Tracks exist for feature 318, defect A329, and defect B312 in reference to the graphix11 release. To indicate that these three tracks are corequisites, type:

```
Coreq -create -feature 318 -defect A329 B312 -release graphix11
```

The tracks now belong to a corequisite group.

2. To add defect 322 to the group of corequisite tracks created in example 1, type:

```
Coreq -create -defect A329 322 -release graphix11
```

By naming one of the tracks from the existing corequisite group along with a new track, you identify the new track as a corequisite of each of the tracks in the existing group.

3. To delete the track for feature 318 from the corequisite group defined in example 2, when your CMVC\_RELEASE environment variable is set to the graphix11 release, type:

```
Coreq -delete -feature 318
```

The track for feature 318 in release graphix11 is no longer part of the corequisite group of tracks.

## Related Information

See commands: **Defect, Feature, Report, Track.**

While corequisites are defined by users, prerequisites are identified by CMVC. Therefore, you can create and delete corequisite relationships, but you cannot modify prerequisite relationships. For more information about corequisite and prerequisite relationships, refer to the book *IBM CMVC Concepts*.

---

## Chapter 7. Defect

Use the **Defect** command to report problems by opening defects. Also use this command to modify properties of defects, change the state of defects, and view information about defects.

When you open a defect, you become the originator of the reported defect. You must describe the problem you think needs to be resolved and the primary component affected by the problem. The owner of the component you assign the defect to becomes the defect owner. That person must respond to the defect by accepting it, returning it, or assigning it to a different component or user ID. ( If the defectDSR subprocess is included in the managing component's process, then the defect owner must respond to it by designing it, returning it, or assigning it to a different component or user ID.)

As the originator of the defect, you can cancel or reopen it if it is returned by the defect owner, and you can modify selected properties of a defect.

Originators of duplicate defects are also notified when the corresponding active defect or feature is closed or canceled. They can either cancel or reopen the duplicate defect, as appropriate.

The states a defect moves through depends on the CMVC subprocesses included in its associated component process. A component process can include the defectDSR (design, size, review) or defectVerify subprocesses, or none at all. For more information on the defect states and their relationship to CMVC subprocesses, refer to the book *IBM CMVC Concepts*.

**Note:** Because your family administrator can modify or delete certain configurable defect fields and create new fields, the attributes for the **-open** and **-modify** actions listed in this section may be different from those in your family or may not appear at all. Those listed here represent the shipped default fields only. For a list of the field properties and flags in use in your family, use the **Defect -configInfo** command or see your family administrator. For more information on configurable fields, refer to the book *IBM CMVC Server Administration and Installation*.

### Syntax

The syntax statements for the **Defect** command are:

**Defect -accept**    *Number ... -family Name [ -answer Name ]\* [ -remarks Text ] [ -become Name ] [ -verbose ]*

**Defect -assign**    *Number ... -family Name { -component Name -owner Name } [ -remarks Text ] [ -become Name ] [ -verbose ]*

**Defect -cancel**    *Number ... -family Name [ -remarks Text ] [ -become Name ] [ -verbose ]*

**Defect -configInfo** *-family name [ -become Name ] [ -raw ]*

**Defect -design**    *Number ... -family Name [ -remarks Text ] [ -become Name ] [ -verbose ]*

**Defect -modify**     *Number ... -family Name { -severity Name -answer Name -environment Name -reference Name -priority Name -symptom Name -release Name -originator Name -target Name -level Name -abstract Text -phaseFound Name -phaseInject Name -prefix Name -name Number } [-remarks Text] [-become Name] [-verbose]*

**Defect -note**     *Number ... -remarks Text -family Name [-become Name] [-verbose]*

**Defect -open**     *-remarks Text -component Name -family Name [-name Number] [-environment Name] [-severity Name]\* [-reference Name] [-prefix Name]\* [-symptom Name]\* [-phaseFound Name]\* [-level Name] [-abstract Text] [-release Name] [-become Name] [-verbose]*

**Defect -reopen**     *Number ... -family Name [-remarks Text] [-become Name] [-verbose]*

**Defect -return**     *Number ... -family Name [-remarks Text] [-answer Name | -duplicate Name]\* [-become Name] [-verbose]*

**Defect -review**     *Number ... -family Name [-remarks Text] [-become Name] [-verbose]*

**Defect -size**     *Number ... -family Name [-remarks Text] [-become Name] [-verbose]*

**Defect -verify**     *Number ... -family Name [-remarks Text] [-become Name] [-verbose]*

**Defect -view**     *Number ... -family Name [-long | -processInfo] [-become Name] [-verbose]*

**Note:** Arguments marked with an asterisk (\*) are required when no default value is set for the CMVC family.

## Action Flags

The action flags of the **Defect** command and their required authority are listed in Figure 17.

Action Flag and Argument	Purpose	Implicit Authority	Explicit Authority
<b>-accept</b> <i>Number ...</i>	Accepts defects that are in the open or review states as problems to be resolved (depending on the subprocess configuration of the component).	Defect owner	DefectAccept
<b>-assign</b> <i>Number ...</i>	Reassigns defects to another owner or another component. The owner of the user ID or component becomes the new defect owner.	Defect owner, Defect originator	DefectAssign
<b>-cancel</b> <i>Number ...</i>	Cancels defects that are in the open state or returned state.	Defect originator	DefectCancel

Figure 17 (Part 1 of 3). Defect Action Flags

Action Flag and Argument	Purpose	Implicit Authority	Explicit Authority
<b>-configInfo</b>	Shows configurable field properties for defects in the specified family. (The information is returned in a fixed ASCII table format.)	N/A	N/A
<b>-design Number ...</b>	Moves defects to the design state or specifies design text. Defects can move to the design state from the open, returned, design, size, or review state.	Defect owner	DefectDesign
<b>-modify Number</b>	Modifies selected properties of defects:		
	<b>-answer</b>	Defect owner	DefectModify
	<b>-name</b> <b>-originator</b> <b>-severity</b>	Defect originator	DefectModify
	<b>-abstract</b> <b>-environment</b> <b>-level</b> <b>-prefix</b> <b>-reference</b> <b>-release</b> <b>-phaseFound<sup>1</sup></b> <b>-phaseInject<sup>1</sup></b> <b>-priority<sup>1</sup></b> <b>-symptom<sup>1</sup></b> <b>-target<sup>1</sup></b>	Defect originator, Defect owner	DefectModify
<b>-note Number ...</b>	Adds remarks to defects. These notes cannot be modified or deleted once they are in the system.	N/A	N/A
<b>-open</b>	Opens a defect. (A unique identifier is generated by CMVC to identify the new defect, unless you specify an identifier using the optional <b>-name</b> flag.)	N/A	N/A
<b>-reopen Number ...</b>	Reopens defects that are in the returned state or the canceled state.	Defect originator	DefectReopen
<b>-return Number ...</b>	Returns defects that are in the open, design, size, review, or working states. (A working defect can be returned only if it does not have tracks associated with it.)	Defect owner	DefectReturn
<b>-review Number ...</b>	Moves defects from the size state to the review state so that the proposed design implementation and sizing information can be reviewed.	Defect owner	DefectReview
<b>-size Number ...</b>	Moves defects from the design state to the size state for sizing. (Design text must first be entered using <b>Defect -design -remarks</b> ).	Defect owner	DefectSize

Figure 17 (Part 2 of 3). Defect Action Flags

Action Flag and Argument	Purpose	Implicit Authority	Explicit Authority
-verify <i>Number ...</i>	Moves defects from the working state to the verify state.	Defect owner	DefectVerify
-view <i>Number ...</i>	Shows all current information for the specified defects.	Defect owner, Defect originator	DefectView

<sup>1</sup> These shipped default flags can be changed or deleted by your family administrator and may not appear as listed. For a list of the configurable flags in use in your family, use the **Defect -configInfo** command.

Figure 17 (Part 3 of 3). Defect Action Flags

## Attribute Flags

The attribute flags of the **Defect** command are listed in Figure 18.

Attribute Flag and Argument	Purpose
-abstract <i>Text</i>	Enters concise text to summarize a defect. Up to 63 characters are allowed. This text appears in reports and notification messages. (If this flag is not specified when you are opening a defect, the first 63 characters or the text up to the first new-line character of the <b>-remarks</b> flag serve as the abstract.)
-answer <i>Name</i>	Specifies an answer code when accepting, modifying, or returning a defect.
-become <i>Name</i>	Specifies the CMVC user ID to validate your authority to perform this action, only if the CMVC user ID differs from your login. (Environment Variable: CMVC_BECOME)
-component <i>Name</i>	Specifies the name of the component when opening or assigning a defect. The environment variable is not used for <b>Defect -assign</b> . (Environment Variable: CMVC_COMPONENT)
-duplicate <i>Name</i>	Specifies that another defect or feature (that is not canceled, returned, or closed) already exists to address the defect being returned.
-environment <i>Name</i>	Specifies the environment where a defect was discovered, for example, AIX* or OS/2* environments.
-family <i>Name</i>	Specifies the family for which this command is being issued. (Environment Variable: CMVC_FAMILY)
-level <i>Name</i>	Specifies the level in which the defect was discovered.
-long	Displays detailed information for the specified defect; including the defect history, all notes attached to the defect, all tracks and verification records associated with the defect, and any duplicate defects or features and their originators, in reference to the <b>-view</b> action flag.
-name <i>Number</i>	Specifies the defect identifier. Up to 15 alphanumeric characters are allowed for user-generated defect IDs. (CMVC checks the uniqueness of the ID. If the ID already exists in CMVC, the action fails and you receive a message indicating that the identifier is not unique. You must then enter a new identifier or allow CMVC to generate one.)
-originator <i>Name</i>	Specifies the user ID of the new originator when you modify a defect. The originator's verification record must be reassigned manually using <b>Verify -assign</b> when the defect is in either the working state or the verify state.

Figure 18 (Part 1 of 2). Defect Attribute Flags

Attribute Flag and Argument	Purpose
<b>-owner</b> <i>Name</i>	Specifies the user ID of the new owner when you assign a defect.
<b>-prefix</b> <i>Name</i>	Identifies a prefix that categorizes the defect by type. This value precedes the defect identifier in report output.
<b>-processInfo</b>	Displays the current process setting and associated CMVC subprocesses for the component associated with the specified defects when used with the <b>-view</b> action flag.
<b>-raw</b>	Produces report output in raw format: <ul style="list-style-type: none"> <li>Information retrieved from each field is separated by the vertical bar delimiter.</li> <li>Each line of output corresponds to one database record.</li> </ul>
<b>-reference</b> <i>Name</i>	Assigns a value, name, or keyword to a defect. Or refers to a previous defect or feature when opening or modifying a defect.
<b>-release</b> <i>Name</i>	Specifies a particular release to trigger the defect verification process when the track for this release moves to the complete state.
<b>-remarks</b> <i>Text</i>	Describes the change being requested, the actual design for the defect, or the reason for modifying or changing the state of the defect. Once you issue a command which adds remarks, you cannot change the remarks (that is, you cannot use <b>Defect -modify</b> to change the remarks). To move a defect to the size state, you must have entered some design text using the <b>-remarks</b> flag within the <b>-design</b> action.
<b>-severity</b> <i>Name</i>	Specifies the severity of the problem that the defect was opened to resolve.
<b>-verbose</b>	Indicates that you want to see a confirmation message after you issue this command.
<b>-phaseFound</b> <i>Name</i> <sup>1</sup>	When opening or modifying a defect, specifies the development phase in progress when the defect was discovered.
<b>-phaseInject</b> <i>Name</i> <sup>1</sup>	When modifying a defect, specifies the development phase in progress when the defect was injected in the code.
<b>-priority</b> <i>Name</i> <sup>1</sup>	When modifying a defect, specifies the timing or scheduling requirements for resolving a defect.
<b>-symptom</b> <i>Name</i> <sup>1</sup>	Specifies the symptom associated with the defect.
<b>-target</b> <i>Name</i> <sup>1</sup>	Specifies a target (such as, a level or a date) for defect resolution or availability.

<sup>1</sup> These shipped default flags can be changed or deleted by your family administrator and may not appear as listed. For a list of the configurable flags in use in your family, use the **Defect -configInfo** command.

Figure 18 (Part 2 of 2). Defect Attribute Flags

## Examples

The following are examples of **Defect** command actions:

1. Assume default values are set for phaseFound, symptom, and prefix. To open a defect with a severity rating of 3 against the debugr component, using the text from an existing file to describe the defect, type:

```
Defect -open -component debugr -sev 3 -remarks - < /tmp/defect.descr
```

A new defect with the severity rating of 3 is opened against the component called debugr. The dash (-) after the **-remarks** flag indicates the location of the redirected input. The redirection symbol (<) indicates that the file /tmp/defect.descr contains the remarks, that is, the description of the problem. The first 63 characters are used as the abstract.

The defect identifier is displayed on the screen when the command is completed successfully.

You are the originator of this defect because you opened it, and the component owner is the owner of the defect.

2. To change the severity rating for defect 4312 and to change the existing value in the reference field, type:

```
Defect -modify 4312 -sev 3 -reference BADMSG
```

The severity level of defect 4312 is changed to 3, and the reference is changed to BADMSG.

3. To assign defect 4312 to the graphix component, type:

```
Defect -assign 4312 -component graphix
```

Defect 4312 is assigned to the graphix component; therefore, the owner of the graphix component becomes the owner of defect 4312.

4. Assume that you are the originator of defect 4298 and this defect is currently in the returned state. To cancel this defect, type:

```
Defect -cancel 4298 -remarks "This was a user error."
```

As the originator, you could have also canceled this defect if it was in the open state.

5. Assume that you are a defect owner. To return a defect someone opened against your component because it is a duplicate of a defect that is currently in the working state, type:

```
Defect -return 4245 -duplicate 4197
```

Defect 4245 is associated with defect 4197 as a duplicate. Defect 4245 is moved to the returned state, and its answer code becomes duplicate. A verification record is created for the originator of defect 4245 and it exists in reference to defect 4197. Originators of all duplicate defects and features must complete verification records when the active defect is in the verify state.

6. Assume that you are the originator of defect 1424. It is returned to you by the defect owner. To reopen defect 1424, type:

```
Defect -reopen 1424 -remarks "Disagree with restriction classification"
```

Defect 1424 moves to the open state.

7. Assume that you own a component against which someone opened defect 4312. To accept defect 4312, and associate it with the answer code your family administrator has configured to represent an enhancement (enh), type:

```
Defect -accept 4312 -answer enh
```

Defect 4312 is moved to the working state with an answer code for enhancement. (Defect answer codes are defined by the family administrator.)

8. To view information about defect 4244, including its history, all notes, tracks, and verification records, process name and associated subprocess settings, type:

```
Defect -view 4244 -long
```

9. To view the configurable field properties for the defects in family rdev, type:

```
Defect -configInfo -family rdev
```



## Related Information

See commands: **Feature, Fix, Report, Size, Track, Verify.**

Use the **Report** command to get more information on existing configuration table values:

- **Report -vi** config
- **Report -vi** config **-w** "name = 'symptom' "

To see the defect state diagrams, refer to the *IBM CMVC User's Reference*.





## Attribute Flags

The attribute flags of the **Environment** command are listed in Figure 20.

Attribute Flag and Argument	Purpose
-become <i>Name</i>	Specifies the CMVC user ID to validate your authority to perform this action, only if the CMVC user ID differs from your login. (Environment Variable: CMVC_BECOME)
-family <i>Name</i>	Specifies the family name for which this command is being issued. (Environment Variable: CMVC_FAMILY)
-release <i>Name</i>	Specifies the release with which the environment list is associated. (Environment Variable: CMVC_RELEASE)
-tester <i>Name</i>	Specifies the user responsible for testing in a given environment.
-verbose <i>Name</i>	Indicates that you want to see a confirmation message after you issue this command.

Figure 20. Environment Attribute Flags

## Examples

The following are examples of **Environment** command actions:

1. Assume you own the `debugr` release. Work being done in reference to that release needs to be tested in the `PCVersion1` environment as well as in the `PCVersion2` environment. To specify `john` as the tester on the environment list associated with the `debugr` release, type:

```
Environment -create PCVersion1 PCVersion2 -tester john -release debugr
```

Two new environment list entries are created for the `debugr` release: one for the `PCVersion1` environment and one for the `PCVersion2` environment. The owner of the user ID `john` is responsible for testing both environments. Therefore, `john` owns 2 test records for every track that is created for the `debugr` release.

2. To delete all entries for environment `ModelA` from the environment list associated with the release set in your `CMVC_RELEASE` environment variable, type:

```
Environment -delete ModelA
```

The environment `ModelA` is deleted from the list for the release specified by the `CMVC_RELEASE` environment variable.

3. To indicate that the new tester for `PCVersion2` in the `graphix` release is `lisa`, type:

```
Environment -modify PCVersion2 -tester lisa -release graphix
```

The owner of the user ID `lisa` replaces the previous person responsible for testing the `PCVersion2` environment for the `graphix` release.

## Related Information

See commands: **Test, Track, Release, Report.**



---

## Chapter 9. Feature

Use the **Feature** command to open requests for design changes or ideas for future functions. Also use this command to delete, modify properties of, change the state of, and view information about features.

When you open a feature, you become the originator of the feature. You must describe the proposed design change and name the primary component affected by the feature. The owner of the component you assign the feature to becomes the feature owner. That person responds to the feature by moving it to the design state, returning it, or assigning it to a different component or user ID. (If the featureDSR subprocess is not included in the managing component's process, then the feature owner must respond to it by accepting it, returning it, or assigning it to a different component or user ID.)

As the originator of the feature, you can cancel or reopen it if it is returned by the feature owner, and you can modify selected properties of a feature.

Originators of duplicate features are also notified when the corresponding active defect or feature is closed or canceled. Thus they can either cancel or reopen the duplicate feature, as appropriate.

The states a feature moves through depends on the CMVC subprocesses included in its associated component process. A component process can include the featureDSR (design, size, review) or featureVerify subprocesses, or none at all. For more information on the feature states and their associated subprocesses, refer to the *IBM CMVC Concepts* manual.

**Note:** Because your family administrator can modify or delete certain configurable feature fields and create new fields, the attributes for the **-open** and **-modify** actions listed in this section may be different from those in your family or may not appear at all. Those listed here represent the shipped default fields only. For a list of the field properties and flags in use in your family, use the **Feature -configInfo** command or see your family administrator. For more information on configurable fields, refer to the *IBM CMVC Server Administration and Installation* manual.

### Syntax

The syntax statements for the **Feature** command are:

```
Feature -accept      Number ... -family Name [ -remarks Text ] [ -become Name ] [ -verbose ]
Feature -assign     Number ... -family Name [ -remarks Text ] [ -verbose ] { -component Name -owner Name } [ -become Name ]
Feature -cancel     Number ... -family Name [ -remarks Text ] [ -become Name ] [ -verbose ]
Feature -configInfo -family name [ -become Name ] [ -raw ]
Feature -design      Number ... -family Name [ -remarks Text ] [ -become Name ] [ -verbose ]
```

<b>Feature -modify</b>	<i>Number ... -family Name { -prefix Name -name Number -target Name -reference Name -originator Name -abstract Text -priority Name } [ -remarks Text ] [ -become Name ] [ -verbose ]</i>
<b>Feature -note</b>	<i>Number ... -remarks Text -family Name [ -become Name ] [ -verbose ]</i>
<b>Feature -open</b>	<i>-remarks Text -component Name -family Name [ -name Number ] [ -prefix Name ]* [ -reference Name ] [ -abstract Text ] [ -become Name ] [ -verbose ]</i>
<b>Feature -reopen</b>	<i>Number ... [ -remarks Text ] [ -become Name ] [ -verbose ]</i>
<b>Feature -return</b>	<i>Number ... -family Name [ -duplicate Name ] [ -remarks Text ] [ -become Name ] [ -verbose ]</i>
<b>Feature -review</b>	<i>Number ... -family Name [ -remarks Text ] [ -become Name ] [ -verbose ]</i>
<b>Feature -size</b>	<i>Number ... -family Name [ -remarks Text ] [ -become Name ] [ -verbose ]</i>
<b>Feature -verify</b>	<i>Number ... -family Name [ -remarks Text ] [ -become Name ] [ -verbose ]</i>
<b>Feature -view</b>	<i>Number ... -family Name [ -long   -processInfo ] [ -become Name ] [ -verbose ]</i>

**Note:** Arguments marked with an asterisk (\*) are required when no default value is set in the configuration table for the CMVC family.

## Action Flags

The action flags of the **Feature** command and their required authority are listed in Figure 21.

Action Flag and Arguments	Purpose	Implicit Authority	Explicit Authority
<b>-accept</b> <i>Number ...</i>	Accepts features that are in the open or review states as enhancements to be implemented (depending on the subprocess configuration of the component).	Feature owner	FeatureAccept
<b>-assign</b> <i>Number ...</i>	Assigns features to another owner or another component.	Feature owner, Feature originator	FeatureAssign
<b>-cancel</b> <i>Number ...</i>	Cancels features that are in the open state or the returned state.	Feature originator	FeatureCancel
<b>-configInfo</b>	Shows configurable field properties for features in the specified family. (The information is returned in a fixed ASCII table format.)	N/A	N/A

Figure 21 (Part 1 of 2). Feature Action Flags



Action Flag and Arguments	Purpose	Implicit Authority	Explicit Authority
<b>-design</b> <i>Number ...</i>	Moves features to the design state or specifies design text. (Features can move to the design state from the open, returned, design, size, or review state.)	Feature owner	FeatureDesign
<b>-modify</b> <i>Number ...</i>	Modifies selected properties of features: <b>-name</b> <b>-originator</b>	Feature originator	FeatureModify
	<b>-abstract</b> <b>-prefix</b> <b>-reference</b> <b>-priority</b> <sup>1</sup> <b>-target</b> <sup>1</sup>	Feature originator, Feature owner	FeatureModify
<b>-note</b> <i>Number ...</i>	Adds remarks to features. These notes cannot be modified or deleted after they are in the system.	N/A	N/A
<b>-open</b>	Opens a feature. (A unique identifier is generated by CMVC to identify the new feature unless you specify an identifier using the optional <b>-name</b> flag.)	N/A	N/A
<b>-reopen</b> <i>Number ...</i>	Reopens features that are in the returned state or the canceled state.	Feature originator	FeatureReopen
<b>-return</b> <i>Number ...</i>	Returns features from any state except the verify state, closed state, or canceled state. (A working feature can be returned only if it does not have tracks associated with it.)	Feature owner	FeatureReturn
<b>-review</b> <i>Number ...</i>	Moves features from the size state to the review state so that the proposed design implementation and sizing information can be reviewed.	Feature owner	FeatureReview
<b>-size</b> <i>Number ...</i>	Moves features from the design state to the size state for sizing. (Design text must first be entered using <b>Feature -design -remarks</b> ).	Feature owner	FeatureSize
<b>-verify</b> <i>Number ...</i>	Moves features from the working state to the verify state.	Feature owner	FeatureVerify
<b>-view</b> <i>Number ...</i>	Shows information about features.	Feature owner, Feature originator	FeatureView

<sup>1</sup> These shipped default flags can be changed or deleted by your family administrator and may not appear as listed. For a list of the flags in use in your family, use the **Feature -configInfo** command.

Figure 21 (Part 2 of 2). Feature Action Flags

## Attribute Flags

The attribute flags of the **Feature** command are listed in Figure 22.

Attribute Flag and Argument	Purpose
<b>-abstract</b> <i>Text</i>	Enters concise text to summarize a feature. Up to 63 characters are allowed. This text appears in reports and notification messages. (If this flag is not specified when you open a feature, the first 63 characters or the text up to the first new-line character of the <b>-remarks</b> flag serves as the abstract.)
<b>-become</b> <i>Name</i>	Specifies the CMVC user ID to validate your authority to do this action, only if the CMVC user ID differs from your login. (Environment Variable: CMVC_BECOME)
<b>-component</b> <i>Name</i>	Specifies the name of the component when opening or assigning a feature. The environment variable is not used for <b>Feature -assign</b> . (Environment Variable: CMVC_COMPONENT)
<b>-duplicate</b> <i>Name</i>	Specifies that another defect or feature (that is not canceled, returned, or closed) already exists to address the feature being returned.
<b>-family</b> <i>Name</i>	Specifies the family for which this command is being issued. (Environment Variable: CMVC_FAMILY)
<b>-long</b>	Displays information for the specified feature including the feature history, all notes attached to the feature, all sizing, track, and verification records associated with the feature and any duplicate defects or features and their originators, in reference to the <b>-view</b> action flag.
<b>-name</b> <i>Number</i>	Specifies the feature identifier. Up to 15 alphanumeric characters are allowed for user-generated feature IDs. (CMVC checks the uniqueness of the ID. If the ID already exists in CMVC, the action fails and you receive a message indicating that the identifier is not unique. You must then enter a new identifier or allow CMVC to generate one.)
<b>-originator</b> <i>Name</i>	Specifies the user ID of the new originator when modifying a feature. The originator's verification record must be reassigned manually using <b>Verify -assign</b> when the feature is in either the working state or the verify state.
<b>-owner</b> <i>Name</i>	Specifies the user ID of the new owner when you assign a feature.
<b>-prefix</b> <i>Name</i>	Categorizes features by type. This value precedes the feature identifier in report output.
<b>-processInfo</b>	Displays the current process setting and associated CMVC subprocesses for the component associated with the specified features when used with the <b>-view</b> action flag.
<b>-raw</b>	Produces report output in raw format: <ul style="list-style-type: none"><li>• Information retrieved from each field is separated by the vertical bar delimiter.</li><li>• Each line of output corresponds to one database record.</li></ul>
<b>-reference</b> <i>Name</i>	Assigns a value, name, or keyword to a feature. Or refers to a previous defect or feature when opening or modifying a feature.
<b>-remarks</b> <i>Text</i>	Describes the change being requested, the actual design for the feature, or the reason for modifying or changing the state of the feature. Once you issue a command which adds remarks, you cannot change the remarks (that is, you cannot use <b>Feature -modify</b> to change the remarks.) To move a feature to the size state, you must have entered some design text using the <b>-remarks</b> flag within the <b>-design</b> action.
<b>-verbose</b>	Indicates that you want to see a confirmation message after you issue this command.

Figure 22 (Part 1 of 2). Feature Attribute Flags

Attribute Flag and Argument	Purpose
<code>-priority Name</code> <sup>1</sup>	When modifying a feature, specifies the timing or scheduling requirements for implementing a feature.
<code>-target Name</code> <sup>1</sup>	When opening or modifying a feature, specifies a target (such as, a level or a date) for feature implementation or availability.

<sup>1</sup> These shipped default flags can be changed or deleted by your family administrator and may not appear as listed. For a list of the flags in use in your family, use the **Feature -configInfo** command.

Figure 22 (Part 2 of 2). Feature Attribute Flags

## Examples

The following are examples of **Feature** command actions:

1. To open a feature against the debugr component, assuming there is a default prefix value set, type:

```
Feature -open -rem "Change format of parameter values display"
-component debugr
```

A feature change request is created against the debugr component.

The feature identifier appears on the screen when the command is completed.

You are the originator of this feature because you opened it. The owner of component debugr is the owner of the feature.

2. Assume that you are the owner of feature 4312. To assign it to another component, type:

```
Feature -assign 4312 -component graphix
```

Feature 4312 is assigned to the graphix component; the owner of graphix becomes the new owner of this feature.

3. Assume that you are the originator of feature 4298 and that it is currently in the returned state. To cancel this feature, type:

```
Feature -cancel 4298
```

4. Assume that you are the originator of feature 4245 and that it is currently in the canceled state. To reopen that feature, type:

```
Feature -reopen 4245 -remarks "Disagree with restriction
classification"
```

Feature 4245 is now in the open state. It is reopened against the component that owned it when the feature was canceled.

5. Assume that you are a component owner and that feature 4312 was opened against your component. To move that feature to the design state, type:

```
Feature -design 4312
```

Feature 4312 is moved to the design state. You can issue the **Feature -design** command with the **-remarks** flag when you are ready to enter actual design information. You can issue the **Feature -design** command more than once.

6. Assume that you are the owner of feature 3129. This feature is in the design state and text has been entered using **Feature -design -remarks**. To move the feature to the size state, type:

```
Feature -size 3129
```

Feature 3129 is moved to the size state. Once it is in the size state, you can create sizing records using the **Size** command. (One sizing record is required for each component and release combination affected by the feature change.)

7. Assume that you own feature 4312 and that it is currently in the review state. After you have reviewed the feature information, you decide to accept the feature for implementation. To accept the feature, and therefore move it to the working state, type:

```
Feature -accept 4312
```

Feature 4312 moves from the review state to the working state. Tracks and fix records are created according to the sizing records for this feature.

8. To view information about feature 1244, including its purpose, originator, owner, and current state, type:

```
Feature -view 1244
```

9. To view the field properties for the features in family rdev, type:

```
Feature -configInfo -family rdev
```

## Related Information

See commands: **Defect, Fix, Report, Size, Track, Verify**. To see the feature state diagrams, refer to the *IBM CMVC User's Reference*.

---

## Chapter 10. File

Use the **File** command to bring files into the CMVC development environment and to work with individual files once they are in this environment. You must bring a file into the CMVC development environment by creating it on the CMVC server using **File -create**. The file must already exist on the CMVC client's file system before you can bring it under CMVC control.

You create and access files in CMVC by issuing **File** commands from your current working directory. For details concerning the relative placement of files when you create them using the command line interface, refer to "Related Information" on page 51.

When you create a file in CMVC, you must associate it with a release (to relate the file to a development effort) and with a component (to control the ownership of and the access to a file). You also have the option of specifying a file mode when creating a file using the **-fmode** flag. (If file mode is not specified, the current file mode is used.) After a file is successfully created in CMVC, CMVC modifies the file permissions of the working copy of the file left on the client to read-only. All subsequent **File** access commands must specify the file name and release name to identify the correct file.

By default, files are created as text files, although you can specify whether a file is a text or binary file at the time you create it. If your CMVC uses Source Code Control System (SCCS) as the underlying version control system, the following text type files are created as binary type files by CMVC:

- An ASCII control character, SOH (start of header or control-A), at the beginning of a line
- An ASCII control character, NUL, anywhere in the file

The contents of the files are not affected. There are no restrictions on text files if your CMVC uses PVCS Version Manager\*\* as the underlying version control system.

File path names within the CMVC environment must be unique for a release. Files that have a unique base name within a release can be specified by their base name; other files (for example, **Makefile**) must be specified by their full path name when performing CMVC actions against the file.

You can perform various actions against files in CMVC, depending on the authority you have in the access lists for the components that manage the files. Copies of files can be *extracted*, or files can be checked out for editing and subsequently checked in to save the changes. Various properties of files can be modified, such as, the path name, release, component, and file mode. Files can be deleted and then recreated. Destroying files permanently removes the database record for the files from the CMVC environment. Destroyed files cannot be recreated. However, the names of destroyed files can be used to create new files within the CMVC development environment.

Actions performed on files can be undone, although certain limitations apply. The **-undo** action negates the most recent action that changed a file. For files in a

release whose process includes the track subprocess, each uncommitted change can be undone in backward sequential order.

When working with files in a release whose process includes the track and level subprocesses, multiple sets of file changes can be checked in and included in one level; however, other types of file changes can be specified only once in a level. For example, a file can be created or deleted or renamed or recreated or linked in one level, but a file cannot be created and renamed in one level. However, an existing uncommitted **-create** action can be undone so that you can create a file with the desired name.

Files can be linked to identify them as either shared or *common files*. A file is shared if it is a member of more than one release, and different versions of the file are used in each release. Shared files follow separate paths of development but are based on the same initial file. A file is common if it is a member of more than one release, and the same version of the file is being used in those releases. Common files follow a single path of development.

You can define a common file for releases whose process does not include the track subprocess; however, commonality is broken when a change is made to the file. For a discussion of the CMVC track subprocess, refer to the book *IBM CMVC Concepts*. If the track subprocess is included in the release process, CMVC maintains commonality of files unless common versions are forced using the **-force** flag. For more information about breaking commonality, refer to “Related Information” on page 51.

**Note:** Because your family administrator can create new fields, the attributes for the **-create** and **-modify** actions listed in this section may be different from those in your family. Those listed here represent the shipped default fields only. For a list of the field properties and flags in use in your family, use the **File -configInfo** command or see your family administrator. For more information on configurable fields, refer to the book *IBM CMVC Server Administration and Installation*.

## Syntax

The syntax statements for the **File** command are:

```
File -checkin    Name ... -release Name -family Name [ -common Name ... ]**  
                  [ -force ] [ -remarks Text ] [ -relative Name | -top Name ]  
                  [ -defect Number ... -feature Number ... ]*  
                  [ -become Name ] [ -verbose ]  
  
File -checkout  Name ... -release Name -family Name [ -force ] [ -stdout ]  
                  [ -relative Name | -top Name ] [ -become Name ] [ -verbose ]  
  
File -configInfo -family name [ -become Name ] [ -raw ]  
  
File -create    Name ... -component Name -release Name -family Name  
                  [ -fmode Octal_number ] [ -relative Name | -top Name ]  
                  [ -binary ] [ -defect Number ... -feature Number ... ]*  
                  [ -remarks Text ] [ -become Name ] [ -verbose ]  
  
File -delete    Name ... -release Name -family Name [ -top Name ]  
                  [ -defect Number ... -feature Number ... ]* [ -force ]  
                  [ -common Name ... ] [ -become Name ] [ -verbose ]  
  
File -destroy   Name ... -release Name -family Name [ -top Name ]  
                  [ -become Name ] [ -verbose ]
```

**File -extract**     *Name ... -release Name -family Name [ -version Name ]*  
                           [ **-nokeys** ] [ **-stdout** ] [ **-relative Name | -top Name** ]  
                           [ **-dmask Octal\_number** ] [ **-fmask Octal\_number** ]  
                           [ **-become Name** ] [ **-verbose** ]

**File -link**         *Name ... -to Name -release Name -family Name*  
                           [ **-version Name** ] [ **-defect Number ... -feature Number ...** ]\*  
                           [ **-top Name** ] [ **-become Name** ] [ **-verbose** ]

**File -lock**         *Name ... -release Name -family Name [ -force ] [ -top Name ]*  
                           [ **-become Name** ] [ **-verbose** ]

**File -modify**       *Name ... -release Name -family Name*  
                           { **-fmode Octal\_number -component Name** }  
                           [ **-top Name** ] [ **-become Name** ] [ **-verbose** ]

**File -recreate**     *Name ... -release Name -family Name [ -top Name ]*  
                           [ **-defect Number ... -feature Number ...** ]\* [ **-force** ]  
                           [ **-common Name ...** ]\*\* [ **-become Name** ] [ **-verbose** ]

**File -rename**       *Name -path Name -release Name -family Name [ -top Name ]*  
                           [ **-defect Number ... -feature Number ...** ]\* [ **-force** ]  
                           [ **-common Name ...** ]\*\* [ **-become Name** ] [ **-verbose** ]

**File -resolve**      *Name ... -release Name -family Name [ -quiet ] [ -top Name ]*  
                           [ **-become Name** ] [ **-verbose** ]

**File -undo**         *Name ... -release Name -family Name [ -top Name ]*  
                           [ **-defect Number ... -feature Number ...** ]\* [ **-force** ]  
                           [ **-common Name ...** ]\*\* [ **-become Name** ] [ **-verbose** ]

**File -unlock**       *Name ... -release Name -family Name [ -become Name ]*  
                           [ **-relative Name | -top Name** ] [ **-verbose** ]

**File -view**         *Name ... -release Name -family Name [ -long ] [ -top Name ]*  
                           [ **-become Name** ] [ **-verbose** ]

**Note:** Arguments marked with an asterisk (\*) are required for files associated with a release whose process includes the CMVC track subprocess. Arguments marked with a double asterisk (\*\*) can be specified if the files are associated with a release whose process includes the CMVC track subprocess.

## Action Flags

The action flags of the **File** command and their required authority are listed in Figure 23.

Action Flag and Arguments	Purpose	Implicit Authority	Explicit Authority
<b>-checkin</b> <i>Name ...</i>	Submits to the CMVC server the changes made to a specified file. <sup>1</sup> (Any associated tracks must be in the fix state and the associated fix records in the ready or active state.)	Component owner User who checked out or <i>locked</i> the file	FileCheckIn [FileForceIn]

Figure 23 (Part 1 of 3). File Action Flags

Action Flag and Arguments	Purpose	Implicit Authority	Explicit Authority
<b>-checkout</b> <i>Name ...</i>	Retrieves a working copy of a specified file and locks it for editing purposes. Only the most recent version of a file can be checked out.	Component owner	FileCheckOut [FileForceOut]
<b>-configInfo</b>	Shows configurable field properties for files in the specified family. (The information is returned in a fixed ASCII table format.)	N/A	N/A
<b>-create</b> <i>Name ...</i>	Creates files with the specified names; this creates a CMVC record for the file and copies it to the server. <sup>1</sup> Files must have unique path names within a release.	Component owner	FileAdd
<b>-delete</b> <i>Name ...</i>	Deletes the specified files. <sup>1</sup> A file's association with a release whose process includes the track subprocess cannot be deleted if changes are pending for that file.	Component owner	FileDelete [FileDeleteForce]
<b>-destroy</b> <i>Name ...</i>	Destroys the specified files. The CMVC record for the file is removed so that a new file can be created using the same file name. Files associated with a release whose process includes the track subprocess must be deleted and committed before the <i>destroy</i> action can be performed.	Component owner	FileDestroy
<b>-extract</b> <i>Name ...</i>	Retrieves a copy of a specified file. The current version is extracted by default.	Component owner	FileExtract
<b>-link</b> <i>Name ...</i>	Makes common or <i>shared files</i> in the specified release. <sup>1</sup>	Component owner	FileLink
<b>-lock</b> <i>Name ...</i>	Locks a file in the CMVC server. This prevents other users from checking out the file. Only the current version of a file can be locked.	Component owner	FileLock [FileLockForce]
<b>-modify</b> <i>Name ...</i>	Reassigns the file to another component or changes the file permission. When reassigning the file to another component, the component you specify manages access to the file. (Different components can manage different versions of the same file.)	Component owner	FileModify
<b>-recreate</b> <i>Name ...</i>	Recreates previously deleted files. <sup>1</sup>	Component owner	FileRecreate [FileRecreaForce]
<b>-rename</b> <i>Name</i>	Specifies a new path name for a file. <sup>1</sup>	Component owner	FileRename [FileRenameForce]
<b>-resolve</b> <i>Name ...</i>	Displays the full path name in a specific release for specified file base names.	N/A	N/A

Figure 23 (Part 2 of 3). File Action Flags



Action Flag and Arguments	Purpose	Implicit Authority	Explicit Authority
<b>-undo</b> <i>Name ...</i>	<p>Undoes the most recent uncommitted action that changed specified files.</p> <p>If the track subprocess is included in the release process, you can undo:<sup>1</sup></p> <ul style="list-style-type: none"> <li>• The most recent uncommitted delete, rename, recreate, or check-in action in one or more of the releases where the file is common. All tracks related to the most recent change must be in the fix state.</li> <li>• The most recent uncommitted create or link in a single release. The <b>-common</b> flag is ignored. All tracks related to the most recent change must be in the fix state.</li> <li>• Multiple check-in actions, up to the latest committed version of the specified file.</li> </ul> <p>If the track subprocess is not included in the release process, you can undo multiple check-in actions back to the first version of a file created within the CMVC environment (or back to the last version of a file committed by a track).</p>	Component owner	FileUndo
<b>-unlock</b> <i>Name ...</i>	<p>Unlocks a file that is checked out so that it is no longer reserved for editing purposes and so that no changes are submitted to the server. Or, unlocks a file that has been previously locked using <b>-lock</b>.</p>	Component owner User who has the file locked	FileUnlock
<b>-view</b> <i>Name ...</i>	<p>Shows all information for the specified files.</p>	Component owner	FileView

<sup>1</sup> If the file is associated with a release whose process includes the CMVC track subprocess, you must specify the tracks associated with this action by giving the defect or feature identifier and release name.

Figure 23 (Part 3 of 3). File Action Flags

## Attribute Flags

The attribute flags of the **File** command are listed in Figure 24.

Attribute Flag and Argument	Purpose
<b>-become</b> <i>Name</i>	<p>Specifies the CMVC user ID to validate your authority to perform this action, only if the CMVC user ID differs from your login. (Environment Variable: CMVC_BECOME)</p>

Figure 24 (Part 1 of 3). File Attribute Flags

Attribute Flag and Argument	Purpose
<b>-binary</b>	Indicates that the file being created is a binary file. (Default type is text.)
<b>-common</b> <i>Name ...</i>	Specifies the releases in which common files are to be maintained or whether the specific file change is to apply to all releases in which the file is common. All releases must be specified unless the <b>-force</b> flag is specified as well.
<b>-component</b> <i>Name</i>	Specifies the name of the component that manages access and notification for a file. (Different components can manage different versions of the same file.) The environment variable is not used for <b>File -modify</b> . (Environment Variable: CMVC_COMPONENT)
<b>-defect</b> <i>Number...</i>	Specifies the defect identifier if the file being acted upon is associated with a release whose process includes the track subprocess. <sup>1</sup>
<b>-dmask</b> <i>Octal_number</i>	Specifies the read, write, and execute directory permissions in octal notation for the extracted file. Default is 750 (read, write and execute access for directory owner, read and execute access for others in the owner's group, and no access for all other users).
<b>-family</b> <i>Name</i>	Specifies the family name for which this command is being issued. (Environment Variable: CMVC_FAMILY)
<b>-feature</b> <i>Number...</i>	Specifies the feature identifier if the file being acted upon is associated with a release whose process includes the track subprocess. <sup>1</sup>
<b>-fmask</b> <i>Octal_number</i>	Specifies the read, write, and execute file permissions in octal notation for the extracted files. Default is the file's mode less the write permission for the file owner, others in the owner's group and all others. The <b>-fmask</b> flag overrides the <b>-fmode</b> setting.
<b>-fmode</b> <i>Octal_number</i>	Specifies the file mode in CMVC when creating or modifying a file. If no mode is specified, the current file mode is accepted.
<b>-force</b>	Forces a break between common files when using <b>-lock</b> , <b>-checkout</b> , <b>-checkin</b> , <b>-delete</b> , <b>-recreate</b> , <b>-rename</b> , or <b>-undo</b> .
<b>-long</b>	Displays information for the specified files, including the file history, whether the file is checked out for editing, all associated common files, and any change information. (The file change information includes the existing active changes for the file, the defect or feature, and file version associated with those changes.)
<b>-nokeys</b>	Indicates that keywords should not be expanded when a file is extracted.
<b>-path</b> <i>Name</i>	Specifies a new file path name when renaming a file. (File names, consisting of the base name and the path name, must be unique within a release.)
<b>-quiet</b>	Suppresses explanatory remarks and new line characters in the output of <b>File -resolve</b> .
<b>-raw</b>	Produces report output in raw format: <ul style="list-style-type: none"> <li>• Information retrieved from each field is separated by the vertical bar delimiter.</li> <li>• Each line of output corresponds to one database record.</li> </ul>
<b>-relative</b> <i>Name</i>	Creates, checks in, checks out, or extracts the specified file relative to the directory location specified according to the complete path name of the file. Directories are created if necessary when extracting or checking out in order to copy the file by its full path name.
<b>-release</b> <i>Name</i>	Specifies the associated release for the specified files. <sup>1</sup> (Environment Variable: CMVC_RELEASE)

Figure 24 (Part 2 of 3). File Attribute Flags

Attribute Flag and Argument	Purpose
<b>-remarks</b> <i>Text</i>	Adds explanatory remarks when checking in or creating a file. Up to 15 999 characters are permitted.
<b>-stdout</b>	Redirects the specified file to standard output when extracting it from the CMVC server.
<b>-to</b> <i>Name</i>	Specifies the release in which you want to create a link for a common or shared file; use it when you link specified files.
<b>-top</b> <i>Name</i>	Specifies the leading portion of the path name that is a subset of the current working directory on the client machine. (Environment Variable: CMVC_TOP)
<b>-verbose</b>	Indicates that you want to get a confirmation message when you issue this command.
<b>-version</b> <i>Name</i>	Specifies the version of the file you want to extract or link. File versions are specified by an SCCS or PVCS identification number.  <b>Note:</b> PVCS is not supported on the CMVC for Sun systems or CMVC for HP systems products.

<sup>1</sup> If the file is associated with a release whose process includes the CMVC track subprocess, you must specify the tracks associated with this action by giving the defect or feature identifier and release name.

Figure 24 (Part 3 of 3). File Attribute Flags

## Examples

The following are examples of **File** command actions:

1. Assume that your current working directory (cwd) is **/u/jane**, and your CMVC\_TOP environment variable is set to **/u/jane**. You have a file with the path name **/u/jane/src/bar/option/tic.c** on your workstation. To create this file within the CMVC development environment as **src/bar/option/tic.c** and associate it with a release whose process includes the track subprocess, type:  

```
File -create src/bar/option/tic.c -component graphs -release 32charting -defect 341
```

The file `src/bar/option/tic.c` is created as a member of the `32charting` release. The file is managed by the `graphs` component. The file is created as part of the fix for defect 341.
2. To rename an existing file in a release whose process includes the track subprocess, type:  

```
File -rename debugr/src/xyz.c -path debugr/v2/xyz2.c -release 20debugr
```

The file `debugr/src/xyz.c` in release `20debugr` is renamed `debugr/v2/xyz2.c`.
3. Assume that your CMVC\_RELEASE environment variable is set to the release associated with the file `src/bar/option/tic.c`. To reassign that file to another component, type:  

```
File -modify src/bar/option/tic.c -component debugr
```
4. To create a common file between two releases, type:  

```
File -link debugr/x.c -release 10debugr -to 20debugr -defect 866
```

The current version of the file `debugr/x.c` in release `10debugr` is linked to the `20debugr` release. This creates a common file link between the releases `10debugr` and `20debugr` for the current version of the file in release `10debugr`.

If the releases `20debugr` and release `10debugr` both have the track subprocess included in their release process, future changes to the file `debugr/x.c` must reference a track for each release to maintain file commonality.

If the track subprocess is not included in the process of both releases, no track is required but the file becomes a shared file once the file is changed in reference to either release.

5. Assume that your `CMVC_RELEASE` environment variable is set to the release associated with a file you want to extract. To extract a copy of the file, type:

```
File -extract graphix/x.c -stdout > View_x.c
```

File `graphix/x.c` in the release specified by the `CMVC_RELEASE` environment variable is copied to the file `View_x.c` in your current working directory. If the **-stdout** flag is not specified, the file is copied to your current working directory using the base name, `x.c`.

6. Assume that your `cwd` is `/u/jane/graphix`, and your `CMVC_TOP` environment variable is set to `/u/jane`. To check out a working copy of a file and lock it for editing, type:

```
File -checkout x.c -release 10graphix
```

In this example, the value of the `CMVC_TOP` environment variable, `/u/jane`, is stripped from the head of the user's current working directory. The result indicates the name of the file, `graphix/x.c`, within the `CMVC` environment. The file is copied to `/u/jane/graphix` with the name `x.c`, and the current version of the file for the release `10graphix` is locked.

7. Assume that your `CMVC_RELEASE` environment variable is set to the release associated with the file `graphix/x.c` and that the track subprocess is included in the associated release process. Your current working directory is `/u/jane/test`, and your `CMVC_TOP` environment variable is not set. To check in that file after editing, type:

```
File -checkin graphix/x.c -defect 8117 5412
```

Changes made to the file `graphix/x.c` are submitted to the `CMVC` server creating a new version of the file. The changes relate to the tracks corresponding to defects `8117` and `5412` in the release indicated by the `CMVC_RELEASE` environment variable. The edited file `x.c` must exist in your current working directory. If `x.c` was a unique base name within the release, you would only have to specify the base name.

8. To unlock a file that was checked out, type:

```
File -unlock debugr/x.c -release 11debugr
```

The file `debugr/x.c` in the `11debugr` release is no longer locked.

9. To undo the most recent change to a file, type:

```
File -undo debugr/x.c
```

The change submitted most recently for file `debugr/x.c` in the release indicated by the `CMVC_RELEASE` environment variable is reversed or undone. If the track subprocess is not included in the associated release process, the **-undo** action only affects the most recent **File -checkin** command; if the track

subprocess is included in the associated release process, **-undo** affects the most recent **-checkin**, **-create**, **-delete**, **-recreate**, **-rename**, or **-link** action.

10. To view information about a specified file, type:

```
File -view graphix/x.c -release 10graphix
```

To obtain additional information about a specified file, include the **-long** option with **File -view** by typing:

```
File -view graphix/x.c -release 10graphix -long
```

File information for the file `graphix/x.c` is displayed, including the file history, whether or not the file is locked for editing, all common files, and change information.

11. To view the fields properties for the file in family `rdev`, type:

```
File -configInfo -family rdev
```

## Related Information

See commands: **Level**, **Component**, **Defect**, **Feature**, **Release**, **Report**, **Track**.

For a list of supported keywords, refer to the *IBM CMVC User's Reference*.

### Accessing Files and Determining Location

You must always provide a file name and a release when you access a file that is under CMVC control or place a file under CMVC control. Four methods of accessing files are described below. All examples provided assume that your `CMVC_RELEASE` environment variable is set.

#### Method 1

If a file has a base name that is unique within the release, then you can specify just the base name in the command. When you perform a checkout, CMVC writes to the `cwd`. When you perform a checkin, CMVC looks in the `cwd` for the base name.

For example, assume that you have a file named **src/cat/cmvc.msg**, and there is no other file called **cmvc.msg** in the release. You can use `File -checkout cmvc.msg` or `File -checkin cmvc.msg`.

#### Method 2

If a file has a base name that is not unique; that is, the base name is used for multiple files in the release, then you must specify the full path name.

For example, assume that you have a number of **Makefile** files but you only want to work with **src/cat/Makefile**. You must use `File -checkout src/cat/Makefile` to have the file copied to your `cwd`, or `File -checkin src/cat/Makefile` to indicate the location of the file on your *host*.

#### Method 3

You have the option of using the **-relative** flag to specify where to access the file on the client, regardless of `cwd`. This option writes or reads the file according to its full path name as opposed to its base name.

The **-relative** flag can be used only with the **extract**, **create**, **checkout**, **checkin**, and **unlock** actions.

Example 1: File `-checkout src/cat/Makefile -relative /tmp` writes the file to **`/tmp/src/cat/Makefile`**.

Example 2: File `-checkout cmvc.msg -relative /tmp` writes the file to **`/tmp/src/cat/cmvc.msg`**.

In both of these examples, the directories **`/tmp/src`** and **`/tmp/src/cat`** are created if they do not already exist.

## Method 4

You can set a value for the `CMVC_TOP` environment variable. Though this variable is similar to the **`-relative`** flag, it is used only when it matches the leading portion of your current working directory (`cwd`). The `CMVC_TOP` environment variable can also be used with all file actions.

## Determining the Server Pathname by the CMVC Client

The CMVC client determines the server pathname as follows:

1. If the `CMVC_TOP` environment variable is not set, or if `CMVC_TOP` is not a subset of the current working directory (`cwd`), then

server pathname = user-supplied pathname

**Example 1:** `CMVC_TOP` is not set

`CMVC_TOP` = (Not set)

`cwd` = `/u/jane/workspace`

File `-checkout src/cat/cmvc.msg`

**`/src/cat/cmvc.msg`** becomes the server pathname.

**Example 2:** `CMVC_TOP` is not subset of `cwd`

`CMVC_TOP` = `/u/pat`

`cwd` = `/u/jane/workspace`

File `-checkout src/cat/cmvc.msg`

**`/src/cat/cmvc.msg`** becomes the server pathname.

2. If the `CMVC_TOP` environment variable is set, and if `CMVC_TOP` is a subset of the current working directory (`cwd`)

server pathname = (`cwd` - `CMVC_TOP`) + user-supplied pathname

**Example 3:** `CMVC_TOP` is set and is a subset of `cwd`

`CMVC_TOP` = `/u/jane/workspace`

`cwd` = `/u/jane/workspace/src/cat`

File `-checkout cmvc.msg`

**`/src/cat/cmvc.msg`** becomes the server pathname.

If the server pathname is a basename, the server will resolve it to the full path name in the specified release. The resolved pathname must be unique or an error message is issued.

**Example 4:** `CMVC_TOP` is a subset of `cwd` and server pathname is a basename

`CMVC_TOP` = `/u/jane/workspace`

`cwd` = `/u/jane/workspace`

File -checkout cmvc.msg

The server resolves **cmvc.msg /src/cat/cmvc.msg** as the server pathname.

This assumes that the `src/cat/cmvc.msg` is the only pathname in that release that has basename `cmvc.msg`.

### Determining Destination Pathname by the Client

After the CMVC server completes the specified file action, the client determines the destination pathname as follows:

1. If the `CMVC_TOP` environment variable is not set, or if `CMVC_TOP` is not a subset of the current working directory (`cwd`)

destination pathname = `cwd` + resolved server pathname

Example 5: `CMVC_TOP` is not set or is not a subset of `cwd`

`CMVC_TOP` = `/u/jane/workspace`

`cwd` = `/u/pat`

File -checkout `src/cat/cmvc.msg`

or

File -checkout `cmvc.msg`

write the file to **`/u/pat/cmvc.msg`**

2. If the `CMVC_TOP` environment variable is set, and if `CMVC_TOP` is a subset of the current working directory (`cwd`)

destination pathname = `CMVC_TOP` + resolved server pathname

Example 6: `CMVC_TOP` is set and is a subset of `cwd`

`CMVC_TOP` = `/u/jane/workspace`

`cwd` = `/u/jane/workspace/src`

File -checkout `cat/cmvc.msg`

writes the file to **`/u/jane/workspace/src/cat/cmvc.msg`**

If the **-relative** flag is set, then the **-top** flag, `CMVC_TOP` environment variable, and the `cwd` are ignored.

Example 7: The `-relative` flag is set

`CMVC_TOP` = `/u/jane/workspace`

`cwd` = `/u/jane/workspace/src`

File -checkout `cmvc.msg -relative /tmp`

or

File -checkout `src/cat/cmvc.msg -relative /tmp`

write the file to **`/tmp/src/cat/cmvc.msg`**

## Common Files in Releases

When a common file is checked out for editing, it is locked in all releases where it is common. If the release process includes the track subprocess, you only need to do one check-in to have the change reflected in all releases in which the file is common. CMVC maintains commonality of files unless uncommon versions are forced using the **-force** flag. If you need to edit a locked file and cannot wait for the file to be checked in, you can break the common link (and thus the lock on the common version) by specifying the **-force** flag when you issue **File -checkout**. The force applies only in that release so that the file associated with that release is no longer common. You must explicitly link files to make them common after that time.

When a common file is checked in using the **File -checkin** command, a track must be specified for each release in which the file is common. The associated tracks must be in the fix state and the associated fix records in the ready or active state. (A track is identified by a defect identifier and a release name or by a feature identifier and a release name.)

If you want to check in or modify (for example, rename, delete) a common file, and you do not want the changes to be reflected in the other releases in which the file is common, use the **-force** flag to break the common link. If you want to check in or modify a common file, but you want the changes to be reflected in some of the releases to which the file is common but not in others, use the **-force** flag as well as the **-common** flag. Provide names of the releases for which you want to maintain file commonality (excluding the name of the release associated with the file you are checking in or modifying), as arguments to the **-common** flag.



---

## Chapter 11. Fix

Fix records are associated with tracks. A fix record is used to reflect the status of all the file changes made to resolve a defect (or implement a feature) for a release in reference to one component. A track has one or more fix records associated with it, depending on the number of components in which files are changed. The component manages the files that need to be changed in relation to the track.

Use the **Fix** command to create, delete, and reassign fix records and to change the state of fix records.

Each fix record is uniquely identified by a defect or feature identifier, a release, and a component. The owner of a fix record is, by default, the owner of the related component; however, this ownership can be reassigned using the **-assign** action flag.

Each fix record refers to the file changes required within one component. The state of the fix record indicates the state of file changes for that component.

Fix records are created according to the sizing records of a feature or defect at the time a track is created for the feature or defect. They are created in the notReady state if the associated track is in the approve state, otherwise they are created in the ready state. Additional fix records are created if files are changed and checked in to the CMVC development environment for a defect or a feature in a component for which there is no existing fix record. In this case, the fix record is in the active state. The active state means that file changes have been checked in for the defect or feature in the component. You can create fix records using the **-create** action flag if a track is in the approve state or the fix state.

Use the **-complete** action flag to indicate that the file changes necessary to fix the defect or feature within that component are completed. This moves the fix record to the complete state.

When all fix records for the track are completed, it moves from the fix state to the integrate state. Use the **-activate** action flag to reactivate a fix record that is in the complete state if additional file changes are needed. This can only be done if the track is in the fix state.

If you decide that no file changes are required for a component that has a fix record, you can use the **-delete** action flag to delete the fix record from the associated track.

### Syntax

The syntax statements for the **Fix** command are:

```
Fix -activate      { -defect Number ... -feature Number ... }  
                   -family Name -release Name ... -component Name  
                   [ -become Name ] [ -verbose ]
```

```
Fix -assign       -to Name { -defect Number ... -feature Number ... }  
                   -release Name ... -component Name -family Name  
                   [ -become Name ] [ -verbose ]
```

```

Fix -complete    { -defect Number ... -feature Number ... }
                  -family Name -release Name ... -component Name
                  [ -become Name ] [ -verbose ]

Fix -create     { -defect Number ... -feature Number ... } -release Name ...
                  -component Name -family Name
                  [ -developer Name ] [ -become Name ] [ -verbose ]

Fix -delete    { -defect Number ... -feature Number ... } -release Name ...
                  -component Name -family Name
                  [ -become Name ] [ -verbose ]

```

## Action Flags

The action flags of the **Fix** command and their required authority are listed in Figure 25.

Action Flag	Purpose	Implicit Authority	Explicit Authority
<b>-activate</b>	Moves a fix record from the complete state to the active state so that additional file changes can be made. <sup>1</sup> (You can only change the fix record state if the corresponding track is in the fix state.)	Owner of fix record, Component owner, Track owner	FixActive
<b>-assign</b>	Assigns ownership of a fix record to another user ID. <sup>1</sup> (You cannot reassign the component.)	Owner of fix record, Component owner, Track owner	FixAssign
<b>-complete</b>	Moves a fix record to the complete state to indicate that all file changes required in the associated component are completed. <sup>1</sup> If no other fix records exist, or if all other records are completed, this causes the track to change from the fix state to the next valid state governed by the release's process.	Owner of fix record, Component owner, Track owner	FixComplete
<b>-create</b>	Creates a fix record for a track in relation to a component. <sup>1</sup> You can only create a fix record if the track is in the approve state or the fix state.	Defect owner, Feature owner, Track owner	FixCreate
<b>-delete</b>	Deletes the fix record for the specified track and component. (You cannot delete a fix record that is in the active state or the complete state because it has file changes associated with it.)	Defect owner, Feature owner, Track owner	FixDelete

<sup>1</sup> To perform this action, the associated release's process must include the fix subprocess.

Figure 25. Fix Action Flags

## Attribute Flags

The attribute flags of the **Fix** command are listed in Figure 26.

Attribute Flag and Argument	Purpose
<b>-become</b> <i>Name</i>	Specifies the CMVC user ID to validate your authority to perform this action, only if the CMVC user ID differs from your login. (Environment Variable: CMVC_BECOME)
<b>-component</b> <i>Name</i>	Specifies the component that manages the files that need to be changed. (Environment Variable: CMVC_COMPONENT)
<b>-defect</b> <i>Number ...</i>	Specifies one or more defect identifiers for the fix records.
<b>-developer</b> <i>Name</i>	When creating a fix record, specifies the user ID of the owner of the fix record.
<b>-family</b> <i>Name</i>	Specifies the family for which this command is being issued. (Environment Variable: CMVC_FAMILY)
<b>-feature</b> <i>Number ...</i>	Specifies one or more feature identifiers for the fix records.
<b>-release</b> <i>Name</i>	Specifies the release to which this fix record applies. (Environment Variable: CMVC_RELEASE)
<b>-to</b> <i>Name</i>	When assigning a fix record, specifies the user ID of the new owner.
<b>-verbose</b>	Indicates that you want to see a confirmation message after you issue this command.

Figure 26. Fix Attribute Flags

## Examples

The following are examples of **Fix** command actions:

1. Assume that file changes are required to resolve defect 909 in release 21gos. The component `graphix` manages the files that need to be changed. To create a fix record, type:

```
Fix -create -defect 909 -component graphix -release 21gos
```

A fix record is created to monitor changes made to files in the `graphix` component to resolve defect 909 for the release 21gos. The owner of the new fix record is the owner of the `graphix` component.

2. Assume that you own a fix record that monitors changes made to files in the `debugr` component for the track referencing feature 955 in release 21gos. To reassign that fix record to `joe1` if your `CMVC_RELEASE` environment variable is set to 21gos, type:

```
Fix -assign -feature 955 -component debugr -to joe1
```

The fix record is now owned by `joe1`. If your `CMVC_RELEASE` environment variable was not set to the proper release, you would have had to use the **-release** attribute flag.

3. Assume additional file changes are required to files managed by component `graphix` for the track referencing defect 412 and release `font38`. Also, assume that the `CMVC_COMPONENT` and the `CMVC_RELEASE` environment variables are set to `graphix` and `font38`, respectively, and the track is in the fix state. To reactivate the fix record, type:

```
Fix -activate -defect 412
```

The fix record for defect 412 in the component and release specified in the environment variables is moved to the active state and additional file changes can now be checked in.

## **Related Information**

See commands: **Component, Defect, Feature, File, Release, Track.**

---

## Chapter 12. Host (Hostcmd)

Use the **Host** command (or the **Hostcmd** command for the OS/2 client) to create and delete entries on a CMVC user's host list. Each entry identifies client access for a user ID on one host, and consists of a user ID and a host in the format *login@hostName*. The **Host** command is used in conjunction with the **User** command when initially creating a new user ID. A host list is attached to a user ID and must have at least one entry to establish client access for the user. Additional entries can be defined to allow a user to complete CMVC commands from multiple hosts (and logins).

A CMVC superuser must create the first host list entry for a new user ID. The owner of the user ID can make subsequent entries to gain client access on the hosts where he or she has logins. Each user ID can have multiple host list entries.

When using the **-become** flag or the CMVC\_BECOME environment variable, you require an entry on the host list of the user ID specified by **-become**. This gives you authority to act on behalf of that user ID.

Host list entries can be deleted; however, a user ID must always have one host list entry to be able to access CMVC. If all host list entries are deleted for a user ID, only a CMVC superuser can create a host list entry to reestablish client access for that user ID.

### Syntax

The syntax statements for the **Host** command are:

```
Host -create   Name ... -family Name [ -login Name ] [ -become Name ]  
                [ -verbose ]
```

```
Host -delete  Name ... -family Name [ -login Name ] [ -become Name ]  
                [ -verbose ]
```

### Action Flags

The action flags of the **Host** command and their required authority are listed in Figure 27.

---

Action Flag and Argument	Purpose	Implicit Authority	Explicit Authority
<b>-create</b> Name ...	Creates one or more host list entries for an existing user ID, using the format <i>login@hostName</i> . The login is optional if it matches the user's current login.  The initial host list entry for each user must be created by someone with CMVC superuser privilege.	Owner of the User ID	Superuser

---

Figure 27 (Part 1 of 2). Host Action Flags

Action Flag and Argument	Purpose	Implicit Authority	Explicit Authority
-delete <i>Name ...</i>	Deletes one or more host list entries for an existing user ID, using the format <i>login@hostName</i> . The login is optional if it matches the user's current login. Each user must have at least one host list entry to have CMVC access.	Owner of the User ID	Superuser

Figure 27 (Part 2 of 2). Host Action Flags

## Attribute Flags

The attribute flags of the **Host** command are listed in Figure 28.

Attribute Flag and Argument	Purpose
-become <i>Name</i>	Specifies the CMVC user ID to validate your authority to perform this action, only if the CMVC user ID differs from your login. (Environment Variable: CMVC_BECOME)
-family <i>Name</i>	Specifies the family for which this command is being issued. (Environment Variable: CMVC_FAMILY)
-login <i>Name</i>	Specifies the CMVC user ID for which you want to create or delete a host list entry.
-verbose	Indicates that you want to see a confirmation message after you issue this command.

Figure 28. Host Attribute Flags

## Examples

The following are examples of **Host** command actions:

1. Assume that your login on host `lab1` is `jane` and you have an identical CMVC user ID that has a host list entry for `lab1`; that is, the CMVC user ID `jane` has a host entry for `jane@lab1`. You also have the login `jane` on two other hosts, `lab2` and `lab3` (which should also be on the network), and you want to use CMVC on those hosts as well. To give yourself CMVC client access on these additional hosts, type:

```
Host -create lab2 lab3 -login jane
```

The host list entries `jane@lab2` and `jane@lab3` are created for your existing CMVC user ID `jane`. Because your login on the current host is identical to the CMVC user ID for which you are making a host list entry, the command could also be:

```
Host -create lab2 lab3
```

2. Assume that `jane` and `pete` are logins on host `lab2`. To give them access to the admin CMVC user ID at that host, type:

```
Host -create jane@lab2 pete@lab2 -login admin
```

Adding the above host list entries for the user ID `admin` allows logins `jane` and `pete` to perform CMVC commands using the CMVC user ID `admin`, while logged

on to the host lab2. They can use the **-become** attribute flag to move between the admin user ID and their own user ID.

**Note:** You assume the access authority of the user ID you specify.

3. Assume that you are logged on to host lab2 with the CMVC user ID jane. That user ID does not have superuser privilege. You have a host list entry for user ID admin (that is, jane@lab2 is on the host list for admin), and that user ID does have superuser privilege. To give the user ID george superuser privilege, you must become user admin to issue the command successfully. Type:

```
User -modify george +super -become admin
```

The user ID george is given superuser privilege.

4. Assume that your user ID joan has a number of host list entries, one for the host johnson.kap.uwo.com. To delete that entry from the host list associated with your user ID, type:

```
Host -delete joan@johnson.kap.uwo.com
```

Your user ID, joan, can no longer perform CMVC commands from that host.

5. To see all the host list entries for user ID shirley, type:

```
User -view shirley -long
```

## Related Information

See command: **User**.





---

## Chapter 13. Level

Use the **Level** command to create and delete levels, commit the file changes related to levels, extract the file tree represented by levels, and obtain information about existing levels.

A level group is a set of file changes for a release. To create a level, you assign a name to it and relate it to a release. You then define a set of tracks as *level members*. (For information on how to define tracks as level members, refer to Chapter 14, "LevelMember (Levelmem)" on page 69.) These tracks represent the files that have been changed in relation to that level. If you create a level, you become the level owner by default. You can reassign ownership of the level to another user.

A level can be extracted at any time after tracks are added as level members. A *delta file tree*, which contains only the files that have been changed for the level, is extracted by default. You can also extract a *full file tree* that contains all of the files for the level, once the level has been committed.

Combining the delta file tree (for a current level) with a full file tree (for the last committed level) results in a complete directory structure of all files in a release. This directory structure incorporates the new file changes. You can compile this directory structure and test it to make sure that the results are acceptable. This process of making file changes, extracting a delta and a full file tree, combining the two into a new directory structure and compiling it, can be repeated as needed.

When you want to make permanent all file changes associated with the level, you can move the level to the commit state. To do this, all level member tracks must be in the integrate or commit state, and all prerequisite and corequisite tracks must be included in the level. You also need explicit access authority to commit a level.

If you have explicit access authority, you can indicate when a level is ready for formal testing by specifying that the level is complete. This action changes the state of the associated tracks to test if an environment list exists for the release associated with the tracks, otherwise, the tracks move to the complete state.

If a level has been committed, you can extract a full file tree that includes all the files in the associated release at the version that was current when the level was committed. You can process and distribute a committed level for testing; however, you cannot modify a level or the file changes associated with that level after it has been committed.

**Note:** If Network File System\*\* (NFS\*\*) server daemons are running on a host, you can extract a level to that host and specify a directory location for the file tree. The directory to which you are extracting must be exported via the NFS system. The NFS client daemons must also be running on the CMVC server.

## Syntax

The syntax statements for the **Level** command are:

<b>Level -assign</b>	<i>Name ... -to Name -release Name -family Name</i> [ <b>-become</b> <i>Name</i> ] [ <b>-verbose</b> ]
<b>Level -check</b>	<i>Name ... -release Name -family Name</i> [ <b>-long</b> ] [ <b>-become</b> <i>Name</i> ] [ <b>-verbose</b> ]
<b>Level -commit</b>	<i>Name ... -release Name -family Name</i> [ <b>-become</b> <i>Name</i> ] [ <b>-verbose</b> ]
<b>Level -complete</b>	<i>Name ... -release Name -family Name</i> [ <b>-become</b> <i>Name</i> ] [ <b>-verbose</b> ]
<b>Level -create</b>	<i>Name ... -release Name -family Name</i> [ <b>-type</b> <i>Name</i> ]* [ <b>-become</b> <i>Name</i> ] [ <b>-verbose</b> ]
<b>Level -delete</b>	<i>Name ... -release Name -family Name</i> [ <b>-become</b> <i>Name</i> ] [ <b>-verbose</b> ]
<b>Level -extract</b>	<i>Name ... -release Name -family Name -root Name</i> <b>-node</b> <i>Name</i> [ <b>-full</b> ] [ <b>-nokeys</b> ] [ <b>-fmask</b> <i>Octal_number</i> ] [ <b>-dmask</b> <i>Octal_number</i> ] [ <b>-uid</b> <i>Number</i> ] [ <b>-gid</b> <i>Number</i> ] [ <b>-become</b> <i>Name</i> ] [ <b>-verbose</b> ]
<b>Level -modify</b>	<i>Name ... -release Name -family Name</i> { <b>-name</b> <i>Name</i> <b>-type</b> <i>Name</i> } [ <b>-become</b> <i>Name</i> ] [ <b>-verbose</b> ]
<b>Level -view</b>	<i>Name ... -release Name -family Name</i> [ <b>-long</b> ] [ <b>-become</b> <i>Name</i> ] [ <b>-verbose</b> ]

**Note:** Arguments marked with an asterisk (\*) are required when no default value is set for the CMVC family.

## Action Flags

The action flags of the **Level** command and their required authority are listed in Figure 29 on page 65.

Action Flag and Argument	Purpose	Implicit Authority	Explicit Authority
-assign <i>Name</i> ...	Assigns ownership of levels to another user ID.	Level owner	LevelAssign
-check <i>Name</i> ...	Lists the outstanding prerequisite and corequisite tracks for the specified levels.	Level owner	LevelCheck
-commit <i>Name</i> ...	Moves the specified levels to the commit state where they can no longer be modified. <sup>1</sup> All file changes associated with level members become permanent.	N/A	LevelCommit
-complete <i>Name</i> ...	Moves the specified levels to the complete state where they are ready to be tested. All level members change to the test or complete state.	N/A	LevelComplete
-create <i>Name</i> ...	Creates levels with the specified names. <sup>1</sup> The user who creates a level is the level owner by default.	Release owner	LevelCreate

Figure 29 (Part 1 of 2). Level Action Flags

Action Flag and Argument	Purpose	Implicit Authority	Explicit Authority
-delete <i>Name</i> ...	Deletes the specified levels before they are committed.	Level owner	LevelDelete
-extract <i>Name</i> ...	Creates a file tree by extracting the files defined by the member tracks of specified levels. The default is to extract only changed files.  <b>Note:</b> When extracting multiple levels, you must specify the level names in the chronological order in which they were committed or created.	Level owner	LevelExtract
-modify <i>Name</i> ...	Changes the name or type of the specified levels. Type is configured by family. Use the <b>Report</b> command to find out the level types for your family.	Level owner	LevelModify
-view <i>Name</i> ...	Shows all current information for the specified levels.	Level owner	LevelView

<sup>1</sup> To perform this action, the associated release's process must include the level subprocess.

Figure 29 (Part 2 of 2). Level Action Flags

## Attribute Flags

The attribute flags of the **Level** command are listed in Figure 30 on page 66.

Attribute Flag and Argument	Purpose
<b>-become</b> <i>Name</i>	Specifies the CMVC user ID to validate your authority to perform this action, only if the CMVC user ID differs from your login. (Environment Variable: CMVC_BECOME)
<b>-dmask</b> <i>Octal_number</i>	Specifies the read, write, and execute directory permissions for extracted files in octal notation. Default is 750 (read, write and execute access for directory owner, read and execute access for others in the owner's group, and no access for all other users).
<b>-family</b> <i>Name</i>	Specifies the family for which this command is being issued. (Environment Variable: CMVC_FAMILY)
<b>-fmask</b> <i>Octal_number</i>	Specifies the read, write, and execute file permissions for extracted files in octal notation. Default is the file's mode less the write permission for the file owner, others in the owner's group and all others.
<b>-full</b> <i>Name</i>	Includes all of the files in a release in the extracted file tree. Use to extract a full file tree. Only available for committed levels.
<b>-gid</b> <i>Number</i>	Assigns group ownership of extracted files by specifying the internal number that uniquely identifies the group to the system. The default group value assigned to the extracted files is the CMVC family's <i>GID</i> .
<b>-long</b>	Displays a detailed version of current information for levels, including all track members for <b>-view</b> ; details about prerequisites and corequisites for <b>-check</b> .

Figure 30 (Part 1 of 2). Level Attribute Flags

Attribute Flag and Argument	Purpose
<b>-name</b> <i>Name</i>	Renames an existing level.
<b>-node</b> <i>Name</i>	Specifies a remote host on which to place an extracted file tree. Use the <b>-root</b> attribute flag with this attribute flag. NFS must be installed and running on the remote host system.
<b>-nokeys</b>	Indicates that you do not want to substitute assigned values in place of keywords imbedded in the extracted files.
<b>-release</b> <i>Name</i>	Specifies the release associated with the level. (Environment Variable: CMVC_RELEASE)
<b>-root</b> <i>Name</i>	Specifies a directory on the designated remote host where the extracted file tree is to be placed. This attribute flag is used only in conjunction with the <b>-node</b> flag.
<b>-to</b> <i>Name</i>	Specifies the new level owner when assigning a level.
<b>-type</b> <i>Name</i>	Specifies the type of level when creating or modifying a level. A default type might be established for your family. (Use the <b>Report</b> command to find out the level types for your family.)
<b>-uid</b> <i>Number</i>	Assigns user ownership of extracted files by specifying the internal number that uniquely identifies the user to the system. The default owner assigned to the extracted files is the CMVC family's UID.
<b>-verbose</b>	Indicates that you want to see a confirmation message after you issue this command.

Figure 30 (Part 2 of 2). Level Attribute Flags

## Examples

The following are examples of **Level** command actions:

1. Assume that level type has a default value. To create a level called 9032 for the 21debugr release, type:

```
Level -create 9032 -release 21debugr
```

A new level called 9032 is created for the 21debugr release. You are its owner, and it is in the working state. Use the **LevelMember** command to add tracks to the level.

2. Assume level 9032 is in the integrate state. To check whether any outstanding prerequisites or corequisites exist in level 9032 for release 21debugr, type:

```
Level -check 9032 -release 21debugr
```

Any existing unsatisfied prerequisite and corequisite tracks required for level 9032 for the 21debugr release are listed.

3. To commit level 9029 when your CMVC\_RELEASE environment variable is set to the release associated with the level, type:

```
Level -commit 9029
```

Level 9029 for the release defined by the CMVC\_RELEASE environment variable is committed. At this point, all track members of this level move to the commit state, committing all files changed in relation to those tracks.

4. Assume that you want to extract all of the files for the committed level 9032 to a specific directory and host. Also assume that the directory **/tmp** has been exported on the host johnson.kap.uwo.com with write permission given to the CMVC server. To place the full file tree in the **/tmp** directory of the host johnson.kap.uwo.com, type:

```
Level -extract 9032 -release 21debugr -full -node  
johnson.kap.uwo.com -root /tmp
```

The full file tree is placed relative to the **/tmp** directory on the host named johnson.kap.uwo.com. A full file tree produces a snapshot of all the files in a release at the time the level was committed.

5. Assume that you own level b1992 and that your CMVC\_RELEASE environment variable is set to the release associated with that level. To assign the level to user ID sara, type:

```
Level -assign b1992 -to sara
```

The person with the CMVC user ID sara becomes the new owner of the level b1992 for the release defined by the CMVC\_RELEASE environment variable. If the environment variable was set differently, you would have had to use the **-release** attribute flag to specify the appropriate release for the level.

6. To view information about level b1992 for release debugr, including all of its level members, type:

```
Level -view b1992 -release debugr -long
```

## Related Information

Files that have been deleted or renamed in the current level must be deleted from an extracted file tree. CMVC creates a file named **.gone** that specifies the full path name of each file deleted or renamed that has not already been committed. This file is extracted with the files in a delta tree extraction. Extracting the delta file tree of an uncommitted level extracts all files listed in the **.gone** file.

After merging a delta tree with a *base file tree*, or when extracting noncommitted levels, run the following command from the top of the extracted file tree to remove deleted and renamed files from the tree:

```
xargs rm < .gone
```

Where **.gone** is a file created as part of the extraction that contains the names of all of the deleted and renamed files.

When you extract a committed level, any files that were contained in that level when it was committed are accessed. When you destroy a file, the database record for that file is destroyed but the file remains on the server, so that previously committed levels can be re-built.

See commands: **Defect, Feature, File, LevelMember, Report, Track.**

For a list of supported keywords, refer to the *IBM CMVC User's Reference*.

---

## Chapter 14. LevelMember (Levelmem)

Use the **LevelMember** command (or the **LevelMem** command on the OS/2 client) to specify the tracks you want to include in a given level. The tracks must be in the fix state or the integrate state. A single track can be a member of more than one level. After a track is committed in a level, the other levels in which it is a member ignore the committed track.

By making a track part of a level, you associate the files changed in relation to that track with the specified level. These files must be members of the release associated with the level.

You cannot create level members for, or delete level members from, a level after it is committed.

### Syntax

The syntax statements for the **LevelMember** command are:

```
LevelMember -create  -level Name -release Name -family Name
                       { -defect Number ... -feature Number ... }
                       [ -become Name ] [ -verbose ]
```

```
LevelMember -delete -level Name -release Name -family Name
                       { -defect Number ... -feature Number ... }
                       [ -become Name ] [ -verbose ]
```

### Action Flags

The action flags of the **LevelMember** command and their required authority are listed in Figure 31.

Action Flag	Purpose	Implicit Authority	Explicit Authority
-create	Creates tracks as members of a specific level.  <b>Note:</b> To perform this action, the associated release's process must include the level subprocess.	Level owner	MemberCreate
-delete	Deletes tracks as members of a specific level.	Level owner	MemberDelete

Figure 31. LevelMember Action Flags

## Attribute Flags

The attribute flags of the **LevelMember** command are listed in Figure 32.

Attribute Flag and Argument	Purpose
<b>-become</b> <i>Name</i>	Specifies the CMVC user ID to validate your authority to perform this action, only if the CMVC user ID differs from your login. (Environment Variable: CMVC_BECOME)
<b>-defect</b> <i>Number ...</i>	Specifies defects to identify the tracks you want to include in, or remove from, a level.
<b>-family</b> <i>Name</i>	Specifies the family for which this command is being issued. (Environment Variable: CMVC_FAMILY)
<b>-feature</b> <i>Number ...</i>	Specifies features to identify the tracks you want to include in, or remove from, a level.
<b>-level</b> <i>Name</i>	Specifies the name of the level for which you are creating or deleting level members.
<b>-release</b> <i>Name</i>	Specifies the release associated with this level. (Environment Variable: CMVC_RELEASE)
<b>-verbose</b>	Indicates that you want to see a confirmation message after you issue this command.

Figure 32. *LevelMember* Attribute Flags

## Examples

The following are examples of the **LevelMember** command actions:

1. Assume that level 9012 already exists and you own it. To create level members using the tracks for defect 8761 and 8690 in release 21graphix, type:

```
LevelMember -create -defect 8761 8690 -release 21graphix -level 9012
```

All files changed in reference to the tracks for defects 8761 and 8690 in the release 21graphix are included in level 9012. Level 9012 must be associated with release 21graphix.

2. Assume that you own level 9010. To delete a level member, specifically the track for feature 8744 in the release defined in your CMVC\_RELEASE environment variable, type:

```
LevelMember -delete -feature 8744 -level 9010
```

The track for feature 8744 in the release identified by your CMVC\_RELEASE environment variable is deleted from the level 9010. Level 9010 must be associated with the release defined in the CMVC\_RELEASE environment variable.

## Related Information

See commands: **Defect**, **Feature**, **File**, **Level**, **Report**, **Track**.

To delete a track from a level to make more file changes, you must issue the following commands:

1. **LevelMember -delete** for the track.
2. **Track -fix** to move the track to the fix state.
3. **Fix -activate** to indicate changes are not complete for the track. (Do this for components where files need to be changed.)



After you make the file changes for the track, you must issue the following commands:

1. **Fix -complete** to indicate the fixes are complete.
2. **LevelMember -create** to make the track a member of the level once again.



---

## Chapter 15. Migrate

Use the **Migrate** command to migrate all versions or deltas of specified SCCS text files into the CMVC development environment that uses SCCS as the underlying version control mechanism, and associate them with a component and a release. This migrates all versions of the file so that subsequent development can make use of any previous deltas of the file. You must specify one version of the file at the time of migration as the current version of the file under CMVC control. That file will be the current version in its release. Any other migrated version can be linked to another release for a different development effort.

If you want to associate the files with a release whose process includes the track subprocess, you must also specify a defect or feature identifier. The track associated with the defect or feature identifier in the release must be in the fix state, and the fix records must be in either the active state or the ready state (or not exist).

All SCCS files contain information about the creation of a particular delta and the resulting remarks about it. When migrating SCCS files, CMVC captures this information on a version record for a particular delta only if the user who made the delta has a CMVC user ID. Otherwise, the user ID of the person performing the migration is stored on the version record. If you want to ensure that the people who made the changes are identified on the version records, you must create a CMVC user ID for each of them prior to migrating the files. (For more information on creating user IDs, refer to Chapter 22, "User" on page 107.)

The migrate command is primarily designed for use with the shell scripts Filemap and Filemigrate, which generate the required command line syntax for successfully performing the migration. For more information about these shell scripts, refer to the *IBM CMVC Server Administration and Installation* manual. You can, however, use the **Migrate** command without these shell scripts.

### Syntax

The syntax statement for the **Migrate** command is:

```
Migrate -migrate Name ... -component Name -version Name -family Name  
-release Name [ -relative Name | -top Name ]  
[ -defect Number ... -feature Number ... ]  
[ -become Name ] [ -verbose ]
```

## Action Flags

The action flag of the **Migrate** command and its required authority is listed in Figure 33.

Action Flag and Argument	Purpose	Implicit Authority	Explicit Authority
<b>-migrate</b> <i>Name ...</i>	Migrates the named SCCS files to the CMVC development environment. All versions of the specified files will be migrated. Migrating creates a record for the files and a version record for each version.	Component owner	FileAdd

Figure 33. Migrate Action Flag

## Attribute Flags

The attribute flags of the **Migrate** command are listed in Figure 34.

Attribute Flag and Argument	Purpose
<b>-become</b> <i>Name</i>	Specifies the CMVC user ID to validate your authority to perform this action, only if the CMVC user ID differs from your login. (Environment Variable: CMVC_BECOME)
<b>-component</b> <i>Name</i>	Specifies the component that will manage the file. (Environment Variable: CMVC_COMPONENT)
<b>-defect</b> <i>Number ...</i>	If the track subprocess is included in the release process, specifies the defects to identify tracks for the files.
<b>-family</b> <i>Name</i>	Specifies the family for which this command is being issued. (Environment Variable: CMVC_FAMILY)
<b>-feature</b> <i>Number ...</i>	If the track subprocess is included in the release process, specifies the features to identify tracks for the files.
<b>-relative</b> <i>Name</i>	Indicates that the files can be accessed relative to the specified directory location.
<b>-release</b> <i>Name</i>	Specifies the release with which the files will be associated under CMVC control. (Environment Variable: CMVC_RELEASE)
<b>-top</b> <i>Name</i>	Indicates the leading portion of the path name that is a subset of the current working directory on the client workstation. (Environment Variable: CMVC_TOP)
<b>-verbose</b>	Indicates that you want to receive a confirmation message after you issue this command.
<b>-version</b> <i>Name</i>	Specifies the SCCS version to be linked to the file record and used as a base for future changes to the file. Use any one of the existing SCCS version numbers.

Figure 34. Migrate Attribute Flags

## Examples

The following are examples of **Migrate** command actions:

1. To migrate the SCCS file `s.fileAA` from directory `/u/sccstree/version1/userA` to the CMVC development environment and associate it with release `userA.r1` and component `tvlas`, making version 1.4 the current version, type:

```
Migrate -release userA.r1 -migrate version1/userA/fileAA  
-component tvlas -relative /u/sccstree -version 1.4
```

The migrated file is named **version1/userA/fileAA**, and version 1.4 will be the base for future file changes in reference to the development effort under release `userA.r1`.

2. Assume that your `CMVC_RELEASE` environment variable is set to release `44dev6`, and that this is a release whose process includes the track subprocess. To migrate the `s.fileBB` from the directory `/u/sccstree/version1/userA` to CMVC and associate it with release `44dev6` and component `tvlas`, type:

```
Migrate -migrate version1/userA/fileBB -defect 1199 -component  
tvlas -relative /u/sccstree -version 1.4
```

The migrated file is named **version1/userA/fileBB**, and version 1.4 of the file will be used as the base version for future file changes. The changes made to the file will be referencing the track associated with defect 1199.

## Related Information

See commands: **Component, Defect, Feature, File, Release, Report, Track.**



---

## Chapter 16. Notify

By default, you receive implicit notification when an action is required on your part. To receive additional notification (out of interest), entries can be made to notification lists for specific components.

Use the **Notify** command to create entries on a component notification list, and delete entries from it. Each entry associates a user ID with a preconfigured notification interest group. The interest group identifies the set of actions a user ID is notified of in relation to the component. For a list of the notification interest groups shipped with CMVC, refer to the *IBM CMVC User's Reference*. Current interest groups can be modified and new ones can be defined by your family administrator.

Notification messages are sent to the address specified for each user ID when the user ID is created; the address can be modified using **User -modify**.

A user ID can have more than one entry on the notification list for a given component. Interest groups defined on notification lists are inherited down the component hierarchy, even though the notification lists of child components do not show the notification list entries being inherited from ancestor components.

### Syntax

The syntax statements for the **Notify** command are:

```
Notify -create    -login Name ... -interest Name -component Name  
                  -family Name [ -become Name ] [ -verbose ]  
  
Notify -delete  -login Name ... -interest Name -component Name  
                  -family Name [ -become Name ] [ -verbose ]
```

### Action Flags

The action flags of the **Notify** command and their required authority are listed in Figure 35.

Action Flag	Purpose	Implicit Authority	Explicit Authority
<b>-create</b>	Creates one or more notification list entries for a specified component.	Component owner	NotifyCreate
<b>-delete</b>	Deletes one or more notification list entries from the specified component. Owners of user IDs do not need special authority to delete their user IDs from a notification list.	Component owner, Owner of user ID	NotifyDelete

Figure 35. Notify Action Flags

## Attribute Flags

The attribute flags of the **Notify** command are listed in Figure 36.

Attribute Flag and Argument	Purpose
<b>-become</b>	Specifies the CMVC user ID to validate your authority to perform this action, only if the CMVC user ID differs from your login. (Environment Variable: CMVC_BECOME)
<b>-component <i>Name</i></b>	Specifies the component associated with the notification list. (Environment Variable: CMVC_COMPONENT)
<b>-family <i>Name</i></b>	Specifies the family for which this command is being issued. (Environment Variable: CMVC_FAMILY)
<b>-interest <i>Name</i></b>	Specifies a preconfigured notification interest group for the specified user ID.
<b>-login <i>Name ...</i></b>	Specifies one or more CMVC user IDs as members of the notification list.
<b>-verbose <i>Name</i></b>	Indicates that you want to see a confirmation message after you issue this command.

Figure 36. Notify Attribute Flags

## Examples

The following are examples of **Notify** command actions:

1. You own the `graphix` component. To create notification list entries for the owners of user IDs `pam`, `jack`, and `lisa` with general notification interest for that component, type:

```
Notify -create -login pam jack lisa -interest general -component graphix
```

Three entries are made to the notification list associated with the `graphix` component, one for each of the user IDs you specified with the **-login** attribute flag. Each is notified when an action configured in the `general` notification interest group is performed in reference to the `graphix` component or to any child components of that component. If this group includes the `DefectOpen` action, then these users are notified each time a defect is opened against the `graphix` component.

2. You do not own the `debugr` component, but you want to remove a notification list entry for that component for your own user ID, `pam`. To remove the notification list entry that gives you `developer` interest at that component, type:

```
Notify -delete -login pam -interest developer -component debugr
```

You are no longer notified when actions configured for the `developer` interest group are performed in reference to the `debugr` component or its child components.



## Related Information

See commands: **Component, Report.**

Use the **Report** command to view the interest groups and the actions they include.  
For example:

- **Report -vi interest | pg**

For a list of the notification groups shipped with CMVC, refer to the *IBM CMVC User's Reference*.

See your family administrator, or read the *IBM CMVC Server Administration and Installation* manual for information about configuring new notification interest groups and modifying existing ones.



---

## Chapter 17. Release

Use the **Release** command to create, modify, delete, and recreate releases, extract the set of files associated with a release, link files within releases with those in other releases, and view information about existing releases.

A release group is a set of files that must be built, tested, and distributed as a whole. Release names must be unique within a family, and a release must be created in relation to a component to manage access and notification for the release. If you create a release, you become its owner and you have implicit authority to define an approval list and an environment list for that release.

When creating a release, you must choose a preconfigured process for the release using the **-process** flag. A process groups different combinations of CMVC subprocesses. CMVC subprocesses determine the states of the tracks within a release. For release processes, the track, approval, fix, level, and test CMVC subprocesses can be specified. Processes are configured by your family administrator who can modify current processes and define new ones. For a list of the valid release processes and the CMVC subprocesses they include, use the **Report -view cfgrelproc** command.

You can change the process for an existing release using the **-modify** flag. For more information on how CMVC subprocesses relate to the states of CMVC objects, refer to the book *IBM CMVC Concepts*.

To modify an environment or a tester for a release, use the **Environment** command (or **Environ** command if you are using the OS/2 client). See Chapter 8, "Environment (Environ)" on page 33 for more information.

To modify an approver for a release, use the **Approver** command to add or delete an approver. See Chapter 4, "Approver" on page 17 for more information.

You cannot delete releases that have files, outstanding tracks, noncommitted levels, or active sizing records associated with them. You cannot reuse the name of a deleted release, but you can recreate a deleted release and modify the name of a recreated release.

When you link files in one release to those in another, you can link the current or the committed version of each active file. The current version is the default setting. If you are linking files to a release whose process includes the track subprocess, you must supply the defect or feature identifier for the associated tracks.

When you extract files associated with a release, the current version of the associated files is extracted by default. Alternatively, you can extract files changed after a certain date. For a release whose process includes the track and level subprocesses, you can extract the last committed version of the files.

**Note:** If NFS server daemons are running on a host, you can extract a release to that host and specify a directory location for the file tree. The directory to which you are extracting must be exported via the NFS system. The NFS client daemons must also be running on the CMVC server.

## Syntax

The syntax statements for the **Release** command are:

```

Release -create  Name... -component Name -process Name
                  -family Name [-environment Name -tester Name ]*
                  [-approver Name ]* [-description Text ] [-owner Name ]
                  [-become Name ] [-verbose ]

Release -delete Name ... -family Name [-become Name ] [-verbose ]

Release -extract Name ... -node Name -root Name -family Name [-nokeys ]
                  [-date yy/mm/dd | -committed ] [-fmask Octal_number ]
                  [-dmask Octal_number ] [-uid Number ] [-gid Number ]
                  [-become Name ] [-verbose ]

Release -link   Name ... -to Name [-date yy/mm/dd | -committed ]
                  [-defect Number ... -feature Number ...]* -family Name
                  [-become Name] [-verbose]

Release -modify Name ... -family Name { -name Name
                  -component Name [-process Name ]
                  [-environment Name -tester Name ]*
                  [-approver Name ]* -description Text -owner Name }
                  [-become Name ] [-verbose ]

Release -recreate Name ... -family Name [-environment Name -tester Name ]*
                  [-approver Name ]* [-become Name ] [-verbose ]

Release -view   Name ... -family Name [-processInfo ]
                  [-become Name ] [-verbose ]

```

**Note:** Arguments marked with an asterisk (\*) are required only when their related subprocess has been specified for the release.

## Action Flags

The action flags of the **Release** command and their required authority are listed in Figure 37.

Action Flag and Argument	Purpose	Implicit Authority	Explicit Authority
<b>-create</b> <i>Name ...</i>	Creates releases with the specified names.	N/A	ReleaseCreate
<b>-delete</b> <i>Name ...</i>	Deletes the specified releases. Releases cannot have files, outstanding tracks, noncommitted levels, or active sizing records associated with them.	Release owner	ReleaseDelete
<b>-extract</b> <i>Name ...</i>	Extracts the file tree for the specified releases. By default, the current version of all files in the releases are extracted.	Release owner	ReleaseExtract
<b>-link</b> <i>Name ...</i>	Links the active files in a specified release to those in another specified release. If the <b>-committed</b> or <b>-date</b> flag is not supplied, the current version of each file is linked by default.	Release owner	ReleaseLink

Figure 37 (Part 1 of 2). Release Action Flags

Action Flag and Argument	Purpose	Implicit Authority	Explicit Authority
<b>-modify</b> <i>Name ...</i>	Modifies the properties of the specified releases:		
	<b>-process</b> <b>-owner</b> <b>-name</b>	Release owner	ReleaseModify
	<b>-component</b>	Release owner	ReleaseModify and ReleaseCreate in the new component.
<b>-recreate</b> <i>Name ...</i>	Recreate previously deleted releases.	Release owner	ReleaseRecreate
<b>-view</b> <i>Name ...</i>	Shows all current information for the specified releases.	Release owner	ReleaseView

Figure 37 (Part 2 of 2). Release Action Flags

## Attribute Flags

The attribute flags of the **Release** command are listed in Figure 38.

Attribute Flag and Argument	Purpose
<b>approver</b> <i>Name</i>	Specifies the user ID of the approver to be added to the approver list if the approval subprocess is included in the release process.
<b>-become</b> <i>Name</i>	Specifies the CMVC user ID to validate your authority to perform this action, only if the CMVC user ID differs from your login. (Environment Variable: CMVC_BECOME)
<b>-committed</b>	Indicates that the last committed version of the files in the release are to be extracted or linked.
<b>-component</b> <i>Name</i>	Specifies the component associated with the release when creating or modifying the release. The environment variable is not used for <b>Release -modify</b> . (Environment Variable: CMVC_COMPONENT)
<b>-date</b> <i>Date</i>	Indicates that files modified in the release since the specified date are to be extracted or linked.
<b>-defect</b> <i>Number ...</i>	Specifies the defect identifier if the track subprocess is included in the process of the release to which files are being linked.
<b>-description</b> <i>Text</i>	Adds a description of a release when creating or modifying it.
<b>-dmask</b> <i>Octal_number</i>	Specifies the read, write, and execute directory permissions for the extracted files in octal notation. Default is 750 (read, write and execute access for directory owner, read and execute access for others in the owner's group, and no access for all other users).
<b>-environment</b> <i>Name</i>	Specifies the environment in which the testing is to be done if the test subprocess is included in the release process. (The tester/environment name combination becomes an entry on the environment list for the release.)

Figure 38 (Part 1 of 2). Release Attribute Flags

Attribute Flag and Argument	Purpose
<b>-family</b> <i>Name</i>	Specifies the family for which this command is being issued. (Environment Variable: CMVC_FAMILY)
<b>-feature</b> <i>Number ...</i>	Specifies the feature identifier if the track subprocess is included in the process of the release to which files are being linked.
<b>-fmask</b> <i>Octal_number</i>	Specifies the read, write, and execute file permissions for the extracted files in octal notation. Default is the file's mode less the write permission for the file owner, others in the owner's group, and all others.
<b>-gid</b> <i>Number</i>	Specifies ownership of extracted files by identifying the internal number that uniquely identifies the group to the system. The default group value assumed for the extracted files is the CMVC family's GID.
<b>-name</b> <i>Name</i>	Specifies a new name for an existing release when renaming it.
<b>-node</b> <i>Name</i>	Specifies a remote host on which to place the extracted file tree. Use the <b>-root</b> attribute flag in conjunction with this attribute flag.
<b>-nokeys</b> <i>Name</i>	Indicates that you do not want to substitute assigned values in place of keywords imbedded in the files when the files are extracted.
<b>-owner</b> <i>Name</i>	Specifies the user ID of the release owner when creating or modifying a release.
<b>-processInfo</b>	Displays the current process setting and its associated CMVC subprocesses for the specified releases when used with the <b>-view</b> action flag.
<b>-process</b> <i>Name</i>	Specifies a process when creating or modifying a release. Processes for your environment are configured by your family administrator. For a list of the valid release processes and the CMVC subprocesses they include, use the <b>Report -view cfgrelproc</b> command.
<b>-root</b> <i>Name</i>	Specifies the directory on the designated host where the extracted file tree is to be placed.
<b>-tester</b> <i>Name</i>	Specifies the user responsible for testing in the given environment if the test subprocess is included in the release process. (The tester/environment name combination becomes an entry on the environment list for the release.)
<b>-to</b> <i>Name</i>	Specifies the associated release that contains the files you want to link.
<b>-uid</b> <i>Number</i>	Specifies ownership of extracted files by identifying the internal number that uniquely identifies the user to the system. The default owner of the extracted files is the CMVC family's <i>UID</i> .
<b>-verbose</b>	Specifies that you want to see a confirmation message after you issue this command.

Figure 38 (Part 2 of 2). Release Attribute Flags

## Examples

The following are examples of **Release** command actions:

1. To create a release named 10debugr with the process preship (which specifies the CMVC track, approval, fix, level and test subprocesses), type:

```
Release -create 10debugr -process preship -environment PCVersion1
-tester john -approver jack
```

The release 10debugr is created with preship as its process. As the preship process includes the CMVC test subprocess, an environment, PCVersion1, and an initial tester, john, are specified. And because the preship process includes the CMVC approval subprocess, an approver, jack, is specified.

2. You own release 10debugr. To make pam the new owner, type:  
Release -modify 10debugr -owner pam  
Pam now owns the release 10debugr.
3. You own release 10debugr. To change the process associated with the release to prototype, type:  
Release -modify 10debugr -process prototype  
The release now has prototype as its process.
4. To link the committed version of release 10debugr to release 20debugr, type:  
Release -link 10debugr -to 20debugr -committed -defect 12  
The committed version of the files in 10debugr are linked to release 20debgr.
5. You own the 21graphix release. To extract all of the files associated with that release that have been changed since January 18, 1993, and write the files to the **/tmp/test/graphix** directory of the host astro, type:  
Release -extract 21graphix -node astro -root /tmp/test/graphix -date 93/01/18 -gid 2 -uid 210  
A file tree is created on the machine astro relative to the location **/tmp/test/graphix**. This file tree represents all files associated with the 21graphix release that have been changed since January 18, 1993. NFS must be installed and running on astro and the directory identified by the **-root** flag must be exported so that CMVC can write to it.

## Related Information

See commands: **Approver, Component, Environment, File, Track.**

For a list of supported keywords, refer to the *IBM CMVC User's Reference*.





---

## Chapter 18. Report

Use the **Report** command to *query* the tables and views associated with CMVC and generate output showing the results of that query. CMVC uses the information provided in a **Report** command to build an SQL SELECT statement. The **-view** flag specifies the database table or view to query, and the **-where** flag specifies the selection criteria for the query.

You can issue queries to generate reports of data from tables and views with the **-view** action flag. If you do not specify selection criteria, such as the fields and the *search* conditions you want to use, the report query selects all entries for the table or view indicated. The **-help** flag displays a list of valid table and view names that you can use as arguments for the **-view** flag. All view and table names as well as their corresponding fields are listed in the *IBM CMVC User's Reference*.

Views are also available to report all inherited notification list members for a specified component; these are designated by the postfix UpView. You must specify a component in the selection criteria of the **Report** command to query this view.

Views are available to report all objects of a certain type for all descendents of a specified component; these views are designated by the postfix DownView. Downviews are valid for **Defect**, **Feature**, **Access**, and **Notify**. You must specify a component in the selection criteria of the **Report** command to query this view.

The *Text* argument of the **-where** attribute flag defines the search criteria and the conditions of the data you want to select, and it must follow Structured Query Language (SQL) syntax rules for the database used by your CMVC installation. It can include subselects and valid SQL functions. For a discussion of SQL syntax and its use, refer to your database product documentation.

By default, report results are displayed in 132 column tabular format, but you can request the output to be displayed in 80 column stanza format or in long format, which is a combination of stanza and tabular formats. You can also request output in raw format if you want the results to be used in another program or utility.

### Syntax

The syntax statements for the **Report** command are:

```
Report -help          -family Name [ -become Name ] [ -verbose ]
Report -testClient  -family Name [ -become Name ] [ -verbose ]
Report -testServer -family Name [ -become Name ] [ -verbose ]
Report -view       Name -family Name [ -where Text ] [ -become Name ]
                    [ -stanza | -raw | -table | -long ] [ -verbose ]
```

## Action Flags

The action flags of the **Report** command and their required authority are listed in Figure 39.

Action Flag and Argument	Purpose	Implicit Authority	Explicit Authority
<b>-help</b>	Displays a list of the valid view and table names you can use as arguments for the <b>-view</b> flag.	N/A	N/A
<b>-view Name</b>	Specifies the database table or view you want to query. You can use a unique prefix abbreviation of the table and view names.	N/A	N/A
<b>-testClient</b>	Tests the availability of the CMVC message catalog on the client's host, and returns a message informing the user of its availability.	N/A	N/A
<b>-testServer</b>	Tests the availability of the CMVC message catalog on the CMVC server, and returns a message informing the user of its availability.	N/A	N/A

Figure 39. Report Action Flags

## Attribute Flags

The attribute flags of the **Report** command are listed in Figure 40.

Attribute Flag and Argument	Purpose
<b>-become Name</b>	Specifies the CMVC user ID to validate your authority to perform this action, only if your client login differs from your CMVC user ID. (Environment Variable: CMVC_BECOME)
<b>-family Name</b>	Specifies the family for which this command is being issued. (Environment Variable: CMVC_FAMILY)
<b>-long</b>	Produces report output in stanza format, with additional important information shown in tabular format: <ul style="list-style-type: none"><li>• Each database record is a stanza.</li><li>• Each stanza line consists of a field and its corresponding values.</li></ul>
<b>-raw</b>	Produces report output in raw format: <ul style="list-style-type: none"><li>• Information retrieved from each field is separated by the vertical bar delimiter.</li><li>• Each line of output corresponds to one database record.</li></ul> For the order and description of field names that are output for various views, refer to Appendix A, "Report-Raw Output" on page 115.
<b>-stanza</b>	Produces report output in stanza format: <ul style="list-style-type: none"><li>• Each database record is a stanza.</li><li>• Each stanza line consists of a field and its corresponding values.</li></ul>

Figure 40 (Part 1 of 2). Report Attribute Flags

Attribute Flag and Argument	Purpose
<b>-table</b>	<p>Produces report output in tabular format:</p> <ul style="list-style-type: none"> <li>• Each field is displayed as a column heading</li> <li>• Field values appear under respective column headings</li> <li>• Each row corresponds to one database record</li> </ul> <p>This is the default format of report output.</p>
<b>-verbose</b>	Indicates that you want to see a confirmation message after you issue this command.
<b>-where Text</b>	Defines the selection criteria to query the specified table or view using valid SQL syntax.

Figure 40 (Part 2 of 2). Report Attribute Flags

## Examples

The following are examples of **Report** command actions:

1. To display all users who have developer authority for the graphix component, type:

```
Report -view accessView -where "compname = 'graphix' AND
authorityName='developer'"
```

The above command could be abbreviated to:

```
Report -vi accessv -w "compname = 'graphix' AND
authorityName='developer'"
```

This shows access explicitly defined for the graphix component. Additional access may be inherited at the component level.

2. To display all the approval records for the 20graphix release that were updated on or after December 1, 1993, type:

```
Report -view approvalview -where "releasename='20graphix' AND
lastupdate > '93/12/01'"
```

Because date fields include the date of the action as well as the time of the action, the approval records selected using the above example are those that were updated after 12:00 p.m. on November 30, 1993. The date field must be enclosed in single quotation marks because it is a character type field.

3. To display all members of the approver list for any of the debugger releases, type:

```
Report -view approverView -where "releasename like '%debug%'"
```

The percent sign (%) is a wildcard character used with the **like operator** to match zero or more characters. For a more granular search, you can use the underscore (\_) wildcard character instead to match a single character.

4. To display all authority groups that include the LevelCommit action, type:

```
Report -view authority -where "action = 'LevelCommit'"
```

5. To display all actions that are included in the definition of the general authority group, type:

```
Report -view authority -where "name = 'general'"
```

6. To display all levels for the 20debugr release that have been updated on or after April 29, 1993, type:

```
Report -view levelView -where "releaseName='20debugr' AND
lastUpdate > '93/04/29'"
```

Because date fields include date and time, the levels selected using the above example are those that were updated after 12:00 p.m. on April 28, 1993. The date field must be enclosed in single quotation marks because it is a character type field.

7. To display, in raw format, all levels that were committed earlier than March 3, 1993, type:

```
Report -view levelView -where "commitDate < '93/03/03'" -raw
```

The levels committed on or before 12:00 p.m. on March 3, 1993 are selected. The date field must be enclosed in single quotation marks because it is a character type field.

8. To display all returned defects originated by the user ID jack, type:

```
Report -view defectView -where "originLogin = 'jack' AND state =
'returned'"
```

9. To display all defects for the graphix component that are in the working state, type:

```
Report -view defectView -where "state = 'working' AND compName =
'graphix'"
```

10. To display all defects in the open or working state that are owned by users in the area e50, type:

```
Report -view defectView -where "state in ('open','working') AND
ownerArea='e50'"
```

If some user areas are E50, they are not selected.

11. To display all release environment list entries that designate the user jack as the tester of the PCVersion1 environment, type:

```
Report -view envView -where "userlogin = 'jack' AND name = 'PCVersion1'"
```

12. To display the release environment list members for the 21debugr release, type:

```
Report -view envView -where "releaseName = '21debugr'"
```

13. To display member files of the 10debugr release that were last updated on or after August 8, 1993, type:

```
Report -view fileView -where "releaseName = '10debugr' AND lastUpdate >
'93/08/07'"
```

The date field must be in the yy/mm/dd format, and it must be enclosed in single quotation marks because it is a character type field. Because the date field includes date and time, all files updated after August 7, 1993 at 12:00 p.m. are selected.

14. To display member files of the 20graphix release that currently are checked out, type:

```
Report -view filesOutView -where "releaseName = '20graphix'"
```

15. To display all actions that define the developer interest group, type:  

```
Report -view interest -where "Name = 'developer'"
```
16. To display all the notes for defect 7627 that were added before September 2, 1993, type:  

```
Report -view noteView -where "defectName = '7627' AND addDate < '93/09/02'"
```

The date field must be in the yy/mm/dd format, and it must be enclosed in single quotation marks because it is a character type field.
17. To display all the notes for defect 4866 written by the owners of the user IDs sam and sara, type:  

```
Report -view noteView -where "defectName = '4866' AND (userlogin = 'sam' OR userlogin = 'sara')"
```
18. To display all test records for the environment PCVersion2 that have reject or abstain test results recorded, type:  

```
Report -view testView -where "envName = 'PCVersion2' AND (state = 'reject' OR state = 'abstain')"
```
19. To display all test records for the defect 9821 that have an environment name beginning with PCV, type:  

```
Report -view testView -where "envName like 'PCV%' AND defectName = '9821'"
```
20. To display all existing tracks for defect 5490 that are in the fix state, type:  

```
Report -view trackView -where "defectName = '5490' AND state = 'fix'"
```
21. To display all existing tracks created on or after September 17, 1993 for the 21debugr release, type:  

```
Report -view trackView -where "releasename = '21debugr' AND addDate > '93/09/16'"
```
22. To display all users in areas that include tools as part of the area name, type:  

```
Report -view users -where "area like '%tools%'"
```
23. To display all users who have CMVC superuser privilege, type:  

```
Report -view users -where "superuser = 'yes'"
```
24. To display the most recently created defect, type:  

```
Report -view defectview -where "id=(select max(id) from defects where prefix in (select name from Config where type = 'defectPrefix'))"
```
25. To display an order by clause with two column names, type:  

```
Report -view changeview -where "defectName = '1491' AND releaseName = 'projectA_re11' order by versionSID asc, pathName desc"
```

Asc orders the path name column in ascending order and desc orders the path name column in descending order. This query reports changes to files in ascending order, and the path names of the files in descending order.

26. To display all tracks that are in the integrate state and that are not in a level, type:

```
Report -vi trackview -w "state='integrate'  
AND releasename='projectA_r1'  
and id not in (select trackid from levelmembers)"
```

27. To display all file changes for `src/kernel/ibmesa/io/dkios.c` that were committed in a level on or before October 21, 1993, type:

```
Report -vi changeview -w "pathname =  
'src/kernel/ibmesa/io/dkios.c' and levelname in  
(select name from levelview where commitdate < '93/10/21')"
```

## Related Information

For the order and description of field names that are output for various views when you issue the **Report** command using the **-raw** flag, refer to Appendix A, “Report-Raw Output” on page 115. For a list of the views and a description of their fields, or a list of the tables that can be specified as subselects in the **-where** clause and a description of their fields, refer to the *IBM CMVC User's Reference*.

Basic SQL rules for defining queries are:

1. The views and their columns can be typed in any manner, if you supply the full name of the view or column name. You can use all uppercase letters, all lowercase letters, or a mixture of both when typing the names of the views and their columns. You must supply the full name of the view and column; however, views can be abbreviated except as part of a subselect in a **-where** clause.
2. When searching for specific values you must type the value of the field exactly as it exists in the database. The database values are case sensitive.
3. Enclose text with imbedded blanks in single quotation marks.
4. Date fields include date and time. The correct format is *yy/mm/dd hh:mm:ss*.
5. Use the following relational operators (also called *comparison operators*) to describe a relationship between two values:

=	Equal to.
<> or !=	Not equal to.
>	Greater than. The difference for different data types is: <ul style="list-style-type: none"><li>• Character: later in the alphabet (where lowercase letters are greater than uppercase letters, and uppercase letters are greater than numbers)</li><li>• Date: later date.</li></ul>
>=	Greater than or equal to. The difference for different data types is: <ul style="list-style-type: none"><li>• Character: later in the alphabet or equal to</li><li>• Date: later date.<sup>1</sup></li></ul>
<	Less than. The difference for different data types is: <ul style="list-style-type: none"><li>• Character: earlier in the alphabet</li><li>• Date: earlier date.</li></ul>

<b>&lt;=</b>	Less than or equal to. The difference for different data types is: <ul style="list-style-type: none"> <li>• Character: earlier in the alphabet or equal to</li> <li>• Date: earlier date.<sup>1</sup></li> </ul>
<b>in</b>	Search for one or more items you specify in a list.
<b>not in</b>	Search for any items that do not appear on the list you specify.
<b>like</b>	Search for a string similar to the one you specify; use wildcard characters in place of other characters to expand the search.
<b>between</b>	Search for items falling between two items that you type.
<b>is null</b>	Search for values that are set to null or those for which no values have been assigned. Null differs from the values of zero (0) and blank ( ).

<sup>1</sup> Selections in reports for date fields using <= or >= return the same information as if you entered < or >, respectively. This is because the date data type consists of a date and a time. Use the like operator with the < or > operators respectively, to return <= or >= information. See examples 2, 6, and 27 in the “Examples” section of this chapter.

- Use the following wildcard characters in place of other characters in a string:
  - % represents zero or more characters in a string.
  - \_ represents one character in a string.
- Issue search conditions that are connected by keywords such as AND, OR, NOT.
- Enclose values for char fields in single quotation marks.
- Subselects can be defined. For a list of tables and descriptions of their fields, refer to the *IBM CMVC User's Reference*.
- SQL functions can be used.





---

## Chapter 19. Size

Use the **Size** command to create, delete, and reassign sizing records for a defect or feature that is in the size state, or to indicate sizing information. A sizing record must be created explicitly by the defect or feature owner. A sizing record indicates the time and resources needed to resolve a defect or implement a feature in one component for a release. Each sizing record is uniquely identified by a defect or feature identifier, a component, and a release.

If you are the owner of the component in which the defect must be resolved or the feature must be implemented then, by default, you are also the owner of the sizing record. Sizing information must be entered as text on a sizing record.

All sizing records must be marked either with accept or reject in order to move the defect or feature from the size state to the review state. Tracks and fix records are created for all sizing records marked accept, when the defect or feature is accepted.

### Syntax

The syntax statements for the **Size** command are:

```
Size -accept      { -defect Number ... -feature Number ... }  
                  -component Name ... -release Name -family Name  
                  -sizing Text [ -become Name ] [ -verbose ]  
  
Size -assign     -to Name { -defect Number ... -feature Number ... }  
                  -component Name ... -release Name -family Name  
                  [ -become Name ] [ -verbose ]  
  
Size -create     { -defect Number ... -feature Number ... }  
                  -component Name ... -release Name -family Name  
                  [ -become Name ] [ -verbose ]  
  
Size -delete     { -defect Number ... -feature Number ... }  
                  -component Name ... -release Name -family Name  
                  [ -become Name ] [ -verbose ]  
  
Size -reject     { -defect Number ... -feature Number ... }  
                  -component Name ... -release Name -family Name  
                  [ -become Name ] [ -verbose ]
```

### Action Flags

The action flags of the **Size** command and their required authority are listed in Figure 41.

---

Action Flag and Argument	Purpose	Implicit Authority	Explicit Authority
<b>-accept</b>	Indicates that the sizing information is entered and complete for the corresponding defect, feature, release, and component. This action is used to record initial sizing information.	Sizing record owner	SizeAccept

---

Figure 41 (Part 1 of 2). Size Action Flags

Action Flag and Argument	Purpose	Implicit Authority	Explicit Authority
<b>-assign</b>	Reassigns ownership of the specified sizing record to another user ID.	Sizing record owner	SizeAssign
<b>-create</b>	Creates a sizing record for the corresponding defect, feature, release, and component.  <b>Note:</b> To perform this action, the associated component's process must include the DSR subprocess.	Defect or Feature owner	SizeCreate
<b>-delete</b>	Deletes the specified sizing record.	Defect or Feature owner	SizeDelete
<b>-reject</b>	Indicates that resolving the defect or implementing the feature does not require changes in the corresponding component. (If old sizing information exists for this sizing record, it is deleted.)	Sizing record owner	SizeReject

Figure 41 (Part 2 of 2). Size Action Flags

## Attribute Flags

The attribute flags of the **Size** command are listed in Figure 42.

Attribute Flag and Argument	Purpose
<b>-become Name</b>	Specifies the CMVC user ID to validate your authority to perform this action, only if the CMVC user ID differs from your login. (Environment Variable: CMVC_BECOME)
<b>-component Name ...</b>	Specifies the components for these sizing records. (Environment Variable: CMVC_COMPONENT)
<b>defect Number ...</b>	Specifies the defect identifier for these sizing records.
<b>-family Name</b>	Specifies the family for which this command is being issued. (Environment Variable: CMVC_FAMILY)
<b>-feature Number ...</b>	Specifies the feature identifier for these sizing records.
<b>-release Name</b>	Specifies the release for this sizing record. (Environment Variable: CMVC_RELEASE)
<b>-sizing Text</b>	Specifies sizing information for the proposed defect or feature change in the specified component and release.
<b>-to Name</b>	Specifies the user ID of the user who is responsible for sizing the specified defect or feature, when assigning ownership of a sizing record.
<b>-verbose</b>	Indicates that you want to see a confirmation message after you issue this command.

Figure 42. Size Attribute Flags

## Examples

The following are examples of **Size** command actions:

1. To create a sizing record for feature 483 in the graphix component for the 21graphix release, type:

```
Size -create -feature 483 -component graphix -release 21graphix
```

A sizing record is created for feature 483 in the graphix component for the 21graphix release. Ownership defaults to the owner of the graphix component.

2. You own the sizing record for feature 483 in the graphix component for the 20graphix release. To assign the sizing record to user ID mary, type:

```
Size -assign -feature 483 -component graphix -release 20graphix -to mary
```

Ownership, and thus the sizing responsibility, of the sizing record is reassigned to the user mary. The sizing record is uniquely defined by the feature number 483, release 20graphix, and component graphix.

3. Assume you own the sizing record for feature 483 in component graphix for release 21graphix. To specify that 10 person days are required to implement that feature, type:

```
Size -accept -feature 483 -component graphix -release 21graphix -sizing "10 person days"
```

The sizing information is entered for feature 483 in component graphix for the release 21graphix.

4. Assume you also own the sizing record for defect APAR20 in component charting for release 32charting. If no changes are required in that component, and thus no additional resources are required, type:

```
Size -reject -defect APAR20 -component charting -release 32charting
```

The **-reject** action flag indicates that no changes are required for defect APAR20 in the component charting for the release 32charting.

## Related Information

See commands: **Feature**, **Defect**, **Release**, **Report**, **Track**.



---

## Chapter 20. Test

Use the **Test** command to indicate the results of an environment test on a test record associated with a track. A track is referenced by a defect identifier and a release, or a feature identifier and a release.

If a release has an environment list, test records are created according to the entries in that list whenever a new track is created for that release (providing that the release's process includes the test subprocess). Each test record includes the environment name and user ID specified on the release environment list, and the defect or feature identifier of the track. The owner of the user ID is the tester, and owns the test record.

Test records are activated (that is, they are moved to the ready state) when the associated track moves to the test state and the proposed change (whether for resolving a defect or implementing a feature) is ready for environment testing. When results are entered for all the environment test records associated with a track, the state of that track changes to complete. Even if you reject a test record, the track changes to the complete state. You should open another defect or feature to address the changes still required.

### Syntax

The syntax statements for the **Test** command are:

```
Test -abstain    { -defect Number ... -feature Number ... } -family Name  
                  -release Name ... -environment Name ... [ -tester Name ]  
                  [ -become Name ] [ -verbose ]  
  
Test -accept    { -defect Number ... -feature Number ... } -family Name  
                  -release Name ... -environment Name ... [ -tester Name ]  
                  [ -become Name ] [ -verbose ]  
  
Test -assign    -to Name { -defect Number ... -feature Number ... }  
                  -release Name ... -environment Name ... -family Name  
                  [ -tester Name ] [ -become Name ] [ -verbose ]  
  
Test -reject    { -defect Number ... -feature Number ... } -family Name  
                  -release Name ... -environment Name ... [ -tester Name ]  
                  [ -become Name ] [ -verbose ]
```

### Action Flags

The action flags of the **Test** command and their required authority are listed in Figure 43.

Action Flag	Purpose	Implicit Authority	Explicit Authority
<b>-abstain</b>	Abstains from testing for a release environment.	Owner of test record	TestAbstain
<b>-accept</b>	Indicates successful results for a release environment test.	Owner of test record	TestAccept

Figure 43 (Part 1 of 2). Test Action Flags

Action Flag	Purpose	Implicit Authority	Explicit Authority
-assign	Assigns a test record to another user ID.	Owner of test record	TestAssign
-reject	Indicates unsuccessful results for a release environment test.	Owner of test record	TestReject

Figure 43 (Part 2 of 2). Test Action Flags

## Attribute Flags

The attribute flags of the **Test** command are listed in Figure 44.

Attribute Flag and Argument	Purpose
-become <i>Name</i>	Specifies the CMVC user ID to validate your authority to perform this action, only if the CMVC user ID differs from your login. (Environment Variable: CMVC_BECOME)
-defect <i>Number ...</i>	Specifies the defects associated with the test records. Testing must be done to determine whether or not these defects were resolved in the specified environments.
-environment <i>Name ...</i>	Specifies the environment in which testing must be done.
-family <i>Name</i>	Specifies the family for which this command is being issued. (Environment Variable: CMVC_FAMILY)
-feature <i>Number ...</i>	Specifies the features associated with the test records. Testing must be done to determine whether or not these features were properly implemented in the specified environments.
-release <i>Name ...</i>	Specifies the release associated with the test records. (Environment Variable: CMVC_RELEASE)
-tester <i>Name</i>	Performs <b>Test</b> actions on a test record owned by a different user. By naming the tester, this flag identifies the test record to be modified.
-to <i>Name</i>	When reassigning ownership of a test record, specifies the user ID of the user who is to become the new tester.
-verbose	Indicates that you want to see a confirmation message after you issue this command.

Figure 44. Test Attribute Flags

## Examples

The following are examples of **Test** command actions:

1. Assume that you are responsible for testing whether or not defect 7966 was successfully resolved in the PCVersion1 environment for release tripod3. To accept the test record for that track if your CMVC\_RELEASE environment variable is set to tripod3, type:

```
Test -accept -defect 7966 -environment PCVersion1
```

The test record you own is marked accept, indicating successful test results. The track moves to the complete state if this is the last test record for the track to be marked with test results.

2. Assume that you have superuser privilege and that your `CMVC_RELEASE` environment variable is set to `tripod3`. To indicate that jane, the owner of the test record for defect 7966 in release `tripod3`, will abstain from marking test results in the `PCVersion2` environment, type:

```
Test -abstain -defect 7966 -environment PCVersion2 -tester jane
```

The test record owned by the user `jane` for defect 7966 in the environment `PCVersion2` is marked `abstain`. You could mark the test record owned by another user because you have superuser privilege. The track moves to the complete state if this is the last test record for the track to be marked with test results.

3. Assume that you own the test record for feature 7562 in release `20gos` and environment `Model1`. To assign this test record to user ID `bob` so that the owner of that user ID assumes the testing responsibility, type:

```
Test -assign -to bob -feature 7562 -environment Model1 -release 20gos
```

## Related Information

See commands: **Defect**, **Environment**, **Feature**, **Level**, **Release**, **Report**, **Track**.





---

## Chapter 21. Track

The **Track** command is used to create, modify, reassign, delete, and view information about a track, and to change the state of a track. The states a track moves through depends on the CMVC subprocesses included in the associated release process. A release process can include the track, approval, fix, level, or test subprocesses, or none at all. For more information on the track states and their associated subprocesses, refer to the book *IBM CMVC Concepts*.

The purpose of a track is to monitor the progress of changes to resolve a defect or implement a feature.

A track does not have its own name or number; instead, a track is identified by a release and a defect identifier or by a release and a feature identifier. The user who creates the track becomes the owner of the track unless a different owner is specified when the track is created.

If a defect or feature is linked to more than one release, multiple tracks exist for that defect or feature. The tracks required for a defect or feature are created according to the accepted sizing records, when the defect or feature changes to the working state. Defect or feature owners can create additional tracks if the defect or feature is in the working state.

To determine the prerequisite and corequisite tracks for a particular track, use the **Track -check** action. Specify the level name to determine the prerequisite and corequisite tracks relative to an earlier committed level. You will get a list of all prerequisite and corequisite tracks including those for file changes which were committed after the commit date of the specified level.

### Syntax

The syntax statements for the **Track** command are:

<b>Track -assign</b>	<b>-to</b> <i>Name</i> { <b>-defect</b> <i>Number</i> ... <b>-feature</b> <i>Number</i> ... } <b>-release</b> <i>Name</i> ... <b>-family</b> <i>Name</i> [ <b>-become</b> <i>Name</i> ] [ <b>-verbose</b> ]
<b>Track -cancel</b>	{ <b>-defect</b> <i>Number</i> ... <b>-feature</b> <i>Number</i> ... } <b>-release</b> <i>Name</i> ... <b>-family</b> <i>Name</i> [ <b>-become</b> <i>Name</i> ] [ <b>-verbose</b> ]
<b>Track -check</b>	{ <b>-defect</b> <i>Number</i> ... <b>-feature</b> <i>Number</i> ... } <b>-release</b> <i>Name</i> ... <b>-family</b> <i>Name</i> [ <b>-level</b> <i>Name</i> ] [ <b>-become</b> <i>Name</i> ] [ <b>-verbose</b> ]
<b>Track -commit</b>	{ <b>-defect</b> <i>Number</i> ... <b>-feature</b> <i>Number</i> ... } <b>-release</b> <i>Name</i> ... <b>-family</b> <i>Name</i> [ <b>-become</b> <i>Name</i> ] [ <b>-verbose</b> ]
<b>Track -complete</b>	{ <b>-defect</b> <i>Number</i> ... <b>-feature</b> <i>Number</i> ... } <b>-release</b> <i>Name</i> ... <b>-family</b> <i>Name</i> [ <b>-become</b> <i>Name</i> ] [ <b>-verbose</b> ]
<b>Track -create</b>	{ <b>-defect</b> <i>Number</i> ... <b>-feature</b> <i>Number</i> ... } <b>-release</b> <i>Name</i> ... <b>-family</b> <i>Name</i> [ <b>-owner</b> <i>Name</i> ] [ <b>-target</b> <i>Name</i> ] [ <b>-become</b> <i>Name</i> ] [ <b>-verbose</b> ]
<b>Track -fix</b>	{ <b>-defect</b> <i>Number</i> ... <b>-feature</b> <i>Number</i> ... } <b>-release</b> <i>Name</i> ... <b>-family</b> <i>Name</i> [ <b>-become</b> <i>Name</i> ] [ <b>-verbose</b> ]
<b>Track -integrate</b>	{ <b>-defect</b> <i>Number</i> ... <b>-feature</b> <i>Number</i> ... } <b>-release</b> <i>Name</i> ... <b>-family</b> <i>Name</i> [ <b>-become</b> <i>Name</i> ] [ <b>-verbose</b> ]

**Track -modify**     **-target** *Name* { **-defect** *Number* ... **-feature** *Number* ... }  
**-release** *Name* ... **-family** *Name* [ **-become** *Name* ]  
[ **-verbose** ]

**Track -test**        { **-defect** *Number* ... **-feature** *Number* ... } **-release** *Name* ...  
**-family** *Name* [ **-become** *Name* ] [ **-verbose** ]

**Track -view**        { **-defect** *Number* ... **-feature** *Number* ... } **-family** *Name*  
**-release** *Name* ... [ **-long** ] [ **-become** *Name* ] [ **-verbose** ]

## Action Flags

The action flags of the **Track** command and their required authority are listed in Figure 45.

Action Flag	Purpose	Implicit Authority	Explicit Authority
<b>-assign</b>	Reassigns ownership of specified tracks to another user ID.	Track owner	TrackAssign
<b>-cancel</b>	Cancels the specified tracks. This is valid only if no changes have been made to files referencing the tracks.	Defect owner, Feature owner	TrackCancel
<b>-check</b>	Displays the prerequisite and corequisite tracks for the specified tracks.	Track owner	TrackCheck
<b>-commit</b>	Changes the state of the specified tracks from integrate to commit, when no file changes have been made for the track. This is required only if the track is not committed in a level.	Track owner	TrackCommit
<b>-complete</b>	Changes the state of the specified tracks from test to complete when no file changes have been made for the track. No additional state changes can occur after a track reaches the complete state.	Track owner	TrackComplete
<b>-create</b>	Creates a track for the specified defect or feature in a given release. <sup>1</sup> If there is no approver list for the related release, then the new track is created in the fix state. If an approver list exists, the initial state is approve.	Defect owner, Feature owner	TrackCreate
<b>-fix</b>	Moves the specified tracks from the integrate state to the fix state.	Track owner	TrackFix
<b>-integrate</b>	Changes the state of the specified tracks from fix to the next valid state governed by the release's process. For a release whose process includes the level subprocess, this action is only valid if no file changes have been made for the track and the track is not committed in a level.	Track owner	TrackIntegrate

Figure 45 (Part 1 of 2). Track Action Flags

Action Flag	Purpose	Implicit Authority	Explicit Authority
<b>-modify</b>	Modifies the target field for the specified tracks.	Track owner	TrackModify
<b>-test</b>	Changes the state of the specified tracks from commit to test. This is required only if the track is not committed in a level. Normally this occurs when you issue <b>Level -complete</b> .	Track owner	TrackTest
<b>-view</b>	Shows all information for the specified tracks.	Track owner	TrackView

<sup>1</sup> To perform this action, the associated release's process must include the track subprocess.

Figure 45 (Part 2 of 2). Track Action Flags

## Attribute Flags

The attribute flags of the **Track** command are listed in Figure 46.

Attribute Flag and Argument	Purpose
<b>-become Name</b>	Specifies the CMVC user ID to validate your authority to perform this action, only if the CMVC user ID differs from your login. (Environment Variable: CMVC_BECOME)
<b>-defect Number ...</b>	Specifies the defect identifiers if the tracks are created to resolve defects.
<b>-family Name</b>	Specifies the family for which this command is being issued. (Environment Variable: CMVC_FAMILY)
<b>-feature Number ...</b>	Specifies the feature identifiers if the tracks are created to implement features.
<b>-level Name</b>	Specifies the name of the committed level to use as a base when determining prerequisite and corequisite tracks for a given track.
<b>-long</b>	Displays detailed track information including the track approval records, the file changes associated with the track, the fix records and test records for the track, and the levels in which the track is a member.
<b>-owner Name</b>	When creating a track, specifies the user ID of a user who is to be the track owner.
<b>-release Name ...</b>	Specifies the releases associated with the tracks. (Environment Variable: CMVC_RELEASE)
<b>-target Name</b>	Specifies a target, such as a level in which the track is to be included as a level member, or a date on which you expect to complete the fixes for the track.
<b>-to Name</b>	When assigning ownership of a track, specifies the user you want to assume track owner responsibility.
<b>-verbose</b>	Specifies that you want to receive a confirmation message after you issue this command.

Figure 46. Track Attribute Flags

## Examples

The following are examples of **Track** command actions:

1. To create tracks for a defect that requires resolution for three releases, type:

```
Track -create -defect 8734 -release 20graphix 10graphix 21charting
```

Three tracks are created for defect 8734, one for each of the three releases.

2. To reassign owner responsibility for a track to the user ID jack, type:

```
Track -assign -feature 8803 -release 20graphix -to jack
```

3. Assume that the track for defect 8734 and the release specified by your CMVC\_RELEASE environment variable is not a member of a level. To change the track from integrate state to fix state, type:

```
Track -fix -defect 8734
```

The track is moved from the integrate state to the fix state. If a track is in a level, you must delete it from the level before you can move it back to the fix state.

4. To check whether prerequisite or corequisite tracks exist for a track relative to a particular level, type:

```
Track -check -defect 8734 -release 10graphix -level 9028
```

All prerequisite and corequisite tracks that exist for the track for defect 8734 in the 10graphix release are displayed, including those for file changes committed after the commit date of the level 9028.

5. To view information about a specified track associated with the release set in the CMVC\_RELEASE environment variable, type:

```
Track -view -defect 8667
```

## Related Information

See commands: **Approval**, **Coreq**, **Defect**, **Feature**, **Fix**, **Level**, **Release**, **Report**, **Size**, **Test**.

---

## Chapter 22. User

Use the **User** command to create new user IDs, to modify information associated with user IDs, and delete user IDs. Superuser privilege is required to create user IDs for new users, delete other user IDs, and modify the superuser privilege of a user. You can modify your own user ID information, but cannot give yourself CMVC superuser privilege.

Although the **User** command establishes required user ID information, actual host access for a user ID must be created with the **Host** command.

User IDs cannot be deleted if they have work pending or they own objects. You cannot use a deleted user ID but you can recreate it and rename it.

**Note:** Because your family administrator can create new fields, the attributes for the **-create** and **-modify** actions listed in this section may be different from those in your family. Those listed here represent the shipped default fields only. For a list of the field properties and flags in use in your family, use the **User -configInfo** command or see your family administrator. For more information on configurable fields, refer to the book *IBM CMVC Server Administration and Installation*.

### Syntax

The syntax statements for the **User** command are:

```
User -configInfo  -family name [ -become Name ] [ -raw ]
User -create     -login Name -address Name -family Name [ -name Text ]
                  [ -area Name ] [ +super ] [ -become Name ] [ -verbose ]
User -delete     Name ... -family Name [ -become Name ] [ -verbose ]
User -modify     Name ... -family Name { -login Name -name Text -address
                  Name -area Name [ +super | -super ] } [ -become Name ]
                  [ -verbose ]
User -recreate   Name ... -family Name [ -become Name ] [ -verbose ]
User -view       Name ... -family Name [ -long ] [ -become Name ]
                  [ -verbose ]
```

### Action Flags

The action flags of the **User** command and their required authority are listed in Figure 47.

---

Action Flag and Argument	Purpose	Implicit Authority	Explicit Authority
-configInfo	Shows configurable field properties for users in the specified family. (The information is returned in a fixed ASCII table format.)	N/A	N/A
-create	Adds user IDs by specifying a value for the <b>-login</b> flag. User IDs must be unique within a family.	N/A	Superuser

---

Figure 47 (Part 1 of 2). User Action Flags

Action Flag and Argument	Purpose	Implicit Authority	Explicit Authority
-delete <i>Name ...</i>	Deletes specified user IDs.	N/A	Superuser
-modify <i>Name ...</i>	Modifies information associated with specified user IDs. Only a superuser can modify superuser privilege.	Owner of user ID	Superuser
-recreate <i>Name ...</i>	Recreates previously deleted user IDs.	N/A	Superuser
-view <i>Name ...</i>	Shows information about specified user IDs.	N/A	N/A

Figure 47 (Part 2 of 2). User Action Flags

## Attribute Flags

The attribute flags of the **User** command are listed in Figure 48.

Attribute Flag and Argument	Purpose
-address <i>Name</i>	Specifies a user's mail address in the form login@hostname.
-area <i>Name</i>	Specifies the work area or department of a user.
-become <i>Name</i>	Specifies the CMVC user ID to validate your authority to perform this action, only if the CMVC user ID differs from your login. (Environment Variable: CMVC_BECOME)
-family <i>Name</i>	Specifies the family for which this command is being issued. (Environment Variable: CMVC_FAMILY)
-login <i>Name</i>	Specifies a CMVC user ID.
-long	Displays user information plus the hosts associated with this user ID.
-name <i>Text</i>	Specifies the user's full name.
-raw	Produces report output in raw format: <ul style="list-style-type: none"> <li>Information retrieved from each field is separated by the vertical bar delimiter.</li> <li>Each line of output corresponds to one database record.</li> </ul>
+super	Grants CMVC superuser privilege to a specified user ID.
-super	Removes CMVC superuser privilege for a specified user ID.
-verbose	Specifies that you want to receive a confirmation message after you issue this command.

Figure 48. User Attribute Flags

## Examples

The following are examples of **User** command actions:

1. To create a user ID for a new user, type:

```
User -create -login dorrie -address dorrie@cansas -name "Julie Karland"
-area tools
```

A new user is created with the user ID `dorrie` and the mailing address `dorrie@cansas`. Since CMVC superuser authority was not specified, this user does not have this authority.

2. To delete a user ID, type:

```
User -delete jack
```

The user ID jack is deleted. You can delete a user ID only if all associated objects the user ID owns have been deleted or reassigned, and the user ID is removed from approver, environment, access, and notification lists. A deleted user ID can be recreated.

3. To modify information for a user, type:

```
User -modify dorothy +super
```

This grants CMVC superuser privilege for the user with the login name dorothy.

4. To modify information for multiple user IDs, type:

```
User -modify jack sally -area tools07 -super
```

This changes the area specification to tools07 for the user IDs jack and sally. It also removes CMVC superuser privilege for both user IDs.

5. To view information for specified user IDs, type:

```
User -view dorothy jack sally
```

This displays the user information for the specified user IDs.

6. To view the field properties for the User record in family rdev, type:

```
User -configInfo -family rdev
```

## Related Information

See commands: **Host, Report.**





---

## Chapter 23. Verify (Verifycm)

Use the **Verify** command (or the **Verifycm** command for the OS/2 client) to verify the resolution of defects or the implementation of features, or to reassign ownership of existing *verification records*.

A verification record is created for the originator of a defect or a feature when the defect or feature is accepted and the component that manages the defect or feature has a process which includes the defectVerify or featureVerify subprocesses respectively. Additional verification records are created for the originators of duplicate defects or features, and attached to the active defect or feature. Defects can be specified as duplicates of features, and features can be specified as duplicates of defects.

Verification records become active when a defect or feature changes from the working state to the verify state. When results have been recorded for all the verification records for a defect, and when all of that defect's tracks are complete, the defect changes from the verify state to the closed state. The same is true for a feature.

The defect or feature moves to the closed state even if you indicate unsuccessful results by marking your verification with **-reject**. In this case, you should open a new defect or feature to address the changes still required.

### Syntax

The syntax statements for the **Verify** command are:

```
Verify -abstain    { -defect Number ... -feature Number ... } -family Name  
                  [ -tester Name ] [ -become Name ] [ -verbose ]  
  
Verify -accept   { -defect Number ... -feature Number ... } -family Name  
                  [ -tester Name ] [ -become Name ] [ -verbose ]  
  
Verify -assign   -to Name { -defect Number ... -feature Number ... }  
                  -family Name [ -tester Name ] [ -become Name ]  
                  [ -verbose ]  
  
Verify -reject   { -defect Number ... -feature Number ... } -family Name  
                  [ -tester Name ] [ -become Name ] [ -verbose ]
```

### Action Flags

The action flags of the **Verify** command and their required authority are listed in Figure 49.

---

Action Flag and Argument	Purpose	Implicit Authority	Explicit Authority
<b>-abstain</b>	Indicates that the owner of the verification record refrains from verifying defect resolution or feature implementation.	Owner of verification record	VerifyAbstain
<b>-accept</b>	Indicates successful verification of the defect resolution or feature implementation.	Owner of verification record	VerifyAccept

---

Figure 49 (Part 1 of 2). Verify Action Flags

Action Flag and Argument	Purpose	Implicit Authority	Explicit Authority
-assign	Reassigns the ownership of the verification record for a specified defect or feature to another user.	Owner of verification record	VerifyAssign
-reject	Indicates unsuccessful verification of the defect resolution or feature implementation.	Owner of verification record	VerifyReject

Figure 49 (Part 2 of 2). Verify Action Flags

## Attribute Flags

The attribute flags of the **Verify** command are listed in Figure 50.

Attribute Flag and Argument	Purpose
-become <i>Name</i>	Specifies the CMVC user ID to validate your authority to perform this action only if the CMVC user ID differs from your login. (Environment Variable: CMVC_BECOME)
-defect <i>Number ...</i>	Specifies the defect identifiers associated with the verification records.
-family <i>Name</i>	Specifies the family for which this command is being issued. (Environment Variable: CMVC_FAMILY)
-feature <i>Number ...</i>	Specifies the feature identifiers associated with the verification records.
-tester <i>Name</i>	Identifies the owner of the verification record, if you are performing the verification for someone else.
-to <i>Name</i>	When assigning ownership of a verification record, specifies the user ID of the new owner.
-verbose	Specifies that you want to receive a confirmation message after you issue this command.

Figure 50. Verify Attribute Flags

## Examples

The following are examples of **Verify** command actions:

1. To indicate that a defect resolution was verified successfully, type:  
Verify -accept -defect 976
2. To indicate that a defect resolution was not verified successfully, type:  
Verify -reject -defect 1001
3. To reassign ownership of a verification record, type:  
Verify -assign -feature 899 -to lee

If you are a superuser, the current owner of the verification record for feature 899, or you have VerifyAssign explicit authority, then you can type the above line to reassign that verification record to the user ID lee.

## Related Information

See commands: **Defect**, **Feature**, **Release**, **Track**.



---

## Appendix A. Report-Raw Output

This appendix shows the field names for various views and tables, listed in the order that is output by the **-raw** option of the **Report** command. The table and view names that you must use with the **-view** flag are in large bold type in the left margin. Field names in italics provide data output, but cannot be used as search criteria. Specify values for date fields in the format *yy/mm/dd hh:mm:ss*.

When entering a query, do not abbreviate the field names. In most cases you receive only an error message if you abbreviate a field name. However, depending on the database used by your installation, some abbreviations could be interpreted incorrectly and give inaccurate information. For example, if your installation uses an ORACLE\*\* database and you replace the field names **userLogin** or **userName** with **user**, you will not receive an error message; however, the search will return with no records found.

**Note:** The data types listed are for the INFORMIX\*\* and SYBASE\*\* databases. The ORACLE database equivalents are int for number, varchar for char, and text in table for long.

Because CMVC allows your family administrator to modify and delete certain configurable fields and to create new ones, the following field names may differ from the output you receive. For information on the configurable fields used in your environment, see your family administrator.

### AccessDownView

#### Access List Entries, Including Those of Child Components

Fieldname[length]	Datatype	Description
childCompName[63]	char	Child Component Name
userLogin[31]	char	User's CMVC User ID
userName[63]	char	User's Full Name
userArea[31]	char	User's Work Area or Department
authorityName[31]	char	Access Authority Name
authorityType[15]	char	Access Authority Type ('granted' or 'restricted')

### AccessView

#### Component Access List Entries

Fieldname[length]	Datatype	Description
compName[63]	char	Component Name
userLogin[31]	char	User's CMVC User ID
userName[63]	char	User's Full Name
userArea[31]	char	User's Work Area or Department

Fieldname[length]	Datatype	Description
authorityName[31]	char	Access Authority Name
authorityType[15]	char	Access Authority Type ('granted' or 'restricted')

## ApprovalView

### Track Approval Records

Fieldname[length]	Datatype	Description
defectPrefix[31]	char	Defect or Feature Prefix
defectName[31]	char	Defect or Feature Identifier
releaseName[31]	char	Release Name
userLogin[31]	char	Approver's CMVC User ID
userName[63]	char	Approver's Full Name
userArea[31]	char	Approver's Work Area or Department
state[15]	char	Approval Record State
addDate[25]	char	Date Created
lastUpdate[25]	char	Date of Last Update
defectReference[31]	char	Defect or Feature Reference
defectAbstract[127]	char	Defect or Feature Abstract
defectType[7]	char	'defect' or 'feature'

## ApproverView

### Release Approver List Entries

Fieldname[length]	Datatype	Description
releaseName[31]	char	Release Name
userLogin[31]	char	Approver's CMVC User ID
userName[63]	char	Approver's Full Name
userArea[31]	char	Approver's Work Area or Department

## Authority

### Access Authority Groups

Fieldname[length]	Datatype	Description
name[31]	char	Access Authority Group Name
action[15]	char	Name of Action

## Cfgcomproc

### Component Process Names and Subprocesses

Fieldname[length]	Datatype	Description
name[31]	char	Component Process Name
config[15]	char	Name of Active Subprocess

## Cfgrelproc

### Release Process Names and Subprocesses

Fieldname[length]	Datatype	Description
name[31]	char	Release Process Name
config[15]	char	Name of Active Subprocess

## ChangeView

### File Changes

Fieldname[length]	Datatype	Description
releaseName[31]	char	Release Name
defectName[31]	char	Defect or Feature Identifier
levelName[31]	char	Name of Level where Change is Committed
versionSID[47]	char	Version ID of Changed File
pathName[195]	char	File's Full Path Name
type[8]	char	Type of File Change
defectReference[31]	char	Defect or Feature Reference
defectAbstract[127]	char	Defect or Feature Abstract
defectPrefix[31]	char	Defect or Feature Prefix
<i>userLogin</i> [31]	char	CMVC User ID of Person who Made the Change
<i>userName</i> [63]	char	Full Name of Person who Made the Change
<i>userArea</i> [31]	char	Work Area of Person who Made the Change

## CompView

### Component Properties

Fieldname[length]	Datatype	Description
name[63]	char	Component Name
userLogin[31]	char	Component Owner's CMVC User ID
userName[63]	char	Component Owner's Full Name
userArea[31]	char	Component Owner's Work Area or Department
addDate[25]	char	Date Created

Fieldname[length]	Datatype	Description
dropDate[25]	char	Date Deleted
lastUpdate[25]	char	Date of Last Update
description[127]	char	Component Description
compProcess[31]	char	Process Name
featureDSR[3]	char	Feature Design Size Review Subprocess ('yes' or 'no')
featureVerify[3]	char	Feature Verify Subprocess ('yes' or 'no')
defectDSR[3]	char	Defect Design Size Review Subprocess ('yes' or 'no')
defectVerify[3]	char	Defect Verify Subprocess ('yes' or 'no')

## Config

### Configurable Data Definitions

Fieldname[length]	Datatype	Description
type[15]	char	Name of Configuration Data Type
name[31]	char	Data Name
dflt[3]	char	Default Value ('yes' or 'no')
value1	number	not currently used
value2	number	not currently used
description[127]	char	Description of Data Name

## DefectDownView

### Defect Properties, Including Those of Child Components

Fieldname[length]	Datatype	Description
prefix[31]	char	Defect Prefix
name[31]	char	Defect Identifier
childCompName[63]	char	Component Name
releaseName[31]	char	Release Name
ownerLogin[31]	char	Defect Owner's CMVC User ID
state[15]	char	Defect State
answer[31]	char	Accept or Return Answer Type
severity[31]	char	Severity Level
abstract[127]	char	Defect Abstract
age	number	Defect Age
envName[31]	char	Environment where Discovered
levelName[31]	char	Level where Discovered
duplicate[31]	char	Duplicate Defect or Feature Identifier
lastUpdate[25]	char	Date last Updated
addDate[25]	char	Date Created



<b>Fieldname[length]</b>	<b>Datatype</b>	<b>Description</b>
assignDate[25]	char	Date when Reassigned
responseDate[25]	char	Date Accepted or Returned
endDate[25]	char	Date Closed or Canceled
ownerName[63]	char	Defect Owner's Full Name
ownerArea[31]	char	Defect Owner's Work Area or Department
reference[31]	char	Defect Reference
originLogin[31]	char	Defect Originator's CMVC User ID
originName[63]	char	Defect Originator's Full Name
originArea[31]	char	Defect Originator's Work Area or Department

## DefectView

### Defect Properties

<b>Fieldname[length]</b>	<b>Datatype</b>	<b>Description</b>
prefix[31]	char	Defect Prefix
name[31]	char	Defect Identifier
compName[63]	char	Component Name
releaseName[31]	char	Release Name
ownerLogin[31]	char	Defect Owner's CMVC User ID
state[15]	char	Defect State
answer[31]	char	Accept or Return Answer Type
severity[31]	char	Severity Level
abstract[127]	char	Defect Abstract
age	number	Defect Age
envName[31]	char	Environment where Discovered
levelName[31]	char	Level where Discovered
duplicate[31]	char	Duplicate Defect or Feature Identifier
lastUpdate[25]	char	Date last Updated
addDate[25]	char	Date Created
assignDate[25]	char	Date Reassigned
responseDate[25]	char	Date Accepted or Returned
endDate[25]	char	Date Closed or Canceled
ownerName[63]	char	Defect Owner's Full Name
ownerArea[31]	char	Defect Owner's Work Area or Department
reference[31]	char	Defect Reference
originLogin[31]	char	Defect Originator's CMVC User ID
originName[63]	char	Defect Originator's Full Name
originArea[31]	char	Defect Originator's Work Area or Department

Fieldname[length]	Datatype	Description
symptom[171] <sup>1</sup>	char	Symptom or Problem
phaseFound[171] <sup>1</sup>	char	Phase where Discovered
phaseInject[171] <sup>1</sup>	char	Phase where Introduced
priority[171] <sup>1</sup>	char	Priority Type for Resolution Scheduling
target[171] <sup>1</sup>	char	Target for Defect Resolution

<sup>1</sup> These field names are only applicable for the manufacturer default settings.

## EnvView

### Release Environment List Entries

Fieldname[length]	Datatype	Description
name[31]	char	Environment Name
releaseName[31]	char	Release Name
userLogin[31]	char	Tester's CMVC User ID
userName[63]	char	Tester's Full Name
userArea[31]	char	Tester's Work Area or Department

## FeatureDownView

### Feature Properties, Including Those of Child Components

Fieldname[length]	Datatype	Description
prefix[31]	char	Feature Prefix
name[31]	char	Feature Identifier
childCompName[63]	char	Child Component Name
ownerLogin[31]	char	Feature Owner's CMVC User ID
ownerName[63]	char	Feature Owner's Full Name
state[15]	char	Feature State
abstract[127]	char	Feature Abstract
age	number	Feature Age
duplicate[31]	char	Duplicate Defect or Feature Identifier
lastUpdate[25]	char	Date of Last Update
addDate[25]	char	Date Created
assignDate[25]	char	Date Reassigned
responseDate[25]	char	Date Accepted or Returned
endDate[25]	char	Date Closed or Canceled
ownerArea[31]	char	Feature Owner's Work Area or Department
reference[31]	char	Feature Reference
originLogin[31]	char	Feature Originator's CMVC User ID

Fieldname[length]	Datatype	Description
originName[63]	char	Feature Originator's Full Name
originArea[31]	char	Feature Originator's Work Area or Department

## FeatureView

### Feature Properties

Fieldname[length]	Datatype	Description
prefix[31]	char	Feature Prefix
name[31]	char	Feature Identifier
compName[63]	char	Component Name
ownerLogin[31]	char	Feature Owner's CMVC User ID
ownerName[63]	char	Feature Owner's Full Name
state[15]	char	Feature State
abstract[127]	char	Feature Abstract
age	number	Feature Age
duplicate[31]	char	Duplicate Defect or Feature Identifier
lastUpdate[25]	char	Date of Last Update
addDate[25]	char	Date Created
assignDate[25]	char	Date Reassigned
responseDate[25]	char	Date Accepted or Returned
endDate[25]	char	Date Closed or Canceled
ownerArea[31]	char	Feature Owner's Work Area or Department
reference[31]	char	Feature Reference
originLogin[31]	char	Feature Originator's CMVC User ID
originName[63]	char	Feature Originator's Full Name
originArea[31]	char	Feature Originator's Work Area or Department
priority[171] <sup>1</sup>	char	Priority Type for Scheduling
target[171] <sup>1</sup>	char	Target for Feature Implementation

<sup>1</sup> These field names are only applicable for the manufacturer default settings.

## FileView

### File Properties

Fieldname[length]	Datatype	Description
baseName[127]	char	File Base Name
releaseName[31]	char	Release Name
compName[63]	char	Component Name
versionSID[47]	char	Last Committed File Version ID

Fieldname[length]	Datatype	Description
addDate[25]	char	Date Created
dropDate[25]	char	Date Deleted
lastUpdate[25]	char	Date of Last Update
pathName[195]	char	File Path Name
nuVersionSID[47]	char	Current File Version ID
nuAddDate[25]	char	New Creation Date
nuDropDate[25]	char	New Deletion Date
nuPathName[195]	char	New File Path Name
userLogin[31]	char	CMVC user ID who locked or checked out the file. NULL means file not locked or checked out.
fmode[4]	char	File Permission

## FilesOutView

### Files Currently Locked for Editing

Fieldname[length]	Datatype	Description
fileNuPath[195]	char	File Path Name
releaseName[31]	char	Release Name
checkOutDate[25]	char	Date File Locked
newSID[47]	char	New File Version ID
userLogin[31]	char	User's CMVC User ID
userName[63]	char	User's Full Name
userArea[31]	char	User's Work Area or Department

## FixView

### Fix Records

Fieldname[length]	Datatype	Description
defectName[31]	char	Defect or Feature Identifier
releaseName[31]	char	Release Name
compName[63]	char	Component Name
state[15]	char	Fix Record State
userLogin[31]	char	Fix Record Owner's CMVC User ID
userArea[31]	char	Fix Record Owner's Work Area or Department
defectAbstract[127]	char	Defect or Feature Abstract
addDate[25]	char	Date Created
lastUpdate[25]	char	Date of Last Update
userName[63]	char	Fix Record Owner's Full Name
defectPrefix[31]	char	Defect or Feature Prefix

Fieldname[length]	Datatype	Description
defectType[7]	char	'defect' or 'feature'
defectReference[31]	char	Defect or Feature Reference

## HostView

### User Host List Entries

Fieldname[length]	Datatype	Description
login[31]	char	User's Login Name on Host
name[127]	char	Host Name
userLogin[31]	char	User's CMVC User ID
userName[63]	char	User's Full Name
userArea[31]	char	User's Work Area or Department

## Interest

### Notification Interest Groups

Fieldname[length]	Datatype	Description
name[31]	char	Interest Group Name
action[15]	char	Name of Action

## LevelMemberView

### Level Members

Fieldname[length]	Datatype	Description
levelName[31]	char	Level Name
releaseName[31]	char	Release Name
defectName[31]	char	Defect or Feature Identifier
defectReference[31]	char	Defect or Feature Reference
trackUserLogin[31]	char	Track Owner's CMVC User ID
trackUserName[63]	char	Track Owner's Full Name
trackUserArea[31]	char	Track Owner's Work Area or Department
defectPrefix[31]	char	Defect or Feature Prefix

## LevelView

### Level Properties

Fieldname[length]	Datatype	Description
name[31]	char	Level Name
releaseName[31]	char	Release Name
type[31]	char	Level Type
userLogin[31]	char	Level Owner's CMVC User ID

Fieldname[length]	Datatype	Description
userName[63]	char	Level Owner's Full Name
userArea[31]	char	Level Owner's Work Area or Department
addDate[25]	char	Date Created
commitDate[25]	char	Date Committed
lastUpdate[25]	char	Date of Last Update
state[15]	char	Level State

## NoteView

### Defect Notes

Fieldname[length]	Datatype	Description
defectName[31]	char	Defect or Feature Identifier
defectReference[31]	char	Defect or Feature Reference
action[15]	char	Action Occurring when Note Added
addDate[25]	char	Date Note Added
userLogin[31]	char	User's CMVC User ID
userName[63]	char	User's Full Name
userArea[31]	char	User's Work Area or Department
defectPrefix[31]	char	Defect or Feature Prefix
remarks[15999]	long	Text of Remarks

## NotifyDownView

### Notification List Members, Including Descendant Members

Fieldname[length]	Datatype	Description
childCompName[63]	char	Child Component Name
userLogin[31]	char	User's CMVC User ID
userName[63]	char	User's Full Name
userArea[31]	char	User's Work Area or Department
userAddress[159]	char	User's Mailing Address
interestName[31]	char	Interest Group Name

## NotifyUpView

### Notification List Entries, Including Those Inherited from Parent Components

Fieldname[length]	Datatype	Description
parentName[63]	char	Parent Component Name
userLogin[31]	char	User's CMVC User ID
userName[63]	char	User's Full Name
userArea[31]	char	User's Work Area or Department

Fieldname[length]	Datatype	Description
userAddress[159]	char	User's Mailing Address
interestName[31]	char	Interest Group Name

## NotifyView

### Notification List Entries

Fieldname[length]	Datatype	Description
compName[63]	char	Component Name
userLogin[31]	char	User's CMVC User ID
userName[63]	char	User's Full Name
userArea[31]	char	User's Work Area or Department
userAddress[159]	char	User's Sendmail Address
interestName[31]	char	Interest Group Name

## ReleaseView

### Release Properties

Fieldname[length]	Datatype	Description
name[31]	char	Release Name
compName[63]	char	Component Name
relProcess[31]	char	Process Name
userLogin[31]	char	Release Owner's CMVC User ID
userName[63]	char	Release Owner's Full Name
userArea[31]	char	Release Owner's Work Area or Department
addDate[25]	char	Date Created
dropDate[25]	char	Date Deleted
lastUpdate[25]	char	Date of Last Update
description[127]	char	Release Description
track[3]	char	Track Subprocess ('yes' or 'no')
approve[3]	char	Approval Subprocess ('yes' or 'no')
fix[3]	char	Fix Subprocess ('yes' or 'no')
lvl[3]	char	Level Subprocess ('yes' or 'no')
test[3]	char	Test Subprocess ('yes' or 'no')

## SizeView

### Sizing Records

Fieldname[length]	Datatype	Description
featureName[31]	char	Feature or Defect Identifier
featureReference[31]	char	Feature or Defect Reference

<b>Fieldname[length]</b>	<b>Datatype</b>	<b>Description</b>
compName[63]	char	Component Name
releaseName[31]	char	Release Name
sizing[127]	char	Text of Sizing Information
addDate[25]	char	Date Created
state[7]	char	Size Record State
userName[63]	char	Size Record Owner's Full Name
userLogin[31]	char	Size Record Owner's CMVC User ID
userArea[31]	char	Size Record Owner's Work Area or Department
lastUpdate[25]	char	Date of Last Update
featurePrefix[31]	char	Feature or Defect Prefix
featureAbstract[127]	char	Feature or Defect Abstract

## TestView

### Environment Test Records

<b>Fieldname[length]</b>	<b>Datatype</b>	<b>Description</b>
releaseName[31]	char	Release Name
defectPrefix[31]	char	Defect or Feature Prefix
defectName[31]	char	Defect or Feature Identifier
envName[31]	char	Environment Name
state[15]	char	Environment Test Record State
addDate[25]	char	Date Created
lastUpdate[25]	char	Date of Last Update
userLogin[31]	char	Test Record Owner's CMVC User ID
defectAbstract[127]	char	Defect or Feature Abstract
userName[63]	char	Test Record Owner's Full Name
userArea[31]	char	Test Record Owner's Work Area or Department
defectReference[31]	char	Defect or Feature Reference

## TrackView

### Track Properties

<b>Fieldname[length]</b>	<b>Datatype</b>	<b>Description</b>
releaseName[31]	char	Release Name
defectName[31]	char	Defect or Feature Identifier
defectReference[31]	char	Defect or Feature Reference
state[15]	char	Track State
target[31]	char	Target for Completion
addDate[25]	char	Date Created



Fieldname[length]	Datatype	Description
userLogin[31]	char	Track Owner's CMVC User ID
userName[63]	char	Track Owner's Full Name
userArea[31]	char	Track Owner's Work Area or Department
actual[31]	char	Name of Level where Track is Committed
lastUpdate[25]	char	Date of Last Update
defectPrefix[31]	char	Defect or Feature Prefix
defectAbstract[127]	char	Defect or Feature Abstract

## Users

### User ID Properties

Fieldname[length]	Datatype	Description
login[31]	char	User's CMVC User ID
name[63]	char	User's Full Name
area[31]	char	User's Work Area or Department
address[159]	char	Mailing Address
addDate[25]	char	Date Created
dropDate[25]	char	Date Deleted
lastUpdate[25]	char	Date of Last Update
superUser[3]	char	Superuser Privilege ('yes' or 'no')

## VerifyView

### Verification Record Properties

Fieldname[length]	Datatype	Description
defectName[31]	char	Defect or Feature Identifier
state[15]	char	Verification Record State
addDate[25]	char	Date Created
userLogin[31]	char	Verification Record Owner's CMVC User ID
userArea[31]	char	Verification Record Owner's Work Area or Department
type[15]	char	'original' or 'duplicate'
userName[63]	char	Verification Record Owner's Full Name
defectAbstract[127]	char	Defect or Feature Abstract
lastUpdate[25]	char	Date of Last Update
defectPrefix[31]	char	Defect or Feature Prefix
defectReference[31]	char	Defect or Feature Reference
compName[63]	char	Component Name



---

# Glossary

Glossary terms are defined as they are used in this manual. If you cannot find the term for which you are looking, refer to the *IBM Dictionary of Computing*, SC20-1699.

## A

**access list.** A CMVC object that controls access to development data. A list of user ID-authority group pairs attached to a component, designating users and the corresponding authority access they are being granted for all objects managed by this component or any of its descendants. It also contains the user ID-authority group pairs designating users who are restricted from performing actions at a specific component.

**action.** A task performed by the CMVC server and requested by a CMVC client. A CMVC action corresponds to issuing one CMVC command.

**approval record.** A status record on which an approver must give an opinion of the proposed file changes required to resolve a defect or implement a feature in a release.

**approver.** A user who approves changes within a specific release.

**approver list.** A list of user IDs attached to a release, representing the users who must approve file changes required to resolve a defect or implement a feature in that release.

**authority.** The right to access development objects and perform CMVC commands. See also *access list*, *base authority*, *explicit authority*, *implicit authority*, *restricted authority*, and *superuser privilege*.

## B

**base authority.** The set of actions granted to a user whenever a user ID is created within a CMVC family.

**base file tree.** The base set of files, associated with a release, to which changes are applied over time. Each committed level or track for a release updates the base file tree for that release.

## C

**change control.** Controlling changes to files by verifying access authority and the files' status prior to checking files in and out.

**child component.** All components in each CMVC family, with the exception of the root component, must be created in reference to an existing component. The existing component is referred to as the parent component, while the new component becomes known as the child component. A parent component can have more than one child component. See also *component*.

**command.** A request to perform an operation or run a program from the command line interface. In CMVC, a command consists of the command name, one action flag, and zero or more attribute flags.

**common file.** A file that is contained in two or more releases and the same version of the file is the current version for those releases. See also *shared file*.

**comparison operator.** An operator used in comparison expressions, such as, > (greater than), < (less than), >= (greater than or equal to), <= (less than or equal to), and = (equal to).

**component.** A CMVC object that simplifies project management, organizes project data into structured groups, and controls configuration management properties. Component owners can control access to development data (see *access list*) and configure notification about CMVC actions (see *notification list*). Components exist in a parent-child hierarchy, with descendent components inheriting access and notification information from ancestor components.

**configuration management.** The process of identifying, managing, and controlling software modules as they change over time.

**corequisite tracks.** Two or more tracks designated as corequisites by a user so that all tracks in the corequisite group must be included as members in the same level. If a track is added to a level then all tracks that have a corequisite relationship with that track must also be included in the same level before the level is committed.

## D

**database.** A systemized collection of data that can be accessed and operated upon by a data processing system for a specific purpose.

**default query.** A database search, defined for a specific CMVC GUI window, that is issued each time the CMVC GUI window is opened. See also *search*.

**defect.** A CMVC object used to formally report a problem. The user who opens a defect is the defect originator.

**delete.** Deleting a development object, such as, a file or a user ID. Certain objects can be deleted only if certain criteria are met. Most objects that are deleted can be re-created.

**delta file tree.** A directory structure representing only those files that have been changed in reference to the tracks included in a specified level.

**destroy.** The only CMVC development object that can be destroyed in CMVC is a file. Destroying a file removes the file record from the database on the CMVC server.

## E

**environment.** A user-defined testing domain for a particular release. Also used as a defect field, in which case it is the environment where the problem occurred.

**environment list.** A CMVC object used to specify environments in which a release should be tested. A list of environment-user ID pairs attached to a release, representing the user responsible for testing each environment. Only one tester can be identified per environment.

**explicit authority.** The ability to perform an action against a CMVC object because you have been granted the authority to perform that action.

**extract.** A CMVC action you can perform on a file, level, or release. A file extraction results in the specified file being copied to the client workstation. A level extraction and release extraction result in copying the files associated with the level or release to a designated workstation.

## F

**family.** A logical organization of related development data. A single CMVC server can support multiple families. The data in one family cannot be accessed from another family.

**family administrator.** A user who is responsible for all nonsystem related tasks for one or more CMVC families such as planning, configuring, and maintaining the CMVC environment and managing user access to those families.

**feature.** A CMVC object used to formally request a functional addition or enhancement. The user who opens a feature is the feature originator.

**file.** A collection of data that is stored by the CMVC server and retrieved by a path name. Any text or binary file used in a development project can be created as a CMVC file. For example, source code, executable programs, documentation, or test cases. See also *common file*, *shared file*.

**fix record.** A status record that is associated with a track and is used to monitor the phases of change within each component that is affected by a defect or feature for a specific release.

**full file tree.** A directory structure representing a complete set of active files associated with a release.

## G

**GID.** A number which uniquely identifies a files' group to the operating system.

**GUI.** The OSF/Motif\*\*-based CMVC graphical user interface program.

## H

**host.** Host node, host computer, or host system.

**host list.** A list associated with each CMVC user ID which indicates the client hosts that can access CMVC and act on behalf of the CMVC user. The list is used by the CMVC server to authenticate the identity of a CMVC client upon receipt of a CMVC command. Each entry consists of a login, a CMVC user ID, and a host name.

## I

**implicit authority.** The ability to perform an action against a CMVC object without being granted explicit authority. This authority is implicitly granted due to object ownership. Contrast with *explicit authority* and *base authority*.

**inheritance.** The passing of configuration management properties from parent component to child component. The configuration management properties that are inherited are access and notification. Inheritance within a component hierarchy is cumulative.

## L

**level.** A collection of tracks which represent a set of changed files within a release. Levels are only associated with releases whose processes include the track and level subprocesses.

**level member.** A track that has been added to a level.

**lock.** Prevent editing access to a file stored within the CMVC development environment so that only one user can make changes to a given file at one time.

**login.** User identification.

## N

**notification list.** A CMVC object allowing component owners to configure notification. A list of user ID-interest group pairs attached to a component, designating users and the corresponding notification interest they are being granted for all objects managed by this component or any of its descendants.

## O

**operator.** A symbol that represents an operation to be done. See *comparison operators*.

**originator.** The user who opens a defect or feature and is responsible for verifying the outcome of the defect or feature on a verification record. This responsibility can be reassigned.

**owner.** The user who is responsible for a CMVC object within a CMVC family, either because they created the object or because they were assigned ownership of that object.

## P

**parent component.** See *child component* and *component*.

**prerequisite tracks.** If a file has been changed to resolve more than one defect or feature then the track referenced by the first change is a prerequisite of the track referenced by the later changes. A track is a prerequisite to another track if:

1. File changes have been checked in, but not committed, in reference to the first track, and
2. One or more of those same files is then checked out, changed, and checked in again in reference to the second track.

**problem tracking.** The process of tracking all reported defects through to resolution and proposed features through to implementation.

**process.** A combination of CMVC subprocesses, configured by the family administrator, that controls the general movement of CMVC objects (defects, features, tracks and levels) from state to state within a component or release. See also *subprocess* and *state*.

## Q

**query.** A structured request for information from a database. For example, search for all defects that are currently in the open state. See also *default query* and *search*.

## R

**release.** A CMVC object that groups all of the files that make up one version of a product.

**restricted authority.** The restriction of a user's ability to perform certain actions at a specific component.

**root component.** The initial component that is created when a CMVC family is configured. All components in a CMVC family are descendants of the root component. Only the root component has no parent component.

## S

**search.** The act of scanning one or more data elements of a set in a database to find elements that have certain properties.

**shared file.** A file that is shared between two or more releases. See also *common file*.

**sizing record.** A status record created for each component-release pair affected by a proposed defect

or feature. The sizing record owner must indicate whether the defect or feature affects the specified component-release pair and the approximate amount of work needed to resolve the defect or implement the feature within the specified component-release pair.

**state.** Tracks, levels, features, and defects all move through various states during their life cycles. An object's current state determines which actions may be performed against it. See also *process* and *subprocess*.

**subprocess.** CMVC subprocesses govern the state changes for CMVC objects. The design, size, review (DSR) and verify subprocesses are configured for component processes. The track, approve, fix, level, and test subprocesses are configured for release processes. See also *process* and *state*.

**superuser privilege.** A user who is granted superuser privilege. Superuser privilege allows a user to perform any action available in the CMVC family.

**Note:** Superuser privilege is internal to CMVC and not related to your operating system.

## T

**test record.** A status record used to record the outcome of an environment test performed for a specific level of a release once the defect is resolved or the feature is implemented.

**tester.** A user responsible for testing the resolution of

a defect or the implementation of a feature for a specific level of a release and recording the results on a test record.

**track.** A CMVC object created to monitor the progress of changes within a release to resolve a specific defect or implement a specific feature.

## U

**UID.** A number which uniquely identifies a login on a host and controls the ownership of a file within the file system.

**user.** A person with an active user ID and access to one or more CMVC families.

## V

**verification record.** A status record which must be marked by the originator of a defect or a feature before the defect or feature can move to the closed state. This allows the originator to verify the resolution or implementation of the defect or feature they opened.

**version control.** The storage of multiple versions of a single file along with information about each version.

**view.** An alternative and temporary representation of data from one or more tables.



---

# Please Tell Us What You Think!

IBM Configuration Management Version Control  
Commands Reference  
Version 2 Release 2  
Publication No. SC09-1635-01

We hope you found this book useful and informative. If you like what we've done, please let us know; if not, please tell us why. We'll use your comments to make the book better.

Please use one of the methods listed below to send your comments to IBM. Whichever method you choose, make sure you send your name, address, and telephone number to receive a reply.

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 obligation.

- To send comments by mail or fax, use the form titled "What Do You Think?" on the following page.

If you're mailing from a country other than the United States, you can give the form to the local IBM branch office or IBM representative for postage-paid handling.

To fax the form, use this number: (919) 469-7718.

- To send comments electronically, use one of the following network IDs:

**IBM Mail Exchange    USIB5DNQ at IBMMAIL**  
**Internet                KFRYE@CARVM3.VNET.IBM.COM**

Thank you! Your comments help us make the information more useful for you.




# What Do You Think?

**IBM Configuration Management Version Control  
Commands Reference  
Version 2 Release 2  
Publication No. SC09-1635-01**

We're in business to satisfy you. If we're succeeding, please tell us; if not, let us know how we can do better.

## Overall, how satisfied are you with this book?

	Very Satisfied	Satisfied	Neither Satisfied nor Dissatisfied	Dissatisfied	Very Dissatisfied	No Opinion
Overall satisfaction						

## How satisfied are you that the information in this book is:

Accurate						
Complete						
Easy to find						
Easy to understand						
Well organized						
Applicable to your tasks						

## Please tell us how we can improve this book:

---

---

---

---

---

---

---

---

---

---

May we contact you to discuss your responses?  Yes  No ☎ \_\_\_\_\_

Fax \_\_\_\_\_

Note that IBM may use or distribute the responses to this form without obligation.

\_\_\_\_\_  
Name

\_\_\_\_\_  
Address

\_\_\_\_\_  
Company or Organization

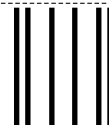
\_\_\_\_\_  
Phone No.



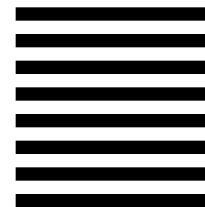
Fold and Tape

Please do not staple

Fold and Tape



NO POSTAGE  
NECESSARY  
IF MAILED IN THE  
UNITED STATES



# BUSINESS REPLY MAIL

FIRST-CLASS MAIL PERMIT NO. 40 ARMONK, NEW YORK

POSTAGE WILL BE PAID BY ADDRESSEE

International Business Machines Corporation  
Information Development  
Department T45  
PO Box 60000  
Cary, NC 27511-8519



Fold and Tape

Please do not staple

Fold and Tape





Printed in U.S.A.

SC09-1635-01

